LEARNING TO COPE WITH SMALL NOISY DATA IN SOFTWARE EFFORT ESTIMATION

by

LIYAN SONG

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
January 2019

Abstract

Though investigated for decades, Software Effort Estimation (SEE) remains a challenging problem in software project management. However, there are several factors hindering the practical use of SEE models. One major factor is the scarcity of software projects that are used to construct SEE models due to the long process of software development. Even given a large number of projects, the collected effort values are usually corrupted by noise due to the participation of humans. Furthermore, even given enough and noise-free software projects, SEE models may have sensitive parameters to tune possibly causing model sensitivity problem.

The thesis focuses on tackling these three issues. It proposes a synthetic data generator to tackle the data scarcity problem, introduces/constructs uncertain effort estimators to tackle the data noise problem, and analyses the sensitivity to parameter settings of popular SEE models. The main contributions of the thesis include:

1. Propose a synthetic project generator and provide an understanding of when and why it improves prediction performance of what baseline models.
2. Introduce relevance vector machine for uncertain effort estimation.
3. Propose a better uncertain estimation method based on an ensemble strategy.
4. Provide a better understanding of the impact of parameter tuning for SEE methods.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Prof. Xin Yao, who has shown his constructive supervision, patience, and substantial support throughout my PhD. He has rarely given me answers but enlightened me with questions; he has guided me away from big wrongs but allowed for small faults and helped me learn from them. His ways and attitudes of doing research and his personality have great influence on mine. I would also like to present my greatest thankfulness to my co-supervisor Dr. Leandro Lei Minku, who has enlightened me on research topics, gave me valuable guidance on overcoming difficulties throughout my PhD, and educated me with concrete research skills. We have many intensive discussions. Without their supervision and encouragement, this thesis would never have been completed.

Many thanks to the thesis group members (John Bullinaria, Rami Bahsoon, and Leandro Minku) for their helpful instructions and discussions. Many thanks also to Machine Learning Reading Group members: Ata Kaban, Peter Tino, Yuan Shen, Fengzhen Tang, Luca Rossi, Momodou Sanyan, Frank-Michael Scheleif, and other PhD fellows for the three-year interesting and in-depth discussions on various research topics. Special thanks to Dr. Haiping Lu for his nice and helpful supervision in HKBU leading to a fruitful year. These experience has illuminated my truly passion and enthusiasm for doing research, which I would treasure as a lifelong career.

I would like to show my special gratitude to my close friend Dr. Fengzhen Tang for her companion and lots of discussions on my daily life and research work. I am also grateful

to Xiaoning Shen, Jiangshan Yu, Yuan Shen, Luca Rossi, Xiaofen Lu, William Zhong, Lukas Adam, Roya Soufi, Ulrike Brauer, Silvia Umiliacchi, Klulood AlYahya, Yini Wang, Hans Hofland, Michael Wang, Michael LoCicero, Dan Liu, Catherine Harris, Vladislav Kramarev, and many other colleagues and friends for the interesting conversations and their nice help during this long and painful journey. I have so many first-time in life spending with them. All these experiences and memories have made up my personality.

Many thanks to Qing Bao, Qiquan Shi, Yang Zhou, Shuyi Zhang, Bo Yuan, Yunwen Lei, and many other friends and colleagues in HKBU and SUSTech. They encouraged me to go through the challenging time, making my life relaxing and colourful.

I should show my deepest gratitude to my parents, who give me unconditional love and forever encouragement and support. If it were not for them, I would be a very different and nasty person. I should also show my great thankfulness to my husband Yunsheng Kuang for being at my back for many years. Lastly, I have to show my gratitude to my younger self. She used to be troublesome, stubborn, and erratic, which makes it much easier to grow better.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

k-NN	k-Nearest Neighbours
ANN	Artificial Neural Network
CDF	Cumulative Distribution Function
CI	Confidence Interval
CL	Confidence Level
CV	Cross validation
LOC	Line Of Code
LSD	Logarithmic Standard Deviation
MAE	Mean Absolute Error
MdAE	Median Absolute Error
MdMRE	Median Magnitude of Relative Error
ML	Machine learning
MLR	multivariate linear regression
MMRE	Mean Magnitude of Relative Error
MRE	Magnitude of Relative Error
PDF	Probabilistic Distribution Function
PI	Prediction Interval
RT	Regression Tree
RVM	relevance vector machine
SA	Standardised Accuracy
SDP	Software Defect Prediction

SEE	Software effort estimation
SMOTE	Synthetic Minority Over-sampling Technique
STD	Standard Deviation
SVR	Support Vector Regression

CHAPTER 1

Introduction

Software Effort Estimation (SEE) is the process of predicting the effort (e.g. in person-month) required to develop a software project. From the Machine Learning (ML) viewpoint, SEE is a regression problem with continuous output values. The examples of input features are *software development type*, *staff team skill*, and *functional size*. It often takes place in the early stage of software development and continues throughout the entire progress. Based on it, the project managers make important decisions such as the budget, the subsequent planning and control, and the bidding price [19]. Good effort estimation is important in software project management. Both overestimation and underestimation can cause serious problems: overestimation may result in a company losing bids for contracts or wasting resources, while underestimation may lead to poor product quality, delay or even unfinished software systems, and unsatisfied customers [187, 49, 49].

Though investigated for decades, SEE remains a weak link in software project management. Project managers have mainly relied on expert judgement to make effort estimation based on the experience of the experts who are familiar with the development of software applications [83, 74]. This is partially due to the inconclusive evidence that whether model-based methods can produce more accurate estimation than expert judgement [85].

Expert-based methods suffer from several problems: they are not repeatable; the process of deriving estimation is not explicit; they contain personal bias and are often sensitive to political pressure; it is costly to find experienced experts for every new project.

In the past three decades, many model-based SEE methods have been developed in an attempt to circumvent the problems of expert-based methods, where some formal models or ML methods are proposed to elaborate the completed software projects for predicting the effort of new software projects [131, 187, 49]. Popular model-based SEE methods include Boehm's COCOMO [26], linear regression [99, 162], regression tree [136], neural network [7], and ensembles of learning machines [110, 136]. They have the advantages of being repeatable, objective, efficient, and giving better understanding of the estimation process [187, 49, ~~49,~~ 89, 187]. Model-based SEE methods can be used as decision support tools for the project managers to assist expert judgement (rather than replace it). They can provide additional information based on which the experts can justify and criticize their estimation [74, 83]. This thesis concentrates on model-based SEE methods.

Despite loads of research papers, the practical use of SEE methods is still not popular, and SEE remains a challenging problem both for researchers and project managers [83, 85]. There are several factors hindering their practical use [102, 66]. One major factor is the scarcity of software projects that are used to construct SEE models due to the long process of software development. Even given a large number of software projects, the collected effort values are usually corrupted by data noise due to the participation of humans. And even given enough and noise-free software projects, SEE models usually have tuning parameters possibly causing model sensitivity problem.

This thesis proposes a synthetic project generator to tackle the data scarcity problem, introduces and proposes uncertain SEE methods to tackle the data noise problem, and provides a systematic analysis on the sensitivity to parameter settings of popular SEE methods. We list the three issues mainly investigated in the thesis as below.

1. Synthetic project generation. This study provides an alternative and much cheaper way to alleviate the data scarcity problem than proposing advanced SEE methods.
2. Uncertain effort estimation. This study tackles the data noise problem and supports more informative decision making by providing uncertain prediction. This study is divided into two steps: (1) introduce an existing ML method that can provide uncertain effort estimation and is suitable for SEE and (2) propose an ensemble strategy based on this baseline method to improve uncertain prediction performance.
3. Sensitivity to parameter settings of SEE methods. This study draws the attention to formal and fair parameter tuning in the SEE community. It can also reveal a possible reason for unsatisfactory prediction performance of SEE methods in view of inadequate parameter tuning.

More details on the research questions and methodologies presented in the thesis will be discussed in section 1.1 and chapters 3, 4, 5, and 6.

This chapter is organized as follows. Section 1.1 discusses the research questions answered by the thesis. Section 1.2 formulates the issues of SEE, including the training and predicting processes, uncertain effort estimation, and the data augmentation for SEE. Section 1.3 summarizes the contributions and the organisation of the thesis.

1.1 Research Questions

This section explains the research questions answered by the thesis and their motivations. More detailed information and the methodology to tackle each research question will be given in chapters 3, 4, 5, and 6 respectively.

1.1.1 Synthetic Data Generation

The first research question answered by the thesis is:

> *RQ1. Can we generate synthetic software projects to enlarge the training set size for obtaining better prediction performance? If so, how?*

One of the core challenges of SEE is the lack of software projects due to the expensive or long process of data collection [169, 49, 106, 107, 116]. Consequently, companies usually have small numbers of completed projects to construct their SEE models, leading to unsatisfactory prediction performance since the information contained in such small data probably cannot support appropriate training of SEE models [69, 182, 106].

Existing literature in SEE has frequently attempted to address this problem by creating advanced models [116, 136, 109]. Rather than introducing sophisticated SEE models or collecting as many completed software projects as possible, an alternative and much cheaper way to tackle this issue is to augment data sets by generating synthetic projects based on the completed ones. However, little work has been done to investigate such strategies to alleviate the data scarcity problem in SEE literature.

The answer to RQ1 will provide a novel data augmentation method and investigate how well it tackles the data scarcity problem of SEE. The proposed approach should be general so that it can be used with any state-of-the-art SEE method. Ideally, it should be simple and hardly have negative effect on SEE performance.

Chapter 3 proposes our synthetic project generation approach. Experimental results show that our synthetic projects can improve the performance of SEE methods especially when the training examples are insufficient; when the generated projects cannot improve the performance, they are not detrimental either. Therefore, the proposed data generation approach is helpful in alleviating the data scarcity problem of SEE.

1.1.2 A Bayesian Approach for Uncertain Effort Estimation

The second research question answered by this thesis is:

RQ2. Is there any ML approach that can provide uncertain effort estimation? How well can it perform in terms of point and uncertain prediction?

Most SEE models produce only point effort estimation, i.e., a single estimate of the effort required to develop a given software project [187, 49]. However, there are several sources of uncertainty in the context of SEE; simply relying on point estimation may ignore these uncertain factors and lead project managers to wrong decision-making [174, 91, 84].

Besides providing a point estimate, SEE methods should ideally support the handling of uncertainty by accessing the probability of falling within a specific interval consisted of a minimum and maximum effort values, namely Prediction Interval (PI). The certainty on this PI can be characterized by a Confidence Level (CL). Such uncertain effort estimation presents more reasonable representation of reality, potentially helping project managers to make better informed decisions and enabling more flexibility in these decisions [174]. However, just a few studies have been developed for uncertain effort estimation methods.

The answer to RQ2 will provide the first step to tackle the uncertain effort estimation task by introducing and evaluating an existing ML model that may be suitable for SEE. More advanced uncertain estimation method can be built on this baseline.

Chapter 4 introduces a Bayesian regression model, namely Relevance Vector Machine (RVM), to the context of SEE for tackling the uncertain effort estimation task straightly. RVM explicitly models effort noise and can provide uncertain estimation. Experimental results show that RVM can achieve competitive performance in terms of point prediction; statistical analyses on its uncertain estimation shows that the derived PIs can usually cover the actual effort values. Therefore, RVM is a very promising method for SEE and should be further exploited.

1.1.3 Synthetic Bootstrap Ensemble for Uncertain Estimation

The third research question answered by the thesis is:

> *RQ3. Can we improve the prediction intervals of RVM? How? How well does this method perform compared to state-of-the-art point/uncertain methods?*

Recently, uncertain effort estimation has attracted increasing attention in the SEE community [97, 103]. PIs with CLs can be considered as a representative format for uncertain effort estimation, which allow for risk management and provide more flexibility to project managers. Chapter 4 introduces RVM to the context of SEE and evaluates its prediction performance: RVM is competitive to other SEE methods in terms of point prediction, but the derived PIs with CLs are sometimes too wide to be informative.

The answer to RQ3 will provide a more advanced uncertain effort estimation approach based on RVM for better PIs with CLs: on the one hand, the PIs with CLs should be wide enough to capture the actual effort values of many software projects; on the other hand, they should be sufficiently narrow to be informative and of practical use.

Chapter 5 proposes a novel uncertain effort estimation method, namely *Synthetic Bootstrap ensemble of Relevance Vector Machines* (SynB-RVM). SynB-RVM adopts Bootstrap resampling technique to construct multiple RVMs based on modified Bootstrap training bags whose replicated training examples are replaced by their synthetic counterparts. Experimental results show that when used as a point estimator, SynB-RVM can either significantly outperform or perform similarly compared to state-of-the-art SEE methods. When used as an uncertain estimator, SynB-RVM can achieve significantly narrower (and thus more informative) PIs with CLs compared to its baseline RVM. SynB-RVM is superior to other uncertain methods in term of point/uncertain prediction measured in at least one performance metric.

1.1.4 Sensitivity to Parameter Setting of SEE Methods

The fourth research question answered by the thesis is:

> *RQ4. To what extent do parameter settings affect the performance of SEE methods, and should we pay attention to their parameter tuning?*

Most SEE studies involve comparisons among different methods, which usually have more than one parameter that need to be tuned [165, 89, 187]. However, as explained by Minku and Yao [136], the methodology used to choose parameter settings is frequently omitted from the experimental framework of SEE literature, seemingly making an implicit assumption that parameter settings would not change the outcomes significantly. There has been little work investigating the impact of parameter settings for SEE methods.

The answer to RQ4 will provide a better understanding of the impact of parameter tuning for SEE methods and reveal the possible reasons for unsatisfactory prediction performance of some model-based SEE methods in view of being absent or unfair parameter tuning in their performance evaluation.

Chapter 6 proposes an analysis methodology and performs systematic experiments to investigate to what extent parameter settings affect the performance of SEE methods. Experimental results show that different SEE methods have different sensitivity to their parameter settings. For instance, regression tree and Bagging ensembled with regression trees are not very sensitive to their parameter settings, while multilayer Perceptron and Bagging ensembled with multilayer Perceptrons are extremely sensitive to their parameter settings. Thus, the ways of parameter settings should be formally reported in the experimental framework of SEE literature.

It is worth noting that the thesis does not intend to address all problems about parameter tuning of SEE methods, but to encourage studies to use principled schemes for tuning the parameters. Developing automatic tuning methods is considered as future work.

1.2 Formulation of the Problem

This section formally introduces the training and prediction processes of point and uncertain SEE methods. It also formulates the data generation process for SEE.

1.2.1 SEE: A Regression Problem

From the ML viewpoint, SEE is a regression problem with continuous output values. SEE methods utilize the accomplished software projects with the collected effort (training examples) to predict the effort of the future unknown ones (testing examples).

Denote the training set of N software projects as $\mathscr{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N$, where $\mathbf{x}^n = [x_1^n, \cdots, x_D^n]^T \in \mathbb{R}^D$ is the n^{th} training example consisting of D features[1], and $y^n \in \mathbb{R}^1$ is the effort for developing this software. Examples of input features $\{x_1, \cdots, x_D\}$ are *software development type*, *hardware platform*, *user interface*, *tool use*, *team expertise quality requirements*, *staff tool skills*, *staff team skill*, *functional size*, and so on.

Denote $\mathscr{F}$ as a set of learning machines such as regression trees and neural networks and $\boldsymbol{\theta}$ as their model parameters. The training process of point effort estimation aims to determine the optimal function $f(\cdot, \boldsymbol{\theta}^*) \in \mathscr{F}$ from input features to output effort as

$$\begin{aligned} \mathscr{F} &: \mathbb{R}^D \rightarrow \mathbb{R}^1 \\ f(\mathbf{x}; \boldsymbol{\theta}^*) &= \hat{y}, \end{aligned} \tag{1.1}$$

by minimizing a loss function $\mathscr{L}(\cdot)$ with respect to model parameter $\boldsymbol{\theta}$ as

$$\mathscr{L}(\mathscr{D}) = \sum_{n=1}^{N} (||f(\mathbf{x}^n; \boldsymbol{\theta}) - y^n||_*). \tag{1.2}$$

If $||\cdot||_1$ is employed, $\mathscr{L}(\mathscr{D})$ is the mean absolute error, being a popular instantiation of $\mathscr{L}$ in SEE. More discussion on its implementation can be found in section 2.4.2. The training process determines the optimal model parameter $\boldsymbol{\theta}^*$.

[1]Categorical features can be converted into real-valued ones, and thus can be represented as in $\mathbb{R}^D$.

For a testing software project $(\mathbf{x}, y)$, the prediction process aims to obtain an estimated effort value $\hat{y}$ based on the input features $\mathbf{x}$ using the constructed SEE model as

$$\hat{y} = f(\mathbf{x}; \boldsymbol{\theta}^*), \tag{1.3}$$

hoping for small deviation between y and $\hat{y}$ as $\mathscr{L}(||y - \hat{y}||)$.

In practice, we usually align all training examples into the matrix as

$$\mathbf{X} = [\mathbf{x}^1, \cdots, \mathbf{x}^N]^T \in \mathbb{R}^{N \times D}, \tag{1.4}$$

where T denotes the transpose and the comma (semicolon) in $[u, v]$ ($[u; v]$) represents to concatenate variable u and v in row (column). The matrix of training examples contains all training information for building an SEE method with respect to a loss function.

1.2.2 Uncertain Effort Estimation

The training process of uncertain effort estimation aims to determine the optimal function $f(\cdot, \boldsymbol{\theta}^*) \in \mathscr{F}$ with the modelling of effort noise/uncertainty ε as

$$\begin{aligned} \mathscr{F} : \mathbb{R}^D &\rightarrow \mathscr{P} \\ f(\mathbf{x}; \boldsymbol{\theta}^*) + \varepsilon(\boldsymbol{\beta}^*) &= p(\hat{y}), \end{aligned} \tag{1.5}$$

by minimizing a loss function $\mathscr{L}$ with respect to $\boldsymbol{\theta}$ and the parameter of the effort noise modelling $\varepsilon(\boldsymbol{\beta})$. Here, $\mathscr{P}$ denotes the distribution space of the effort values. The modelling assumes that effort noise has zero mean and is addictive to and independent of input features. The training process determines the optimal parameters $\boldsymbol{\theta}^*$ and $\boldsymbol{\beta}^*$.

For a testing software project $(\mathbf{x}, y)$, the testing process aims to estimate Probability Distribution Function (PDF) of effort y as

$$p(\hat{y}) \sim \mathscr{F}(\mathbf{x}; \boldsymbol{\theta}^*, \boldsymbol{\beta}^*). \tag{1.6}$$

The value with the highest probability can be used as the point estimation.

1.2.3 Data Augmentation in SEE

SEE usually suffers from the data scarcity problem due to the expensive or long process of data collection. As a result, companies usually have limited training examples for effort estimation, causing unsatisfactory prediction performance of SEE models. Rather than introducing sophisticated SEE models or collecting as many completed software projects as possible, we can generate synthetic projects based on the completed projects $\mathscr{D}$, and augment the SEE data set as

$$\mathscr{D}^{(new)} = \mathscr{D} \bigcup \mathscr{D}^{(syn)}, \tag{1.7}$$

where $\mathscr{D}^{(syn)}$ consists of the generated synthetic projects with the size $\lceil \gamma N \rceil$, $\gamma > 0$ is a predefined *synthetic ratio*, and $\lceil \cdot \rceil$ denotes the upward rounding operator (e.g. $\lceil 1.4 \rceil = 2$). The *synthetic ratio* γ should be small enough not to hinder the information provided by the real projects, but large enough to be helpful. Hereafter, the model training would be based on $\mathscr{D}^{(new)}$.

Given a randomly chosen training example $(\mathbf{x}, y)$ where $\mathbf{x} = [x_1; \cdots; x_D]$, the generation of a synthetic project can be formulated as

$$\begin{aligned} \mathscr{G} : (\mathbb{R}^D, \mathbb{R}) &\rightarrow (\mathbb{R}^D, \mathbb{R}) \\ \left(\begin{bmatrix} g_1(x_1; \boldsymbol{\sigma}) \\ \cdots \\ g_D(x_D; \boldsymbol{\sigma}) \end{bmatrix}, g_{D+1}(y; \boldsymbol{\sigma}) \right) &= (\mathbf{x}^{(syn)}, y^{(syn)}), \end{aligned} \tag{1.8}$$

where $\mathscr{G}$ denotes the process of generating a synthetic project, $\mathbf{g} = [g_1, \cdots, g_D, g_{D+1}] \in \mathscr{G}$ are the generation functions, and $\boldsymbol{\sigma}$ denotes the parameters of the synthetic generator. Detailed discussion on the realization of the synthetic project generation will be discussed in section 3.2.

1.3 Thesis Contributions and Organisation

The contributions of the thesis (and its organisation) are:

1. We propose a synthetic data generator to alleviate the data scarcity problem of SEE and investigate when and why our synthetic projects can improve the prediction performance of the baseline methods. This is the answer to RQ1 of section 1.1.1 and will be presented in chapter 3, which is based on our paper [*3].
2. We investigate the impact of SEE data set sizes on prediction performance. This is a by-product for answering RQ1 of section 1.1.1 and will be discussed in the second last subsection of section 3.4.1, which is based on our paper [*3].
3. We introduce RVM to the context of SEE for uncertain effort estimation and provide the ways of deriving PIs with CLs for software projects. This is the answer to RQ2 of section 1.1.2 and will be presented in chapter 4, which is based on our paper [*2].
4. We conduct a comprehensive review and categorization for uncertain effort estimators. This is a by-product for answering RQ2 of section 1.1.2 and RQ3 of section 1.1.3 and will be presented in section 2.2, which is based on our paper [*4].
5. We propose a better uncertain effort estimator based on Bootstrap ensemble of RVMs and perform a thorough comparison against other uncertain SEE methods. To the best of our knowledge, this is the most thorough experimental comparison on this topic. This is the answer to RQ3 of section 1.1.3 and will be presented in chapter 5, which is based on our paper [*4].
6. We systematically evaluate the sensitivity to parameter settings based on statistical analyses and gain a better understanding of the impact of parameter tuning in the context of SEE. This is the answer to RQ4 of section 1.1.4 and will be presented in chapter 6, which is based on our paper [*1].

Overall, the main contributions of the thesis are the answers to RQ1 to RQ4, in particular the proposal of a synthetic project generator (chapter 3), the introduction/proposal of uncertain SEE methods (chapters 4 and 5), and the investigation on sensitivity to parameter settings of SEE methods (chapter 6). The thesis also contains chapter 2 with background knowledge and literature review and chapter 7 with conclusions and future work.

Some contents of this thesis are contained in the following papers:

- [*1] **Liyan Song, Leandro L. Minku, and Xin Yao (2013).** *The Impact of Parameter Tuning on Software Effort Estimation Using Learning Machines.* International Conference on Predictor Models in Software Engineering (PROMISE'13), USA, pp. 9:1-9:10.

- [*2] **Liyan Song, Leandro L. Minku, and Xin Yao (2014).** *The Potential Benefit of Relevance Vector Machine to Software Effort Estimation.* International Conference on Predictor Models in Software Engineering (PROMISE'14), Italy, pp. 52-61.

- [*3] **Liyan Song, Leandro L. Minku, and Xin Yao (2018).** *A Novel Automated Approach for Software Effort Estimation based on Data Augmentation.* ACM Symposium on the Foundations of Software Engineering (FSE), USA.

- [*4] **Liyan Song, Leandro L. Minku, and Xin Yao (2018).** *Software Effort Interval Prediction via Bayesian Inference and Synthetic Bootstrap Resampling.* ACM Transactions on Software Engineering and Methodology (TOSEM).

CHAPTER 2

Background and Literature Review

Many papers have been published in the area of SEE, most of which concentrate on proposing models that can provide good point prediction [187, 49, 187]. However, the majority of these studies have ignored three practical issues of SEE. First, the collection of software projects for constructing SEE models usually takes time due to the long process of software development, leading to the data scarcity problem. Thus, modern methods that can generate synthetic data to augment the training set should be reviewed, which can alleviate the small data problem of SEE. Second, given large amount of training examples, the collected effort is highly likely to be contaminated by noise due to the participation of humans in the data collection process. Without providing uncertain effort prediction to reflect such noise, the estimates could be misleading and the project managers can make inadequate decisions. Thus, uncertain effort estimators methods should be reviewed. Finally, the sensitivity of SEE methods to their parameter settings is essential to gain a better understanding of the methods. Thus, such studies should be reviewed to frame the most recent outcomes.

This chapter reviews SEE literature on the three issues and provides background knowledge of SEE data sets and performance metrics. This chapter is organized as follows.

Section 2.1 discusses data generation methods for catering small data problem, which is related to RQ1 of the thesis. Section 2.2 discusses uncertain effort estimators, which is related to RQ2 and RQ3 of the thesis. Section 2.3 presents previous studies in SEE about the sensitivity to parameter settings of SEE methods, which is related to RQ4 of the thesis. Section 2.4 describes performance evaluation for SEE including the data sets investigated in the thesis and the performance metrics for point and uncertain effort estimation. This chapter is summarised in section 2.5. Discussion on point effort estimation methods can be found in appendix A. These studies are related to the thesis as background knowledge and are used in the experimental comparisons against our proposed methods.

2.1 Data Augmentation for Small Data Problem

Data augmentation technique is data-level that adjusts the number of training examples and their distribution directly. For regression, it has the potential to enlarge the training set to alleviate the small data problem; for classification, it increases the number of minority class to alleviate the imbalance between minority and majority classes.

Section 2.1.1 discusses the data augmentation techniques for classification. Section 2.1.2 discusses software defect prediction that is a typical example of imbalance classification in the context of software engineering. The two subsections are discussed for being the most relevant to data augmentation for SEE and being inspiring for our approach. Section 2.1.3 discusses data augmentation techniques for regression. Finally, section 2.1.4 discusses the existing literature for SEE.

It is noteworthy that the approaches discussed in sections 2.1.1 and 2.1.2 are for classification problem with discrete output labels (e.g. defect-prone or non-defect), but SEE is a regression problem with continuous output values. Thus, the data augmentation methods designed for classification cannot be directly used for SEE since by nature there are no minority or majority classes in regression.

2.1.1 Data Augmentation in ML Classification

There have been many studies in classification that augment the data set size for better prediction performance. Imbalance classification is a typical example, where the difference between the numbers of training examples in different categories is huge [71]. One challenge of learning from imbalance data is that the classifiers would often predict a new example with the majority label though its label should equal to the minority [186].

Data oversampling is a popular and effective way to tackle the imbalance classification problem, where synthetic data is generated and added to the minority class for a more balanced data set [39, 58]. There are several ways to enlarge the minority class.

Random oversampling

It is a non-heuristic method that makes replicates of randomly selected training examples and adds them to the original training set. The degree of class distribution can be adjusted to any desired level. It is simple and sometimes effective, but it has been argued that exact copies can lead to overfitting [39, 20].

Introduction of Gaussian Noise

Lee [119, 120] proposed an easy oversampling technique that generated synthetic minority data by injecting small Gaussian noise to the features of training examples of the minority class. The number of synthetic data and the covariance matrix of multivariate Gaussian noise were predefined in the paper (without explanation). The experimental result demonstrated the effectiveness of the generated data in improving the classification performance in terms of ROC area and Kullback-Leibler, probably due to a regularization induced by the synthetic examples of the minority class for the classification model.

Algorithm 1 Classical SMOTE algorithm [39].

1: **Input:** minority training class $\mathscr{D}_{min}$, SMOTE rate τ, and the k in k-NN.
2: **Aim:** Generate $m = \lceil |\mathscr{D}_{min}| * \tau \rceil$ synthetic minority examples to to alleviate the imbalance problem, where $\lceil \cdot \rceil$ denotes the ceiling integer.
3: **Procedures:**
4: For a randomly chosen minority example $\mathbf{x} \in \mathscr{D}_{min}$
5: (1) Compute its k-NN set $\{\mathbf{x}^s\} \subset \mathscr{D}_{min}$, where $s = \{1, \cdots, k\}$.
6: (2) Randomly choose a neighbour $\mathbf{x}^s$.
7: (3) Compute difference vector $\mathbf{z} = \mathbf{x}^s - \mathbf{x}$.
8: (4) Multiply the value on each dimension of $\mathbf{z}$ by a random value $\sigma_d \in [0, 1]$, and denote the result as $\mathbf{z}'$.
9: (5) $\mathbf{x}^{(syn)} = \mathbf{x} + \mathbf{z}'$.
10: (6) Repeat step (1)$\sim$(5) until generating synthetic minority examples.
11: **Output:** The generated M synthetic examples.

Synthetic Minority Oversampling Technique

Synthetic Minority Oversampling Technique (SMOTE) is an advanced and probably the most popular oversampling method showing great success in many applications [92, 39]. It generates synthetic data of the minority class based on the similarities between them. Specifically, to create a synthetic example, SMOTE randomly selects one of the k-nearest neighbours in the feature space of a minority example $\mathbf{x}$, denoted by $\mathbf{x}'$, calculates the difference between them, and then adds the difference vector where each dimension is multiplied by a random number $\sigma \in [0, 1]$ to $\mathbf{x}$. It is a point along the line between $\mathbf{x}$ and $\mathbf{x}'$. By doing so for certain rounds, it forces the decision boundaries of the minority class to spread towards the majority class effectively. Algorithm 1 summarizes SMOTE. The oversampling rate τ and the number of nearest neighbours k are pre-defined parameter. There are many variants of SMOTE algorithms, some of which are discussed as follows.

Borderline-SMOTE

Borderline-SMOTE [68] is a modification of SMOTE. It assumes that the examples near classification boundaries are more likely to be misclassified and thus more important. Only borderline examples are used to generate new synthetic data by applying SMOTE.

Adaptive Synthetic Sampling

The main idea of adaptive synthetic sampling [70] is to generate minority class examples adaptively according to their distributions, where more synthetic examples are created for 'difficult' ones than 'easy' ones. Similar to borderline-SMOTE, 'difficult/easy' is determined by the neighbourhood of a minority class examples. Specifically, for each training example $\mathbf{x}$ in the minority training set, this technique calculates the proportion of majority class examples in the k-NN set of $\mathbf{x}$, denoted by $\Delta_{\mathbf{x}}$. Normalize $\Delta_{\mathbf{x}}$ so that $\sum_{\mathbf{x}} \Delta_{\mathbf{x}} = 1$. Then, the number of synthetic examples that need to be generated for $\mathbf{x}$ equals to $\Delta_{\mathbf{x}} * T$, where T is the desired number of synthetic examples. Intuitively, Δ describes an empirical density distribution that decides the number of synthetic examples.

Majority Weighted Minority SMOTE

Similar to adaptive synthetic sampling [70], the main idea of Majority Weighted Minority SMOTE [18] is to identify the hard-to-learn informative minority class samples and assign weights individually according to their Euclidean distance from the nearest majority class samples. The synthetic examples are then generated from the weighted informative minority class samples using a clustering approach.

2.1.2 Data Augmentation in Software Defect Prediction

Software Defect Prediction (SDP) is a typical example of imbalance classification in the context of software engineering, since the defect samples are much less likely to happen than the non-defect ones. This subsection is discussed for being among the most related to the synthetic data generation for SEE task in the context of software engineering.

Data imbalance usually undermines the SDP methods [186, 148], where the defect predictors often rarely predict the faulty modules. 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 [186, 92, 148, 185]. For instance, some studies [92, 148, 185]

employed data augmentation methods, such as random oversampling that reproduces the data of minority class randomly or SMOTE [39] that produces new data based on k-NN, to enlarge the data set size of minority class. Their experimental studies showed promising or better effect of the synthetic data in performance improvement. Drown et al. [54] proposed a genetic algorithm-based data augmentation method, which outperformed the baseline predictor without the synthetic data as well as other augmentation methods.

2.1.3 Data Augmentation in ML Regression

Despite many studies on data augmentation for classification, there have been few studies for regression. It may be due to the difficulty in defining minority and majority values for regression problem.

SMOTE-like data augmentation for regression, usually namely *utility-based regression* or *imbalance regression*, is the mainstream, which assumes that the distribution of the training examples is imbalanced and bias towards those that are not in accordance with the user's preference [153, 179, 178, 16]. In other words, the minority examples are assumed to be those with small probability of the data distribution.

Specifically, Torgo et al. [179, 178] adapted SMOTE from classification to regression, namely SMOTER, which quantified the minority examples by an *utility function* that was determined by the users or distributions of training examples. For instance, the *utility* of a training example is assigned to be inverse proportional to the density of training examples surrounding it. This training example belongs to the minority or majority class is determined by a predefined threshold. Their experimental results showed better performance than the situations without synthetic data in terms of *precision and recall* defined for regression in [153]. Later in 2017, Branco et al. [16] combined SMOTER with random under-sampling strategy that might achieve better performance. Their experimental results showed further performance improvement in terms of F_1^ϕ defined for regression

in [30], when combined with the base learners random forest or multivariate adaptive regression splines.

2.1.4 Data Augmentation in SEE

To our best knowledge, there has been only one work in the SEE community that tackles the data scarcity problem by data augmentation [93].

The augmentation method extended SMOTE [39] from classification to regression by attributing class imbalance from the most predictive SEE feature with the following procedures. First, Pearson correlation between each feature and effort is calculated, and the one with the largest correlation is considered to be the most predictive, which is usually a size-related feature or estimation of completion date or effort such as *functional size* and *line of codes*. Then, the entire training examples are cast into three classes (i.e. small, medium, and large) with similar amount of data size. For instance, if the functional size has the largest correlation to the effort with its minimum and maximum being 60 and 780 respectively, the entire data set would be divided into 3 parts according to the feature value of functional size as [60,300), [300,540) and [540,780). Finally, the conventional SMOTE was used to generate synthetic projects to small and medium classes to balance the data distribution. The entire data set size was thus increased. These synthetic projects together with the real ones were passed to analogy-based estimation method (i.e. k-NN) for the purpose of getting better prediction performance.

Their experiments showed promising results based on Desharnais data set in terms of MMRE and Pred(25): their proposed data augmentation approach can play none or sometimes slightly better effect on the baseline model to enhance the prediction performance. However, no statistical tests were conducted, leaving the significance of performance improvement obscure. We suspect that the improvement would be insignificant if taking statistical tests since the magnitude of the performance superiority was usually very tiny.

Despite the fact that the synthetic data generator [93] was designed for k-NN, it can be easily extended to be used with other SEE methods by treating the data generator as a data preprocessor. In the thesis, we compare this data augmentation approach with our proposed approach in term of improving the performance of the baseline SEE models. Our experiments and analyses showed that our synthetic generator is either significantly superior to or has no significant difference from the data generator of [93].

2.2 Uncertain Effort Estimation Methods

Few studies have considered the development of automated models that are able to provide uncertain effort estimation. The existing approaches in SEE can be cast into the following five categories.

2.2.1 Bootstrap Wrapping

Angelis et al. [9] took the first attempt to provide uncertain effort estimation, where the authors compared the effort estimation derived from a Bootstrap-based model with the ones from regression-based methods. However, the method was actually producing confidence intervals (of the mean effort of training examples) rather than prediction intervals (of testing examples) [81, 10]. Later, Bootstrap resampling was integrated with a hybrid software model called CoBRA [33] in order to provide PIs for SEE [104]. The authors wrapped Bootstrap resampling into the CoBRA training process by replacing a single *nominal project* of CoBRA with an empirical distribution. To construct CoBRA, domain experts were asked to decide the *causal factors* of CoBRA and their possible values. Experimental results showed realistic uncertainty prediction performance. However, this method requires intensive human participation in constructing CoBRA and is very specific to CoBRA. More recently, Laqrichi et al. [118] considered uncertainty when using ANN for SEE via Bootstrap mechanism. The proposed method generated a prob-

ability distribution of point estimates, based on which the PIs can be computed. Their experimental results showed better point prediction performance compared to traditional SEE methods based on linear regression, but there was no clear support of good enough uncertain prediction performance.

The uncertain SEE methods in this category wrap Bootstrap resampling to reproduce multiple training sets [118] or estimates of model components [104], from which the PIs can be computed. They are different from our method proposed in chapter 5 in the following aspects. (1) *Base learner:* their base learners can only provide point estimation [9, 118]. In contrast, our method is based on probabilistic estimator, from which the uncertainty can be retained and calculated automatically leading to ideally more sensible uncertain estimation. (2) *Usage of Bootstrap bags:* they use all bags regardless of their prediction being unreasonable, whereas our method prunes those unreasonable ones.

2.2.2 Empirical Error Probability Consistency Assumption

Jørgensen and Sjøberg [90] proposed and evaluated an uncertan SEE method based on the assumption that empirical distribution of the estimation accuracy was consistent between the historical and the future software projects. Its implementation can be found in the third subsection of section 5.3.4. Comparing their method with a regression-based method and a human-based method showed that different methods performed well in some data while failed in others. Another work [28] with the same assumption was proposed later, but aims to produce CIs rather than PIs.

The uncertain SEE methods in this category assume that the estimation accuracy of earlier software projects predicts the uncertainty of future testing software projects. However, when this is not the case, the results are misleading. In contrast, the source of uncertainty of our method proposed in chapter 5 is assumed to originate from the Gaussian noise of effort values, which lays its foundation on *central limit theorem* [145], stating that

the summation of several independent random processes tends to a normal distribution even if the original variables themselves are not normally distributed. Considering the errors/noises that generate model uncertainty as random variables, their overall effect is reasonable to be simulated by a Gaussian distribution. However, this assumption still has problem for disregarding the fact that effort values have to be positive. Better performance can be expected with more proper noise assumption. More discussions can be found in the last subsection of section 5.7.2.

2.2.3 Categorical Conversion

Sentas et al. [161, 162] employed ordinal regression to classify a new project into an effort category (e.g. low, nominal, or high). The completed software projects were required to predefine the effort categories based on the expert judgement. The point estimate of a testing example was the mean/median of the effort values in this category the testing example falls in. Bibi et al. [23] performed experimental comparisons between the models producing point and interval estimation, but found no general conclusions as the best performed method could behave relatively bad depending on the data set. Later, Bakir et al. [17] applied clustering method to automatically define the effort categories. Its main contribution was the removal of human intervention in predefining the effort intervals.

The uncertain SEE approaches in this category have the following problems: (1) The performance of the uncertain prediction heavily relies on the goodness of the predefined intervals. (2) They suffer inferior point prediction performance as they simply use median or mean of the categorical intervals for the corresponding point estimation. (3) The interval and point estimation would be exactly the same for all projects in the same category, which may be improper and highly limit the representativeness of effort estimation. (4) It is hard to interpret the category intervals and the confidence on these interval prediction is not provided.

Mensah et al. [129] provided an advanced method that automatically determined the categorical intervals. Their method provided duplex outputs for a testing example: one for the *effort estimation* as done in most SEE studies and one for the *effort level* (high, moderate or low) for interpretation purpose. It categorized the effort values of the training examples into *high*, *moderate*, and *low* according to the density quantile function. The obtained effort of a testing example would be subsequently assigned to one of these levels. This method allows practitioners for better interpretation of the estimated effort.

Both our method proposed in chapter 5 and Mensah et al.'s method in [129] can provide additional information for the testing example helping the project managers for better decision making. However, they mainly differ in the catering problems and the types of information provided. Mensah et al.'s method aims to improve the interpretability of the point effort estimates and thus provides duplex outputs: one for the point estimation and the other for its high/moderate/low level; however, our method aims to cater the inherent uncertainty within the SEE data and to support the project managers in their decision making by providing PIs.

2.2.4 Uncertain Prediction from Bayesian Inference

RVM is a Bayesian regression model that can provide uncertain prediction as discussed in appendix A.2. As a contribution of the thesis, chapter 4 will investigate its point prediction performance in the context of SEE and provide a simple way of constructing PIs with CLs. The construction of PIs based on RVM is also novel to the thesis and no such description has been discussed explicitly in ML community. Specifically, based on the properties of the Gaussian distribution that is derived by RVM, we can present the PI with any CL $\alpha \in (0, 1)$ by using the Cumulative Distribution Function (CDF) of the derived Gaussian effort estimation.

There are some other methods based on Bayes' Theorems [45, 175, 149, 128] to infer

uncertain effort estimation. However, these methods do not aim to and cannot provide PIs, and is thus out of the scope of the thesis. In the thesis, we aim to improve the performance of RVM-based uncertain methods in terms of a narrower and more informative PI and at the same time maintain or improve the point prediction performance.

2.2.5 Other Methods Optimizing Uncertainty

Sarro et. al. [159] proposed a bi-objective SEE approach to optimise the accuracy of the point prediction performance and the uncertainty associated with the estimation model simultaneously. However, it cannot provide uncertain estimate, and is thus out of the scope of the thesis.

2.3 Parameter Settings for SEE Methods

By the publication of [169], there had been little work on quantifying the impact of different parameter settings on the performance of model-based SEE methods, and the methodologies used to choose parameter settings were frequently omitted from the experimental framework of SEE papers [136].

Early concerns of a proper representation of parameter settings came from Minku and Yao [136] and Menzies and Shepperd [134]. Minku and Yao [136] emphasized the importance of explaining clearly how the parameters were chosen involving comparisons of different SEE approaches, as they may have significant influence in the obtained conclusions. Menzies and Shepperd [134] also expressed concern regarding the effect that spending more time tuning one approach than another may cause different conclusions of their performance superiority. However, these studies did not provide an analysis of the impact that different parameter settings can have in SEE.

There have been few studies analyse the impact of different parameter settings on model-based SEE methods. In 2012, Dejaeger et al. [49] performed a comparison of

several model-based SEE methods: some adopted their default parameters, one (k-NN) was directly included in the analysis using four different parameter choices, and the others were tuned by a validation procedure. Their analyses revealed that the different values of k in k-NN did not significantly affect k-NN's performance. However, the impact of parameter settings of the other approaches was not analysed. In 2013, Kocaguneli et al. [112] performed an analysis on the effect of different kernel functions and bandwidth parameters in an analogy-based effort estimator through kernel methods. They concluded that these parameters did not affect the performance of the approach significantly. In 2017 after the publication of our paper [169], Hosni et al. [73] investigated the impact of parameter settings for heterogeneous ensemble methods using grid search optimization and particle swarm algorithms. Their results showed that heterogeneous ensembles based on optimized base learners provided better performance and the two tuning techniques generated ensembles with the same predictive capability.

There have been a few meta-heuristic algorithms applied to tune parameters of SEE models, especially for COCOMO-related models [6, 166, 158, 152, 117]. Evolutionary computation techniques such as differential evolution [6] and genetic algorithms [166, 147, 5, 65, 158] have been used as a general tuning approach for SEE models. Swarm intelligence based techniques, including particle swarm optimization [152, 117] and bee colony optimization [96], have also been used to tune the model parameters in SEE. The experimental results showed better (and sometimes significantly better) performance with the optimal parameter settings than the default ones. However, none of the above studies aimed to investigate the impact of parameter settings on the prediction performance of SEE models systematically.

A few comprehensive studies of the impact of parameter settings can be found in general software engineering areas. For instance, Arcuri and Fraser [11] performed a analysis of parameter settings in the field of test data generation using genetic algorithms. Their

analysis showed that parameter tuning had critical impact on algorithmic performance, and that overfitting of parameter tuning was a serious threat to external validity of empirical analysis in search based software engineering. In 2016, Tantithamthavorn et al. [176] investigated the performance of several popular defect prediction models by applying an automated parameter optimization technique. Their analyses showed that parameter settings had a large impact on the performance of defect prediction models, suggesting that researchers should pay attention to the parameter tuning of the classification techniques.

Overall, the impact of parameter settings of model-based SEE methods, including those that have been shown to obtain relatively good performance, was unknown [169]. A study analysing the sensitivity of the popular model-based SEE methods to their parameter settings would be important to provoke formal report of the experimental design and for a more informed choice of what SEE approach to adopt. This motivates our work [169] in investigating the sensitivity to parameter settings for SEE methods.

2.4 Performance Evaluation of SEE Methods

Consider a testing set of T project examples,

$$\mathscr{D}_T = \{\mathbf{x}_i, y_i\}_{i=1}^{T} \tag{2.1}$$

where $\mathbf{x}_i \in \mathbb{R}^D$ is the i^{th} testing example, and y_i is the actual effort for developing this software project. Note that unlike the training set of Eq. (A.1), the testing set is denoted by lower subscripts. It is to retain a simple notation when measuring prediction error. The framework for evaluation of SEE models consist of the description of SEE data sets together with the preprocessing procedures on each data set, the performance metrics in terms of both point and interval effort estimation based on the testing set $\mathscr{D}_T$, and the statistical approaches for evaluating models across multiple data sets.

Table 2.1: SEE data sets investigated in the thesis.

Repository	Name	#(Project)	#(feature)
SEACRAFT	Maxwell	62	23
	Kitchenham	145	3
	Cocomo81	63	17
	Nasa93	93	17
	Desharnais	81	8
	Albrecht	24	7
	Kemerer	16	6
ISBSG	Org1	76	3
	Org2	32	3
	Org3	162	3
	Org4	122	3
	Org5	21	3
	Org6	22	3
	Org7	21	3

2.4.1 SEE Data Sets

The experimental results in the thesis are based on several data sets from the Software Engineering Artifacts Can Really Assist Future Tasks (SEACRAFT) Repository [130][1] and from the International Software Benchmarking Standards Group (ISBSG) Repository Release 10 [77]. We choose these data sets to cover a wide range of features, such as number of projects, type of features, countries and companies. Table 2.1 contains an overall description of the data sets. This section provides detailed description and explanation on their preprocessing procedures.

SEACRAFT Repository

The SEACRAFT data sets used in the thesis include Maxwell, Kitchenham, Cocomo81 and Nasa93.

Maxwell [52] was first presented in [127] for illustration of linear regression models in SEE, and then described in [162] for ordinal regression models. It contains 62 projects from one of the biggest commercial banks in Finland, covering the years from 1985 to 1993

[1]The repository was previously called PROMISE [133].

and both in-house and outsourced development. The following steps were performed to process this data set for use in the thesis:

1. *Features:* Remove the input features start year (*syear*) and *duration* (*duration* = *syear*$-1985+1$). Start year was removed since my thesis mainly targeted the offline scenario (rather than the online), which was thus an irrelevant feature. Previous work [162] followed the same the preprocessing. Duration was removed because we usually could not know the project delivery time in reality during effort prediction process. This preprocessing resulted in the 23 input features listed in table 2.2.
2. *Categorical conversion:* The categorical features are converted into numerical values so that all the investigated methods are applicable on the same data set. Take *development type* with categorical values {organic, embedded, semi-detached} as an example to illustrate our numerical representation for categorical features. The categorical feature values can be converted as {organic=1,embedded=2, semi-detached=3}. For SEE methods that handle numerical (ordinal) and categorical features separately such as RT and k-NN, the modified Euclidean distance of Eq. (A.17) is adopted in the thesis. In this way, the ordering of the categorical features has no impact on distance. For SEE methods that can only handle numerical features such as RVM, our preprocessing raises ordering information among categorical features which can be misleading. Other preprocessing method takes 'dummy variable' that uses 0 (1) to indicate the absence (presence) of a categorical feature value. However, such method can drastically increase the number of features.
3. *Normalization:* Normalize each of the 23 input features to have zero-mean and unit-variance as Eq. (A.19). Zero-mean can usually simplify ML methods and unit-variance of each feature can avoid scalability problem among different features.
4. *Missing values:* There were no missing values in this data set.
5. *Output:* The output effort was measured in hours and remained unchanged.

Table 2.2: Detailed description of Maxwell features.

ID	name	description	type	value
1	App	application type	categorical	information/online service, transaction service, customer service, management info system, production control logistics order processing
2	Har	hardware platform	categorical	mini computer, multi-platform, networked, mainframe, PC
3	Dba	database	categorical	relational, sequential, other, none
4	Ifc	user interface	categorical	GUI, text user interface
5	Source	where developed	categorical	in-house, outsourced
6	Telonuse	Telon use	categorical	No, Yes
7	Nlan	#development languages	ordinal	1: one language used, 2: two languages used, 3: three languages used, 4: four languages used
8	T01	customer participation	ordinal	
9	T02	development environment adequacy	ordinal	
10	T03	staff availability	ordinal	
11	T04	standards use	ordinal	
12	T05	methods use	ordinal	
13	T06	tools use	ordinal	1: very low, 2: low, 3: normal, 4: high, 5: very high
14	T07	software's logical complexity	ordinal	
15	T08	requirements volatility	ordinal	
16	T09	quality requirements	ordinal	
17	T10	efficiency requirements	ordinal	
18	T11	installation requirements	ordinal	
19	T12	staff analysis skills	ordinal	
20	T13	staff application knowledge	ordinal	
21	T14	staff tool skills	ordinal	
22	T15	staff team skills	ordinal	
23	size	application size	numerical	real-valued value

Kitchenham [51] was fully described in [100]. It comprises 145 maintenance and development projects undertaken between 1994 and 1998 by a single software development company. The following steps were performed to process this data set for use in the thesis:

1. *Features:* Remove features *project ID*, *actual start date*, *actual duration*, *estimated completion date*, *first estimate*, and *first estimate method*. Project ID was removed because it was irrelevant for training an SEE model. Actual start date was removed following the same preprocessing as [100]. 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. The other features were removed because they were themselves estimation of completion date or effort, or represent the method used for such estimation. This preprocessing resulted in the three remaining input features: *adjusted function points*, *project type* and *client code* listed in table 2.3.

Table 2.3: Detailed description of Kitchenham features.

ID	name	description	type	value
1	func	adjusted functional size	numerical	continuous
2	proj	project type	categorical	A, C, D, P, Pr, U
3	client	client code	categorical	C1, C2, C3, C4, C5, C6

2. *Categorical conversion*: The categorical features are converted into numerical values, for the same reason and with the same method as for Maxwell.
3. *Normalization:* Normalize each of the three input features to have zero-mean and unit-variance as Eq. (A.19), for the same reason as for Maxwell.
4. *Missing values:* Treat missing values using 1-NN imputation method that had shown to improve SEE [37]. This imputation method is based on k-NN. It first finds the k most similar complete projects to the target project to be imputed where similarity is measured by Euclidean distance. After that, the missing values for the feature are assigned with the same values of their nearest neighbours or determined by vote counting when $k > 1$. There were in total ten projects with missing values.
5. *Output:* The output effort was measured in hours and remained unchanged.

Cocomo81 [26] and Nasa93 [53] follow the COCOMO data format [26], which has 17 input features consisting of 15 cost drivers, lines of code (*loc*) and the development type. The detailed description can be found in table 2.4. The data sets were processed to use the COCOMO numeric values for the cost drivers. Cocomo81 consists of 63 projects analysed by Boehm to introduce COCOMO [26]. Nasa93 contains 93 Nasa projects developed between 1970's and 1980'. The following steps were performed to process this data set for use in the thesis:

1. *Categorical conversion:* The categorical features are converted into numerical values, for the same reason and with the same method as for Maxwell.
2. *Normalization:* Normalize each of the 17 features to have zero-mean and unit-variance as Eq. (A.19), for the same reasons as for Maxwell.

Table 2.4: Detailed description of COCOMO-format features [26]. The term 'corr' denotes correlation to effort. In particular, 'U-shape' means giving programmers either too much or too little time to develop a system can be detrimental.

ID	name	description	corr	type	value
1	loc	line of codes	none	numerical	continuous
2	dev	develop type	none	categorical	{organic, embedded; semidetached}
3	rely	required software reliability	pos	ordinal	{1 = very low; 2 = low; 3 = normal; 4 = high; 5 = very high; 6 = extra high}
4	data	data base size	pos	ordinal	
5	cplx	process complexity	pos	ordinal	
6	time	time constraint for CPU	pos	ordinal	
7	stor	main memory constraint	pos	ordinal	
8	virt	machine volatility	pos	ordinal	
9	turn	turnaround time	pos	ordinal	
10	acap	analysts capability	neg	ordinal	
11	aexp	application experience	neg	ordinal	
12	pcap	programmers capability	neg	ordinal	
13	vexp	virtual machine experience	neg	ordinal	
14	lexp	language experience	neg	ordinal	
15	modp	modern programming practices	neg	ordinal	
16	tool	use of software tools	neg	ordinal	
17	sced	schedule constraint	U-shape	ordinal	

3. *Missing values:* There were no missing values in these two data sets.
4. *Output:* The output effort was measured in person-month and remained unchanged.

Desharnais contains 81 projects with 9 features from a Canadian software company. Four projects contained missing values, so they were excluded from our investigation. The input feature *YearEnd* was removed and thus there were 8 features in use, including *TeamExp*, *ManagerExp*, *Transactions*, *Entities*, *PointsNonAdjust*, *Adjustment*, *PointsAjust* and *Language*. The categorical features are converted into numerical values, for the same reason and with the same method as for Maxwell. The depended feature *effort* was recorded in 1,000 hours and remained unchanged.

Albrecht contains 24 projects developed in IBM using the third generation languages in the 1970s [4]. There were no categorical features nor missing values in this data set. Specifically, 18 out of 24 projects were written in COBOL, four were written in PL1,

and two were writen in DMS languages. There are 7 input features, including *InputFP*, *OutputFP*, *EnquiryFP*, *FileFP*, *FPAdj*, *RawFPcounts*, and *AdjFP*. The dependent effort was recorded in 1,000 hours and remained unchanged.

Kemerer contains 16 projects with 7 features donated by Dr. Jacky W. Keung in 2010. The input feature *project ID* was removed since it is irrelevant to the effort prediction. Accordingly, there are 6 input features including *language*, *hardware type*, *estimated duration*, *KSLOC*, *AdjFP* and *RawFP*. The categorical features (i.e. *language* and *hardware type*) are converted into numerical values, for the same reason and with the same method as for Maxwell.

Note that Desharnais, Albrecht, and Kemerer will only be investigated in chapter 3 (for RQ1 of the thesis) to gain an idea of the prediction performance of SEE methods on these three data sets. It is because most of their input features are either size-related or estimation of completion date or effort, which is usually not practical in reality. So we exclude them in the experiments for answering RQ2, RQ3, and RQ4 of the thesis.

ISBSG Repository

ISBSG release 10 contains a large body of completed software projects (5,052 projects), covering many different companies, several countries, organisation types, application types, etc. The data can be used for several different purposes, such as evaluating the benefits of changing a software or hardware development environment, improving practices and performance, and estimation [77].

First, we preprocessed the ISBSG repository following the same procedures as [136], resulting in 621 projects, by maintaining only projects with:

- Data and function points quality A (assessed as being sound with nothing being identified that might affect their integrity) or B (appears sound but there are some factors which could affect their integrity/integrity cannot be assured).
- Recorded effort that considers only development team.

Table 2.5: Groups data sets from ISBSG repository according to organization type. Only the groups with at least 20 projects were maintained following ISBSG's guideline.

ID	Organisation Type	#Projects
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	21

Table 2.6: Detailed description of ISBSG features.

ID	name	description	type	value
1	func	functional size	numerical	continuous
2	dev	development type	categorical	{enhancement, re-development, new development}
3	lang	primary programming language	categorical	{3GL, 4GL, ApG}

- Normalize effort equal to total recorded effort, meaning that the reported effort is the actual effort across the whole life cycle.
- Functional sizing method IFPUG version 4+ or NESMA.
- No missing organisation type field.

After that, a set of relevant comparison data sets need to be selected in order to produce reasonable SEE using ISBSG data. The selected projects were grouped into several ISBSG data sets according to the *organisation type* feature [136], and only the groups with at least 20 projects were maintained, following ISBSG's data set size guidelines. The resulting organisation types are shown in table 2.5.

Finally, the following steps were performed to these data sets for use in the thesis:

1. *Features:* ISBSG suggests that the most important criteria for estimation purpose are the functional size, the development type (new development, enhancement or re-development), the primary programming language (3GL, 4GL or ApG) and the 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 listed in table 2.6.

2. *Categorical conversion:* The categorical features are converted into numerical values, for the same reason and with the same method as for Maxwell.
3. *Normalization:* Normalize *functional size* to have zero-mean and unit-variance as Eq. (A.19), for the same reason as for Maxwell.
4. *Missing values:* There were no missing values for the features of *functional size* and *development type*, but *language type* had missing values across several data sets. For instance, Org1 had 25 out of 76 projects (about 33%) with their values of *language type* missing. We would lose too many data that were potentially useful in improving and evaluating a model's performance if we further eliminated those projects [138]. Thus, instead of discarding the projects in which the values of *language type* were absent, we treated these missing values by 1-NN imputation method [37], with the same procedures as in Kitchenham.
5. *Output:* The output effort was measured in hours and remained unchanged.

2.4.2 Metrics for Point Prediction

There are several performance metrics that implement the loss function in Eq. (1.2) and can be used for evaluation of point effort estimation. Popular examples are mean/median magnitude of the relative error, percentage of estimates within $N\%$ of actual values, mean absolute error, logarithmic standard deviation, and standardised accuracy.

Magnitude of Relative Error (MRE) was widely used for SEE [165, 63]. It measures the error ratio between the actual effort y_i and the predicted value $\hat{y}_i$ of the testing data set in Eq. (2.1) as:

$$MRE_i = \frac{|\hat{y}_i - y_i|}{y_i}. \tag{2.2}$$

The smaller the MRE_i, the better the prediction performance. A summary of $\{MRE_i\}$ can be derived as the Mean Magnitude of Relative Error (MMRE) or Median Magnitude of Relative Error (MdMRE) defined respectively as:

$$MMRE = \frac{1}{T}\sum_{i=1}^{T} MRE_i,$$
$$MdMAE = \text{median}\{MRE_1, \cdots, MRE_T\}. \tag{2.3}$$

There have been some criticisms on this type of metrics. For instance, [101, 63, 142, 164] showed that MMRE was unreliable since it penalized overestimates more than underestimates, and thus was biased towards prediction systems that underestimate effort and could be misleading. Underestimation (i.e. overoptimism) is the direction of the error that practitioners are more unwilling to see [88, 86].

Logarithmic Standard Deviation (LSD) is a more reliable percentage-based criterion than MMRE [63], defined as

$$LSD = \sqrt{\frac{\sum_{i=1}^{T}(r_i + \frac{s^2}{2})^2}{T-1}}, \tag{2.4}$$

where $r_i = ln\, y_i - ln\, \hat{y}_i$, $i \in \{1, \cdots, T\}$, T is the number of testing examples, and s^2 is an estimator of the variance of these residuals $\{r_i\}$. LSD is an analogy of STD by replacing $(y_i - \hat{y}_i)$ with $log\frac{y_i}{\hat{y}_i}$ alleviating the influence of large effort values; the rationale of s^2 is to compensate the bias when estimating the model parameter, where effort values in the logarithm scale and functional point are assumed to be linear [63]. Smaller LSD values correspond to better prediction performance. LSD is appropriate to evaluate multiplicative models, but it may be inappropriate for comparing additive models [63, 142].

Mean Absolute Error (MAE) has been recommended by Shepperd and MacDonell for SEE studies, for being a symmetric metric and not biased towards under or overestimation [164]. It is defined as

$$MAE = \sum_{i=1}^{T} |y_i - \widehat{y}_i|/T. \tag{2.5}$$

MAE in the logarithm effort scale can also be reported for alleviating the impact of very large error. It is computed based on the effort values in their logarithm scale (see chapter 3). Without specified, MAE is calculated based on the original effort scale. Median Absolute Error (MdAE) is defined as the median value of the prediction residues

$$MdAE = \text{median}\{|y_i - \hat{y}_i|, i = 1, \cdots, T\}. \tag{2.6}$$

It has shown to be less sensitive than MAE to occasional projects with very large efforts and is thus a useful addition to MAE [63]. One disadvantage of MAE and MdAE is that they are difficult to interpret and comparisons cannot be made across data sets since the residuals/errors are not standardised [164].

Standardised Accuracy (SA) can provide interpretable results [164]. Given T actual effort values $\{y_i\}_{i=1}^{T}$ and their estimates $\{\widehat{y_i}\}_{i=1}^{T}$ predicted by method P, SA is defined as

$$SA = 1 - \frac{MAE_P}{\overline{MAE}_{P_0}}, \tag{2.7}$$

where method P_0 denotes the random guessing, and $\overline{MAE}_{P_0}$ is the prediction performance (measured in MAE) of a large number (typically 1000) runs of random guessing. Estimating $\widehat{y_i}$ by random guessing P_0 is to take $\widehat{y_i} = y_r$ where r is drawn randomly from all the remaining $(T-1)$ effort values (i.e. $r \in \{1, \cdots, T\} \bigwedge \{r \neq i\}$) with equal probability. The value of SA can be interpreted to be how much better P is than random guessing P_0. It gives a good idea of how well the method performs. The larger the ratio, the better the prediction performance is. A value close to zero is discouraging and a negative value would be even worse.

Percentage of successful estimation falling within $N\%$ of the actual values (Pred(N)) is a common alternative to MMRE and MdMRE, defined as

$$Pred(N) = \frac{100}{N} \sum_{i=1}^{T} \begin{cases} 1, & \text{if } MRE_i \leqslant \frac{N}{100} \\ 0, & \text{if otherwise.} \end{cases} \tag{2.8}$$

For instance, $Pred(25)$=50% means that half of the estimates fall within 25% of the actual effort values [165]. The larger the percentage, the better the prediction performance is. One disadvantage of Pred(N) is that it is not analytical, and the testing set should be large enough for providing a meaningful result.

Different performance metrics emphasize different factors and can behave differently in evaluating SEE methods [135]. It is highly unlikely to exist a single, simple-to-use and universal goodness-of-fit performance metric for SEE [63]. In practice, practitioners should choose the performance metrics according to their particular emphasis and interests.

2.4.3 Metrics for Prediction Interval

An effort *Prediction Interval* (PI) can be considered as a representative form of uncertain effort estimation, and it comprises a minimum and maximum values between which the future effort value is expected to lie at a *Confidence Level* (CL). It is usually associated to a most likely point estimate. For instance, a project manager may be 95% certain that the estimated effort of a project will fall between 500 and 2,500 person-hour with the most likely effort value at 1,500. *Confidence Interval* (CI) is another uncertainty concept, which usually refers to the uncertainty associated with the unknown population statistics, such as the uncertainty of the mean value of an unknown distribution [13, pp.761-824]. For instance, a project manager may be 95% certain that the mean effort of all developed software projects is 1,500 person-month. Overall, PIs are related with an unknown project to be predicted, while CIs are connected with the mean effort of existing projects. In the thesis, we are more interested in providing PIs with CLs for an unknown project.

PIs with CLs allow for risk management and provide more flexibility to project managers. For instance, when bidding for a project, if the competition is very fierce the project manager can report a lower price within the interval to enhance the winning chances; on the contrary, when the competition is less fierce, he/she can propose a higher price for bringing more profit to the organization. The performance of the PIs with CLs is typically measured by the following two metrics.

Hit rate is the most commonly used evaluation metric for PIs [104, 84, 104]. The underlying idea is that: if PIs with CL_α are evaluated by T software projects, it is expected

that around $\alpha \times T$ projects have actual efforts falling inside the corresponding predicted PIs. The hit rate can be calculated by first counting the number of projects whose efforts are within the PIs, and then dividing that by the total number of projects. When the number of testing examples is sufficiently large, the obtained hit rate should be around the chosen CL: when the hit rate is higher, the estimated PIs are too wide; otherwise, the estimated PIs are too narrow. However, we should note that due to the small SEE data sets, we usually do not have sufficient testing examples. Hence, hit rate may deviate from its corresponding CL although the two values should be very close in essence.

In practice, the hit rates that are either equivalent to or greater than their CLs are considered to be satisfactory. When the hit rates are smaller than their CLs, the method fails in terms of hit rate and smaller values represent worse performance. When the hit rates are equal or greater than their CLs, the performance is satisfactory and succeeds in terms of hit rate. If the hit rates usually surpasses their CLs, it means that this method can be further improved by having more informative PIs. In formula, the loss function of hit rate is defined as

$$\mathscr{L}(h) = \begin{cases} cl - h, & cl > h \\ 0, & cl \leqslant h \end{cases} \tag{2.9}$$

where h denotes the actual hit rate and cl is the corresponding CL. When the hit rate equals to or surpasses its CL, the loss is zero; when the hit rate is lower than its CL, the loss equals to their difference.

Relative width is another useful performance metric for the PIs [91]. The underlying idea is that: of two sets of PIs with similar hit rates, the set with the narrower intervals is more informative and indicative for a higher level of expertise or more efficient use of the uncertainty information than the wider intervals. For example, a person who is only guessing may end up with an adequate hit rate, but his/her 90% PIs are extremely wide and thus of little practical use.

To compare PIs of different magnitudes, the relative width of a PI is defined as

$$rWidth = \frac{upB - lowB}{|Est|}, \tag{2.10}$$

where $upB/lowB$ denotes the upper/lower bound of the PI, and Est is the most likely point estimation. The overall performance of uncertain prediction is measured by the average relative width across all testing examples.

Larger hit rate may associate to a worse PI, whereas lower hit rate may associate to a better PI. Therefore, if two methods have different hit rates, their relative widths are not comparable. Conversely, if two methods have the same hit rate, we can say that the one providing the narrowest relative width is more informative.

2.5 Summary and Discussion

There have been loads of SEE literature proposing point effort estimation methods (see appendix A). However, most of them do not tackle the three issues in SEE: data scarcity, data noise, and sensitivity to model parameters. Hence, we reviewed the literature that investigated synthetic data generation to tackle the small data problem, uncertain SEE methods to cater the effort noise problem, and the sensitivity to parameter settings.

As discussed in section 2.1, Kamei et al.'s [93] has been the only synthetic data generator in SEE prior to ours in chapter 3. Their method extended SMOTE from classification to regression by attributing class imbalance from the most predictive input feature. However, their experimental results showed similar and sometimes slightly better prediction performance than the baseline k-NN and no statistical tests were taken, leaving the significance of performance improvement obscure. Therefore, it is essential to evaluate their data generator based on statistical tests. If no significant superiority over the baseline can be detected, a novel synthetic generator that plays no worse and sometimes significantly better impact on the prediction performance of the base models is demanding.

As discussed in section 2.2, there have been a few studies providing uncertain effort estimation to cater effort noise. They can be mainly cast into four categories: *Bootstrap wrapping* (section 2.2.1), *empirical error probability consistency* (section 2.2.2), *categorical conversion* (section 2.2.3), and *Bayesian inference* (section 2.2.4). However, most of the uncertain methods had not been thoroughly evaluated in terms of uncertain prediction performance and no comparisons among them had been conducted. Therefore, a thorough experimental comparison on their point/uncertain prediction performance is needed.

As discussed in section 2.3, there have been few studies that analyse the impact of parameter settings to their (point) prediction performance. Previous studies mainly focused on proposing parameter tuning approaches for COCOMO-related models based on meta-heuristic algorithms. Therefore, a study that analyses the sensitivity to parameter settings of SEE methods is demanding.

This chapter also presents the performance evaluation for SEE methods in section 2.4, which includes SEE data sets and their preprocessing procedures in section 2.4.1 and performance metrics of point and uncertain prediction in sections 2.4.2 and 2.4.3 respectively. More discussion on the statistical tests for validating the significance of performance difference in terms of point prediction of SEE methods can be found in appendix B.

CHAPTER 3

A Synthetic Project Generation Approach for SEE

3.1 Introduction

The collection of completed software projects may require considerable amount of time and workload [106, 107, 116]. Consequently, companies usually have limited training examples to construct SEE models, causing unsatisfactory prediction performance. SEE literature has frequently attempted to tackle this problem by creating advanced SEE methods that are suitable for small training set [116, 136, 109]. Rather than that, we can augment SEE data set by generating synthetic projects. The data generator can provide an alternative and much cheaper way to address the data scarcity problem. However, there have been few SEE studies investigating this strategy to assist such learning.

This chapter aims to address the data scarcity problem of SEE by answering the first research question of the thesis:

> *RQ1. Can we generate synthetic software projects to enlarge the training set size for obtaining better prediction performance? If so, how?*

The proposed data augmentation approach should be general so that it can be used

[0]This chapter corresponds to RQ1 in section 1.1.1, and is based on our published paper [171].

with any SEE method. Ideally, it should hardly have negative effect on the baseline performance that does not use synthetic projects.

To answer RQ1, we propose a synthetic project generator that can be used as a pre-processor and encoded with any SEE method. Our approach produces synthetic projects by slightly displacing the completed software projects that are chosen randomly, each associated with one training example. Though the synthetic projects are not 'real' software data, they can enrich the representativeness of the area they are generated and potentially improve the prediction performance.

Given an SEE method, the first research question of the thesis can be further divided into the following sub-research questions:

- RQ1.1: Can our synthetic data generator help improve prediction performance over the baseline that does not use synthetic projects? When? Could it be detrimental?
- RQ1.2: If our synthetic projects are helpful for prediction performance, why are they helpful? If they are detrimental, why are they detrimental?
- RQ1.3: How well does our data generator perform compared to other data generators in SEE literature?

The main contribution of this chapter is the proposal of a data augmentation approach and the answers to RQs 1.1∼1.3. Especially, we provide the understanding of when and why our synthetic projects can help improve the baseline performance that does not use our synthetic projects.

The remaining of this chapter is organised as follows. Section 3.2 proposes our synthetic project generation approach, including synthetic feature generation in subsection 3.2.1 and synthetic effort generation in subsection 3.2.2. Section 3.3 presents the experimental design to evaluate the effectiveness of our synthetic data generator. Experimental results are discussed in section 3.4. This chapter is summarized in section 3.5.

3.2 Our Synthetic Data Generator

Different from the data generator in SEE literature [93], 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.

As outlined in section 1.2.3, consider a training set of N software projects:

$$\mathscr{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^{N}, \tag{3.1}$$

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 $y^n \in \mathbb{R}^1$ is the actual effort for developing this software. Our synthetic data generator will produce $\lceil \gamma N \rceil$ synthetic projects to augment the training set and tackle the data scarcity problem of SEE, where γ is the *synthetic rate* and $\lceil \cdot \rceil$ denotes the upward rounding operator (e.g. $\lceil 1.4 \rceil = 2$). The *synthetic rate* γ should be small enough not to hinder the information provided by the real project, but large enough to be helpful. In the thesis, γ is chosen from $\{0.25, 0.5, 0.75, 1\}$ as shown in table 3.2. Ultimately, our approach will generate $M = \lceil \gamma N \rceil$ synthetic projects, each of which consists of two steps: synthetic feature generation and synthetic effort generation.

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

Synthetic Categorical Feature Generation

For a categorical feature $x_c \in \mathbf{x}$ with k values $\{v_{c1}, \cdots, v_{ck}\}$, our 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 parameter $\tau \in [0, 1)$ to the synthetic categorical feature generation, such that with probability $(1-\tau)$ the synthetic feature retains to be v_{c,x_c}, and with probability τ 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

$$x_c^{(syn)} = \begin{cases} v_{c,x_c} & \text{if } \tau < \eta \leqslant 1 \\ \sim U(\{v_{c1}, \cdots, v_{ck}\} \backslash \{v_{c,x_c}\}) & \text{if } 0 \leqslant \eta \leqslant \tau \end{cases} \tag{3.2}$$

where η 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 τ as listed in table 3.2.

Taking the categorical feature *development type* with the feature 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 τ.

Synthetic Ordinal Feature Generation

For an ordinal feature $x_o \in \mathbf{x}$ with k values $\{v_{o1}, \cdots, v_{ok}\}$ where $v_{oi} \leqslant v_{oj}$ for $1 \leqslant i \leqslant j \leqslant k$, our approach generates the synthetic $x_o^{(syn)}$ 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 and fails with probability $(1 - p)$ [184, 27]. For a Binomial random variable $\xi \sim B(n, p)$, its *expectation* satisfies $E[\xi] = np$. Binomial distribution is suitable for ordinal feature

modelling since it can manifest the ordered relationship between discrete feature values. Figure 3.1(a) illustrates the histogram of a Binomial distribution $B(n = 10, p = 1/5)$.

We take an example to demonstrate our procedures in deciding Binomial distribution 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 *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*, Binomial parameters should satisfy $n \cdot p = 3$. To guarantee the same chance for *2=low* and *4=high*, p should be $1/2$. Taking the two equations together, Binomial distribution of the ordinal feature is $B(n = 6, p = 1/2)$. Figure 3.1(b) shows a solution of Binomial distribution for *team expertise*. To retain *3=normal* situating at the distribution centre, three *dummy* values are added.

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

$$x_o^{(syn)} \sim B(n = 2 \cdot v_{o,x_o}, p = 1/2), \tag{3.3}$$

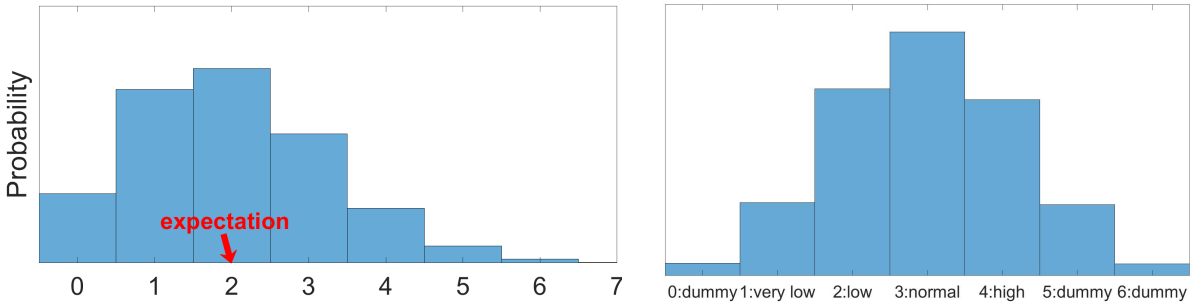
where v_{o,x_o} is the ordinal feature value of the training example.

Synthetic Numerical Feature Generation

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 $\varepsilon \in \mathcal{N}(0, \sigma^2)$ to its baseline value x_f as

$$x_f^{(syn)} = x_f + \varepsilon_f, \;\; \varepsilon_f \sim \mathcal{N}(0, \sigma^2). \tag{3.4}$$

Usually the numerical SEE features are size-related such as *functional point* or estimation of completed effort such as *line of code*.



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

Figure 3.1: Binomial distribution and its ordinal feature modelling. Figure 3.1(a) shows the PDF of a Binomial distribution, and figure 3.1(b) illustrates the Binomial modelling for the ordinal feature *team expertise*.

In this work, we normalize each numerical feature to have zero-mean and unit-variance as Eq. (A.19) and assign Gaussian σ^2 with small values $\{0.1, 0.2, 0.3\}$ as shown in table 3.2 to restrict the impact of Gaussian displacement.

3.2.2 Synthetic Effort Generation

Denote y as the actual effort of a 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 $\varepsilon \sim \mathcal{N}(0, \sigma'^2)$ to its baseline effort value as

$$y^{(syn)} = y + sign(\varepsilon_f) \cdot |\varepsilon|, \quad \varepsilon \sim \mathcal{N}(0, \sigma'^2), \tag{3.5}$$

where $sign(\varepsilon_f)$ is the positive/negative sign of the Gaussian distributed numerical feature in Eq. (3.4). When there are more than one numerical features, ε_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 [122, 40]. In this work, we confine $\sigma' = \sigma$ for simplicity. Exploration of a separate parameter σ' can be conducted in future. The proposed synthetic generator is summarized in algorithm 2.

Algorithm 2 Our synthetic project generator.

Input: (1) Training data set $\mathscr{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N$, (2) synthetic rate γ, (3) synthetic categorical feature parameter τ, and (4) Gauss variance σ^2 of synthetic numerical feature.

Aim: Generate $M = \lceil \gamma N \rceil$ synthetic projects $\{(x^{m,(syn)}, y^{m,(syn)})\}_{m=1}^M$ based on $\mathscr{D}$.

Procedures:

(1) Randomly choose a training example $\mathbf{x} \in \mathscr{D}$.

Synthetic feature generation.

(2) For each categorical feature $x_c \in \mathbf{x}$, generate its synthetic counterpart $x_c^{(syn)}$ according to a discrete uniform distribution as Eq. (3.2).

(3) For each ordinal feature $x_o \in \mathbf{x}$, generate its synthetic counterpart $x_o^{(syn)}$ according to a Binomial distribution $B(n, p)$ as Eq. (3.3).

(4) For each numerical feature $x_f \in \mathbf{x}$, generate its synthetic counterpart $x_f^{(syn)}$ by adding a zero-mean Gaussian variable as Eq. (3.4).

Synthetic effort generation.

(5) Assign synthetic effort $y^{(syn)}$ by adding a zero-mean Gaussian variable with the same increasing/decreasing direction as the numerical features to y as Eq. (3.5).

Loop.

(6) Repeat steps (1)$\sim$(5) until M synthetic projects are generated.

Output: M synthetic projects $\{(x^{m,(syn)}, y^{m,(syn)})\}_{m=1}^M$.

3.2.3 Further Discussion on Our Data Generator

There are a few problems that may hinder the effectiveness of our data generator:

(1) 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 with *high* rather than *low* expertise. Therefore, it would be interesting to study whether other non-symmetric distributive modellings of ordinal/categorical features would further improve prediction performance. This would depend on expert knowledge of the data distribution.

(2) Our approach assumes that synthetic effort values are only affected by the change in numerical features. Assigning synthetic effort from the changes of ordinal/categorical features is very challenging, requiring expert knowledge or data analyses with large training sets. This is potentially a harder problem than SEE itself. Moreover, changing some or-

dinal/categorical features would increase the effort, whereas changing some others would decrease it. Altogether, this would cause small variations in effort. Since our strategy had achieved good results, we did not investigate the impact of changing 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 one training example that is chosen randomly. Thus, the 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 SEE method. Thus, it can be used as a preprocessor with any SEE model.

3.3 Experimental Design

This section presents our experimental design to justify the effectiveness of the proposed data generator, including the data sets and the preprocessing procedures on them, the evaluation strategies for prediction performance, the baseline SEE methods and their parameter settings.

3.3.1 Data Sets

The analyses in this chapter is based on 14 data sets from the SEACRAFT Repository [130] and the ISBSG Repository Release 10 [77]. Seven data sets including Maxwell, Cocomo81, Nasa93, Albrecht, Kemerer, Desharnais, and Kitchenham, are from SEACRAFT repository; Seven data sets, namely Org1∼Org7, are the subsets of ISBSG being grouped according to organization type as table 2.5. Section 2.4.1 describes these data sets and the basic preprocessing procedures on them. As explained in section 2.4.1, Desharnais, Albrecht, and Kemerer are only investigated in this chapter to gain an idea of their SEE performance. It is because most of their input features are either size-related or estimation of completion date or effort, being not practical in reality.

Table 3.1: SEE data sets that are cast into 3 groups representing *small*, *medium* and *large* data set sizes according to the ratio of the data number over the feature number. 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			

To investigate the effect of training set size, SEE data sets are grouped into *small*, *medium*, and *large* according to the ratio of the number of data over the number of 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. This exception was removed, resulting in 20 projects with a single input feature. Table 3.1 contains the basic description of the investigated data sets in this chapter.

Data preprocessing. For each data set in table 3.1, 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 [132, 173]. Logarithm preprocessor has shown to be non-harmful to or even sometimes improve the performance of the defect prediction [132, 173]. Therefore, all numeric features are replaced with their natural logarithm values in this chapter. This preprocessing also minimizes the effect of the occasional very large feature values. The

dependent effort values are also converted into their logarithm scale to make the effort distribution more Gaussian. This procedure can alleviate the prediction problem when treating testing examples with very large effort (section 2.4.2).

3.3.2 Performance Evaluation

The performance metric used in this chapter is MAE defined as

$$\sum_{i=1}^{T} \frac{|y_i - \widehat{y}_i|}{T}, \tag{3.6}$$

where $y_i/\widehat{y}_i$ denotes the actual/predicted effort of the i^{th} testing data from $\mathscr{D}_T$, and T is the number of testing data. MAE was recommended by Shepperd and MacDonell in SEE for being a symmetric measure not bias towards under- or overestimation [164]. As the effort is in their logarithm scale, MAE becomes less affected by the project size.

There are three typical approaches of data partition to evaluate the prediction performance [69]: holdout [115], k-fold cross-validation (CV) [69], and leave-one-out (LOO) [69, 108]. *Holdout* keeps a certain percentage p out of the entire data set as the training set, and the model will be evaluated on the remaining testing data. The larger the p, the more training examples used for model construction. The *k-fold CV* partitions the entire data set into k subsets, each of which is used once to validate the model performance and the rest data samples are used to train the predictor. *LOO* is a special case of k-fold CV when the number of folds equals to the number of data samples.

In this chapter, we apply *holdout* evaluation to control the training set size deliberately and validate the proposed data generator by investigating the impact of synthetic projects when the training set size is small, medium, and large respectively. We first randomly split the data set into training and test subsets. Each SEE method is constructed 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.

When we are in a data-rich situation, the best way to evaluate an SEE method is to randomly divide the entire data set into 3 parts: a training set, a validation set and a testing set. The training set is used to fit the model, the validation set is used to choose the optimal model parameters, and the testing set is used to evaluate the model performance. However, SEE data sets are usually very small. Taking a separate testing set will result in an even smaller number of examples for training and validating (model selection). A small validation set may not be able to find the optimal model parameters, failing to evaluate the method with its best capability. A small testing set may not represent the data space very well, possibly causing invalid evaluation of SEE methods. Therefore, we use no separate testing sets in the thesis. This experimental setting is also consistent to many previous SEE studies [109, 111, 169, 170].

3.3.3 Baseline SEE Methods

To evaluate the effectiveness of our data generator, we investigate six SEE methods: Multivariate Linear Regression (MLR), Automatically Transformed Linear Model (ATLM), k-Nearest Neighbours (k-NN), Relevance Vector Machine (RVM), Regression Tree (RT), and Support Vector Regression (SVR). Please refer to appendix A for basic description of these methods.

MLR and ATLM [188] are chosen because they have been shown to be good baselines after appropriate data transformations [188, 99]. *R.matlab* package [22] was used to configure the R implementation of ATLM into the MATLAB framework. Note that sometimes the estimates from ATLM can be NaN (Not A Number), which may be caused by the automatic transformation on input features. When this happened, we use the prediction of MLR to replace those NaNs.

K-NN is chosen for being among the simplest SEE method and due to its intuitive interpretation that mimics the human decision-making [169, 165, 122, 114]. Some studies

have showed that k-NN is comparable and sometimes superior to other SEE methods [165, 87, 122, 9, 114]. To predict the effort of a testing project, the distance of the project to all training examples are computed in terms of Euclidean distance in Eq. (A.17). Based on them, k nearest neighbours to the testing project are determined, and their median effort value is returned as the estimate effort value [111].

RVM is chosen because it has been shown to be very competitive to other state-of-the-art SEE methods [170, 177, 59]. 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

$$\phi_j(\mathbf{x}) = exp\{-(\mathbf{x} - \boldsymbol{\mu_j})^2/(2s^2)\}$$

to be our basis function for its *locality* property [136], where the $\boldsymbol{\mu}_j$ is the j-th training example and the width s controls their spatial scale.

RT is chosen for being among the most frequently used SEE methods which has presented potential advantage for SEE [136, 187]. 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 [136].

SVR is designed for small data problem [55], which seems suitable for SEE. However, SVR has not been popularly used in the SEE community partially because of the contradictory conclusions drawn from previous studies [160, 146, 7, 41]. Some claimed its superior performance in SEE [146, 160, 41], while others claimed inferior performance of SVR compared with other SEE methods [7]. There are several choices for SVR kernel. We use linear kernel for having been shown to be a better choice [146].

Each of these SEE methods will be used as a baseline to investigate whether or not the generated synthetic projects can improve their prediction performance. SEE methods in the chapter are implemented in MATLAB and specified if otherwise.

Table 3.2: Parameter values of the SEE methods investigated.

ID	Method	Parameters
1	MLR	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} M (min #node per leaf) = {1, 2, 4} E (stopping error) = {0.0001, 0.01, 0.5}
6	SVR	*kernel* = 'linear' C (regularization) = {0.01, 0.1, **1**, 10 } ε (slack variables) = {0.1, **0.3**, 0.5, 1}
7	syn.our	γ (synRate) = {0.25,0.5,0.75,1} τ (categorical) = {0,0.2,0.4} σ^2 (GaussVar) = {0.1,0.2,0.3}
8	syn.cmp	k (neighbours in SMOTE) = {1,2,3,5}

3.3.4 Parameter Settings

The parameter values of the SEE methods investigated in this chapter are listed in Table 3.2. For RT, the maximum tree depth of -1 means unlimited tree depth. For SVR, we use the conventional settings for regularization parameter C and slack variable ε [41, 140]. For the models that have more than one parameters, we investigate all possible parameter settings. Our discussion is based on the performance with the best parameter settings.

3.4 Experimental Result and Discussion

This section evaluates our synthetic data generator by comparing the performance of SEE methods with and without the generated synthetic projects. The performance of an SEE method that does not use our synthetic projects is denoted as *bsl.SEEr*, and the performance of an SEE method that uses our synthetic projects is denoted as *syn.SEEr*. Moreover, section 3.4.3 compares our synthetic generator against its only competitor in SEE literature [93]. The performance of an SEE method that uses Kamei et al.'s [93] synthetic projects is represented by *syn.cmp.SEEr*.

3.4.1 Effect of Synthetic Data on Prediction Performance

This subsection aims to answer RQ1.1 presented in section 3.1: *Given an effort model, can our synthetic data generator help improve prediction performance over the baseline that does not use synthetic data? When? Could it be detrimental?* To answer RQ3.1, we investigate the effect of our synthetic projects by comparing the performance of *syn.SEEr* against *bsl.SEEr* across 14 data sets with small, medium and large training sizes respectively. Table 3.3 lists the performance comparisons in all data set sizes. We can see that our synthetic projects can usually improve the prediction 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. In this chapter, we adopt the non-parametric effect size A_{12} that makes no assumptions about the underlying distribution [183, 12]. In table 3.3, large/medium/small effect size is highlighted in orange bold/yellow bold/bold indicating the performance improvement of using our synthetic projects. Detailed description on effect size can be found in appendix B.3.

We perform Wilcoxon signed rank tests with Holm-Bonferroni correction at the level of significance 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 [189, 50]. 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 [50].

Table 3.3: Performance comparison between pairs of *syn.SEEr* vs *bsl.SEEr* across 14 data sets in terms of MAE for small, medium, and large training set sizes. Different training set sizes refer to different *holdout* values in table 3.1. The reported values are the mean of 30 runs followed by their 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. The last two rows show Wilcoxon tests with Bonferroni correction. The overall comparison of *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.MLR	bsl.MLR	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.MLR	bsl.MLR	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.MLR	bsl.MLR	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

MLR and ATLM

Since ATLM is a variant of MLR 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 3.3(a) that the synthetic projects generated by our approach can drastically improve the performance of MLR/ATLM with large effect size in 5 out of 7 SEACRAFT data sets. The improvement is less significant for the ISBSG data sets: the effect size is medium for one and small for four out of seven data sets. The synthetic data never hurts the performance of MLR/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.MLR*/*syn.ATLM* over *bsl.MLR*/*bsl.ATLM*.

It is noteworthy that the performance of MLR/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 $A_{12} = 0.6373$.

As discussed in appendix A.1.3, the unstable performance of MLR/ATLM may be due to the scarcity of training examples. When the few training examples are close to each other, being more likely to happen given inadequate training data, MLR/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 examples.

For medium training set size, we can see from tables 3.3(a) vs 3.3(b) that *bsl.MLR* and *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 insuffi-

cient number can improve the performance of MLR/ATLM. Similar observation can also seen for *syn.MLR*/*syn.ATLM*.

We can also see that our synthetic projects can improve the performance of MLR/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.MLR*/*syn.ATLM* over *bsl.MLR*/*bsl.ATLM*, showing an overall superiority when the training set size is medium.

For large training set size, the superiority of *syn.MLR* (*syn.ATLM*) over *bsl.MLR* (*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.MLR* (*syn.ATLM*) over *bsl.MLR* (*bsl.ATLM*) with p-value 0.016255 (0.00067).

Summary. Our synthetic projects can always improve the performance of MLR and ATLM, and the improvement magnitude is usually significant with 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. They can also help with stable performance especially for small training set. Therefore, we suggest to apply the proposed synthetic generator when using MLR and ATLM as SEE methods in insufficient data sets.

RVM

Table 3.3 shows that our synthetic projects can always improve the performance of RVM. Their effect size is sometimes large or medium, showing substantial performance improvement. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all data sets detect significant superiority of using our synthetic projects for all the training set sizes.

RT

Table 3.3 shows that our synthetic projects can always improve the performance of RT. 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 projects for medium and large training set sizes.

k-NN

Table 3.3 shows that our synthetic projects can usually improve the performance of k-NN. The improvement for medium and large training sets is a bit more obvious than those for small training sets. Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across all data sets detect significantly better overall performance of using our synthetic projects for medium and large training set sizes. However, the effect size shows small or insignificant superiority.

SVR

Table 3.3 shows that our synthetic projects can usually improve the performance of SVR. The performance improvement for small and medium training sets is more obvious than those for large training sets. 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 projects for small and medium training set sizes. However, the effect size usually shows insignificant superiority.

Superior performance of SVR. We can see from table 3.3 that SVR usually outperforms other SEE methods. Friedman tests at the significance level 0.05 across all data sets reject the null hypothesis (H0) stating that all models are equivalent. Nevertheless, our synthetic projects can further improve its performance a little when there are insufficient training examples.

Factors that impact the prediction performance of SEEr. The superiority of SVR over SEE methods is consistent with some previous works [146, 160, 146, 41], but contradicts some others [7]. One of the possible reasons would be the usage of different evaluation approaches that result in different training sizes. The performance of SEE methods can be affected by the training 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, MLR and ATLM. Some other factors that may affect the results of SEE model comparison 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 [169, 11, 125].

Effect of Training Size on Point Prediction Performance

This subsection studies the effect of training set size (i.e., small, medium, and large) on point prediction performance. We can see from table 3.3 that the larger the training set, the better the performance of SEE methods. Taking MLR in Maxwell as an example, the average prediction error is decreased from 1.314 for small training set to 0.656 for medium set, and further to 0.536 for large set. This observation also shows that the performance improvement is more significant when the training set increases from small to medium than from medium to large.

SEE methods gain performance improvement from having more training examples differently. For instance, the profit of more training examples is usually less significant to k-NN or RVM than to SVR or RT; more training examples can drastically improve the performance of MLR and ATLM. Taking Maxwell as an example, the performance is 1.314 (small) vs 0.656 (medium) for MLR, 0.693 (small) vs 0.574 (medium) for RT, and 0.731 (small) vs 0.681 (medium) for k-NN. A possible reason can be contributed to the *locality* (*globality*) property of SEE methods as discussed in section 3.4.2.

Brief Summary

Our synthetic projects usually have positive effect on and are rarely detrimental to the baseline performance that does not use synthetic projects. Specifically, our synthetic projects can usually significantly improve the performance of MLR and ATLM with large/medium effect size; they can often improve the performance of RVM and RT and slightly improve the performance of k-NN and SVR. Note that the performance is measured in the logarithm scale of effort values in this chapter. The prediction performance in the original scale of effort values will be reported in table C.1. The key conclusion that syn.SEEr always performs similarly/better than bsl.SEEr remains the same.

3.4.2 Reasons for Effectiveness of Our Synthetic Projects

This subsection aims to answer RQ1.2 outlined in section 3.1: *Given an SEE method, if our synthetic projects are helpful for prediction performance, why? If they are detrimental, why?* Given the results summarised in section 3.4.1, RQ1.2 can be further divided as

- RQ1.2.1. Why do our synthetic projects usually have positive effect on SEE models?
- RQ1.2.2. Why do our synthetic projects have different improvement magnitude for different SEE methods?

When the training size is large, an SEE method can usually achieve relatively good performance, leaving limited improvement space for using our synthetic projects. Therefore, our discussion will focus on the cases of insufficient training examples.

Positive Effect of Our Synthetic Projects

This subsection aims to answer RQ1.2.1 in view of the augmentation of training set and the enhanced ability to handle data noise.

The main possible reason is the augmentation of SEE training sets by encompassing our synthetic projects into the construction of SEE models, directly tackling the data scarcity problem of SEE.

Another possible reason is the enhanced ability of tackling large data noise that can lead to large variations from the actual effort values. Effort values are highly likely to contain noise due to the participation of humans in data collection [170, 84, 91]. When training examples are insufficient, such noise is more likely to mislead the construction of SEE models, causing less correct and unstable prediction performance. When the training examples contain 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 3.2 illustrates the positive effect of our synthetic projects.

Our data generator emphasizes the more typical areas of learning space, helping avoid being misled by large noise. Specifically, the training examples that locate in crowded regions, which are less likely to contain large variations, are more likely to be chosen to generate our synthetic projects. In this way, our synthetic projects emphasize 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 happen. In this way, we can circumvent the issue of introducing the data noise that can cause large variations from the actual values.

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

Effect of Synthetic Projects on Each SEE Method

This subsection aims to answer RQ1.2.2 in view of the *locality/globality* of SEE methods.

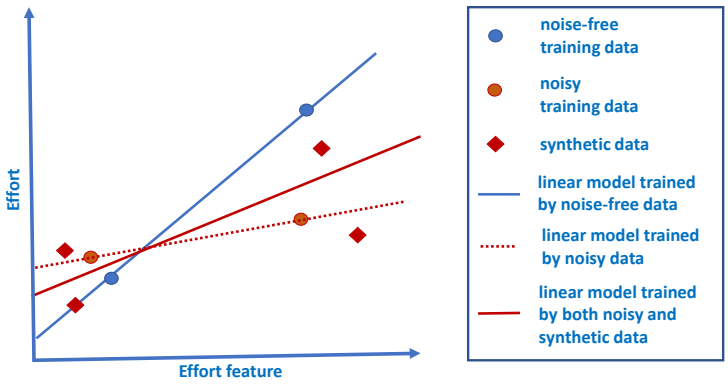


Figure 3.2: A demonstration where four synthetic projects help construct MLR for SEE. The synthetic projects (square) enhance the robustness of its neighbourhood and alleviate detrimental effect of the noisy training examples.

Locality and globality of SEE methods. SEE methods performing effort estimation only based on training examples that are similar to the testing project are referred to have *locality* property [136, 64]. The opposite terminology is referred as *globality* in the thesis, where effort estimation is performed based on all training examples regardless of the similarity to the testing project. Recall that our synthetic projects can only impact their neighbourhood, so the *locality*/*globality* property of an SEE method is a primary avenue to spread the effect of synthetic projects from the neighbourhood to other areas.

MLR/ATLM is an example of SEE methods with thorough *globality*. All training examples, regardless of their similarity to the project to be estimated, are used to estimate the optimal model parameters. The constructed model is then used to estimate the effort of testing projects. Therefore, the effect of our synthetic projects in one area will impact the prediction in the entire space, leading to remarkable effect of our synthetic projects on prediction performance. Particularly, if synthetic projects are created in an area with training examples that the SEE method is confident on their effort estimation, this would improve the prediction in other areas with less training examples, where the model is originally not confident on.

***K*-NN** is an example of SEE methods with thorough *locality*, where the effort esti-

mation of a testing example is only based on the training examples in its neighbourhood. Therefore, the effect of our synthetic projects in one area will not impact the prediction in other areas, causing little effect of our synthetic projects.

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 estimation is based on the training subset. Therefore, the effect of our synthetic projects in one area will impact the prediction in other areas to some extent.

RVM is another example of SEE methods 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 effort estimation of RVM is a weighted summation of the distances between each training example and the testing project. In this sense, only a subset of training examples is used to predict the effort. Therefore, the effect of our synthetic projects in one area can impact the prediction in the other areas in some degree.

SVR has a *tolerance margin* (ε in table 3.2), with which data noise is tolerant to some extent. When a 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 a synthetic project locates on the *tolerance margin*, namely a *support vector*, it can impact 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*.

3.4.3 Comparison of Synthetic Project Generators

This subsection aims to answer RQ1.3 outlined in section 3.1: *How well does our data generator perform compared to other data generators in SEE literature?* To answer RQ1.3, we compare the performance of our data generator against its only competitor in SEE literature [93], denoted by *syn.cmp.SEEr*. As presented in section 2.1.4, though Kamei et al's [93] only uses k-NN as their base learner, their data generator can be encoded with other SEE methods straightforwardly.

Syn.SEEr* vs *Syn.Cmp.SEEr

Table 3.4 lists 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*. For example, the data cell in the first row and first column of table 3.4(a) is in orange bold indicating that the effect size between *syn.MLR* and *syn.cmp.MLR* across 30 runs in Maxwell is large. 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 methods. The superiority magnitude of *syn.SEEr* over *syn.cmp.SEEr* is often large for MLR and ATLM, moderate for RT and RVM, and small for k-NN and SVR.

To analyse the performance superiority across data sets, we perform Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 to judge whether performance difference between *syn.SEEr* and *syn.cmp.SEEr* is statistically significant. The Wilcoxon results between *syn.SEEr* and *syn.cmp.SEEr* are associated with the

Table 3.4: Performance comparison of *syn.SEEr* vs *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. 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 effect size. The last two rows show Wilcoxon tests with Bonferroni correction across all data sets: the rows associated to *syn.SEEr* list Wilcoxon results between *syn.SEEr* vs *syn.cmp.SEEr*, and the rows associated to *syn.cmp.SEEr* list 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.MLR	syn.Cmp.MLR	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.MLR	syn.Cmp.MLR	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.MLR	syn.Cmp.MLR	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

columns of *syn.SEEr*, and significant difference is highlighted in orange. For example, the data cell in the last row and first column of table 3.4(a) is orange indicating that *syn.MLR* performs significantly better than *syn.cmp.MLR* across all data sets.

Overall, Wilcoxon signed rank tests with Holm-Bonferroni correction between *syn.SEEr* and *syn.cmp.SEEr* show that when the training set is not large, our synthetic generator is always superior to its competitor for having significantly better prediction performance. When our synthetic generator is not superior to its competitor, it is not worse either.

Syn.Cmp.SEEr* vs *bsl.SEEr

Comparing the performance of *bsl.SEEr* in table 3.3 and *syn.cmp.SEEr* in table 3.4, we can see that *syn.cmp.SEEr* often fails to outperform *bsl.SEEr*. 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 projects generated by the literature do not have great impact on 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 k-NN [93]. Further studies found that the experiments of [93] were based on Desharnais data set only. The reported superiority of using their synthetic projects was small and no statistical test was conducted. We suspect that with more data sets into their experiments and using statistical tests, their conclusions would probably be no significant difference with or without using their synthetic projects.

3.5 Summary and Discussion

This chapter proposes a synthetic project generator to address the data scarcity problem of SEE. A synthetic project is generated by slightly displacing one training example that is chosen randomly. The generated synthetic projects are then added to the training set and used to construct SEE models.

The effectiveness of our synthetic project generator has been validated by answering the sub-research questions outlined in section 3.1 as follows.

• *RQ1.1: Given an SEE method, can our synthetic data generator help improve prediction performance over the baseline that does not use synthetic projects? When? Could it be detrimental?* Experimental results show that our synthetic projects usually have positive effect on and are rarely detrimental to the performance of SEE methods. They are particularly helpful for small and medium data set sizes for MLR 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.

• *RQ1.2: Given an SEE method, if our synthetic projects are helpful for prediction performance, why are they helpful? If they are detrimental, why are they detrimental?* The effectiveness of our synthetic projects is mainly due to the data augmentation and the robustness enhancement in the areas that data noise may injure the quality of SEE model construction. Different SEE methods have different improvement magnitude which is affected by their *locality* and *globality* properties. Our synthetic projects are rarely detrimental to the baseline performance that does not use synthetic projects.

• *RQ1.3: How well does our data generator perform compared to other data generators in SEE literature?* Experimental results show that our synthetic project generator is significantly superior to or has no significant difference from Kamei et al's [93] that is the only competitor in SEE literature. Their data generator [93] probably brings no significant improvement to the baseline performance that does not use synthetic projects.

This chapter also studies the impact of training set size on prediction performance based on the *holdout* evaluation. Experimental results show SEE methods usually achieve better performance with a larger training set; the magnitude of superiority decreases on the increase of the training size. This is a by-product of answering RQ1.

CHAPTER 4

RVM: A Promising Uncertain Effort Estimator

4.1 Introduction

Uncertainty is inherent in SEE [82, 102] and there are several sources of uncertainty in the context of SEE [174, 91, 84]. For instance, uncertainty may arise from SEE model limitation and/or from the noise in the data used for constructing the models [97, 91, 84, 104]. Appropriate SEE methods should ideally handle uncertainty explicitly and support decision-making in assessing estimation risks based on such uncertainty.

Despite of a vast literature in creating SEE methods, most of them can only produce point estimation [19, 187]. Relying on point estimation may ignore the uncertain factors and lead project managers to wrong decision-making. Consequently, there is no feasible way to manage risks and uncertainty. If a point estimate had to ensure against all possible uncertainty, the price of constructing such an SEE model would be prohibitive [97].

For decades, software estimation experts have argued that besides point estimation, effective project management also requires information about effort-related risks [103]. Particularly, they have suggested that an effort estimate of a software project should be

[0]This chapter corresponds to RQ2 in section 1.1.2, and is based on our published paper [170].

a range of values with a specific probability, namely a PI with a CL, within which the software development can be completed [97]. This can be considered as a more reasonable representation of reality than a mere point estimate and can provide flexibility in the subsequent management activities. Taking bidding for a software project as an example, if the competition is very fierce, the project managers can report a lower price within the PI to enhance the winning chance; when the competition is less fierce, they can propose a higher price for bringing more profit to the organization.

This chapter aims to introduce a Bayesian regression model to the context of SEE, which can provide uncertain effort estimation and is suitable for small training set for answering the second research question of the thesis:

RQ2. Is there any ML approach that can provide uncertain effort estimation? How well can it perform in terms of point and uncertain prediction?

To accomplish the answer to RQ2, we will validate RVM in terms of point and uncertain prediction performance in the context of SEE, forming two sub-research questions:

- RQ2.1: How well does RVM perform compared to other SEE methods when used as a point estimator?
- RQ2.2: How to provide PIs with CLs based on RVM? How well can the PIs with CLs derived from RVM perform?

Answering RQ2.1 allows us to know how promising RVM is in the context of SEE. If it performs very badly, none of the conclusions based on it are reliable. If RVM is competitive to other SEE methods, its conclusions are reliable. It is worth noting that our aim is not to show that RVM is a SEE method that outperforms other SEE methods, but to investigate whether it presents competitive performance and should be investigated further, given its capability to provide uncertain effort estimation.

We will show that RVM is a promising SEE method with competitive point prediction performance and can provide PIs with CLs. On the one hand, experimental results show that RVM is very competitive to state-of-the-art SEE methods, being usually ranked the first or second in 7 across 11 data sets in terms of MAE. On the other hand, RVM explicitly models effort noise and provides a probabilistic effort estimate for a new project, based on which PIs of CLs are derived. The PIs with CLs can usually pass the justification in terms of hit rate, but they can be too wide to be informative.

The remaining of this chapter is organized as follows. Section 4.2 presents its adaptation to SEE in providing uncertain effort estimation. After the experimental design in section 4.3, the performance of RVM as a point and uncertain estimator is evaluated in section 4.4. This chapter is concluded in section 4.5.

4.2 RVM for Uncertain Effort Estimation

This section discusses the derivation of PIs with CLs from RVM. More discussion on RVM in the context of SEE can be found in appendix A.2.

RVM models effort noise by a Gaussian distributed random variable ε in Eq. (A.9). The probabilistic effort estimation is derived accordingly. Note that Gaussian assumption of noise variable ε leads to but does not equal to Gaussian distribution of predicted effort of a testing project. In other words, the Gaussian assumption comes before any training example is observed; in contrast, the effort of a testing project is predicted to be Gaussian distributed after training examples are used.

In the thesis, uncertainty is considered in probability, which originates from the unpredictable and non-deterministic nature of future software projects. Specifically, uncertainty is interpreted as the factors influencing effort values. Accordingly, SEE uncertainty can be characterized through two types of interval predictions:

- *Prediction intervals* of estimated effort comprise minimum and maximum values, be-

tween which the future effort is expected to lie with a *confidence level* (see section 2.4.3). For instance, a project manager may be 95% certain that the estimated effort of a project will fall between 500 and 2,500 person-hours with the most likely effort value 1,500.

• *Confidence interval* (CI) is another uncertainty concept, which usually refers to the uncertainty associated with the unknown population statistics, such as the uncertainty of the mean of an unknown distribution [13, pp.761-824]. For instance, a project manager may be 95% certain that the mean of the effort values of all developed software projects is 1,500 person-months.

• Overall, PIs relate with an unknown software project to be estimated, while CIs relate with (e.g.) the mean of effort values of the known software projects. In the thesis, we focus on providing PIs with CLs for an unknown software project.

Based on Gaussian effort estimation from RVM, a PI with CL $\alpha\%$ can be derived by CDF of this probability namely $\mathcal{F}$ as:

$$PI_{\alpha} = [max\{0, \mathcal{F}^{-1}(\frac{1-\alpha}{2})\}, \mathcal{F}^{-1}(\frac{1+\alpha}{2})] \tag{4.1}$$

where $\mathcal{F}^{-1}(\beta)$ denotes the effort value of the β-percentile of Gaussian CDF. According to the symmetry of Gaussian PDF, the proportion between $\frac{1-\alpha}{2}$ and $\frac{1+\alpha}{2}$ equals to α which equals to the requested CL. Since effort values should be non-negative, we confine the non-negative value for the left-hand-side. A point estimate is assigned to be the mean of the Gaussian distribution, since it is the most likely achievable effort value.

There is an easier way to derive the PIs with CLs 68.27%, 95.45%, and 99.70% based on "68-95-99.7" rule of the Gaussian distribution [191]. Specifically, the PIs with $\text{CL}_{0.6827}$, $\text{CL}_{0.9545}$, and $\text{CL}_{0.997}$ can be easily derived as:

$$[max\{0, \widehat{y} - i\widehat{\sigma})\}, \widehat{y} + i\widehat{\sigma}] \tag{4.2}$$

where $\widehat{y}$ denotes the estimated mean effort, $\widehat{\sigma}$ is the estimated STD of effort noise, and

$i \in \{1, 2, 3\}$ correspond $CL_{0.6827}$, $CL_{0.9545}$, and $CL_{0.997}$ respectively. Note that we can also derive PIs with these CLs according to Eq. (4.1) and it is just easier by Eq. (4.2).

In this chapter, PIs with $CL_{0.6827}$ and $CL_{0.9545}$ will be provided and evaluated for being reasonable as a first step of uncertain effort estimation. Specifically, $CL_{0.6827}$ is more or less sufficient for practical use since usually only 3 of 10 projects may go below the lower bound or exceed the upper bound; the presence of $CL_{0.9545}$ is to have an idea that how well the PIs derived by RVM can achieve when considering higher CLs. $CL_{99.70\%}$ is excluded since it is too strict requiring not a mere violation of 100 PIs.

4.3 Experimental Design

The experiments are designed to answer RQ2.1 and RQ2.2 by evaluating the point and uncertain prediction performance of RVM.

4.3.1 Data Sets

Experimental analyses in this chapter are based on 11 data sets from SEACRAFT [130] and ISBSG [77] Repositories. Four data sets including Maxwell, Cocomo81, Nasa93, and Kitchenham are from SEACRAFT repository; Seven data sets, namely Org1∼Org7, are the subsets of ISBSG being grouped according to organization type as table 2.5. Table 4.1 lists basic description of these data sets. Detailed description and the preprocessing procedures on them can be found in section 2.4.1. As explained in section 2.4.1, Desharnais, Albrecht, and Kemerer are not studied in this chapter since most of their input features are size-related or estimation of completion effort, being not practical in reality.

4.3.2 Performance Evaluation

We use 10 times 10-fold CV to evaluate the performance of SEE methods, which repeats ten times 10-fold CV with different sampling orders in order to cancel out the impact of project orders. We used 10-fold CV because the SEE data are usually small, and it may

Table 4.1: The investigated SEE data sets.

Repository	Name	#(Project)	#(feature)
SEACRAFT	Maxwell	62	23
	Kitchenham	145	3
	Cocomo81	63	17
	Nasa93	93	17
ISBSG	Org1	76	3
	Org2	32	3
	Org3	162	3
	Org4	122	3
	Org5	21	3
	Org6	22	3
	Org7	21	3

cause high bias if using small k ($k = 2$ in k-fold) due to the lack of training data; whereas large k, such as leave-one-out with k equals to the size of data, may result in high variance [69]. Our preliminary empirical results with 5 times 2-fold CV, leave-one-out and 10 times 10-fold CV also indicate this tendency. Therefore, we consider 10-fold as a suitable choice.

The same to the experimental design in section 3.3.2, we use no further spare testing sets because SEE data sets are too small. If using a spare testing set, we will have an even smaller number of projects for training and validating (model selection). Moreover, a small testing set may not represent the whole space very well, so that the evaluation of the learning model would be potentially invalid.

The point prediction performance is measured by MAE as

$$\sum_{i=1}^{T} \frac{|y_i - \widehat{y}_i|}{N},$$

for being symmetric and not bias towards under or overestimation [164]. We do not report (e.g.) MMRE because it has been shown to be a biased measurement [63] and we do not want to include a misleading measurement in our results. More performance metrics will be reported in chapter 5 for more thorough performance evaluation.

4.3.3 Benchmark SEE Methods

We compare point prediction performance of RVM against the following five approaches: k-NN, RT, MLP, Bagging+RT, and Bagging+MLP in this chapter.

We do not investigate Bagging+k-NN because Bagging is known to improve accuracy for unstable models[1] such as MLP and RT, whilst it may slight degrade the performance of stable models such as k-NN [29, 32]. We use WEKA [67] to implement the five SEE methods: k-NN was based on IBK with normalized attributes and Euclidean distance, RT was based on REPTree without pruning, and the others were based on the corresponding classes with the same name. MLP were set to automatically normalize dependent and independent variables. RVM was implemented in MATLAB due to the advantage in matrix computation and the absence of the implementation in WEKA.

RT, Bagging+RT and Bagging+MLP were chosen due to their good performance in comparison to several other ML approaches in SEE literature [136]. K-NN is among the simplest SEE approaches, and shown to perform frequently well [165]. MLP have not been shown well performed in SEE, however, a recent study showed that after finely tuned it could achieve competitive performance [169]. In summary, the main reason of including the five models is their relatively good performance reported by current literature.

4.3.4 Parameter Settings

The parameter values of the SEE methods investigated are shown in table 4.2. The default parameter values are emphasized in bold and correspond to the default values of WEKA. For RT, the maximum depth of -1 means unlimited depth. For MLP, the default value a in #(hidden nodes) represents: $a = [\#(attributes)+1]/2$. For RVM, $0.1 : 1.0 : 15$ denotes the values counting from 0.1 to 15 with step = 1.0.

All methods except for RVM or k-NN have more than one tuned parameters, and

[1] Unstable means that small changes in the training sample can result in large changes in the model.

Table 4.2: Parameter values of the investigated SEE methods. The integers in parentheses are the numbers of investigating parameter values.

Approach	Parameters
k-NN	k(#neighbours)={**1**, 3, 5, 7, 9, 11, 13} (#7)
RT	M(mim.#instance/leaf)={1, **2**, 3, 6, 12, 20} (#6) V(mim.variance for split)= {0.0001,**0.001**, 0.01, 0.1, 10} (#5) L(max.tree depth)={**-1**, 2, 6, 10, 15, 20} (#6)
MLP	L(Learning rate)={0.1, 0.2, **0.3**, 0.4, 0.5} (#5) M(Momentum)={0.1, **0.2**, 0.3, 0.4, 0.5} (#5) N(#epochs)={100, **500**, 1000} (#3) H(#hidden nodes)={***a***, 1, 3, 5, 9} (#4)
Bagging	I(iteration for Bagging)={5, **10**, 25, 50, 75} (#5) Use all possible parameter settings of the adopted base learners.
RVM	One parameter in basis function = 0.1 : 1.0 : 15 (#15)

parameter settings are consisted by enumerating all values for each parameter with all the others set to the default ones. Taking RT as an example, we will compute performance of 17 parameter settings (6 for varying M and L, and 5 for varying V) by first fixing V and L to the defaults and varying M, then fixing M and L to the defaults and varying V, and finally fixing M and V to the defaults and varying L.

Our analyses are based on the performance with the best parameter settings, with which the SEE method can achieve the best prediction performance in terms of MAE among all parameter settings in table 4.2.

4.4 Experimental Result and Discussion

This section aims to evaluate the performance of RVM compared to other investigated SEE methods to accomplish the answer to RQ2 of the thesis.

4.4.1 Evaluation of Point Estimation of RVM

This subsection aims at answering RQ2.1 presented in section 4.1: *How well does RVM perform compared to other SEE methods when used as a point estimator?* As described in appendix A.2 and illustrated in figure A.1, a point prediction of a software project is

Table 4.3: Point prediction performance of the investigated SEE methods in terms of MAE. The ranks of the methods are in the parentheses. *AveRank* denotes the average rank of each method across the data sets.

Data Set	RVM	RT	Bagging+RT	MLP	Bagging+MLP	k-NN
Maxwell	4192.40(2)	4881.22(4)	4181.26(1)	5057.00(5)	10284.98(6)	4501.18(3)
Kitchenham	1701.00(1)	2202.64(6)	1963.32(4)	2182.23(5)	1899.25(3)	1804.88(2)
Cocomo81	591.21(4)	573.09(1)	588.29(3)	654.73(5)	582.25(2)	666.27(6)
Nasa93	359.30(2)	393.40(3)	357.04(1)	664.38(6)	456.23(5)	448.87(4)
Org1	3235.50(2)	3641.39(5)	3068.28(1)	4356.59(6)	3523.73(4)	3274.68(3)
Org2	1829.00(1)	2130.05(4)	2112.60(3)	2637.30(6)	2150.11(5)	1983.81(2)
Org3	1007.29(2)	1011.94(3)	1012.59(4)	1215.63(6)	1058.60(5)	986.51(1)
Org4	3579.10(1)	4374.10(4)	4300.92(3)	4997.01(6)	4554.39(5)	4019.28(2)
Org5	5938.10(6)	5353.56(5)	5154.14(2)	5181.34(3)	4958.01(1)	5351.23(4)
Org6	2811.20(6)	2806.88(5)	2615.18(2)	2707.65(4)	2640.45(3)	2432.00(1)
Org7	4915.00(3)	4809.73(1)	5114.43(5)	5134.44(6)	4939.80(4)	4874.32(2)
aveRank	2.73	3.73	2.64	5.27	3.91	2.73

assigned as the *expectation* of the Gaussian distribution derived by RVM, for having the highest probability among the estimated effort values.

Table 4.3 lists the performance of SEE methods across 11 data sets in terms of MAE. The integers in the parentheses are the ranks of the associated SEE method. We can see that RVM and k-NN can achieve the same average rank of 2.73, ranking the second after Bagging+RT that has the best average performance across data sets. This indicates the competitive performance of RVM used as a point effort estimator.

We conduct statistical tests for a more thorough idea of the performance comparison. Friedman test with significance level 0.05 across data sets rejects the null hypothesis (H0) stating that all methods are equivalent. The p-value is 4.35 being much larger than the critical value of 2.40. Post-hoc tests with Holm-Bonferroni corrections detect significant superiority of RVM over MLP with the p-value 0.00142. Post-hoc tests cannot detect significant differences of RVM against the other methods.

Overall, RVM is competitive with other SEE methods when used as a point effort estimator. Thus, it is a promising SEE method, being worthwhile of further investigation considering its advantage of providing PIs with CLs.

Table 4.4: Hit rate values in line with $CL_{0.6827}$ and $CL_{0.9545}$. The hit rate that is much smaller than the corresponding CL is highlighted in yellow (light grey).

Data Set	$HitRate_1(\%)$	$HitRate_2(\%)$
Maxwell	44.61	74.84
Kitchenham	85.31	95.24
Cocomo81	30.79	53.17
Nasa93	68.60	82.15
Org1	86.84	91.97
Org2	33.13	56.25
Org3	83.83	95.12
Org4	76.07	90.16
Org5	74.76	86.67
Org6	90.91	95.46
Org7	79.05	89.52

4.4.2 Evaluation of Uncertain Estimation of RVM

Section 4.2 presents how to derive PIs with any CLs in Eq. (4.1) and with certain CLs in Eq. (4.2). This gives the answer to the first part of RQ2.2 outlined in section 4.1: *How to provide PIs with CLs based on RVM? How well can the PIs with CLs derived from RVM perform?* This subsection aims to accomplish answering RQ2.2 by evaluating the PIs with CLs in terms of the metrics presented in section 2.4.3.

Hit Rate of PIs with CLs

The most common evaluation metric of PIs is hit rate (HitRate) [104, 84]. When the PIs with CL_α are evaluated by T testing examples, there should be around $\alpha \times T$ projects whose effort values fall within the PIs. Hit rate can be calculated by first counting the number of projects whose effort is within the PIs and then dividing that by the total number of software projects. If the PIs with CLs are realistic, the hit rate should be around CL_α. In this thesis, hit rate is evaluated in Eq. (2.9), where the values that are equivalent or greater than their CLs are considered to be satisfactory.

This chapter evaluates the PIs with $CL_{0.6827}$ and $CL_{0.9545}$ as explained in section 4.2. The evaluation on more CLs will be reported in chapter 5. Specifically, PIs with $CL_{0.6827}$

and $CL_{0.9545}$ in terms of hit rate can be validated as follows. (1) Compute $\{f_s(y_i)\}$ as

$$f_s(y_i) = \begin{cases} 1, & \text{for } |y_i - \widehat{y_i}| < s * \widehat{\sigma_i} \\ 0, & \text{otherwise,} \end{cases} \tag{4.3}$$

where $s \in \{1, 2\}$, $\{y_i\}_{i=1}^{T}$ are effort of testing examples, and $\{\widehat{y_i}\}_{i=1}^{T}$ and $\{\widehat{\sigma_i}\}_{i=1}^{T}$ are the estimated means and STDs of probabilistic effort estimation of testing examples respectively. (2) Compute the hit rate across all testing examples for the s^{th} CL as

$$HitRate_s = \frac{\sum_{i=1}^{T} f_s(y_i)}{T}, \tag{4.4}$$

which is in line with $CL_{0.6827}$ ($CL_{0.9545}$) for s=1 (s=2). (3) If $\{HitRate_s\}_{s\in\{1,2\}}$ satisfies the criteria: $HitRate_1 \geqslant 68.27\%$ ($HitRate_2 \geqslant 95.45\%$), the PIs pass the validation.

Table 4.4 lists the hit rate in line with $CL_{0.6827}$ and $CL_{0.9545}$. We can see that the PIs with CLs can often achieve good hit rate. For instance, only three PIs with $CL_{0.6827}$ and $CL_{0.9545}$ (in yellow/light grey) cannot pass the validation with large magnitude.

Relative Width of PIs with CLs

Table 4.5 shows the median effort values, PIs with $CL_{0.6827}$ and $CL_{0.9545}$, and estimated STD of the Gaussian effort noise modelling for each data set. The median values of PIs are obtained by taking the median of lower/upper bound across all lower/upper bounds of the PIs of the testing projects. This table presents a general idea of the derived PIs in the context of SEE. We can see that the actual effort falls within the PIs with $CL_{0.6827}$ and $CL_{0.9545}$ for all data sets, showing the validation of the derived PIs.

We can also see that the PIs are relatively informative for the first seven data sets. Taking Maxwell as an example, a project manager would have 68.27% confidence that the actual effort will fall within the interval $[3177, 7339]$. Since the best MAE of SEE methods is around 4000, being approximately equivalent to the width of PI with $CL_{0.6827}$, the derived PI is considered to be of practical use. However, the PIs in Org4∼Org7 are

Table 4.5: Examples of PIs of RVM in each data set. For each data set, the *median* of the actual effort values, PIs with $CL_{0.6827}$ and $CL_{0.9545}$, and the estimated STDs are shown. The listed PIs are obtained by taking the median across all lower/upper bounds of the predicted effort values. This table provides a general idea of RVM's PIs for SEE.

Data Set	Actual effort	PI ($CL_{0.6827}$)	PI ($CL_{0.9545}$)	Estimated STD
Maxwell	5190	[3177, 7339]	[1170, 9379]	2128
Kitchenham	1557	[0, 4107]	[0, 6407]	1784
Cocomo81	98	[81, 237]	[21, 312]	70
Nasa93	252	[0, 492]	[0, 755]	250
Org1	1213	[0, 3536]	[0, 5861]	2465
Org2	2045	[1185, 2116]	[694, 2564]	439
Org3	1090	[0, 3278]	[0, 5119]	1779
Org4	3520	[0, 8382]	[0, 12624]	4360
Org5	5506	[307, 12257]	[0, 18645]	5281
Org6	2943	[0, 8024]	[0, 12530]	4453
Org7	4456	[897, 12664]	[0, 18569]	5904

too wide to be informative. Further studies found that the estimated STDs of the four data sets are much larger, all greater than 4000, directly causing wide PIs.

Overall, the PIs of CLs derived by RVM can achieve relatively good hit rate, but they can be too wide to be informative. Chapter 5 aims in the improvement on this issue.

4.5 Summary and Discussion

We introduce RVM to the context of SEE and present the way of deriving PIs with CLs. The potential of RVM in the context of SEE has been validated by answering the sub-research questions outlined in section 4.1 as follows.

• *RQ2.1: How well does RVM perform compared to other SEE methods when used as a point estimator?* Experimental results show that RVM is very competitive compared to other SEE methods, being usually ranked top two out of seven methods across 11 data sets in terms of MAE. Friedman tests detect its significant superiority to MLP and show similar performance with other SEE methods. Thus, RVM is a promising SEE method and worthwhile for further investigation.

• *RQ2.2: How to provide PIs with CLs based on RVM? How well can the PIs with CLs derived from RVM perform?* We provide the way of deriving PIs with CLs in section 4.2 and validate the PIs with two specific cases $CL_{0.6827}$ and $CL_{0.9545}$. Experimental results show that the PIs can often achieve relatively good hit rate, but they can be too wide to be informative. Thus, further studies should focus on providing better PIs with CLs.

In summary, this chapter has showed that RVM is a very promising SEE method and should be further investigated. Specifically, we encourage future research in exploiting and improving its PIs with CLs while retaining point prediction performance and hit rate. Chapter 5 will focus on this issue.

CHAPTER 5

SynB-RVM: Synthetic Bootstrap Ensemble of RVMs

5.1 Introduction

Chapter 4 introduces a Bayesian regression model, namely RVM, as the first step to implement PIs with CLs to tackle the data noise problem of SEE. Experimental results have shown that when used as a point estimator, RVM is very competitive compared with other SEE methods. But when used as an uncertain estimator, the derived PIs can be too wide to be informative.

This chapter aims to develop a novel uncertain estimation approach based on RVM for better PIs with CLs, answering the third research question of the thesis:

RQ3. Can we improve the PIs of the baseline RVM? How well is its performance compared to the state-of-the-art point/uncertain methods?

On the one hand, the proposed PIs of CLs should be wide enough to capture the effort of many software projects; on the other hand, they should be sufficiently narrow to be informative and of practical use.

[0]This chapter corresponds to RQ3 in section 1.1.3, and is based on our submitted paper [172].

Inspired by the ensemble strategy that turns weak methods into a stronger one [8, 193], we adopt Bootstrapping resampling to construct multiple 'weak' RVMs, each of which is trained on part of the training examples. Due to sampling with replacement, the Bootstrap training bag contains replicated training examples, making RVM incapable of computing the inverse of the kernel matrix in the training phrase. To address this problem, we replace the repeated training examples in each Bootstrap training bag with their *synthetic* counterparts. We also propose three ways of incorporating RVMs into a single one for a 'stronger' and 'better' final uncertain estimation. We name our method as *Synthetic Bootstrap ensemble of RVMs* (SynB-RVM).

To accomplish the answer to RQ3 of the thesis, we further divide this research question into the following three sub-research questions:

- RQ3.1: When used as a point estimator, how well can SynB-RVM perform compared with other SEE methods?
- RQ3.2: When used as an uncertain estimator, can SynB-RVM's PIs achieve adequate hit rate with narrower and more informative PIs? This can be divided into two sub-questions: (1) can the PIs adequately cover the effort of testing projects? And (2) are the PIs sufficiently narrow to be informative and of practical use?
- RQ3.3: If SynB-RVM can improve the point and uncertain estimation of its baseline RVM, which components of SynB-RVM contribute to the point and uncertain prediction performance improvement?

Answering RQ3.1 allows us for the information that how promising SynB-RVM is in terms of point prediction performance compared with other SEE methods. Answering RQ3.2 is our main objective that evaluates how well our goal of developing a 'better' uncertain effort estimator has been achieved. Answering RQ3.3 enables us to gain a better understanding of our proposed uncertain effort estimator and to find the reasons why it outperforms its base learner.

Experimental results show that when used as a point estimator, our method can either significantly outperform or have similar performance compared to other SEE methods. When used as an uncertain estimator, SynB-RVM can achieve significantly narrower (and thus more informative) PIs compared to the baseline RVM. The hit rate and relative width are no worse than other uncertain effort estimators. Overall, SynB-RVM is effective in improving uncertain prediction while offering competitive point prediction performance.

The main contribution of this chapter is to propose and validate SynB-RVM in terms of point and uncertain prediction. Moreover, we present a thorough comparison between SynB-RVM and state-of-the-art uncertain effort estimators (section 2.2). To the best of our knowledge, this is the most thorough experimental comparison on this topic.

The remaining of this chapter is organized as follows. Section 5.2 proposes our uncertain estimator for the training phase in subsection 5.2.1 and for the prediction phase in subsection 5.2.2. Section 5.3 describes the experimental design. Experimental results for answering RQ3.1 and RQ3.2 are discussed in section 5.4. Section 5.5 studies the effectiveness of the three components of SynB-RVM, answering RQ3.3. Section 5.6 discusses its implications to practice, providing a deeper understanding on our proposed method and guidelines on the choice of uncertain SEE methods. This chapter is concluded and further discussed in section 5.7.

5.2 SynB-RVM: The Proposed Uncertain Estimator

This section proposes an uncertain SEE method that adopts Bootstrap resampling to produce multiple RVMs and incorporate them for better prediction performance. The idea is similar to that of ensemble learning where several 'weak learners' can be incorporated for a 'stronger' one [8, 193]. In this sense, each Bootstrap bag emphasizes certain parts of the training space and prompts a well-behaved SEE method in this subspace. Incorporating those constructed methods may lead to an overall good uncertain prediction

5.2.1 Training Phase of SynB-RVM

Consider a training set of N software projects $\mathscr{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N$, where $\mathbf{x}^n \in \mathbb{R}^D$ denotes the input features of the n^{th} training example and y^n denotes its output effort. In the training phase, several RVMs are trained with the following three steps.

Step 1. Bootstrap Training Bag Construction

By using Bootstrap resampling with replacement on the original data set $\mathscr{D}$, SynB-RVM creates M Bootstrap training bags of size N, denoted as $\{\mathscr{D}^{(m)}\}_{m=1}^M$. *Sampling with replacement* is reasonable for SEE because it is a small data problem, and thus resampling will not take an excessive amount of time. Each Bootstrap bag $\mathscr{D}^{(m)}$ will be used to develop one RVM estimator, and we will have M trained RVMs.

Step 2. Synthetic Project Displacement

$\mathscr{D}^{(m)}$ is highly likely to contain duplicated training examples due to sampling with replacement to create training bags of size N. The replicated projects will cause invertibility problem of the kernel matrix when training RVMs [177]. To this end, we propose a displacement technique to generate synthetic counterparts for these repetitions. The effectiveness of this displacement technique will be verified in section 5.5.2.

Suppose that a training example $(\mathbf{x}, y) \in \mathscr{D}^{(m)}$ has been resampled K times. Retain one copy and displace the others along certain directions to form (K-1) different *synthetic software projects* as shown in algorithm 3. It can be interpreted as a shift of the replicated project towards a different but similar *data cloud* in the training space. After replacing all repeated training examples, we obtain a non-repeatable revised Bootstrap bag $\mathscr{D}^{(m)}$.

The reasons for displacing a repeated training example along its furthest neighbour are twofold. (1) Choosing the furthest neighbour suggests a more *diverse* Bootstrap training bag and thus is more likely to relieve the invertibility problem. This may also enhance the capability of the method for heterogeneous SEE data. (2) Disturbance of the repeated

Algorithm 3 Synthetic project displacement of SynB-RVM.

1: **Input:** (1) A software project $(\mathbf{x}, y)$ that has been resampled K times in $\mathscr{D}^{(m)}$, and (2) the initial training pool $\mathscr{D}' = \mathscr{D}$.
2: **Aim:** Retain one copy of $(\mathbf{x}, y)$ and replace its K-1 repetitions with synthetic counterparts.
3: **Procedures:**
4: (1) Find the *furthest neighbour* $\mathbf{x}'$ of $\mathbf{x}$ in $\mathscr{D}'$ according to Euclidean distance. To avoid scalability problem, each feature is standardized to have zero-mean and unit-variance.
5: (2) A synthetic counterpart is produced as a linear combination of $(\mathbf{x}, y)$ and $(\mathbf{x}', y')$ as

$$\begin{cases} \mathbf{x}^{syn} = (1-\rho)\mathbf{x} + \rho\mathbf{x}', \\ y^{syn} = (1-\rho)y + \rho y', \end{cases}$$

where the parameter $\rho \in (0, 1)$ controls the displacement degree.
6: (3) Replace one copy of $(\mathbf{x}, y)$ with $(\mathbf{x}^{syn}, y^{syn})$ in $\mathscr{D}^{(m)}$. Reset $\mathscr{D}'$ to be $\{\mathscr{D}' - (\mathbf{x}', y')\}$.
7: (4) Repeat step (1)$\sim$(3) until K-1 repetitions are replaced by their synthetic counterparts.
8: **Output:** The revised Bootstrap bag $\mathscr{D}^{(m)}$ with non-repeatable project $(\mathbf{x}, y)$.

software project based on another real SEE data can avoid the synthetic project to be too far away from the actual data. Moreover, small displacement parameter ρ is confined to further prevent too large deviation of the synthetic project from the actual one.

It is noteworthy that our data generator in chapter 3 produces one synthetic project based on one training example, which is more likely to stay in the same 'data cloud' as the expectation of the generated synthetic project stays the same. Therefore, the synthetic data generation approach in chapter 3 cannot be directly used in this chapter for being incapable of addressing the invertibility problem of the kernel matrix.

The synthetic counterpart of this chapter may not and are not necessary to be composed of 'real' software features. For instance, some synthetic feature may be decimal for an ordinal feature due to a linear combination of two integers. To keep the notation simple, we use $\{\mathscr{D}^{(m)}\}$ to denote the revised Bootstrap bags from this point onwards.

Step 3. RVM Training

The last step of the training phase is to train RVM models from Bootstrap training bags $\{\mathscr{D}^{(m)}\}$. An RVM is trained on one Bootstrap training bag $\mathscr{D}^{(m)}$ using the training procedure described in appendix A.2. The M RVM models can be trained in parallel.

Algorithm 4 Training phase of SynB-RVM.

1: **Aim:** Train M RVMs that are used in prediction phase.
2: **Input:** (1) Training software projects $\mathscr{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N$, (2) the number of Bootstrap bags M and (3) the degree of synthetic displacement ρ.
3: **Procedures**
4: (1) Bootstrap Training Bag Construction: Create M Bootstrap training bags from the data set $\mathscr{D}$ by using Bootstrap resampling with replacement.
5: (2) Synthetic Project Generation: For each Bootstrap training bag, replace the repeated training examples with their synthetic counterparts as algorithm 3.
6: (3) RVM Training: Train M RVM models, each of which is based on one revised Bootstrap training bag. The M RVM models can be trained in parallel.
7: (4) Calculate the training errors of the M RVM models according to some performance metric.
8: **Output:** (1) M trained RVM models and (2) their training errors.

The training phase of SynB-RVM is summarised in algorithm 4.

5.2.2 Prediction Phase of SynB-RVM

For a testing software project $\mathbf{x}$ with unknown effort y, the prediction phase aims to provide PIs with CLs that are wide enough to capture its actual effort and at the same time sufficiently narrow to be informative of practical use based on the trained RVMs. A second aim of prediction phase is to provide a competitive point estimate in comparison with RVM-related methods and SEE methods that have been shown to perform well. Our prediction phase consists of the following 4 steps.

Step 1. Bootstrap Uncertain Estimates

From the trained RVM models, we can obtain M Gaussian PDFs $\{\mathcal{N}(y^{(m)}, \sigma^{(m)})\}$ as the probabilistic estimates for the testing example $\mathbf{x}$, where each $y^{(m)}$ and $\sigma^{(m)}$ are the Gaussian mean and STD respectively for Bootstrap bag $\mathscr{D}^{(m)}$. To generate the final probabilistic prediction, we will combine these PDFs. As Gaussian distribution is uniquely determined by its mean and STD, this issue can be simplified into combining M pairs of $\{(y^{(m)}, \sigma^{(m)}\}$.

Algorithm 5 Prediction phase of SynB-RVM.

1: **Aim:** Provide the PI with a CL for a testing example.
2: **Input:** (1) CL_α, (2) the testing example $\mathbf{x}$, (3) the trained RVM models, (4) the training errors of these RVM models, and (5) the pruning rate τ.
3: **Procedures**
4: (1) Bootstrap Uncertain Estimates: Compute the M Gaussian PDFs of the testing example $\mathbf{x}$ using the trained RVMs and denote them as $\{\mathcal{N}(y^{(m)}, \sigma^{(m)})\}_{m=1}^{M}$.
5: (2) Bootstrap Estimate Pruning: Prune those RVMs with (a) negative estimated mean values, and (b) bad training performance in term of MAE.
6: (3) Final Probabilistic Prediction: Three methods to calculate the final probabilistic estimation of the testing example using one of the equations (5.1)∼(5.3).
7: (4) PI Construction: Convert the Gaussian PDF of the estimated effort values to CDF and derive the PI $[y_{lb}, y_{ub}]$ with CL_α using Eq. (5.4) or Eq. (5.5).
8: **Output:** PI $[y_{lb}, y_{ub}]$ with CL_α.

Step 2. Bootstrap Estimate Pruning

Before framing the final prediction, we note that: (1) some $\{y^{(m)}\}$ may be improperly negative due to the base learner RVMs being weak models, and (2) the estimates from some Bootstrap bags may not perform well in the training set and are improper to be retained in the prediction phase. Thus, it would be reasonable to prune these improper Bootstrap bags before constructing the final estimation.

Case 1. Pruning RVM Bags with Negative Estimated Mean. According to the domain knowledge of SEE, the effort values should be positive, and thus those bags with negative point estimates will be pruned.

Case 2. Pruning RVM Bags with Bad Training Performance. According to ML theory, high training error usually indicates bad prediction performance [25], and thus those bags with high training errors will be pruned. We rank Bootstrap bags according to their point prediction performance on training examples, and prune those bags among the worst $\tau \in [0, 1]$ percentage. People can choose the pruning performance metric based on practical preference. In our implementation, we use MAE in line with our main evaluation metric. Denote $M' \leqslant M$ as the number of remaining Bootstrap bags.

Step 3. Final Probabilistic Prediction

We derive the final probabilistic prediction in three ways based on $\{(y^{(m)}, \sigma^{(m)})\}_{m=1}^{M'}$.

Case 1. Empirical Mean. One of the simplest ways to derive the final probabilistic estimation is the sample means of these Bootstrap estimates as

$$\begin{cases} \hat{y} = \frac{1}{M'} \sum_{m=1}^{M'} y^{(m)}, \\ \hat{\sigma} = \frac{1}{M'} \sum_{m=1}^{M'} \sigma^{(m)}. \end{cases} \tag{5.1}$$

We are aware that the integration of multiple Gaussian PDFs is a Gaussian mixture model with multiple peaks [14, 191]. However, we assume the mixture to be a uni-peak Gaussian PDF by treating each individual PDF as one sampling of an underlying Gaussian PDF. Experimental results in section 5.4 justify the effectiveness of this treatment. Further exploitation into the Gaussian mixture model to derive PIs is left for future work.

Case 2. Uni-variant Empirical PDFs. We simulate the PDFs of $\{y^{(m)}\}$ and $\{\sigma^{(m)}\}$ based on the estimates of the trained RVM models. Then, we set the mean and STD of the final probabilistic estimate as the expectation of those two PDFs respectively. Specifically, develop the frequency histogram of $\{y^{(m)}\}$ ($\{\sigma^{(m)}\}$), where the number of bins B is automatically determined by the binning algorithm[1] with uniform width that can cover the range of elements and reveal the underlying shape of the distribution. Then, characterize the b^{th} bin by its middle point $y(b)$ ($\sigma(b)$) and calculate its frequency $f_y(b)$ ($f_\sigma(b)$). Finally, the mean and STD of the final probabilistic estimation are calculated as

$$\begin{cases} \hat{y} = E\{y(b)\} = \sum_{b=1}^{B} y(b) \cdot f_y(b), \\ \hat{\sigma} = E\{\sigma(b)\} = \sum_{b=1}^{B} \sigma(b) \cdot f_\sigma(b). \end{cases} \tag{5.2}$$

Case 3. Bi-variant Empirical PDFs. Similar to Eq. (5.2), this method is also based on empirical PDFs. However, bi-variant empirical PDF is used, so that the correlation between y and σ can be taken into account. First, develop the 2D frequency histogram for

[1] See Matlab's histogram() function.

$\{(y^{(m)}, \sigma^{(m)})\}$, denoted by $f_{(y,\sigma)}(b_1, b_2)$, where the numbers of bins (B_1, B_2) are automatically determined by the binning algorithm[1] to cover the data range and reveal the shape of the underlying distribution. Then, characterize each rectangle bin by its geometric middle point $\{(y(b_1), \sigma(b_2))\}$ and calculate its frequency $f_{(y,\sigma)}(b_1, b_2)$. Finally, mean and STD of the final probabilistic estimation are calculated as

$$(\hat{y}, \hat{\sigma}) = E\{(y(b_1), \sigma(b_2))\} = \sum_{b_1, b_2} (y(b_1), \sigma(b_2)) \cdot f_{(y,\sigma)}(b_1, b_2). \tag{5.3}$$

Step 4. Prediction Intervals Construction

Denote $\widehat{y}$ as the final estimated mean and $\widehat{\sigma}$ as the final estimated STD from one of the equations (5.1)$\sim$(5.3). Since the final estimated effort values follows the Gaussian distribution $\mathcal{N}(\hat{y}, \hat{\sigma})$, the PI with CL_α can be calculated as

$$PI_\alpha = [max\{0, \mathcal{F}^{-1}(\frac{1-\alpha}{2})\}, \mathcal{F}^{-1}(\frac{1+\alpha}{2})], \tag{5.4}$$

where $\mathcal{F}^{-1}(\beta)$ denotes the effort value located on the β-percentile of this Gaussian CDF. In particular, based on "68-95-99.7" rule of Gaussian distribution [191], the PIs with $\text{CL}_{0.6827}$, $\text{CL}_{0.9545}$, and $\text{CL}_{0.9973}$ can be simply derived as

$$[max(0, \widehat{y} - j\widehat{\sigma}), \widehat{y} + j\widehat{\sigma}], \tag{5.5}$$

for $j \in \{1, 2, 3\}$ respectively. Note that we could also derive PIs with $\text{CL}_{0.6827}$, $\text{CL}_{0.9545}$, and $\text{CL}_{0.9973}$ according to Eq. (5.4) and it is just easier when derived by Eq. (5.5). The testing phase of SynB-RVM is summarized in algorithm 5.

5.3 Experimental Design

The experiments are designed to answer RQ3.1-RQ3.3 in section 5.1. To answer RQ3.1, section 5.4.1 compares SynB-RVM and other SEE methods in terms of point prediction

[1] See Matlab's histcounts2() function.

performance. To answer RQ3.2, section 5.4.2 compare SynB-RVM and other uncertain methods in terms of hit rate and relative width to investigate whether SynB-RVM can improve uncertain prediction performance. To answer RQ3.3, section 5.5 studies SynB-RVM and its variants to explore which components of SynB-RVM are more effective for better point/uncertain prediction performance.

5.3.1 Data Sets

Experimental analyses of this chapter are based on the same data sets in section 4.3.1. Table 4.1 contains basic description of these data sets.

5.3.2 Performance Evaluation

Performance metrics in this chapter include MAE, MdAE, LSD, and SA for point prediction (section 2.4.2) and hit rate and relative width for uncertain prediction (section 2.4.3).

We apply 30 runs of 10 fold CV to validate the performance of SEE methods. The procedure is to repeat 30 times 10-fold CV with different sampling orders, for alleviating the impact of training example orders and Bootstrap displacement. The 10-fold CV is adopted because SEE data sets are usually small and there would be high bias if using small k (e.g. $k = 2$ in k-fold); whereas large k, such as leave-one-out with k equal to the size of data, may result in high variance [69]. The process is repeated 30 times (rather than 10) for stable overall prediction performance since SynB-RVM has more tuning parameters (than RVM).

We report the results using the model parameters that achieve the best point prediction performance based on the 30 runs of 10 fold CV, indicating the best performance the investigated approaches can achieve. All analyses and statistical tests are based on the mean performance across 30 runs, each corresponding to one 10-fold CV. We use no further spare testing sets for the same reasons as in section 4.3.2.

5.3.3 Point Estimation Benchmark Methods

This chapter will evaluate the performance of SynB-RVM as a point estimator against the state-of-the-art point SEE estimators including RVM, ATLM, k-NN, MLP, RT, SVR, Bagging with RVM (*Bagging+RVM*), Bagging with ATLM (*Bagging+ATLM*), Bagging with RT (*Bagging+RT*), and Bagging with SVR (*Bagging+SVR*). As long as SynB-RVM performs no worse (hopefully better) than them in terms of point estimates, its superiority can be justified considering the additional uncertain prediction provided.

RVM is chosen for being the baseline of the proposed method. ATLM is a newly proposed SEE benchmark and having been shown to performance well [188]. K-NN is chosen for being among the most popular SEE models, and due to its simplicity and intuitive interpretation that mimics the human instinctive decision-making [169, 165, 122, 114]. Several empirical studies showed its comparable and sometimes superior performance to other SEE models [165, 87, 122, 9, 114]. ANNs have been widely used in SEE, and MLP are the most common form of ANN [76]. RT are chosen for being among the most frequently used SEE models and having potential advantage for SEE [136, 187]. SVR is designed for small data problems [55], and existing work implies that it is suitable to SEE [146, 160, 146, 41]. There are several choices for SVR kernel, and linear kernel is adopted for being a better choice for SEE [146]. Bagging+RVM plays as an ensemble baseline of the proposed SynB-RVM, and Bagging+ATLM is expected to perform well due to the good performance of its base model. Bagging+RT have shown to be frequently among the best approaches across different data sets and rarely perform considerably worse than the best approach for any data set [136]. Bagging+SVR has shown to be more accurate than those based on other base learners such as MLP [76].

As discussed in appendix A.1.3, one potential issue of ATLM is that it may suffer certain numerical problems while giving effort estimate for some testing projects. For instance, it may produce an extremely large or even infinite effort prediction for a testing

example. To circumvent this numerical issue, we set up a threshold for predicted effort value of ATLM at the value of 10^6. Those estimates that surpass this threshold will not take part in performance evaluation for ATLM. The threshold is reasonable because the actual effort values of the investigated data sets are much smaller than it. This treatment is actually giving advantage to ATLM-related methods in the performance comparison.

5.3.4 Prediction Interval Benchmark Methods

We select three categories of uncertain methods in sections 2.2.1, 2.2.2, and 2.2.4 to justify the uncertain performance of SynB-RVM. The methods in section 2.2.3 are not included because they do not provide intervals that are specific to the project being predicted. Other methods in sections 2.2.4 and 2.2.5 are not compared because they cannot provide PIs and are thus out of the scope of this chapter. As discussed in chapter 4 and appendix A.2, RVM can also provide PIs with CLs, and is thus included in this comparison. Our implementation of the three groups of uncertain SEE methods is described below.

RVM

As SynB-RVM uses RVM as its base leaner and RVMs have shown competitive performance against state-of-the-art point estimators in chapter 4, we compare our uncertain approach against RVM in terms of both point and uncertain prediction.

RVM-/ATLM-Based Bootstrap Wrapped PIs

Section 2.2.1 describes previous work on this category and clarifies their major differences to SynB-RVM. This section details our implementation and denotes RVM-based (ATLM-based) Bootstrap wrapped methods by *BtstrpRVM* (*BtstrpATLM*).

We choose Laqrichi et al. [118]'s method as the implementation for this category because it does not present the issues of other Bootstrap methods: it provides PIs but not CIs as Angelis and Stamelos's [9], and it does not require expert knowledge as Klas et al.'s [104]. Our implementation follows the same procedures as Laqrichi et al.'s [118]

except that the base learner we use is RVM/ATLM instead of MLP. RVM is used for a fair comparison with the proposed RVM-based method. The replacement with RVM may even improve the performance because RVM has been shown to outperform MLP for point estimates [170]. ATLM is also used so that BtstrpATLM plays as a baseline for uncertain prediction.

Our implementation for Laqrichi et al. [118]'s method is as follows: (1) Generate a number of M training sets via Bootstrap resampling, where M is a parameter of this baseline method. (2) For each resampled training set, build a RVM/ATLM model and only consider their point predictions. Hereafter, we have M trained RVM/ATLM models. (3) To predict a testing example, we can get M point estimates from the RVM/ATLM models. (4) The point prediction produced by the previous step composes a distribution that can be used to compute the PI for the testing example corresponding to the CL α. The PI is given by $[e_{(1-\alpha)/2}, e_{(1+\alpha)/2}]$, where $e_{(1-\alpha)/2}$ and $e_{(1+\alpha)/2}$ are the effort values corresponding to the $\frac{(1-\alpha)}{2}\%$ and the $\frac{1+\alpha}{2}\%$ percentiles of the distribution respectively.

RVM-/ATLM-Based Empirical Error PIs

Section 2.2.2 describes previous uncertain methods on this category and clarifies their major differences with respect to our method. We detail our implementation and denote RVM-based (ATLM-based) empirical error-based methods by *EmpRVM* (*EmpATLM*).

We follow the procedures of the empirical version of Jørgensen and Sjøberg's [90] except that: (1) the base learner is RVM/ATLM instead of MLR, and (2) all training examples are used for prediction as the training set is too small for further selection. EmpRVM is built for a fair comparison to our method, and EmpATLM plays as a baseline for uncertain prediction. The replacement of ATLM with MLR can even improve the point prediction performance since ATLM has been shown to outperform MLR [188].

Our implementations for uncertain prediction work as follows: (1) Build the RVM/ATLM model based on all training examples. (2) Calculate the empirical (training) error dis-

tribution measured by Balanced Relative Error (BRE) that is defined for each training example:

$$BRE = \begin{cases} (Act - Est)/Act, & Act \leqslant Est, \\ (Act - Est)/Est, & Act > Est, \end{cases} \tag{5.6}$$

where Act/Est is the actual/estimated effort of the training example. Here, we have a number of BREs, each corresponding to one training example. (3) Calculate the empirical distribution of the training errors based on α-percentiles of those BREs, where α is the corresponding CL. Specifically, we use the percentile $(1-\alpha)/2$ as the minimum BRE value and the percentile $(1+\alpha)/2$ as the maximum BRE value. (4) The lower bound (lowB) and upper bound (upB) of effort PI with CL α for a testing example are calculated using the decided minimum and maximum BRE values as

$$lowB/upB = \begin{cases} Est' \div (1 - BRE), & BRE \leqslant 0 \\ Est' \times (1 + BRE), & BRE > 0 \end{cases} \tag{5.7}$$

where Est' denotes the point prediction of the testing example from RVM/ATLM.

Note that as a point estimator, the prediction of EmpRVM/EmpATLM equals to that of RVM/ATLM precisely. They differ only in the capabilities of constructing PIs.

5.3.5 Parameter Settings

The parameter values of the methods investigated in this chapter are shown in table 5.1. In particular, there are 4 tuning parameters for the proposed SynB-RVM: (1) the basis width c in RVM, (2) the number of Bootstrap bags M, (3) the degree of displacement in synthetic project generation ρ in Sec. 5.2.1 and (4) the pruning rate τ in Sec. 5.2.2. Our analyses are based on the performance with the optimal parameter combinations.

For RVM, its parameter c has been chosen from the values counting from 0.1 to 15 with step 0.2 (i.e. $\{0.1 : 0.2 : 15\}$). Deciding the optimal c for RVM, other RVM-related methods have their specific parameters tuned based on grid search while keeping c fixed, for being considered as possible ways of improving the baseline performance of RVM.

Table 5.1: Parameter values of the investigated SEE methods for SynB-RVM. The integers in parentheses are the numbers of investigating parameter values. The amount of parameter settings is designed to be similar among base learners, and to have three values for the Bagging ensemble.

Approach	Parameters
RVM	c (width) $= \{0.1 : 0.2 : 15\}$ (#75)
BtstrpRVM	c: Use the optimal c from RVM M (Bootstrap bags) $= \{30, 50, 80\}$ (#3)
BtstrpATLM EmpRVM	M (Bootstrap bags) $= \{30, 50, 80\}$ (#3) c: Use the optimal c from RVM
SynB-RVM	c: Use the optimal c from RVM M (Bootstrap bags) $= \{30, 50, 80\}$ (#3) ρ (synthetic displace) $= \{0.01, 0.1, 0.3\}$ (#3) τ (prune rate) $= \{0, 0.1, 0.2\}$ (#3)
k-NN	k (#neighbour) $= \{1 : 1 : 75\}$ (#75)
RT	L (max tree depth) $= \{-1, 2, 6, 10\}$ (#4) M (min #node per leaf) $= \{1, 2, 4, 6\}$ (#4) E (stopping error) $= \{0.0001, 0.001, 0.01, 0.1, 0.5\}$ (#5)
SVR	*kernel* = 'linear' C (regularization) $= \{0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$ (#9) ε (slack variables) $= \{0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.5, 1\}$ (#8)
MLP	L (learning rate) $= \{0.1,\ 0.3\ ,0.5\}$ (#3) M (Momentum) $= \{0.1, 0.3, 0.5\}$ (#3) N (#epochs) $= \{100, 500, 1000\}$ (#3) H (#hidden nodes) $= \{1, 5, 9\}$ (#3)
Bagging with RVM, ATLM, RT and SVR	M (Bootstrap bags) $= \{30, 50, 80\}$ (#3) Use the best parameter setting of the base learners

ATLM and EmpATLM do not have tuning parameters. For RT, the maximum tree depth -1 means the unlimited tree depth. For SVR, the conventional settings for regularization parameter C and slack variable ε are used [41, 140].

For a fair comparison, the number of parameter settings of the base learners is similar to that of RVM. Similar to those RVM-based methods, Bagging with RVM, ATLM, RT and SVR have their specific parameters tuned based on grid search while keeping the optimal parameter settings of the based learners fixed.

5.4 Evaluation of the Proposed SynB-RVM

This section aims to evaluate our SynB-RVM compared with other SEE methods of sections 5.3.3 and 5.3.4 in terms of point prediction. Hereafter, the three versions of our method in section 5.2.2 are denoted as *SynB-RVM_SpMn*, *SynB-RVM_1Dhist*, and *SynB-RVM_2Dhist* respectively.

5.4.1 Evaluation of Point Estimation

This subsection aims to answer RQ3.1 outlined in section 5.1: *When used as a point estimator, how well can SynB-RVM perform compared with other SEE methods?* Note that we have four RVM-related methods: RVM, EmpRVM, BtstrpRVM, and Bagging+RVM; similarly, we have four ATLM-related methods: ATLM, EmpATLM, BtstrpATLM, and Bagging+ATLM. However, point prediction of EmpRVM/EmpATLM/BtstrpRVM/BtstrpATLM equals to that of RVM/ATLM/Bagging+RVM/Bagging+ATLM respectively

Tables 5.2(a)∼5.2(d) list the performance in terms of MAE, MdAE, LSD, and SA. We perform Friedman tests for statistical comparisons of all methods across all data sets. The null hypothesis (H0) states that all methods are equivalent in terms of point prediction performance. The alternative hypothesis (H1) states that at least one pair of methods differs. The Friedman test with the significance level 0.05 rejects H0 with the p-values 3.11×10^{-13}, 4.11×10^{-10}, 0 and 3.38×10^{-11} for MAE, MdAE, LSD and SA respectively.

Friedman tests provide rankings of the methods computed by Eq. (B.6). Let $r_j^{(i)}$ be the rank of the j-th method on the i-th data set, and N be the number of data sets. The average rank of method j is calculated as

$$R_j = \frac{1}{N} \sum_i r_j^{(i)}. \tag{5.8}$$

The average ranks provide a reasonable idea of how SEE methods compare to each other given rejection of the null hypothesis [50]. In table 5.2, the integer in parentheses along

Table 5.2: Point prediction performance of SEE methods in terms of MAE, MdAE, LSD, and SA. The reported values are the mean of 30 runs of 10-fold CV. The first three columns correspond the three versions of our method. The ranks of an SEE method at each data set are in parentheses. The last row lists their average rank ± STD, where significant difference of Friedman tests across all data sets is highlighted in yellow (light grey). Effect size across 30 runs of each data set against the control method is computed. SynB-RVM_2Dhist is chosen as the control method as often having the best average rank among the three versions. Cells in green (light grey)/orange (dark grey) indicate better or worse performance against the control method with medium/large effect size.

(a) Point performance of all investigated methods in terms of MAE.

Data Set	SynB-RVM _SpMn	SynB-RVM _1Dhist	SynB-RVM _2Dhist	RVM (EmpRVM)	BtstrpRVM (Bagging+RVM)	ATLM (EmpATLM)	BtstrpATLM (Bagging+ATLM)	k-NN	SVR	MLP	RT	Bagging+RT	Bagging+SVR
Maxwell	3964.0(2)	3965.3(3)	3949.2(1)	4193.4(6)	4067.3(4)	4085.9(5)	6644.5(13)	4325.0(8)	5360.0(11)	6436.5(12)	4196.5(7)	4404.7(9)	5315.9(10)
Kitchenham	1682.6(5)	1674.6(4)	1668.6(3)	1725.9(6)	1779.1(7)	1603.6(2)	1389.7(1)	1784.2(8)	2230.2(11)	2399.7(13)	1989.8(10)	1879.3(9)	2231.5(12)
Cocomo81	553.4(3)	553.9(4)	554.1(5)	569.7(10)	566.9(7)	254.7(1)	277.2(2)	596.5(12)	568.6(8)	744.9(13)	589.9(11)	564.3(6)	568.8(9)
Nasa93	339.4(3)	340.7(5)	340.4(4)	355.5(6)	332.8(2)	271.4(1)	447.7(12)	438.6(11)	369.3(7)	602.8(13)	414.6(10)	376.1(9)	375.8(8)
Org1	3087.7(7)	3059.1(6)	2941.4(4)	2868.7(3)	3043.6(5)	2633.6(1)	2675.2(2)	3104.1(8)	3323.9(11)	3561.2(13)	3370.9(12)	3215.7(9)	3321.3(10)
Org2	1687.1(1)	1689.2(2)	1696.7(3)	1728.2(6)	1726.6(5)	1718.3(4)	1746.5(7)	1773.3(8)	1833.3(9)	2134.5(13)	2004.6(12)	1998.5(11)	1856.4(10)
Org3	1023.7(7)	1051.9(8)	1067.0(9)	1005.7(5)	1084.0(10)	952.4(1)	954.9(2)	993.4(3)	1208.2(11)	1543.6(13)	999.3(4)	1008.1(6)	1211.7(12)
Org4	3685.7(3)	3662.9(2)	3654.6(1)	3689.0(4)	3769.3(5)	3808.7(7)	3910.5(8)	3788.8(6)	4109.0(10)	4657.9(13)	4394.7(12)	4293.6(11)	4094.9(9)
Org5	5488.6(8)	5252.2(6)	5183.3(5)	5862.7(9)	5876.9(10)	3734.7(2)	3057.4(1)	5390.8(7)	6411.6(11)	7533.4(13)	4725.4(4)	4681.8(3)	6705.2(12)
Org6	2772.2(8)	2697.1(4)	2701.5(5)	2741.1(7)	2958.3(9)	4731.1(13)	2718.2(6)	2541.4(3)	3505.1(10)	3527.2(11)	2508.2(1)	2525.4(2)	3656.3(12)
Org7	4835.9(7)	4829.5(6)	4846.8(8)	4897.4(10)	4951.6(11)	4378.8(2)	4583.9(4)	4007.3(1)	4745.7(5)	5192.9(13)	4467.9(3)	4877.5(9)	5140.1(12)
aveRank	4.91±2.59	4.55±1.86	4.36±2.50	6.55±2.30	6.82±2.89	3.55±3.70	5.27±4.31	6.82±3.37	9.45±2.02	12.73±0.65	7.82±4.14	7.64±3.01	10.55±1.51

(b) Point performance of all investigated methods in terms of MdAE.

Data Set	SynB-RVM _SpMn	SynB-RVM _1Dhist	SynB-RVM _2Dhist	RVM (EmpRVM)	BtstrpRVM (Bagging+RVM)	ATLM (EmpATLM)	BtstrpATLM (Bagging+ATLM)	k-NN	SVR	MLP	RT	Bagging+RT	Bagging+SVR
Maxwell	2104.4(1)	2105.3(2)	2109.6(3)	2176.6(6)	2220.0(8)	2116.1(4)	2259.2(9)	2146.0(5)	3022.6(12)	3499.2(13)	2180.3(7)	2602.4(10)	3005.9(11)
Kitchenham	527.2(2)	529.0(4)	528.5(3)	643.3(8)	717.9(10)	586.7(6)	575.4(5)	473.1(1)	807.2(11)	856.1(13)	623.0(7)	656.2(9)	837.7(12)
Cocomo81	77.1(8)	76.2(6)	76.1(5)	91.9(10)	76.5(7)	33.3(1)	37.1(2)	74.6(4)	78.8(9)	172.7(13)	67.2(3)	109.8(11)	121.0(12)
Nasa93	96.3(6)	97.4(8)	96.9(7)	100.1(9)	86.5(4)	66.8(1)	68.3(2)	147.3(12)	111.1(10)	283.0(13)	84.7(3)	95.6(5)	115.9(11)
Org1	603.8(11)	602.3(9)	603.8(10)	574.5(6)	532.3(4)	488.5(2)	492.0(3)	448.7(1)	546.3(5)	712.4(13)	600.2(8)	645.4(12)	595.2(7)
Org2	674.0(2)	679.4(4)	687.2(5)	624.8(1)	675.4(3)	797.3(6)	831.9(9)	813.2(7)	828.4(8)	1015.4(11)	1052.6(12)	1219.5(13)	908.9(10)
Org3	414.7(5)	414.9(6)	413.9(4)	460.6(8)	559.0(12)	405.7(1)	407.6(2)	410.7(3)	517.6(10)	645.7(13)	458.2(7)	489.7(9)	522.9(11)
Org4	1968.0(5)	1968.6(7)	1968.6(6)	2080.8(9)	2031.3(8)	1573.7(2)	1591.7(3)	1441.5(1)	2118.1(11)	1938.2(4)	2145.9(12)	2310.1(13)	2081.3(10)
Org5	3189.2(9)	3120.8(7)	3173.3(8)	2616.5(6)	3319.3(10)	2078.0(4)	1702.7(1)	2460.9(5)	3955.1(13)	3667.9(11)	1790.5(3)	1751.6(2)	3708.1(12)
Org6	1629.3(7)	1642.3(9)	1641.4(8)	1478.4(5)	1683.7(10)	1121.7(1)	1229.2(2)	1600.2(6)	2032.2(12)	1728.7(11)	1330.1(3)	1464.0(4)	2358.8(13)
Org7	4454.8(11)	4436.6(9)	4452.0(10)	4612.1(13)	4560.7(12)	2861.1(3)	2931.3(4)	2371.7(1)	3807.7(7)	3393.8(5)	2545.6(2)	3617.0(6)	4028.6(8)
aveRank	6.09±3.51	6.45±2.34	6.27±2.53	7.36±3.11	8.00±3.19	2.82±1.94	3.82±2.79	4.18±3.40	9.82±2.40	10.91±3.30	6.09±3.62	8.55±3.78	10.64±1.80

(c) Point performance of all investigated methods in terms of LSD.

Data Set	SynB-RVM _SpMn	SynB-RVM _1Dhist	SynB-RVM _2Dhist	RVM (EmpRVM)	BtstrpRVM (Bagging+RVM)	ATLM (EmpATLM)	BtstrpATLM (Bagging+ATLM)	k-NN	SVR	MLP	RT	Bagging+RT	Bagging+SVR
Maxwell	0.6846(4)	0.6820(3)	0.6814(2)	0.8125(7)	0.8416(8)	0.7711(6)	0.9291(10)	0.8861(9)	1.0263(12)	1.8773(13)	0.6525(1)	0.6910(5)	1.0178(11)
Kitchenham	0.6462(4)	0.6439(2)	0.6434(1)	0.6460(3)	1.3559(13)	0.6602(6)	0.6500(5)	0.7368(8)	1.0394(12)	1.0211(10)	0.7580(9)	0.7081(7)	1.0351(11)
Cocomo81	1.6187(7)	1.6206(8)	1.6207(9)	1.9450(11)	1.8043(10)	0.5473(2)	0.5456(1)	2.0354(12)	1.5720(6)	2.1404(13)	1.2296(4)	1.0069(3)	1.4570(5)
Nasa93	0.8633(4)	0.8674(6)	0.8666(5)	1.1090(11)	0.9291(7)	0.7153(2)	0.6928(1)	1.4506(12)	0.9624(9)	2.1780(13)	0.9313(8)	0.8337(3)	0.9779(10)
Org1	0.9128(6)	0.8956(4)	0.8889(3)	1.0030(9)	0.9523(8)	0.8748(2)	0.8678(1)	1.0312(10)	1.2370(12)	1.5076(13)	0.9116(5)	0.9452(7)	1.2264(11)
Org2	0.7650(6)	0.7658(7)	0.7685(8)	0.7776(9)	0.8106(10)	0.6783(3)	0.6704(2)	0.6949(5)	0.8765(12)	0.8781(13)	0.6842(4)	0.6612(1)	0.8633(11)
Org3	0.7399(2)	0.7435(3)	0.7455(4)	0.7390(1)	2.1272(13)	0.7625(6)	0.7586(5)	0.8237(9)	1.0330(10)	1.2311(12)	0.7773(7)	0.7781(8)	1.0358(11)
Org4	0.8858(7)	0.8836(5)	0.8842(6)	0.9203(10)	0.8936(8)	0.8422(1)	0.8468(2)	0.9159(9)	1.0779(13)	1.0610(11)	0.8712(4)	0.8517(3)	1.0768(12)
Org5	0.9804(8)	0.9574(7)	0.9430(5)	1.0931(10)	1.0772(9)	0.8194(3)	0.8565(4)	0.9544(6)	1.1288(11)	1.7135(13)	0.7924(1)	0.8173(2)	1.2279(12)
Org6	0.9510(8)	0.9230(7)	0.9058(6)	0.9886(9)	1.0039(10)	0.8178(2)	0.7994(1)	0.8397(3)	1.2471(12)	1.2020(11)	0.8464(4)	0.8491(5)	1.2967(13)
Org7	0.9539(4)	0.9513(3)	0.9512(2)	0.9487(1)	1.1148(11)	1.0220(9)	0.9841(6)	0.9699(5)	1.0637(10)	1.1437(13)	0.9857(7)	0.9972(8)	1.1280(12)
aveRank	5.45±1.97	5.00±2.10	4.64±2.54	7.36±3.85	9.73±2.00	3.82±2.52	3.45±2.88	8.00±2.93	10.82±1.99	12.27±1.10	4.91±2.63	4.73±2.49	10.82±2.09

(d) Point performance of all investigated methods in terms of SA.

Data Set	SynB-RVM _SpMn	SynB-RVM _1Dhist	SynB-RVM _2Dhist	RVM (EmpRVM)	BtstrpRVM (Bagging+RVM)	ATLM (EmpATLM)	BtstrpATLM (Bagging+ATLM)	k-NN	SVR	MLP	RT	Bagging+RT	Bagging+SVR
Maxwell	0.5458(2)	0.5457(3)	0.5475(1)	0.5164(6)	0.5340(4)	0.5289(5)	0.0382(13)	0.5002(8)	0.3816(11)	0.2626(12)	0.5153(7)	0.4953(9)	0.3909(10)
Kitchenham	0.5500(4)	0.5526(3)	0.5540(2)	0.5435(5)	0.3834(12)	0.5750(1)	0.5327(6)	0.5259(7)	0.4090(10)	0.3572(13)	0.4727(9)	0.4966(8)	0.4022(11)
Cocomo81	0.4954(3)	0.4949(4)	0.4948(5)	0.4833(8)	0.4831(9)	0.7692(1)	0.7473(2)	0.4590(12)	0.4841(7)	0.3208(13)	0.4650(11)	0.4854(6)	0.4814(10)
Nasa93	0.5940(3)	0.5924(5)	0.5929(4)	0.5778(6)	0.6019(2)	0.6768(1)	0.4644(12)	0.4784(11)	0.5610(7)	0.2789(13)	0.5068(10)	0.5502(9)	0.5504(8)
Org1	0.5004(7)	0.5051(6)	0.5241(4)	0.5398(3)	0.5076(5)	0.5751(1)	0.5672(2)	0.4999(8)	0.4653(10)	0.4238(13)	0.4579(12)	0.4797(9)	0.4626(11)
Org2	0.4156(1)	0.4149(2)	0.4123(3)	0.3993(6)	0.4020(5)	0.4031(4)	0.3951(7)	0.3820(8)	0.3625(9)	0.2607(13)	0.3024(12)	0.3078(11)	0.3570(10)
Org3	0.5413(7)	0.5287(8)	0.5219(9)	0.5514(5)	-1.2072(13)	0.5752(1)	0.5722(2)	0.5563(3)	0.4608(10)	0.3084(12)	0.5538(4)	0.5483(6)	0.4571(11)
Org4	0.4295(4)	0.4330(2)	0.4343(1)	0.4301(3)	0.4165(5)	0.4103(7)	0.3947(8)	0.4132(6)	0.3640(10)	0.2790(13)	0.3200(12)	0.3354(11)	0.3661(9)
Org5	0.4207(7)	0.4456(5)	0.4529(4)	0.3562(8)	0.3264(9)	0.5021(3)	-0.1153(13)	0.4303(6)	0.3245(10)	0.2049(12)	0.5027(2)	0.5058(1)	0.2923(11)
Org6	0.4662(7)	0.4806(4)	0.4798(5)	0.4736(6)	0.4279(8)	0.0890(12)	-0.1062(13)	0.5114(3)	0.3270(9)	0.3207(10)	0.5181(1)	0.5137(2)	0.2959(11)
Org7	0.2370(7)	0.2380(6)	0.2353(8)	0.2233(10)	0.1030(13)	0.3043(2)	0.2582(4)	0.3636(1)	0.2471(5)	0.1807(12)	0.2898(3)	0.2305(9)	0.1890(11)
aveRank	4.73±2.33	4.36±1.86	4.18±2.56	6.00±2.10	7.73±3.82	3.45±3.47	7.45±4.66	6.64±3.35	8.91±1.81	12.36±0.92	7.55±4.32	7.36±3.32	10.27±1.01

with a method is its rank over all methods on each data set ($r_j^{(i)}$), and the last row lists the average rank of each method across all data sets (R_j). We can see that the three versions of SynB-RVM can usually outperform the other RVM-related methods. Specifically, when measured in MAE, SynB-RVM_2Dhist achieves the best average rank, SynB-RVM_1Dhist performs the second followed by SynB-RVM_SpMn with slightly worse average rank, and BtstrpRVM (and Bagging+RVM) performs the worst among all RVM-related methods. ATLM (and EmpATLM) can always have slightly better average rank than SynB-RVM, and the performance of BtstrpATLM (and Bagging+ATLM) shifts according to the performance metrics. Nevertheless, SynB-RVM usually significantly outperforms other SEE methods in terms of at least one performance metric.

Next, we conduct post-hoc tests for a more formal comparison. SynB-RVM_2Dhist is chosen as the control method for often performing the best among the three versions. For each data set, we compute the effect size across the 30 runs against SynB-RVM_2Dhist and highlight the difference in medium/large magnitude. Vargha and Delaney's A_{12} is adopted for being a non-parametric effect size and making no assumptions on the underlying distribution [183, 12]. According to Vargha and Delaney's categories [183], it is interpreted as small ($\geqslant$0.56), medium ($\geqslant$0.64), and large ($\geqslant$ 0.71). The results are as follows:

• In terms of MAE, post-hoc tests with Holm-Bonferroni corrections for comparing each method against SynB-RVM_2Dhist detect significant superiority to SVR, MLP, RT and Bagging+SVR. No significant difference can be found with respect to the three versions of SynB-RVM or to the RVM/ATLM-related methods. SynB-RVM_2Dhist has superiority to RVM (EmpRVM), BtstrpRVM (Bagging+RVM) and BtstrpATLM (Bagging+ATLM) in most data sets with medium/large effect size, but performs worse than ATLM (EmpATLM) in many data sets with medium/large effect size.

• In terms of MdAE, post-hoc tests detect that SynB-RVM_2Dhist performs significantly better than RVM (EmpRVM), BstrpRVM (Bagging+RVM), SVR, MLP, Bagging+RT

and Bagging+SVR. No significant difference can be found among the three versions of SynB-RVM. RT and k-NN have better average ranks in terms of MdAE than in terms of MAE, and perform similarly to SynB-RVM_2Dhist. ATLM (EmpATLM) and BtstrpATLM (Bagging+ATLM) have similar overall performance, but outperform our method in many data sets with medium/large effect size.

• In terms of LSD, post-hoc tests detect that SynB-RVM_2Dhist performs significantly better than BtstrpRVM (Bagging+RVM), k-NN, SVR, MLP and Bagging+SVR. No significant difference can be found among the three versions of SynB-RVM. ATLM (EmpATLM) and BtstrpATLM (Bagging+ATLM) have similar overall performance, and are superior or inferior to SynB-RVM_2Dhist with medium/large effect size depending on the data sets.

• In terms of SA, post-hoc tests detect that SynB-RVM_2Dhist performs significantly better than BtstrpRVM (Bagging+RVM), SVR, MLP, and Bagging+SVR. No significant difference is found among the three versions of SynB-RVM. Similar to MAE, SynB-RVM_2Dhist outperforms RVM (EmpRVM), BtstrpRVM (Bagging+RVM), and BtstrpATLM (Bagging+ATLM) in most data sets with medium/large effect size, but performs worse than ATLM (EmpATLM) in many data sets with medium/large effect size.

• Overall, the performance comparison slightly varies in terms of the metric, consistent with the observations in [135]. ATLM (EmpATLM) can outperform SynB-RVM with medium/large effect size in many data sets, but their performance is statistically similar across data sets. Nevertheless, our method is more consistently among the best methods regardless of the metrics.

Moreover, we discuss the magnitude of performance difference in terms of SA for its interpretability. We can see from table 5.2(d) that the magnitude of performance difference varies depending on the data sets and the competing methods. In the data sets where SynB-RVM performs worse with medium/large effect size, the magnitude is usually

of little practical significance except for EmpATLM and BtstrpATLM on Cocomo81, where SynB-RVM performs worse than EmpATLM/BtstrpATLM with 0.49 vs 0.76/0.74. In the data sets where SynB-RVM performs better with medium/large effect size, the magnitude can be very large in Maxwell against BtstrpATLM with 0.54 vs 0.038, in Org3 against BtstrpRVM with 0.52 vs -1.207, in Org5 against BtstrpATLM with 0.45 vs -0.113, and in Org6 against EmpATLM with 0.47 vs 0.089 and against BtstrpATLM with 0.47 vs -0.106. The performance improvement of SynB-RVM over RVM is usually small, but it can achieve better relative width as will be discussed in section 5.4.2. Overall, these results suggest that SynB-RVM is more likely to perform better and sometimes much better in practice.

We should note one limitation of ATLM, as discussed in sections A.1.3 and 5.3.3, ATLM-related methods may suffer numerical problems when giving effort estimation for the testing example that is very distant form any of the training examples. We circumvent it by setting up a reasonable threshold, surpassing which the predicted effort will not take part in the final prediction calculation (for BtstrpATML and Bagging+ATLM) and performance evaluation (for ATLM and EmpATLM).

An interesting observation is that ATLM-related methods can largely outperform all the others in some data sets such as Cocomo81, Nasa93 and Org5. It suggests fairly good multiple linear fittings between the transformed input features and the transformed efforts. The variable transformation includes *logarithm*, *square-root* and *none* [188]. Pearson correlations between the logarithm of (e.g.) line of codes and logarithm of effort attain fairly large values at 0.8466, 0.8435, and 0.8236 for these data sets respectively, suggesting good linearity between inputs and outputs. These results also confirm the arguments from the paper [99] saying that "with appropriate transformations, multiple linear regression can produce suitable and accurate predictive models".

Overall, experimental results show that SynB-RVM significantly outperforms RVM

(EmpRVM) and BtstrpRVM (Bagging+RVM), and performs similar to ATLM (EmpATLM) and BtstrpATLM (Bagging+ATLM). Nevertheless, SynB-RVM holds its merits over ATLM on its capability of making uncertain prediction and absence of numerical problems. It also confirms the results of Whigham et al.'s [188] stating that ATLM is competitive to state-of-the-art SEE methods.

5.4.2 Evaluation of Uncertain Estimation

This subsection aims to answer RQ3.2 outlined in section 5.1: *Can SynB-RVM's PIs achieve adequate hit rate with narrower and more informative PIs?* Evaluation of uncertain prediction should be based on the best parameter settings with respective to some performance metric such as hit rate, relative width, MAE, MdAE, LSD, or SA. In this chapter, the best parameter settings are decided in accordance with the best MAE.

Evaluation on Hit Rate

To evaluate the derived PIs in terms of hit rate, we produce PIs with twelve CLs as $\{10, 20, \cdots, 90, 68.27, 95.45, 99.73\}\%$ according to equations (5.4) and (5.5). In this settings, CLs $\{68.27, 95.45, 99.73\}\%$ are included due to their easy computation.

In practice, a hit rate that is either equivalent to or greater than their CL is considered to be satisfactory. When a hit rate is smaller than its CL, the method fails in achieving the required hit rate. In this case, the smaller the hit rate the worse the uncertain performance. When a hit rate is equivalent or greater than its CL, the method succeeds in reaching the required hit rate. In formula, the loss function of hit rate is defined as

$$\mathscr{L}(h) = \begin{cases} cl - h, & cl > h \\ 0, & cl \leqslant h \end{cases} \tag{5.9}$$

where h is the actual hit rate and cl is the corresponding CL. When the hit rate equals to or surpasses its CL, the loss is zero; when the hit rate is lower than its CL, the loss

equals to their distance.

For an idea of the achievable hit rate of uncertain methods, table 5.3(a) lists the median hit rate across 11 data sets for each method at each CL. The values in parentheses are the percentages (in 100%) of data sets that succeed in reaching the desired hit rate. We can see that our method can usually succeed in reaching the required hit rate, while most others fail in reaching them. In particular, BtstrpRVM/BtstrpATLM always has much lower hit rate than required. Though having better hit rate than BtstrpRVM/BtstrpATLM, EmpRVM/EmpATLM still rarely succeeds in reaching the CLs, i.e., the percentage of success is almost always zero. The magnitude of superiority of SynB-RVM in terms of hit rate is usually large compared to all the methods except for RVM. Taking $\text{CL}_{80\%}$ as an example, SynB-RVM_1Dhist can achieve 81.6% hit rate that is superior to the best 70.3% achieved by EmpATLM. The difference between SynB-RVM and RVM in terms of hit rate is usually small, but SynB-RVM usually surpasses the CLs. This means that adjusting the method to reduce its hit rate could potentially help improving the width of the PIs produced by this method. A possible enhancement would be to provide a non-symmetric PIs as will be discussed in the last subsection of section 5.7.2.

Table 5.3(b) lists the average rank of each method across all data sets in terms of hit rate. To perform a thorough comparison, for each of the 12 CLs, we conduct one Friedman test with the significance level 0.05 on the hit rate values. The null hypothesis (H0) states that the hit rate values are equivalent across data sets. The alternative hypothesis (H1) states that at least one pair of methods differs in terms of hit rate. All the 12 Friedman tests reject H0 with very small p-values ranging from 4.0301×10^{-14} to 1.1826×10^{-4}. After that, we conduct post-hoc tests with Holm-Bonferroni corrections for each CL. Positive/negative signs in the parentheses denote significant difference/none-difference against the control methods that have $\star$ in their parentheses. The control methods may vary for different CLs and are chosen for having the best average ranks. For instance,

Table 5.3: Evaluation of uncertain SEE methods in terms of hit rate measured in Eq. (5.9).

(a) Median hit rate across 11 data sets for each method at each CL. The values in the parentheses are the percentages (in 100%) of data sets that succeed in hit rate. Cells in yellow (light grey) highlight methods whose median values succeed in reaching or surpassing the corresponding hit rate.

CL%	RVM	BtstrpRVM	EmpRVM	BtstrpATLM	EmpATLM	#_SpMn	#_1Dhist	#_2Dhist
10.00	16.6(72.7)	4.4(0.0)	7.7(0.0)	2.5(0.0)	8.4(0.0)	12.5(81.8)	12.4(81.8)	12.5(81.8)
20.00	36.9(63.6)	9.1(0.0)	16.1(9.1)	5.7(0.0)	16.9(9.1)	24.8(72.7)	24.8(72.7)	24.7(81.8)
30.00	50.7(63.6)	14.7(0.0)	24.9(0.0)	8.8(0.0)	26.2(0.0)	39.5(63.6)	39.2(63.6)	39.2(63.6)
40.00	56.6(63.6)	19.8(0.0)	32.8(0.0)	14.4(0.0)	34.8(0.0)	50.5(63.6)	51.7(63.6)	51.6(63.6)
50.00	61.9(72.7)	24.6(0.0)	43.2(0.0)	18.8(0.0)	44.9(0.0)	57.0(54.5)	59.5(54.5)	62.5(54.5)
60.00	66.8(63.6)	29.8(0.0)	53.8(0.0)	23.2(0.0)	52.5(0.0)	67.0(54.5)	68.8(54.5)	74.8(54.5)
68.27	75.4(72.7)	31.9(0.0)	59.5(0.0)	25.0(0.0)	61.4(0.0)	77.1(63.6)	75.1(63.6)	78.6(63.6)
70.00	76.4(72.7)	33.8(0.0)	59.5(0.0)	26.6(0.0)	61.9(0.0)	78.3(63.6)	77.6(63.6)	79.8(63.6)
80.00	81.0(63.6)	40.5(0.0)	69.7(0.0)	31.6(0.0)	70.3(0.0)	81.3(63.6)	81.6(63.6)	82.2(63.6)
90.00	85.9(36.4)	50.9(0.0)	79.5(0.0)	41.1(0.0)	83.8(0.0)	86.3(27.3)	86.0(27.3)	85.9(27.3)
95.45	89.0(9.1)	53.0(0.0)	86.2(0.0)	45.4(0.0)	89.0(0.0)	88.7(9.1)	88.7(18.2)	88.8(18.2)
99.73	94.9(0.0)	61.7(0.0)	88.4(0.0)	49.1(0.0)	91.2(0.0)	93.5(0.0)	94.3(0.0)	93.4(0.0)

(b) Average ranks and statistical tests of uncertain SEE methods across 11 data sets with respect to hit rate at each CL. Positive/negative signs in the parentheses denote significant difference/none-difference against the control method by Friedman test with the significance level 0.05. The control methods for CLs have $\star$ in their parentheses. Given significant difference, medium/large effect size against the control method across all data sets is highlighted in orange (dark grey)/yellow (light grey).

CL%	RVM	BtstrpRVM	EmpRVM	BtstrpATLM	EmpATLM	#_SpMn	#_1Dhist	#_2Dhist
10.0	2.73 (−)	6.91 (+)	5.95 (+)	6.91 (+)	5.32 (+)	2.73 (−)	2.45 ($\star$)	3.00 (−)
20.0	2.77 (−)	6.95 (+)	5.86 (+)	7.09 (+)	5.18 (+)	2.68 (−)	2.86 (−)	2.59 ($\star$)
30.0	2.77 (−)	6.91 (+)	5.82 (+)	7.00 (+)	4.64 (−)	2.68 ($\star$)	3.23 (−)	2.95 (−)
40.0	2.68 ($\star$)	6.82 (+)	5.82 (+)	6.91 (+)	5.18 (+)	2.82 (−)	2.95 (−)	2.82 (−)
50.0	2.55 ($\star$)	6.91 (+)	6.00 (+)	6.82 (+)	5.00 (+)	2.82 (−)	3.09 (−)	2.82 (−)
60.0	2.45 ($\star$)	7.00 (+)	5.91 (+)	6.82 (+)	4.82 (+)	3.23 (−)	2.91 (−)	2.86 (−)
68.3	2.59 ($\star$)	6.91 (+)	6.09 (+)	6.55 (+)	4.73 (−)	2.95 (−)	3.32 (−)	2.86 (−)
70.0	2.68 ($\star$)	6.91 (+)	6.09 (+)	6.45 (+)	4.73 (−)	2.95 (−)	3.14 (−)	3.05 (−)
80.0	2.59 ($\star$)	6.73 (+)	5.91 (+)	6.64 (+)	4.45 (−)	3.05 (−)	3.50 (−)	3.14 (−)
90.0	2.77 ($\star$)	6.36 (+)	5.45 (+)	6.73 (+)	3.73 (−)	3.82 (−)	3.59 (−)	3.55 (−)
95.5	3.05 ($\star$)	6.82 (+)	5.18 (−)	6.45 (+)	3.45 (−)	3.95 (−)	3.59 (−)	3.50 (−)
99.7	3.64 (−)	6.91 (+)	4.45 (−)	6.36 (+)	2.73 ($\star$)	4.09 (−)	3.82 (−)	4.00 (−)

SynB-RVM_1Dhist has the best average rank for $CL_{10\%}$ and is chosen as the control method; RVM has the best average rank for $CL_{50\%}$ and is chosen as the control method.

We can see from table 5.3(b) that no significant difference has been found between the control methods and RVM and the three versions of SynB-RVM based on the post-hoc tests with Holm-Bonferroni corrections with the significance level 0.05 (see the negative signs associated to them). Actually, the control methods are either RVM or SynB-RVM over all CLs except for $CL_{99.70\%}$ (see the star signs). All PIs derived from BtstrpRVM

perform significantly worse than the ones from the control methods. One possible reason for their worse performance than RVM may be the invertibility problem when training RVM with replicated training examples. SynB-RVM overcomes this problem by replacing the replicated training examples with their synthetic counterparts as in algorithm 3. Similarly, post-hoc tests have found significantly worse performance for EmpRVM compared to the control methods in most hit rate. For ATLM-related methods, all PIs derived from BtstrpATLM are significantly worse than the ones from the control methods. The PIs derived from EmpATLM with CLs that are equivalent or lower than 60% perform significantly worse than the ones from the control methods; post-hoc tests cannot detect significant difference for the CLs greater than 60%, but table 5.3(a) shows large superiority of SynB-RVM to EmpATLM for the CLs until 90%.

Though higher CL is more appealing to industry, Jørgensen et al. [84] suggested "*not to ask for high confidence (90 percent, or worse, 98 percent) effort prediction intervals*" because "*lower confidence intervals are much likely to be realistic*". This is because lower CLs are more likely to be achieved in practice and thus provide more precise and useful information to the project managers. Several studies in industry and academia had also showed strong bias towards overconfidence for effort PIs [84, 91, 104], where higher CLs are not really reachable in practice. Even when we can reach higher hit rate, the width of the PI is usually too wide to be informative [84, 9, 170]. Therefore, we opt to use PIs with lower CLs such as 80% or even 60%, allowing only one/two projects to exceed the upper bound and only one/two to fall below the lower bound on average.

In summary, the experimental results and statistical tests show that SynB-RVM can usually achieve significantly better hit rate than other uncertain methods except for RVM, where they perform similarly. The performance superiority is always very large in practice especially for CLs below 90% that are more pragmatic.

Evaluation on Relative Width

We evaluate relative width based on the PIs generated previously. Larger hit rate may be associated to wider PIs, whereas smaller hit rate may be associated to narrower PIs. Therefore, if two methods have different hit rate, their relative widths are not comparable. Conversely, if two methods have the same hit rate, the one providing the narrower relative width is considered to be more informative. One question is what hit rate should be selected for the comparison. Hit rate equivalent to or greater than the CL is satisfactory. However, most methods were unable to reach their CLs. In the end, we fixed the hit rate to the values that are similar to the largest hit rate achievable by all methods.

We set up the following evaluation procedures to find the similar hit rate (table 5.4) and their corresponding relative widths (table 5.5): (1) For each data set, find the minimum of the highest hit rate across all methods and set this value as the benchmark hit rate denoted as B_HitR. The benchmark hit rate of each data set is chosen to be the highest hit rate that can be achievable by all uncertain methods. In other words, given the set composed by the highest hit rate achieved by uncertain methods for a data set, the benchmark hit rate is the lowest hit rate in this set. The benchmark hit rate values are reported in the second column of table 5.4, each corresponding to one data set. (2) For each method, find the closest hit rate to the benchmark values across all data sets and form the main body of table 5.4. Friedman test with the significance level 0.05 does not find significant difference on these hit rate values with the p-value 0.8763, indicating the similarity of these values as desired for a fair comparison of their widths. (3) Find the relative widths in line with these hit rat values and produce table 5.5. In this way, we can compare the relative widths with similar hit rate. Smaller values represent better exploitation of uncertainty and more informative PIs. Friedman test with the significance level 0.05 rejects null hypothesis (H0) with the p-value 2.28×10^{-4}, indicating that at least one pair of the methods differs.

Next, we conduct post-hoc tests for a more thorough comparison. We can see from table 5.5 that the three versions of our method can usually produce much narrower PIs compared with RVM and BtstrpRVM while reaching similar hit rate. SynB-RVM_1Dhist has the best average rank, and thus is chosen as the control method. Post-hoc tests with Holm-Bonferroni corrections have found significant difference over RVM and BtstrpRVM. No significant difference has been found over EmpRVM, BtstrpATLM or EmpATLM. Nevertheless, SynB-RVM_1Dhist has large magnitude of superiority to these methods with medium/large effect size in terms of relative width in most data sets. SynB-RVM_1Dhist performs similarly to SynB-RVM_2Dhist, but is superior to SynB-RVM_SpMn with medium/large effect size in many data sets.

In practice, SynB-RVM_1Dhist can outperform other RVM-based methods with large magnitude. For instance, SynB-RVM_1Dhist has much narrower PIs in Maxwell at 1.1964 against RVM at 5.3690 and against BtstrpRVM at 3.5965, in Cocomo81 at 1.7579 against RVM at 7.8990, against BtstrpRVM at 4.0331 and against EmpRVM at 5.9306, in Nasa93 at 2.4906 against RVM at 24.7529 and against BtstrpRVM at 8.3049. When it performs worse than some uncertain methods with medium/large effect size, the magnitude of performance inferiority is small. For ATLM-based uncertain methods, the magnitude of performance difference becomes smaller. Nevertheless, SynB-RVM still holds superiority for having better relative widths with medium/large effect size in more data sets.

Evaluation of Relative Width with Higher CLs. We compare SynB-RVM_1Dhist against EmpRVM following the same evaluation procedures in terms of relative width. SynB-RVM_1Dhist (EmpRVM) is chosen for having the best average rank among the three versions of SynB-RVM (the competitors) according to table 5.5. In this manner, we can reach higher benchmark hit rate, and thus the relative widths in line with higher hit rate can be evaluated. The first part of table 5.6 lists the benchmark hit rate values and the chosen hit rate. Wilcoxon sign-rank test with the significance level 0.05 does not find

Table 5.4: Similar hit rate of uncertain SEE methods. B_HitR denotes the benchmark hit rate being the minimum of the highest hit rate across all methods. The chosen hit rate may correspond to different CLs. The reported values are the mean of 30 runs of 10-fold CV.

Data Set	B_HitR	RVM	BtstrpRVM	EmpRVM	BtstrpATLM	EmpATLM	#_SpMn	#_1Dhist	#_2Dhist
Maxwell	0.7161	0.7538	0.7161	0.7177	0.6694	0.7099	0.7134	0.7161	0.7113
Kitchenham	0.3306	0.3687	0.2791	0.2874	0.3306	0.2917	0.2693	0.2630	0.2562
Cocomo81	0.6587	0.6587	0.6545	0.7228	0.6201	0.6201	0.6556	0.6524	0.6556
Nasa93	0.8032	0.7971	0.7921	0.8122	0.8416	0.8043	0.8036	0.8032	0.8036
Org1	0.4912	0.4890	0.4474	0.4592	0.4912	0.4759	0.5689	0.4105	0.4114
Org2	0.5010	0.5396	0.5094	0.4844	0.5010	0.5062	0.5333	0.5271	0.5292
Org3	0.3358	0.2957	0.3331	0.2963	0.3358	0.3805	0.5286	0.5185	0.5278
Org4	0.2858	0.2183	0.2945	0.2596	0.2858	0.2915	0.2470	0.2484	0.2481
Org5	0.6175	0.6175	0.6175	0.5794	0.6095	0.5730	0.6254	0.6286	0.6349
Org6	0.4758	0.3576	0.4455	0.4318	0.4758	0.4530	0.5515	0.5045	0.4530
Org7	0.3349	0.3190	0.3349	0.3651	0.3270	0.3397	0.3460	0.3508	0.3492

Table 5.5: Relative width of similar hit rate of uncertain SEE methods. The reported values are the mean of 30 runs of 10-fold CV. The last row lists the average ranks in terms of better relative width, where significant difference of Friedman tests across all data sets is highlighted in yellow (light grey). Effect size across 30 runs of each data set against the control method is computed. SynB-RVM_1Dhist is chosen as the control method for having the best average rank among the three versions of SynB-RVM. Cells in green (light grey)/orange (dark grey) indicate significantly better/worse in the control method with medium/large effect size.

Data Set	RVM	BtstrpRVM	EmpRVM	BtstrpATLM	EmpATLM	#_SpMn	#_1Dhist	#_2Dhist
Maxwell	5.3690(8)	3.5965(7)	1.3463(5)	1.1753(1)	1.5491(6)	1.2105(4)	1.1964(3)	1.1920(2)
Kitchenham	0.5440(6)	0.6119(8)	0.3736(4)	0.5602(7)	0.4309(5)	0.2880(3)	0.2782(2)	0.2758(1)
Cocomo81	7.8990(8)	4.0331(6)	5.9306(7)	0.8547(1)	0.8838(2)	1.7850(5)	1.7579(4)	1.7434(3)
Nasa93	24.7529(8)	8.3049(7)	2.6709(6)	2.4244(2)	1.4536(1)	2.5382(5)	2.4906(4)	2.4635(3)
Org1	1.2073(8)	1.0177(7)	0.7698(4)	0.9296(6)	0.7343(3)	0.8693(5)	0.5790(1)	0.5807(2)
Org2	0.7568(5)	0.8353(6)	0.7420(4)	1.0378(8)	0.8598(7)	0.6395(1)	0.6397(2)	0.6451(3)
Org3	0.5582(2)	1.1071(8)	0.4536(1)	0.6992(3)	0.7540(4)	0.8216(5)	0.8444(6)	0.9103(7)
Org4	0.5955(5)	0.7351(8)	0.5452(4)	0.6661(7)	0.6005(6)	0.4627(1)	0.4630(2)	0.4635(3)
Org5	1.5556(7)	6.3404(8)	1.1957(5)	1.1817(4)	0.8633(1)	1.2568(6)	1.1067(2)	1.1165(3)
Org6	0.8125(1)	1.6334(7)	0.8477(2)	2.5781(8)	1.2453(5)	0.9132(3)	1.5098(6)	1.2052(4)
Org7	0.9659(4)	2.4802(8)	0.7741(1)	1.1289(7)	0.9961(5)	0.8943(2)	0.9441(3)	1.0276(6)
aveRank	5.64 ± 2.38	7.27 ± 0.75	3.91 ± 1.83	4.91 ± 2.64	4.09 ± 1.98	3.64 ± 1.67	3.18 ± 1.59	3.36 ± 1.67

significant difference on the hit rate with p-value 0.8984, indicating the similarity of these values as desired for a fair comparison of their relative width.

The second part of table 5.6 lists the relative widths in line with the chosen hit rate. Wilcoxon sign-rank test with the significance level 0.05 cannot reject the H0 with p-value 0.3652. This means that in some datasets our approach is better, and in some others EmpRVM is better. Nevertheless, SynB-RVM_1Dhist can achieve better relative widths than EmpRVM with medium/large effect size in most data sets. The magnitude of performance superiority is often very large.

Table 5.6: Relative width of similar hit rate of SynB-RVM_ht1D and EmpRVM. The reported values are the mean of 30 runs of 10-fold CV. Effect size across 30 runs of each data set against SynB-RVM_1Dhist is computed. Cells in green (light grey)/orange (dark grey) indicate better/-worse in the control method with medium/large effect size.

Data Set	Hit rate			Relative width	
	B_HitR	EmpRVM	#_1Dhist	EmpRVM	#_1Dhist
Maxwell	0.7957	0.8446	0.7957	3.2417	1.6536
Kitchenham	0.9623	0.9623	0.9664	2.5938	4.1532
Cocomo81	0.7794	0.7794	0.7995	44.5693	2.7822
Nasa93	0.8032	0.8122	0.8032	2.6709	2.4906
Org1	0.9325	0.9408	0.9325	6.9109	4.2131
Org2	0.6542	0.6542	0.6333	1.0897	0.8198
Org3	0.9842	0.9842	0.9827	3.5895	8.3333
Org4	0.9347	0.9544	0.9347	5.9212	3.5339
Org5	0.8175	0.8175	0.8127	2.5726	1.8341
Org6	0.8682	0.8682	0.8576	1.7567	3.4535
Org7	0.8619	0.8619	0.8635	2.7669	2.3117

Table 5.7: Summary of performance comparison of SynB-RVM against other SEE methods. The point prediction metrics include MAE, MdAE, LSD and SA, and the uncertain prediction metrics include hit rate and relative width. Equality/positive/negative sign denotes insignificantly different/significantly better/significantly worse performance of SynB-RVM against other methods. Non-existing comparison is denoted as N/A. Note that the comparison in hit rate is an overall description across the 12 CLs.

PF Metric	RVM	Bagging+RVM	BtstrpRVM	EmpRVM	ATLM	Bagging+ATLM	BtstrpATLM	EmpATLM	k-NN	SVR	MLP	RT	Bagging+RT	Bagging+SVR
MAE	=	=	=	=	=	=	=	=	=	+	+	+	=	+
MdAE	+	+	+	+	=	=	=	=	=	+	+	=	+	+
LSD	=	+	+	=	=	=	=	=	+	+	+	=	=	+
SA	=	+	+	=	=	=	=	=	=	+	+	=	=	+
hitR	=	N/A	+	+	N/A	N/A	+	+	N/A	N/A	N/A	N/A	N/A	N/A
rWidth	+	N/A	+	=	N/A	N/A	=	=	N/A	N/A	N/A	N/A	N/A	N/A

Overall, SynB-RVM produces significantly better relative widths than those from RVM and BtstrpRVM across data sets, and is superior to EmpRVM, EmpATLM and BtstrpATLM with medium/large effect size in most data sets. The superiority magnitude can be very large over other RVM-related uncertain methods in practice.

5.4.3 Brief Summary

Table 5.7 summarizes the performance comparison in terms of point and uncertain prediction, showing the superiority of SynB-RVM in terms of overall performance:

- In terms of point prediction, SynB-RVM significantly outperforms RVM (EmpRVM),

BtstrpRVM (Bagging+RVM), k-NN, SVR, MLP, RT, Bagging+RT, and Bagging+SVR with respect to at least one performance metric; it performs similarly to ATLM (EmpATLM) and BtstrpATLM (Bagging+ATLM).

• In terms of hit rate, SynB-RVM usually achieves significantly better performance than other SEE methods except for RVM. In terms of relative width, SynB-RVM usually produces significantly better PIs with CLs than those from RVM and BtstrpRVM; it performs similarly to EmpRVM, BtstrATLM, and EmpATLM. Nevertheless, SynB-RVM has large magnitude of performance superiority to EmpRVM, BtstrATLM, and EmpATLM with medium/large effect size in most data sets.

• Therefore, SynB-RVM is a robust winner and never performs significantly worse than its competitors.

5.5 Investigation into SynB-RVM Components

This section aims to answer RQ3.3 outlined in section 5.1: *If SynB-RVM can improve the point and uncertain estimation of its baseline RVM, which components of SynB-RVM contribute to the point and uncertain performance improvement?* It can be further divided into the following sub-questions:

- RQ3.3(a): Are the three methods for deriving the probabilistic prediction similar in terms of final point and uncertain prediction?
- RQ3.3(b): Do the synthetic displacement and Bootstrap pruning of SynB-RVM contribute towards improving the final point and uncertain prediction?

This section also investigates SynB-RVM vs. Bagging ensemble in section 5.5.3 and the correlation between point and uncertain prediction in section 5.5.4. These studies provide a more thorough understanding of SynB-RVM.

5.5.1 Three Methods that Derive Final Uncertain Prediction

To answer RQ3.3(a), we can apply statistical tests on SynB-RVM_SpMn, SynB-RVM_1Dhist and SynB-RVM_2Dhist to investigate the significance of their difference. This is possible because the only difference between the three versions of SynB-RVM is the method for deriving the final probabilistic predictions according to section 5.2.2.

For point prediction, Friedman test with the significance level 0.05 is used for comparing SynB-RVM_SpMn, SynB-RVM_1Dhist, and SynB-RVM_2Dhist in table 5.2. The null hypothesis (H0) states that they are equivalent. The alternative hypothesis (H1) states that at least one pair of the three versions of the proposed SynB-RVM differs. No significant difference is detected with the p-values 0.9204, 0.7145, 0.4617 and 0.9204 in terms of MAE, MdAE, LSD, and SA respectively.

Next, we consider the difference of the three versions of SynB-RVM for uncertain prediction. In term of hit rate, we conduct a Friedman test with the significance level 0.05 on the hit rate values of SynB-RVM_SpMn, SynB-RVM_1Dhist and SynB-RVM_2Dhist shown in table 5.3(b) for each of the 12 CLs. For each CL, the null hypothesis (H0) is that the three versions of SynB-RVM are equivalent in terms of hit rate. The alternative hypothesis (H1) states that at least one pair of the three versions of SynB-RVM differs. No significant difference has been found for either of the 12 CLs with the p-values ranging from 0.4617 to 0.9966. In terms of relative width, Friedman test with the significance level 0.05 is applied to the relative widths of SynB-RVM_SpMn, SynB-RVM_1Dhist and SynB-RVM_2Dhist shown in table 5.5 that achieve similar hit rate. The null hypothesis (H0) states that they are equivalent in terms of relative width. The alternative hypothesis (H1) states that at least one pair of the methods differs. No significant difference has been found with the p-value 0.9204.

Overall, the three versions of the proposed SynB-RVM perform similarly across data sets in terms of both point and uncertain effort prediction.

5.5.2 Synthetic Displacement and Bootstrap Pruning

To answer RQ3.3(b), SynB-RVM_1Dhist is compared with its variants where *synthetic displacement* and/or *Bootstrap pruning* are/is removed contributing to *#_rmAll, #_rmSyn, and #_rmPru*. It enables us to explore the effectiveness of the two components in improving the performance of its base model RVM. SynB-RVM_1Dhist is chosen among the three versions for its best relative width and since they are shown to be statistically similar in section 5.5.1. We follow the same validation design in section 5.3 to compare the three variations against SynB-RVM_1Dhist and RVM.

Comparison of Point Estimation

Table 5.8 shows the point prediction performance of the five investigated methods across data sets. We perform Friedman tests for the statistical comparison. The null hypothesis (H0) states that all methods are equivalent in terms of point prediction performance. The alternative hypothesis (H1) states that at least one pair of methods differs. Friedman tests with the significance level 0.05 reject H0 in terms of all metrics except for MdAE where H1 has to be taken. Next, we conduct post-hoc tests for a more formal comparison between methods. Syn-RVM_1Dhist is the control method for having the best MAE, and is compared with all the others. Post-hoc tests with Holm-Bonferroni corrections for each method against Syn-RVM_1Dhist detect significant difference with respect to RVM, #_rmAll and #_rmSyn in term of MAE, LSD, and SA. No significant difference is detected with respect to #_rmPru.

Overall, analytical results verify the effectiveness of synthetic displacement and the two components as a whole in improving point prediction performance of RVM. They also suggest that synthetic displacement has more significant impact than Bootstrap pruning for point prediction performance.

Table 5.8: Point prediction performance of RVM, SynB-RVM_1Dhist, and its three variants. The reported values are the mean of 30 runs of 10-fold CV. The rank for a data set is in parentheses. The last row lists the average rank, and significant difference of Friedman tests across all data sets is highlighted in yellow (light grey).

(a) MAE. Significant difference with p-value $4.10 * 10^{-6}$.

Data Set	RVM	#_1Dhist	#_rmAll	#_rmPru	#_rmSyn
Maxwell	4193.43(5)	3965.25(1)	3982.31(4)	3965.25(2)	3980.15(3)
Kitchenham	1725.89(3)	1674.57(1)	1753.69(4)	1674.83(2)	1753.88(5)
Cocomo81	569.72(5)	553.92(1)	564.99(3)	553.98(2)	565.02(4)
Nasa93	355.52(5)	340.72(1)	341.83(3)	340.72(2)	342.91(4)
Org1	2868.65(1)	3059.13(2)	3069.56(5)	3059.86(3)	3061.54(4)
Org2	1728.16(5)	1689.17(2)	1727.95(4)	1689.17(1)	1724.15(3)
Org3	1005.73(1)	1051.88(3)	1143.41(4)	1051.88(2)	1147.44(5)
Org4	3689.02(3)	3662.91(2)	3742.61(4)	3662.91(1)	3747.12(5)
Org5	5862.69(5)	5252.19(1)	5743.85(4)	5315.49(2)	5721.11(3)
Org6	2741.10(5)	2697.05(2)	2628.74(1)	2697.05(3)	2725.18(4)
Org7	4897.38(5)	4829.49(1)	4872.56(3)	4829.49(2)	4896.32(4)
aveRank	3.91	1.55	3.55	2.00	4.00

(b) MdAE. No significant difference with p-value 0.132.

Data Set	RVM	#_1Dhist	#_rmAll	#_rmPru	#_rmSyn
Maxwell	2176.64(5)	2105.34(3)	2067.03(1)	2113.98(4)	2084.35(2)
Kitchenham	643.28(3)	529.01(1)	697.68(5)	529.01(2)	692.93(4)
Cocomo81	91.94(5)	76.16(1)	76.91(4)	76.27(2)	76.35(3)
Nasa93	100.08(5)	97.39(3)	96.34(2)	97.74(4)	93.70(1)
Org1	574.48(3)	602.31(4)	562.31(2)	605.39(5)	558.05(1)
Org2	624.81(1)	679.41(2)	698.53(4)	679.47(3)	701.90(5)
Org3	460.62(3)	414.93(1)	482.25(5)	415.46(2)	469.70(4)
Org4	2080.82(5)	1968.64(2)	2021.74(4)	1968.64(1)	2009.22(3)
Org5	2616.50(1)	3120.77(2)	3231.82(5)	3185.10(3)	3218.52(4)
Org6	1478.45(1)	1642.26(3)	1671.11(5)	1654.37(4)	1549.50(2)
Org7	4612.05(4)	4436.62(1)	4690.90(5)	4468.52(2)	4525.38(3)
aveRank	3.27	2.09	3.82	2.91	2.91

(c) LSD. Significant difference with p-value 0.008.

Data Set	RVM	#_1Dhist	#_rmAll	#_rmPru	#_rmSyn
Maxwell	0.81(5)	0.68(3)	0.68(2)	0.68(4)	0.68(1)
Kitchenham	0.65(3)	0.64(1)	0.69(4)	0.64(2)	0.69(5)
Cocomo81	1.95(5)	1.62(2)	1.67(3)	1.62(1)	1.67(4)
Nasa93	1.11(5)	0.87(3)	0.87(1)	0.87(4)	0.87(2)
Org1	1.00(3)	0.90(1)	1.01(5)	0.90(2)	1.00(4)
Org2	0.78(3)	0.77(1)	0.81(5)	0.77(2)	0.80(4)
Org3	0.74(1)	0.74(2)	1.78(5)	0.74(3)	1.58(4)
Org4	0.92(5)	0.88(2)	0.88(1)	0.88(3)	0.88(4)
Org5	1.09(5)	0.96(1)	0.99(3)	0.96(2)	1.01(4)
Org6	0.99(4)	0.92(1)	1.05(5)	0.92(2)	0.93(3)
Org7	0.95(1)	0.95(2)	0.96(4)	0.95(3)	0.96(5)
aveRank	3.64	1.73	3.45	2.55	3.64

(d) SA. Significant difference with p-value $1.68 * 10^{-7}$.

Data Set	RVM	#_1Dhist	#_rmAll	#_rmPru	#_rmSyn
Maxwell	0.52(5)	0.55(1)	0.54(4)	0.55(2)	0.54(3)
Kitchenham	0.54(3)	0.55(1)	0.16(4)	0.55(2)	0.15(5)
Cocomo81	0.48(5)	0.49(1)	0.48(3)	0.49(2)	0.48(4)
Nasa93	0.58(5)	0.59(2)	0.59(3)	0.59(1)	0.59(4)
Org1	0.54(1)	0.51(2)	0.46(5)	0.50(3)	0.48(4)
Org2	0.40(5)	0.41(2)	0.40(4)	0.41(1)	0.40(3)
Org3	0.55(1)	0.53(2)	-0.15(4)	0.53(3)	-0.41(5)
Org4	0.43(3)	0.43(2)	0.42(4)	0.43(1)	0.42(5)
Org5	0.36(3)	0.45(1)	-0.16(4)	0.44(2)	-0.27(5)
Org6	0.47(4)	0.48(2)	-0.20(5)	0.48(1)	0.48(3)
Org7	0.22(4)	0.24(2)	0.22(5)	0.24(1)	0.22(3)
aveRank	3.55	1.64	4.09	1.73	4.00

Comparison of Uncertain Estimation

For effort uncertain estimation, we follow the same procedure as in section 5.4.2 to evaluate the performance of these methods based on the 12 CLs.

Regarding hit rate, we conduct Friedman tests at significance level 0.05 for each of the 12 CLs. The null hypothesis (H0) states that the hit rate values of the methods are equivalent across data sets. The alternative hypothesis (H1) states that at least one pair of the methods differs in terms of hit rate for this CL. None of the 12 Friedman tests can reject H0 with the p-values ranging from 0.0071 to 0.8945, indicating that #_rmAll, #_rmPru and #_rmSyn produce similar hit rate to those of RVM and SynB-RVM_1Dhist.

Regarding relative width, we compare the width of PIs with similar hit rate. Table 5.9

Table 5.9: Similar hit rate of RVM, SynB-RVM_1Dhist, and its three variants. The reported values are the mean of 30 runs of 10-fold CV. B_HitR denotes the benchmark hit rate. The chosen hit rate may correspond to different CLs.

Data Set	B_HitR	RVM	#_1Dhist	#_rmAll	#_rmPru	#_rmSyn
Maxwell	0.7237	0.7538	0.7161	0.7274	0.7161	0.7237
Kitchenham	0.9664	0.9749	0.9664	0.9630	0.9667	0.9621
Cocomo81	0.4392	0.4545	0.4698	0.4392	0.4646	0.4397
Nasa93	0.7993	0.7971	0.8032	0.8014	0.8032	0.7993
Org1	0.9272	0.9289	0.9325	0.9272	0.9316	0.9307
Org2	0.7677	0.7615	0.7677	0.7781	0.7677	0.7677
Org3	0.9765	0.9823	0.9739	0.9765	0.9739	0.9770
Org4	0.9339	0.9489	0.9347	0.9344	0.9347	0.9339
Org5	0.9286	0.9302	0.9571	0.9349	0.9032	0.9286
Org6	0.9636	0.9636	0.9530	0.9636	0.9530	0.9591
Org7	0.9492	0.9111	0.9492	0.9698	0.9492	0.9603

lists the 11 benchmark hit rate and the closest actual hit rate values of the investigated methods to their corresponding benchmark values over all data sets. Friedman test with the significance level 0.05 does not find significant difference on these hit rate values. The p-value is 0.8077, showing the similarity of these values as desired for a fair comparison of their corresponding relative widths.

Relative width in line with these similar hit rate is listed in table 5.10. Friedman test with significance level 0.05 rejects null hypothesis (H0) with the p-value 3.46×10^{-4}, where the alternative hypothesis (H1) is accepted that at least one pair of the methods differs. Post-hoc tests with Holm-Bonferroni corrections by comparing each method against SynB-RVM_1Dhist detect significant difference over RVM, #_rmAll, and #_rmSyn with p-values 3.74×10^{-4}, 2.28×10^{-4}, and 2.161×10^{-2} respectively. No significant difference has been found over #_rmPru.

Overall, analytical results verify the effectiveness of synthetic displacement and the two components as a whole in improving the uncertain prediction performance of RVM. They also suggest that synthetic displacement has a more significant impact than Bootstrap pruning for uncertain prediction, being consistent with the conclusions on point prediction.

Table 5.10: Relative width of similar hit rate of the investigating methods. The reported values are the mean of 30 runs of 10-fold CV. The last row lists the average rank. Significant difference of Friedman tests across all data sets is highlighted in yellow (light grey).

Data Set	RVM	#_1Dhist	#_rmAll	#_rmPru	#_rmSyn
Maxwell	5.3690	1.1964	1.2165	1.2039	1.1998
Kitchenham	4.2984	4.1532	11.4185	4.0922	7.7285
Cocomo81	3.5037	1.0393	0.9260	1.0462	0.9203
Nasa93	7.6685	2.4906	2.5251	2.5062	2.5054
Org1	6.5292	4.2131	4.8810	4.2311	4.8688
Org2	1.7322	1.2572	1.5419	1.2651	1.4916
Org3	8.2039	6.9896	29.4600	7.0333	29.4159
Org4	4.5481	3.5339	3.2107	3.5560	3.2071
Org5	4.7894	3.3688	23.2198	2.4735	17.8844
Org6	11.2810	5.0420	13.7238	5.0735	11.8890
Org7	2.8612	3.0682	7.3683	3.0874	7.8995
aveRank	3.91	1.64	4.00	2.45	3.00

The Parameter of Bootstrap Pruning

It has been shown that Bootstrap pruning has less impact than synthetic displacement in enhancing the baseline performance of RVM. This subsection further investigates how the parameter choices of Bootstrap pruning affect point/uncertain prediction performance. This analysis also provides the information whether this component should be removed if practitioners have no time to tune its value.

Among the tuning parameters of SynB-RVM (shown in table 5.1), we find 3 pairs of (M, ρ), each corresponding to one of the pruning rates $\{\tau_0, \tau_{0.1}, \tau_{0.2}\}$. Specifically, we find the best parameter setting of (M, ρ) for τ_0 and the worst settings of (M, ρ) for $\tau_{0.1}$ and $\tau_{0.2}$ in terms of MAE, MdAE, LSD, and SA respectively. In this manner, we can compare the best performance without pruning against the worst performance with pruning. Then, their performance is compared across data sets to investigate the impact of the pruning rate. The null hypothesis (H0) states that their performance is equivalent across data sets in terms of point/uncertain prediction.

Regarding point prediction performance, Friedman tests with the significance level

0.05 across data sets reject H0 with the p-values 5.31×10^{-9}, 8.78×10^{-7}, 9.42×10^{-8} and 5.31×10^{-9} for MAE, MdAE, LSD and SA respectively. Post-hoc tests with Holm-Bonferroni corrections using the best performance without pruning as the control group detect significant superiority over the worst performance with pruning rates $\tau_{0.1}$ and $\tau_{0.2}$ in terms of all performance metrics. Effect size values across 30 runs for each data set against the control method are large/medium in all data sets except for Maxwell and Org6 in terms of MAE. This demonstrates consistent superiority of using Bootstrap pruning. The magnitude of performance difference in terms of SA varies depending on the data sets. It is usually larger for ISBSG data sets than for SEACRAFT. For instance, the values of SA in Maxwell are similar being all around 0.54, but they have larger difference for Org7 with values of 0.2380, -0.6479 and -0.6132 for τ_0, $\tau_{0.1}$ and $\tau_{0.2}$ respectively. These results indicate that the choice of Bootstrap pruning parameter is important for point prediction. Given bad parameter settings of (M, ρ), using pruning may result in worse performance than not using it.

Regarding hit rate, we conduct the same evaluation procedures as in section 5.4.2. In terms of all performance metrics, Friedman tests with the significance level 0.05 across data sets cannot reject H0 for all CLs with p-values from between 0.0518 to 1. These results indicate that the choice of Bootstrap pruning parameter is insignificant for hit rate. Regarding relative width, Friedman tests with the significance level 0.05 across data sets cannot reject H0 with the p-value 0.3209, 0.2421, 0.0719 and 0.1812 for MAE, MdAE, LSD and SA respectively. These results indicate that choice of Bootstrap pruning parameter is insignificant for relative width.

Overall, analytical results show that parameter choice of Bootstrap pruning has significant impact on point prediction performance. Bad parameter settings of Bootstrap pruning can cause worse prediction performance than not to use this technique. Whereas, the parameter choice of Bootstrap pruning has insignificant impact on uncertain predic-

tion performance significantly. In other words, hit rate and relative width are robust to the pruning rate. In this sense, practitioners are suggested not to adopt Bootstrap pruning when they do not have time to tune its parameter in case of bad point prediction performance.

5.5.3 More Comparisons with Bagging for Point Prediction

Considering that the difference between SynB-RVM_SpMn and Bagging+RVM for point prediction is *synthetic project displacement* and *Bootstrap pruning*, we can compare the two methods to judge the effectiveness of the two components as a whole. Wilcoxon sign-rank test [189] is recommended to compare two methods across multiple data sets [50]. The null hypothesis (H0) states that SynB-RVM_SpMn and Bagging+RVM equal. The alternative hypothesis (H1) states that they differ. Wilcoxon sign-rank test with the significance level 0.05 rejects H0 with the p-values 0.0098, 0.0420, 0.000977, and 0.0049 in terms of MAE, MdAE, LSD, and SA respectively, verifying the effectiveness of *synthetic displacement* and *Bootstrap pruning* together in producing better point prediction performance, which is consistent with the conclusion by comparing SynB-RVM_SpMn and #_rmAll from Friedman post-hoc tests in section 5.5.2.

Moreover, we conduct statistical tests between RVM and Bagging+RVM to study whether Bagging ensemble is sufficient to improve point prediction performance of RVM. The null hypothesis (H0) states that the two methods are equivalent. The alternative hypothesis (H1) states that they differ. Wilcoxon sign-rank test with the significance level 0.05 cannot reject H0 with the p-values 0.1475, 0.2061, 0.6377, and 0.0537 for MAE, MdAE, LSD, and SA respectively, indicating Bagging cannot essentially promote point prediction performance. It is probably because of the invertibility problem when training RVM with replicated training examples.

Overall, *synthetic displacement* and *Bootstrap pruning* of SynB-RVM have the merits

in improving the point prediction performance, but Bagging ensemble alone cannot.

5.5.4 Correlation Between Point Performance and Relative Width

This section aims to investigate how much of good relative width of SynB-RVM is contributed by good point prediction based on Spearman correlation.

Spearman's rank correlation $r_s \in [-1, +1]$ is a non-parametric statistic that assesses how well the relationship between two variables can be described by a monotonic function [60]. The value +1/-1 means a perfectly increasing/decreasing monotone of one variable over the other. Conventionally, the correlation strength $|r_s|$ is interpreted according to Fieller and Pearson's [60] as 0.00 – 0.19: very weak, 0.20 – 0.39: weak, 0.40 – 0.59: moderate, 0.60 – 0.79: strong, and 0.80 – 1.00: very strong.

Specifically, Spearman correlation between point prediction and relative width across all data sets is computed as illustrated in table 5.11. The left data column consists of the point prediction performance across data sets in terms of MAE, MdAE, LSD, or SA; the right data column consists of the relative width. The compared relative width is decided following the procedures in section 5.4.2. In the end, we have four groups of data columns as table 5.11, each corresponding to one performance metric of point prediction.

Table 5.12 lists the results. We can see that the correlation is very weak in terms of MAE, MdAE, and SA, where good point prediction has little effect on narrower PIs. Whereas, there is a *weak* correlation in terms of LSD, where better point prediction leads to narrower PIs with CLs. Therefore, point prediction sometimes has an influence on uncertain prediction. As a result, we should choose a good point estimator for use with the uncertain method. Nevertheless, the uncertain method plays a more important role in contributing narrower PIs, since the correlation between point prediction performance and relative width is *(very) weak*. The choice of uncertain method also has an impact as shown in table 5.5. In particular, some uncertain methods do better than the others.

Table 5.11: Data columns of point prediction error and relative width across SEE data sets and uncertain methods. 'PF' is the acronym of performance in terms of MAE, MdAE, LSD or SA. There are four such tables to compute Spearman correlation, one for each performance metric.

Group 1. Point Performance	Group 2. relative width
PF of Btstrp-ATLM in Maxwell	rWidth of Btstrp-ATLM in Maxwell
⋮	⋮
PF of Btstrp-ATLM in Org7	rWidth of Btstrp-ATLM in Org7
PF of Btstrp-RVM in Maxwell	rWidth of Btstrp-RVM in Maxwell
⋮	⋮
PF of Btstrp-RVM in Org7	rWidth of Btstrp-RVM in Org7
PF of Emp-ATLM in Maxwell	rWidth of Emp-ATLM in Maxwell
⋮	⋮
PF of Emp-ATLM in Org7	rWidth of Emp-ATLM in Org7
PF of Emp-RVM in Maxwell	rWidth of Emp-RVM in Maxwell
⋮	⋮
PF of Emp-RVM in Org7	rWidth of Emp-RVM in Org7
PF of SynB-RVM_SpMn in Maxwell	rWidth of SynB-RVM_SpMn in Maxwell
⋮	⋮
PF of SynB-RVM_SpMn in Org7	rWidth of SynB-RVM_SpMn in Org7
PF of SynB-RVM_1Dhist in Maxwell	rWidth of SynB-RVM_1Dhist in Maxwell
⋮	⋮
PF of SynB-RVM_1Dhist in Org7	rWidth of SynB-RVM_1Dhist in Org7
PF of SynB-RVM_2Dhist in Maxwell	rWidth of SynB-RVM_2Dhist in Maxwell
⋮	⋮
PF of SynB-RVM_2Dhist in Org7	rWidth of SynB-RVM_2Dhist in Org7

5.5.5 Brief Summary

The three versions of SynB-RVM perform similarly in terms of both point and uncertain prediction, indicating that the three ways of deriving the final prediction in section 5.2.2 are possibly similar. Another reason for the similar performance could be the small training set, based on which $\hat{y}$ and $\hat{\sigma}$ are estimated.

Table 5.13 summarizes the effectiveness of SynB-RVM's components. Synthetic displacement and the two components as a whole have the merits in improving the performance of RVM in terms of both point and uncertain prediction. Synthetic displacement has more significant impact than Bootstrap pruning for point and uncertain prediction.

Table 5.12: Spearman correlation between point prediction performance and relative width across data sets and uncertain methods. Data columns for Spearman calculation is in table 5.11.

PF metric	r_s
MAE	-0.005
MdAE	-0.091
LSD	0.297
SA	0.007

Table 5.13: Summary of the effectiveness of SynB-RVM components. The point prediction metrics include MAE, MdAE, LSD, and SA; the uncertain prediction metrics include hit rate and relative width. Equality/positive sign denotes no-different/significantly better performance of SynB-RVM_SpMn against the other method.

PF Metric	RVM	#_rmAll	#_rmPru	#_rmSyn
MAE	+	+	=	+
MdAE	=	=	=	=
LSD	+	+	=	+
SA	+	+	=	+
hitR	=	=	=	=
rWidth	+	+	=	+

5.6 Implications to Practice

5.6.1 Prediction Performance and Data Set Characteristics

This section investigates the correlation between *improvement ratio* of point and uncertain prediction and SEE data characteristics including *complexity*, *linearity*, and *clustering*.

- *Improvement ratio* of method P_1 over P_2 in terms of metric γ is defined as

$$imp\_ratio = \begin{cases} \frac{\gamma(P_2)-\gamma(P_1)}{min\{\gamma(P_1),\gamma(P_2)\}}, & \gamma \in \{\text{relative width, MAE, MdAE, LSD}\} \\ \frac{\gamma(P_1)-\gamma(P_2)}{min\{\gamma(P_1),\gamma(P_2)\}}, & \gamma \in \{\text{SA}\} \end{cases} \tag{5.10}$$

Improvement ratio of P_1 over P_2 is positive if P_1 is superior to P_2 and negative if otherwise.

- *Complexity* of a data set is defined as the division of the number of features over the number of data samples as

$$complexity = \frac{\#fea}{\#data}. \tag{5.11}$$

Larger values mean that the data set is harder for SEE.

• *Linearity* of a data set is defined by the Pearson correlation between effort values in the logarithm scale and the features such as *line of codes* and *functional size* in the logarithm scale. Logarithm scale is applied because these features and effort values of SEE data are often skewed and thus appropriate transformations, such as logarithm, are often required to form a proper and normal shape [99]. The size-related features or estimation of completion date or effort are chosen due to the well-known fact that they are usually the most correlated with the effort on the data sets used in this study [40, 122].

• *Clustering* of a data set is defined by the number of clusters the projects could be divided into. We adopt k-Means [155], for its popularity and effectiveness to improve the performance of SEE based on normalised features and Euclidean distance [163, 136].

The cluster number k is determined among $k = \{2, 3, 4, 5\}$ based on the criterion *silhouette values* [157], which measures the similarity of a project to its own cluster in comparison to the other clusters. The *silhouette value* of each project p_i is calculated as follows. (1) Compute $a(i)$ as the average distance between p_i and the other projects of the cluster where p_i locates. (2) Compute $b(i)$ as the smallest average distance of p_i to the projects across the clusters where p_i is not a member. (3) The *silhouette value* of p_i is defined as

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \in [-1, 1].$$

The average $\{s(i)\}$ across all projects measures how appropriately the projects have been clustered. The larger the *silhouette value*, the more appropriate the cluster configuration. We adopt this metric of clustering since it can be used to determine the number of clusters, provide a succinct graphical representation of how well each object lies within its cluster, and was suggested in Minku et al.'s [136].

The validating procedures are implemented by a MATLAB built-in function *evalclusters()*. Clusters with less than three projects are not considered as a valid division, and thus not counted towards the total number of clusters of the data set. As k-Means is not

a deterministic method, i.e., it cannot always retrieve the same clusters when the same data projects are used, we run ten times of the validating procedures and choose the k with the largest silhouette as the final clustering.

Specifically, we calculate the improvement ratio of SynB-RVM_1Dhist over RVM and EmpATLM on each data set, and then compute the Spearman correlation between these improvement ratios and complexity/linearity/clustering characteristics across data sets. SynB-RVM_1Dhist is chosen among the three versions for usually performing the best, RVM is chosen for being the baseline of SynB-RVM, and EmpATLM is chosen for performing the best among the ATLM-based methods. Table 5.14(a) lists the characteristics of data sets and the improvement ratios of SynB-RVM over RVM/EmpATLM. Some data sets such as Maxwell and Cocomo81 are more linear than others such as Org7 and Org4. The complexity of the data sets varies ranging from Org3 at 0.0185 (relatively easy) to Maxwell at 0.3710 (relatively hard). The number of clusters is usually not more than four. Overall, we have multiple types of data sets to represent SEE in practice.

Table 5.14(b) lists the Spearman correlation between the improvement ratio and the characteristic across SEE data sets as follows

• With respect to RVM, the improvement ratio of our method is larger in the data sets that are more complex or more linear in terms of point and uncertain prediction, shown by the large positive correlation in yellow (light grey) in table 5.14(b). Regarding complexity, a possible reason could be that by using Bootstrap resampling, our method would enlarge the training set of this Bootstrap bag, and thus obtain larger improvement over the prediction performance of the baseline RVM on more complex data sets. Regarding linearity, a possible reason could be the synthetic displacement technique of our method. The synthetic project is generated by a linear combination of a replicated training example and its furthest neighbour. Thus, a more linear data set would lead to synthetic projects with higher quality, potentially contributing to better prediction performance.

Table 5.14: Analysis of the correlation between performance and data characteristics.

(a) Characteristics of SEE data sets with respect to complexity/linearity/clustering and improvement ratio of SynB-RVM_1Dhist over RVM/EmpATLM quantified by Eq. (5.10).

Data Set	Complexity	Linearity	#Clusters	Improvement Ratio over RVM					Improvement Ratio over EmpATLM				
				rwdth	MAE	MdAE	LSD	SA	rwdth	MAE	MdAE	LSD	SA
Maxwell	0.3710	0.8184	2	3.7643	0.0575	0.0339	0.1915	0.0568	0.2940	0.0304	0.0051	0.1307	0.0318
Kitchenham	0.0207	0.7263	1	0.0372	0.0306	0.2160	0.0032	0.0167	-0.1538	-0.0443	0.1090	0.0253	-0.0406
Cocomo81	0.2698	0.8466	2	3.4993	0.0285	0.2071	0.2002	0.0240	-0.9902	-1.1752	-1.2891	-1.9609	-0.5541
Nasa93	0.1828	0.8435	3	8.8860	0.0434	0.0276	0.2786	0.0253	-0.7137	-0.2552	-0.4587	-0.2126	-0.1424
Org1	0.0395	0.7422	4	0.5504	-0.0664	-0.0484	0.1199	-0.0687	-0.0582	-0.1616	-0.2329	-0.0238	-0.1386
Org2	0.0938	0.6763	4	0.5631	0.0231	-0.0874	0.0154	0.0393	0.0109	0.0172	0.1735	-0.1290	0.0293
Org3	0.0185	0.8019	2	-0.0168	-0.0459	0.1101	-0.0061	-0.0429	-1.7163	-0.1045	-0.0227	0.0256	-0.0879
Org4	0.0246	0.6263	4	0.2869	0.0071	0.0570	0.0415	0.0067	0.1111	0.0398	-0.2510	-0.0493	0.0552
Org5	0.1429	0.8236	3	0.4375	0.1162	-0.1927	0.1417	0.2509	0.4768	-0.4063	-0.5018	-0.1684	-0.1267
Org6	0.1364	0.7845	1	1.5207	0.0163	-0.1108	0.0710	0.0148	-0.5763	0.7542	-0.4641	-0.1286	4.3986
Org7	0.1429	0.5138	2	0.2805	0.0141	0.0395	-0.0027	0.0660	0.9841	-0.1029	-0.5507	0.0743	-0.2784

(b) Spearman correlation between the improvement ratio and complexity/linearity/clustering across SEE data sets. The moderate/(very) strong correlation is highlighted in yellow (light grey).

Metric	SynB-RVM_1Dhist vs RVM			SynB-RVM_1Dhist vs EmpATLM		
	Complexity	Linearity	Clustering	Complexity	Linearity	Clustering
rwdth	0.7927	0.5727	0.1180	0.2323	-0.4818	0.2691
MAE	0.6241	0.5364	-0.2171	-0.2642	-0.6000	-0.1605
MdAE	-0.2460	-0.0727	-0.3540	0.1454	-0.3091	0.1416
LSD	0.7563	0.7455	0.1794	-0.2597	-0.5091	-0.2785
SA	0.6606	0.1273	-0.0566	-0.3235	-0.3273	-0.0755

• With respect to EmpATLM, the improvement ratio of SynB-RVM is smaller in more linear data sets in terms of point and uncertain prediction. A possible reason is that EmpATLM is designed (and thus should be more suitable) for more linear data sets, and such factor is stronger than the enhancement of more linear data to SynB-RVM. With respect to clustering, the correlation is (very) weak and thus neglectable for the improve ratios over both RVM and EmpATLM. It means that the clustering of a data set does not have impact on the choice of adopting our method against other uncertain methods.

Overall, when the data sets are harder or more linear, practitioners are suggested to use SynB-RVM rather than RVM for better point and uncertain prediction. The more linear the data set, the smaller the improvement ratio of SynB-RVM over EmpATLM.

5.6.2 Understandability vs. Better Performance

This section aims to discuss the trade-off between understandability and better prediction performance of uncertain methods.

ATLM is easy to understand as a variant of MLR; whereas RVM lays its foundation on Bayesian framework and thus requires more background knowledge to understand. As our proposed methods are based on RVM, the model is more difficult to interpret than those based on ATLM. Nevertheless, the mechanisms used to produce the PIs of the uncertain approaches have conceptually equal understandability: EmpSEEr[1] employs the error distribution of training examples to decide the upper and lower bounds of the PI of a testing example; BtstrpSEEr[2] extracts two values from multiple point estimates to form the upper and lower bounds of the PI; SynB-RVM integrates multiple uncertain predictions into one through averaging. Their final point estimate is either a single value (for EmpSEEr) or the mean of multiple values (for BtstrpSEEr and SynB-RVM).

When practitioners are more keen to understand the SEE models, ATLM-related methods would be more attractive with a sacrifice on uncertain prediction performance. In particular, if practitioners do not value PIs, ATLM would provide both interpretability and good point prediction, being recommended. However, when practitioners are more concerned on uncertain prediction, SynB-RVM would be their best option for being robust and having statistically better or similar point and uncertain prediction performance.

Overall, there is a trade-off between superiority in prediction performance and understandability when selecting an SEE method.

5.6.3 Time Complexity of Uncertain Methods

This section analyses time complexity of uncertain methods with respect to training and prediction phases given training set size N. Since uncertain methods are based on ATLM or RVM, we would analyse the complexity of RVM and ATLM at first.

As ATLM is a variant of MLR with automated data transformation (appendix A.1), its time complexity is $\mathbb{O}(N^3)$ for model training and $\mathbb{O}(N)$ for prediction, ignoring the

[1]EmpSEEr denotes EmpRVM or EmpATLM.
[2]BtstrpSEEr denotes BtstrpRVM or BtstrpATLM.

time complexity of data preprocessing. Note that ATLM itself cannot provide uncertain prediction and needs to be integrated with EmpSEEr/BtstrpSEEr.

The model parameters of RVM need to be updated alternatively [177]. Suppose K to be the iterations by which the learning algorithm of RVM converges. At each iteration, the training process includes an $N \times N$ matrix multiplication and a $N \times N$ matrix inversion, leading to the overall complexity $\mathbb{O}(KN^3)$. The testing phase includes the multiplication of an $N \times 1$ vector and an $N \times N$ matrix, leading to the complexity $\mathbb{O}(N^2)$.

BtstrpSEEr's training phase consists of constructing M base models based on the M Bootstrap bags, leading to $\mathbb{O}(M * N^3)$ for BtstrpATLM and $\mathbb{O}(M * KN^3)$ for BtstrpRVM (section 5.3.4). Its prediction phase includes: (1) giving M point estimates, (2) sorting the M point estimates, and (3) extracting certain percentiles for upper and lower bounds of the PI. Thus, the time complexity of the prediction phase is $\mathbb{O}(M * V) + \mathbb{O}(M^2) + \mathbb{O}(1)$, where V is the prediction complexity of the base model and $\mathbb{O}(M^2)$ is the worst complexity of sorting M values. In particular, the prediction complexity is $\mathbb{O}(M * N) + \mathbb{O}(M^2)$ for BtstrpATLM and $\mathbb{O}(M * N^2) + \mathbb{O}(M^2)$ for BtstrpRVM.

EmpSEEr's training phase consists of (1) constructing base model, (2) computing the training errors, (3) sorting the error of size N, and (4) extracting certain percentiles of the error for upper and lower bounds of the PI (section 5.3.4). Accordingly, the time complexity is $\mathbb{O}(V) + \mathbb{O}(N) + \mathbb{O}(N^2) + \mathbb{O}(1)$, where V is the training complexity of the base model and $\mathbb{O}(N^2)$ is the worst time complexity of sorting N values. In particular, the training time complexity is $\mathbb{O}(N^3)$ for EmpATLM and $\mathbb{O}(KN^3)$ for EmpRVM. EmpSEEr's prediction phase consists of (1) giving the point estimate of the testing example and (2) computing the lower and upper bounds of PIs as Eq. (5.7). Accordingly, the time complexity is $\mathbb{O}(V) + \mathbb{O}(1)$. In particular, the prediction time complexity is $\mathbb{O}(N)$ for EmpATLM and $\mathbb{O}(N^2)$ for EmpRVM.

SynB-RVM's training phase consists of constructing M RVMs from the M Boot-

Table 5.15: Time complexity of uncertain SEE methods with respect to training and prediction phases. Denote N as the training size, M as the number of Bootstrap bags, and K as the iterations that the algorithm of RVM converges. Note that ATLM itself cannot provide uncertain prediction. In practice, the training and prediction processes of BtstrpSEEr/SynB-RVM can be proceeded in parallel, largely reducing time complexity.

Complexity	ATLM	RVM	BtstrpATLM	BtstrpRVM	EmpATLM	EmpRVM	SynB-RVM
training	$\mathbb{O}(N^3)$	$\mathbb{O}(KN^3)$	$\mathbb{O}(M*N^3)$	$\mathbb{O}(M*KN^3)$	$\mathbb{O}(N^3)$	$\mathbb{O}(KN^3)$	$\mathbb{O}(M*KN^3)$
prediction	$\mathbb{O}(N)$	$\mathbb{O}(N^2)$	$\mathbb{O}(M*N)+\mathbb{O}(M^2)$	$\mathbb{O}(M*N^2)+\mathbb{O}(M^2)$	$\mathbb{O}(N)$	$\mathbb{O}(N^2)$	$\mathbb{O}(M*N^2)$

strap bags (section 5.2.1), leading to the complexity $\mathbb{O}(M*KN^3)$. SynB-RVM's prediction phase consists of (1) obtaining M uncertain estimates of the testing example, each corresponding to one of the RVMs, (2) combining the M uncertain estimates using Eq. (5.1)∼(5.3), and (3) extracting certain percentiles for upper and lower bounds of PIs (section 5.2.2), leading to the complexity $\mathbb{O}(M*N^2)+\mathbb{O}(M)+\mathbb{O}(1)=\mathbb{O}(M*N^2)$.

Table 5.15 summarizes the time complexity of uncertain methods. We can see that EmpATLM, EmpRVM, and RVM have lower time complexity in both training and prediction phases than SynB-RVM. Nevertheless, SynB-RVM can achieve significantly better point/uncertain prediction performance as shown in table 5.7. In practice, the time complexity of SynB-RVM can be largely reduced by proceeding the training and prediction processes of M Bootstrap ensembles in parallel.

Overall, there is a trade-off between computational cost and good uncertain prediction performance: the practitioners are suggested to take faster methods such as EmpATLM when they are more concerned with the computational efficiency; whereas, they are suggested to take SynB-RVM when they are keen for better uncertain prediction.

5.7 Summary and Discussion

5.7.1 Summary

This chapter ensembles a set of RVMs into a unified uncertain effort estimator, answering the third research question of the thesis. The proposed SynB-RVM adopts Bootstrap

resampling to produce multiple RVMs based on adapted training bags whose replicated training examples are replaced with their synthetic counterparts. Those RVMs are then incorporated RVMs into a unified uncertain effort estimator.

The performance of SynB-RVM has been validated by answering the three sub-research questions outlined in section 5.1 as follows.

• *RQ3.1: When used as a point estimator, how well can SynB-RVM perform compared with other SEE methods?* Experimental results show that SynB-RVM can either significantly outperform or have similar point prediction performance compared to other SEE methods. Specifically, SynB-RVM significantly outperforms RVM (EmpRVM), BtstrpRVM (Bagging+RVM), k-NN, SVR, MLP, RT, Bagging+RT, and Bagging+SVR with respect to at least one performance metric; it performs similarly to ATLM (EmpATLM) and BtstrpATLM (Bagging+ATLM).

• *RQ3.2. When used as an uncertain estimator, can SynB-RVM's PIs achieve adequate hit rate with narrower and more informative PIs?* In terms of hit rate, SynB-RVM usually has significantly better performance than other SEE methods except for RVM. In terms of relative width, SynB-RVM usually produces significantly better PIs with CLs than those from RVM and BtstrpRVM; it performs similarly to EmpRVM, BtstrATLM, and EmpATLM. Nevertheless, SynB-RVM has performance superiority with medium/large effect size over EmpRVM, BtstrATLM, and EmpATLM in most data sets.

• *RQ3.3. Which components of SynB-RVM contribute to the point and uncertain prediction performance improvement?* Analytical results show that the three versions of SynB-RVM perform similarly in terms of point and uncertain prediction. SynB-RVM has two key components: synthetic displacement and Bootstrap pruning. Analytical results show the effectiveness of synthetic displacement and the two components as a whole in improving the performance of RVM in terms of both point and uncertain prediction. They also suggest that synthetic displacement has a more significant impact than Bootstrap

pruning on point and uncertain prediction.

Besides the main contributions in proposing and validating a novel uncertain estimator, we are the first to provide a thorough experimental comparison on uncertain SEE methods. Due to the encouraging results, SynB-RVM is likely to help project managers to make better informed decisions by accessing the project management risks.

5.7.2 Discussion

This subsection discusses some points regarding SynB-RVM as follows.

SynB-RVM's Base Model

SynB-RVM needs a base model, like RVMs, which are capable of deriving uncertain effort estimation themselves. It then combines those uncertain estimates into a unified one when making an estimate for a testing example. Therefore, the prediction uncertainty originates from each base model when giving uncertain prediction for the testing example. However, other uncertain methods utilize only the point estimates of their base models, from which their prediction uncertainty of the testing example is derived. Therefore, our method can be considered to use richer uncertain information to produce PIs with CLs than other uncertain methods. Moreover, our method can be considered as a way to combine potentially 'weaker' uncertain estimates into a stronger one. Experimental results in comparing the uncertain estimation between RVM and SynB-RVM (Sec. 5.4.2) verify this statement.

SynB-RVM's Extension

SynB-RVM can be extended with other base models as long as they can provide an uncertain estimate for a testing example. The base model can be a single estimator like RVM that can provide uncertain prediction itself or an ensemble of point estimators like EmpRVM that can produce uncertain prediction as a group. In this sense, a more general name of the proposed approach can be *Synthetic Bootstrap ensemble of Probabilistic Pre-*

dictors. The experimental investigation and evaluation on the more general framework are left as future work.

Non-Symmetric Uncertain Effort Estimation

SynB-RVM assumes that the predicted effort values of a software project follow Gaussian distribution as discussed in section 2.2.2. The Gaussian assumption is reasonable according to *central limit theorem* [145] by considering the noise factors as independent random processes. However, this effort noise assumption disregards the fact that the software efforts have to be positive.

With the assumption of Gaussian effort noise, the probabilistic prediction is symmetric as shown in figure A.1. However, when the PDF of the estimated effort values overlaps the negative quadrant, the positive requirement of effort values would push the negative values right-moved to be non-negative, making the symmetric probability right-skewed.

Consequently, non-symmetric right-skewed PIs would be better, where the right parts of PIs are wider and heavier than the left. Accordingly, the Gaussian effort noise assumption may be problematic in causing less informative PIs when the predicted effort values overlap the negative quadrant since the symmetric PIs are produced. To cater such issue, we can relax the Gaussian assumption of effort noise in RVM by considering other non-symmetric effort noise models such as righted-skewed Gaussian [14] or Gamma distribution [80]. However, the revision on the noise assumption of RVM would fail the analytical solutions, and lead to considerably complicated deductions. A potential simpler solution is to adjust the predicted effort probability slightly rightward according to the skewness of the point estimates from RVMs. We leave it as our future work for further improving our method's uncertain estimation.

CHAPTER 6

Sensitivity to Parameter Settings for SEE Methods in Online Scenario

6.1 Introduction

Prediction performance of SEE methods can be affected by many factors, frequently leading to different conclusions [98, 83, 125]. Examples of factors include the data sets investigated, the performance metrics, the data preprocessing procedures, the ways to divide data into training and testing sets, and the amount of parameter tuning for SEE methods [134]. The effect of different data sets and performance metrics has been relatively well known [164, 63, 135, 136], but there has been little work investigating the effect that different parameter settings may have for SEE methods.

In SEE literature, the methodology of choosing parameter settings is frequently omitted from the experimental framework, seemingly making an implicit assumption that parameter settings will not change the performance of SEE methods significantly [136]. However, it is good to know to what extent different parameter settings affect the perfor-

[0]This chapter corresponds to RQ4 in section 1.1.4, and is based on our published paper [169].

mance of SEE methods. Such knowledge will be very valuable for the SEE community, which could guide the choice of SEE methods. For instance, it would be useful to know whether a method that frequently performs better is in fact highly dependent on the fine tuning of its model parameters. Therefore, sensitivity to parameter settings should be taken as another criterion for evaluating SEE methods.

Chronology of the software projects is another issue that is worthwhile to consider for evaluating SEE methods in practice. Most SEE methods take an *offline scenario*, where the training and testing projects are taken as a chunk of stationary training examples without change or chronological ordering [136, 170, 171, 187]. However, when the effort of a new project is predicted, SEE methods can only use projects that have already been completed by that time step. Besides, the environment where the SEE methods operate is unlikely to be stationary (e.g. new employees can be hired or lost) and the characteristic of the completed software projects is more likely to change with time. Therefore, it is more rational to take an *online scenario* for SEE, where completed projects keep arriving with time and participating in the training process. As a result, analysis of sensitivity to parameter settings should consider not only the effect of parameter settings across time steps, but also the effect throughout time steps (i.e., at each time step).

This chapter aims to investigate the sensitivity to parameter settings for SEE methods in the online scenario by answering the fourth research question of the thesis:

RQ4. To what extent do parameter settings affect the performance of SEE methods, and should we pay attention to their parameter tuning?

To answer RQ4, systematic experiments will be conducted based on five popular SEE methods with different parameter settings on three SEE data sets that have chronological information. The performance of the *best*, *default* and *worst* parameter settings for each method on each data set will be computed, based on which the sensitivity to different parameter settings will be analysed.

Given an SEE method and a chronological data set, RQ4 can be further divided into the following three sub-research questions:

- RQ4.1: How sensitive is an SEE method to its parameter settings in terms of average performance across time steps?
- RQ4.2: Does the best parameter setting of an SEE method in terms of average performance across time steps perform consistently well throughout time steps in comparison to other parameter settings?
- RQ4.3: Bagging ensemble of SEE methods has been shown to obtain good results. Could Bagging help to lessen the base learners' sensitivity to parameter settings?

Answering RQ4.1 and RQ4.2 can provide an insight on which of the SEE methods are more sensitive to their parameters. Answering RQ4.3 enables us to gain a better understanding of the behaviour of Bagging ensemble in view of the sensitivity to parameter settings.

Experimental results show that while some SEE methods such as Bagging+RT are not very sensitive to parameter settings, others such as MLP are affected dramatically. Combining SEE methods into Bagging can help the performance of the default parameter settings closer to the best parameter ones. In other words, the performance of Bagging ensemble is more robust to different parameter settings. The average performance of k-NN is not so much affected by different parameter settings.

The remaining of this chapter is organised as follows. Section 6.2 presents the analytical methodology and experimental framework to answer RQ4 of the thesis including the online scenario for SEE, the data sets investigated, and the SEE methods. Section 6.3 discusses the experimental results. This chapter is summarized in section 6.4.

6.2 Analysis Methodology and Experimental Design

This section presents our analysis methodology and the experimental design for investigating the sensitivity to parameter settings of SEE methods.

6.2.1 Online Scenario

As discussed in section 6.1, most SEE studies take an *offline scenario* [136, 170, 171, 187], where models are built upon a set of projects (training examples) and tested in another set of projects (testing examples). In the offline scenario, the training and testing projects are taken as a chunk of stationary training examples without order or change.

However, software developing companies and their employees are unlikely to be stationary but evolve with time. For instance, new employees can be hired or lost, training can be provided and the employees can become more experienced, new types of software projects can be accepted, the management strategy can change, and new programming languages can be introduced [138]. As such changing environment is highly likely to affect the performance of SEE methods [136], it is rational to consider the chronology of projects. It is also important to evaluate SEE methods considering not only their overall performance across time steps, but also the performance throughout time.

Similarly to Minku and Yao's [136], we consider the online scenario, where a new project is received as a training example at each time step, forming a *data stream*. At each time step, after an SEE method is trained on the completed projects received so far, next ten projects of the data stream are predicted. In other words, the performance of the SEE method at this time step is calculated based on the following ten projects. We consider ten as reasonable because not many projects are produced per year by a company. SEE methods investigated in this chapter are assessed according to this evaluation process.

6.2.2 Data Sets with Chronological Information

Three data sets including Kitchenham, Maxwell, and SingleISBSG are chosen in this work for being available to access the chronological information that can be used to sort projects to perform online scenario. The detailed description of Kitchenham and Maxwell can be found in section 2.4.1 except that the projects are sorted by the actual starting date plus

the duration, which approximately corresponds to the completion order of the projects.

SingleISBSG is a subset of ISBSG Repository Release 10, previously used in [136]. It comprises 69 projects from a single-company with the following characteristics:

- Data and function points of quality A (assessed as being sound with nothing being identified that might affect their integrity) or B (appears sound but there are some factors which could affect their integrity).
- Recorded effort that considers only development team.
- Normalised effort equal to total recorded effort, meaning that the reported effort is the actual effort across the whole life cycle.
- Functional sizing method IFPUG version 4+ or identified as with addendum to existing standards.
- Implementation date after the year 2001.

The following procedures are performed to preprocess the software projects:

1. Sort the projects according to the implementation date.
2. Select development type, language type, development platform, and functional size as input features, as recommended by ISBSG [77]. The output variable is the effort in hours. Remove all the other variables.
3. Treat missing values using 1-NN imputation [37]. Only two projects contained missing values.

6.2.3 Benchmark SEE Methods

This chapter investigates five SEE methods: RT, k-NN, MLP, Bagging with RT (Bagging+RT), and Bagging with MLP (Bagging+MLP). An online learning class was developed so that the WEKA implementations of these methods could be used. RT was based on REPTree without pruning, k-NN was based on IBK with normalised attributes and Euclidean distance, and the other methods were based on the implementations with the same name in WEKA.

RT, Bagging+RT and Bagging+MLP are chosen because they have been shown to perform well in comparison to other SEE methods [136]. Nevertheless, the evaluation of these methods has not considered their sensitivity to parameter settings. Knowledge on whether these methods are very sensitive to parameter choices would be important for deciding whether to use them. Bagging ensemble has been shown to be able to improve the frequency that their base learners are ranked first in terms of MAE [136]. Thus, it would be good to know whether they could also make these approaches less sensitive to parameter settings.

K-NN is among the simplest SEE methods, and is included in the analysis because it can perform frequently very well, but sometimes considerably worse than the best SEE method depending on the data set [136]. It would be useful to know whether the sensitivity to data sets also applies to the sensitivity to parameter settings.

MLP has not been shown to be frequently among the best approaches as the others in our analysis [136]. However, it is not known whether this method is performing badly because it is not able to achieve good performance, or if it is highly sensitive to parameter settings and thus difficult to tune, or if some guidelines on its parameter choices could improve its performance. Therefore, the main reason to analyse MLP is to provide a better understanding of the behaviour of this method for SEE.

The investigated parameter settings are listed in table 6.1. Their default parameter values are emphasized in bold and correspond to the default values of WEKA, which are considered to perform generally well. For RT, the maximum depth of -1 means unlimited depth. For MLP, the default value a in the number of nodes of each layer is calculated as

$$a = \frac{\#(attribute) + \#(classes)}{2}, \tag{6.1}$$

where $\#(attributes)$ is the number of input attributes, and $\#(classes)$ is the number of outputs that equals to one for SEE.

Table 6.1: Parameter values of the investigated SEE methods. Default parameter values are highlighted in bold and correspond to the default settings of WEKA that perform generally well. The parameter settings of an SEE method consist of traversing all values of one parameter and keeping the others to the default.

Approach	Parameters
RT	M(min.# instance/leaf)={1,**2**,3,6,12,20} V(min.variance for split)= {0.0001,**0.001**,0.01,0.1,10} L(max.tree depth) = { **-1**,2,6,10,15,20}
k-NN	k(# neighbours)={**1**,3,5,7,9,11,13,15,17,19,21}
MLP	L(Learning rate)={0.1, 0.2, **0.3**, 0.4, 0.5} M(Momentum)={0.1,**0.2**,0.3,0.4,0.5} N(# epochs)={100,**500**,1000} H(# nodes of each layer)={***a***,1,3,5,9}
Bagging	# I(iteration for Bagging)={5,**10**,25,50,75} All the possible parameters of the base learners, as shown above.

Note that it is impossible and not necessary to investigate all possible parameter values, which would be infinite. We believe that the parameter values investigated in table 6.1 form a good range for each of the parameters considered in this study. Additional values could be investigated as future work.

6.2.4 Evaluation of Sensitivity to Parameter Settings

This section proposes the analytical methodology that investigates the sensitivity to parameter settings of an SEE method in the online scenario.

Given an SEE method and a data set, we can evaluate the performance of this method in all parameter settings at each time step. Specifically, for non-deterministic methods including MLP, Bagging+RT, and Bagging+MLP, 30 runs are taken to calculate the mean performance at each time step. In this chapter, the performance at each time step is measured in MAE over the estimates on the next ten projects of the data stream as

$$\sum_{t=1}^{T} \frac{|y_t - \widehat{y}_t|}{T} \tag{6.2}$$

where $T = 10$ is the number of testing projects, y_t denotes the actual effort at the next t^{th} time step and $\widehat{y}_t$ is the estimated effort value. MAE is chosen for being a symmetric

measure not biased towards under or overestimation (section 2.4.2). Performance analysis in terms of other point prediction metric can be considered as future work.

We can calculate the average performance of the SEE method across times steps, and determine the best/worst parameter settings in terms of the average performance across time steps. The performance of the default parameter settings can also be obtained. The performance STD across time steps can be calculated for deterministic methods directly. For non-deterministic SEE methods, the average STD across time steps is calculated based on the pooled STDs over all time steps as

$$std_{ave} = \sqrt{\frac{std_1^2 + std_2^2 + ... + std_n^2}{n}}\ , \tag{6.3}$$

where n is the the number of rounds executed (30 runs) and std_i is the STD across time steps at the i^{th} round of execution.

To investigate to what extent an SEE method is sensitive to its parameter setting, the performance of the best and worst parameter settings are compared using Cohen's d (appendix B.3), being calculated based on the *pooled standard deviation* as

$$d_1 = \frac{\overline{MAE_w} - \overline{MAE_b}}{\sqrt{\frac{std_w^2 + std_b^2}{2}}}, \tag{6.4}$$

where $\overline{MAE_w}$ ($\overline{MAE_b}$) is the average MAE across time steps in the worse (best) parameter setting, and std_w (std_b) denotes the corresponding average STD across time steps. The effect size between the best and the default parameter settings is calculated in a similar way as part of the analysis as

$$d_2 = \frac{\overline{MAE_d} - \overline{MAE_b}}{\sqrt{\frac{std_d^2 + std_b^2}{2}}}. \tag{6.5}$$

The effect size between the default and the worst parameter settings is referred to as

$$d_3 = \frac{\overline{MAE_w} - \overline{MAE_d}}{\sqrt{\frac{std_w^2 + std_d^2}{2}}}. \tag{6.6}$$

When it is problematic to use pooled STDs in the analysis due to the infinite value as explained shortly in section 6.3, one of the parameter settings would be chosen as the *control group*, and its corresponding STD, instead of the pooled STD, is used to calculate the effect size. Effect size d is interpreted in terms of Cohen's categories [164] as: small (≈ 0.2), medium (≈ 0.5) and large (≈ 0.8). Cohen's d rather than Vargha and Delaney's A_{12} is chosen since it quantifies the magnitude of performance difference between different parameter settings; whereas, Vargha and Delaney's A_{12} [183] computes the frequency of one parameter setting superior to the other.

It is worthwhile to note that the pooled STD of Eq. (6.3) assumes the independence of individual STDs across time steps, which are probably positively correlated due to the overlap between testing projects, causing over/underestimated pooled STDs. However, the proposed methodology cares for the comparison between d_1, d_2, and d_3, which are computed from pooled STDs, and thus cancels out the mistakes of pooled STDs. In this way, there would be little impact to our judgement on the sensitivity of an SEE method to its parameter settings.

Given an SEE method and a data set, the analytical procedures are listed as follows. (1) If d_1 is small (around 0.2), the performances of the best and the worst parameter settings are considered fairly similar, and we can claim that this method is not sensitive to parameter settings on this data set. If this observation can be generalized to other data sets, this method is considered robust to parameter settings in general. (2) If d_1 is medium (around 0.5) or large (around 0.8), the method is somewhat or highly sensitive to its parameter settings. In this case, d_2 will reveal whether a default parameter setting can provide reasonable performance despite the overall sensitivity to parameter settings. If d_2 is small or medium, it means that though this method is sensitive to the overall parameter choices, its default parameter setting can perform fairly well, and we can simply adopt its default parameter setting. However, if d_2 is large, the performance of

default parameter setting is significantly worse than that of the best parameter setting. Thus, we should pay attention to tuning the parameters of this method. (3) In this case, we step further to calculate d_3. If d_3 is large (more than 0.8), it means that even though the performance of the default parameter setting is significantly worse than the best one, it is still significantly better than the worst one and thus helpful. Therefore, if the project managers do not have enough data or time to do parameter tuning, they could use the default parameter settings that can perform generally well, but there would be large improvement if tuning the parameter settings carefully. On the contrary, if d_3 is small, this method is too sensitive to the model parameters, and a tiny change to its parameter settings could cause a significantly bad effect on its performance. Accordingly, we do not recommend to use this method for SEE even if it could achieve a fairly good performance in its best parameter settings.

6.3 Experimental Result and Analyses

This section aims at answering RQ4 of the thesis by tackling the three sub-questions outlined in section 6.1 using the analytical methodology presented in section 6.2.

6.3.1 Sensitivity of Average Performance Across Time Steps

This section aims at answering RQ4.1: *Given an SEE method, how sensitive is an SEE method to its parameter settings in terms of average performance across time steps?* We will discuss the SEE methods individually in the following subsections.

MLP and Bagging+MLP

Table 6.2 shows the average MAE across time steps of MLP and Bagging+MLP in their best, default, and worst parameter settings, together with the effect size of performance difference between these parameter settings. Tables 6.2(a) and 6.2(b) show that the performance of the worst parameter settings of MLP and Bagging+MLP are so inferior

that most of their STDs across time steps are infinite, making it impossible to compute the effect size using pooled STDs. To cater this issue, the performance in the best parameter setting is taken as the control group, and the effect size of a certain parameter setting is calculated against this control group.

We can see from tables 6.2(a) and 6.2(b) that MLP and Bagging+MLP are extremely sensitive to parameter settings, since the magnitudes of average performance difference between the best and the worst parameter settings are huge. Such supposition can be confirmed by tables 6.2(c) and 6.2(d), where the effect size between the best and the worst performance is extremely large in all investigated data sets. Even worse, our investigation found that the performance of MLP and Bagging+MLP could be sensitive to the *starting points* of the training algorithms for MLP, which may be a possible reason for causing extremely large MAEs and infinite STDs of the worst performance.

Nevertheless, MLP and Bagging+MLP in the best and default parameter settings can achieve fairly good performance, being competitive to other SEE methods in tables 6.3(a), 6.3(b) and 6.4. Though the performance in the default parameter settings are usually rather worse than the best ones, they are acceptable for the practical usage. As their STDs across time steps are all finite (see tables 6.2(a) and 6.2(b)), the default and the best parameter settings may not be sensitive to the starting points.

Further investigation reveals that the best parameter settings of MLP always have the simplest model structures where each layer has only one node, and the best parameter settings of Bagging+MLP always have the simplest base learners. It is reasonable as small training set of SEE may not be adequate to construct complicate network structures.

Overall, MLP and Bagging+MLP can perform fairly well given proper parameter settings. But the performance is very sensitive to the parameter choices and even to the starting point of the learning algorithms. This can potentially explains the contradictory conclusions of SEE literature on MLP or Bagging+MLP [49, 136, 48]. In practice,

Table 6.2: Average performance and effect size across time steps for MLP and Bagging+MLP. STD is the standard deviation across time steps in terms of MAE. Medium/Large effect size is highlighted in yellow/red (light/dark grey).

(a) Performance of MLP.

MAE across time steps		Kitchenham	Maxwell	SingleISBSG
Best PS	MAE	2046.35	5358.02	2754.78
	STD	2868.96	1979.71	1006.01
Default PS	MAE	2474.78	7893.26	3682.47
	STD	2846.06	3629.54	1254.03
Worst PS	MAE	7.42E+138	1.19E+155	1.07E+153
	STD	4.71E+140	Inf	Inf

(b) Performance of Bagging+MLP.

MAE across time steps		Kitchenham	Maxwell	SingleISBSG
Best PS	MAE	1946.18	5089.75	2705.77
	STD	2883.74	1918.31	790.20
Default PS	MAE	2188.81	5932.99	3025.83
	STD	2892.06	2262.39	860.47
Worst PS	MAE	9.26E+151	1.33E+153	4.52E+153
	STD	Inf	Inf	Inf

(c) Effect size of MLP with the best parameter settings as the control.

Effect Size	Kitchenham	Maxwell	SingleISBSG
best vs. worst	2.5863E+135	6.011E+151	1.0636E+150
best vs. default	0.149	1.281	0.922

(d) Effect size of Bagging+MLP with the best parameter setting as the control.

Effect Size	Kitchenham	Maxwell	SingleISBSG
best vs. worst	3.211E+148	6.933E+149	5.720E+150
best vs. default	0.084	0.440	0.405

Table 6.3: Average performance and effect size across time steps for RT and Bagging+RT. STD is the standard deviation across time steps in terms of MAE. Medium/Large effect size is highlighted in yellow/red (light/dark grey).

(a) Performance of RT

MAE across time steps		Kitchenham	Maxwell	SingleISBSG
Best PS	MAE	2249.14	5629.51	2751.86
	STD	2928.71	2426.86	852.77
Default PS	MAE	2618.38	5930.50	3144.56
	STD	2899.82	2611.51	1016.74
Worst PS	MAE	2618.96	6429.93	3621.96
	STD	2935.13	2725.38	1356.32

(b) Performance of Bagging+RT

MAE across time steps		Kitchenham	Maxwell	SingleISBSG
Best PS	MAE	2055.24	5110.56	2814.39
	STD	2908.12	2606.10	941.08
Default PS	MAE	2209.14	5260.25	2915.74
	STD	2926.48	2572.87	986.80
Worst PS	MAE	2634.56	6230.56	3356.18
	STD	2926.55	2327.33	665.64

(c) Effect size of RT using pooled STD.

Effect Size	Kitchenham	Maxwell	SingleISBSG
d_1 (best vs. worst)	0.126	0.310	0.768
d_2 (best vs. default)	0.127	0.119	0.419
d_3 (default vs worst)	-	-	0.398

(d) Effect size of Bagging+RT pooled STD.

Effect Size	Kitchenham	Maxwell	SingleISBSG
d_1 (best vs. worst)	0.199	0.453	0.665
d_2 (best vs. default)	0.053	0.058	0.105
d_3 (default vs worst)	-	-	-

the practitioners are suggested to use the default parameter settings for MLP and Bagging+MLP that can perform well in general cases, if they have little experience of tuning parameters or do not have time to do so.

RT and Bagging+RT

Table 6.3 shows the average MAE across time steps and effect size for RT and Bagging+RT with the best, default, and worst parameter settings. Tables 6.3(c) and 6.3(d) show that the effect size of the best vs worst parameter settings is small (0.126) and small (0.199) in

Kitchenham, small (0.310) and medium (0.453) in Maxwell, and large (0.768) and large (0.665) in SingleISBSG for RT and Bagging+RT respectively. This suggests that RT and Bagging+RT is much less sensitive to parameter settings than MLP and Bagging+MLP.

Though RT are a bit sensitive to parameter settings in SingleISBSG, the effect size with pooled STD between the best and the default, and between the default and the worst parameter settings are medium (0.419) and medium (0.398) respectively, which means that the default parameter settings can achieve relatively good performance. The performance improvement can be expected if the parameter settings are tuned carefully. For Bagging+RT, the effect size between the best and the default parameter settings in SingleISBSG is insignificant (0.105), indicating that the performance difference between the best and the default parameter settings is quite tiny though Bagging+RT is slightly sensitive in SingleISBSG across all parameter settings.

Overall, RT and Bagging+RT are usually not very sensitive to parameter settings for SEE, and thus it is a good option to simply use the default parameter settings if tuning parameters is not affordable. But, we still suggest to tune the parameters in order to achieve better performance. Comparison with other SEE methods such as MLP and Bagging+MLP, the performance of RT and Bagging+RT in the worst parameter settings is not so much worse than the best ones. Therefore, blind parameter tuning will not cause severe problem for RT and Bagging+RT.

K-NN

Table 6.4 lists the average MAE across time steps for k-NN with their best, default and worst parameter settings. As shown in table 6.4(b), k-NN is not very sensitive to parameter choices in SEE.

Furthermore, table 6.4(c) shows that the default parameter setting ($k = 1$) is always the worst, and the best performance is usually achieved when k equals to 3 or 5. A possible reason may arise from the noise of SEE data. On the one hand, the performance of k-NN

Table 6.4: Average performance and effect size across time steps and parameter settings for K-NN. STD is the standard deviation across time steps in terms of MAE. Medium/Large effect size is highlighted in yellow/red (light/dark grey).

(a) Performance of k-NN.

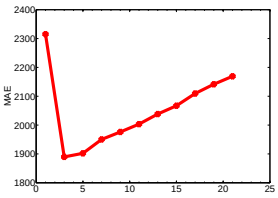
MAE across time steps		Kitchenham	Maxwell	SingleISBSG
Best PS	MAE	1889.67	4642.61	2937.49
	STD	2770.78	2574.22	688.85
Default PS	MAE	2315.12	5667.09	3394.39
	STD	2838.20	2172.91	1562.69
Worst PS	MAE	2315.12	5667.09	3394.39
	STD	2838.20	2172.91	1562.69

(b) Effect size of k-NN using pooled STD.

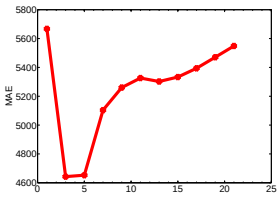
Effect Size	Kitchenham	Maxwell	SingleISBSG
best vs. worst	0.152	0.430	0.378
best vs. default	0.152	0.430	0.378
default vs worst	-	-	-

(c) Parameter Settings of k-NN

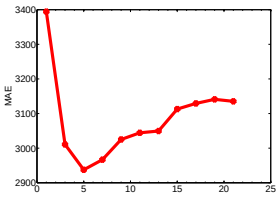
k	Kitchenham	Maxwell	SingleISBSG
Best PS	k=3	k=3	k=5
Defaul PS	k=1		
Worst PS	k=1		



(a) Kitchenham.



(b) Maxwell.



(c) SingleISBSG

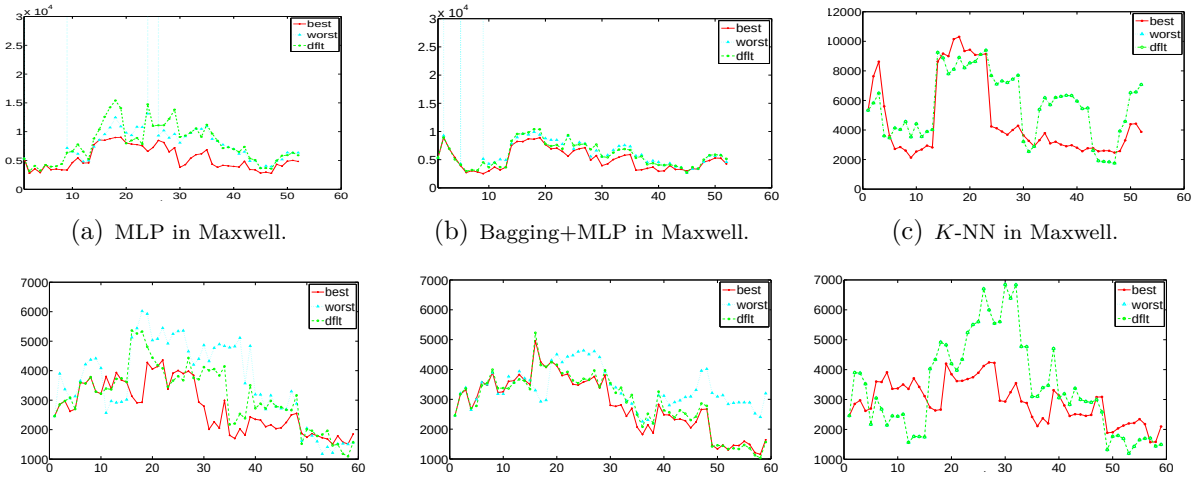
Figure 6.1: Parameter values of k in kNN

can be strongly affected if only using the nearest neighbour due to the corruption of data noise. On the other hand, with the increase of k, more neighbours would lessen the impact of data noise, but more less relevant training examples can involve into predicting, which is not preferred. Considering the small data set for SEE, three or five (or a bit bigger like seven) neighbours may be a good choice to avoid both extremes. Figures 6.1(a)∼6.1(c) further support our conjecture.

Overall, the average performance of k-NN across time steps is not very sensitive to parameter settings, and 1-NN is not recommended due to its inferior overall performance.

6.3.2 Step-Wise Performance of Best Parameter Settings

Previous section analyses the sensitivity of an SEE method to parameter choices based on the average performance across all time steps. This section looks into each time step



(a) MLP in Maxwell. (b) Bagging+MLP in Maxwell. (c) K-NN in Maxwell.

(d) RT in SingleISBSG. (e) Bagging+RT in SingleISBSG. (f) K-NN in SingleISBSG.

Figure 6.2: The performance of SEE methods on each time step with the *best*, *default* and *worst* parameter settings in terms of MAE. We do not list the performance of the method on all data sets, since the trends are similar. Note that the performance of the worst parameter settings of k-NN is overlapped with that of the default.

for answering RQ4.2: *Given an SEE method, do the best parameter settings in terms of the average performance across time steps perform consistently well throughout each time step in comparison to other parameter settings?*

We can see from figure 6.2 that though there are a few time steps where the default or even the worst parameter settings are superior to the best ones, usually the best parameter settings can achieve better performance than the others. Take figure 6.2(d) as an example, the performance in the worst parameter settings outperforms the one in the best and default at the time steps between ten and fifteen, however, the best parameter settings are superior to the default and the worst in the majority of the time steps. The occasional inferiority of the best parameter settings allow for further performance improvement throughout each time step by providing adaptive parameter tuning, which would be a worthwhile future work.

For k-NN, comparing figures 6.2(c) and 6.2(f) with 6.2(a), 6.2(b), 6.2(d), and 6.2(e), we can see that the frequency that the worst parameter settings are superior to the best

ones is higher than other methods. This suggests that the best parameter settings of k-NN are more dependent on the moment in time. In other words, k-NN is less stable in terms of the consistency of the best parameter settings throughout time steps.

For MLP and Bagging+MLP, we can see from figures 6.2(a) and 6.2(b) that the performance with the worst parameter settings are competitive with the best and the default ones at most time steps. Only at a few time steps, the worst parameter settings perform extremely bad: their performance cannot even be shown in the figures for proper visualization of the other parts.

Overall, the performance of the best parameter settings outperform the default and the worst ones at most time steps.

6.3.3 How Could Ensemble Help?

This section aims at answering RQ4.3: *Could Bagging help to lessen the base learners' sensitivity to parameter settings?* We will investigating this issue from two perspectives: (a) Could Bagging help to lessen the base learners' sensitivity to the parameter settings in terms of its average performance across time steps? and (b) Could Bagging also help with the performance at each time step?

Comparing tables 6.2(c) and 6.2(d), we can see that the effect size between the best and the default parameter settings for Bagging+MLP is smaller than those for MLP in all investigated data sets. Specifically, the effect size decrease from 0.149 (Kitchenham), 1.281 (Maxwell), and 0.922 (SingleISBSG) for MLP to 0.084 (Kitchenham), 0.440 (Maxwell), and 0.405 (SingleISBSG) for Bagging+MLP. It means that the performance in the default parameter settings of Bagging+MLP is closer to the best ones than that of MLP. We can also see that the default and the best curves are much closer in figure 6.2(b) than those in figure 6.2(a), meaning that Bagging can help shorten the performance difference in different parameter settings in terms of each time step.

Comparing tables 6.3(c) and 6.3(d), we can see that the effect size between the best and the default parameter settings for Bagging+RT is smaller than that for RT, suggesting that the performance in the default parameter settings for Bagging+RT is closer to the best ones than that for RT. We can also see from figures 6.2(d) and 6.2(e) that Bagging helps drag the curves of the default parameter settings closer to the best one throughout time steps, meaning that Bagging helps shorten the difference in terms of each time step.

Overall, our experimental results show that combining SEE methods into Bagging can help the performance of the default parameters get closer to the best ones.

6.4 Summary and Discussion

This chapter proposes an analysis methodology and conducts systematic experiments to investigate to what extent parameter settings affect the performance of SEE methods and whether different SEE methods are more/less sensitive to their parameter settings.

The model sensitivity problem has been tackled by answering the three sub-research questions outlined in section 6.1 as follows.

• *RQ4.1: How sensitive is an SEE method to its parameter settings in terms of average performance across time steps?* Different SEE methods have different sensitivity to the parameter settings. Specifically, RT and Bagging+RT are not very sensitive to parameter settings in terms of average performance across time steps, but parameter tuning is recommended for gaining better performance. Though MLP and Bagging+MLP can achieve good performance, they are extremely sensitive to their parameter settings, and even to the starting points of their learning algorithms. K-NN is not very sensitive to parameter settings, and 1-NN is not recommended for SEE due to its inferior performance.

• *RQ4.2: Does the best parameter setting of an SEE method in terms of average performance across time steps perform consistently well throughout time steps in comparison to other parameter settings?* The best parameter settings can commonly achieve better

performance than the default and the worst ones, though there are a few time steps where the default or even the worst parameter settings outperform the best ones. In particular, k-NN is less stable in terms of the consistency of the best parameter settings throughout time steps, as it happens more frequently that the best parameter settings perform the worst in some time steps.

- *RQ4.3: Could Bagging help to lessen the base learners' sensitivity to parameter settings?* Incorporating SEE methods into Bagging ensemble can help make the performance of the default parameter settings closer to the best ones. Practically, it would be an acceptable choice to combine MLP and RT into Bagging with their default parameter settings, when there is no time to perform parameter tuning.

In summary, sensitivity to parameter settings should be considered as a criterion for evaluating SEE methods. A good SEE method should not only be the one that is able to achieve superior prediction performance, but also be the one that is either less sensitive to the parameter settings or easy to make good parameter choices.

Future work includes sensitivity investigation of other state-of-the-art SEE methods in more data sets with chronological ordering. Specifically, the adaptation of our SEE methods proposed in chapters 3, 4, and 5 to the online scenario as well their sensitivity to parameter settings in the online scenario is worthwhile for future research.

CHAPTER 7

Conclusions and Future Work

This chapter summarizes the conclusions of the thesis and presents the directions of future work. The thesis focuses on tackling the scarcity and uncertainty issues of SEE and studying the sensitivity to parameter settings of SEE methods. The main contributions are the answers to RQ1 to RQ4 outlined in section 1.1.

7.1 Conclusions

We summarize the conclusions in chapters 3, 4, 5, and 6 as follows.

7.1.1 Synthetic Data Generation for Small Data Problem

SEE usually suffers from data scarcity problem due to the expensive or long process of data collection [106, 107, 116]. SEE literature has frequently tackled this problem by developing sophisticated SEE methods or collecting as many completed projects as possible. An alternative and much cheaper way is to augment the training set with the synthetic software projects.

Chapter 3 proposes a synthetic data generator to tackle the data scarcity problem of SEE, answering the first research question of the thesis:

RQ1. Can we generate synthetic software projects to enlarge the training set size for obtaining better prediction performance? If so, how?

Our method produces synthetic projects by slightly displacing training examples, with each synthetic project associated to one training example that is chosen randomly. The synthetic projects are then added to the training set and used to construct SEE models.

The effectiveness of our synthetic project generator has been validated by answering the sub-research questions outlined in section 3.1 as follows.

• *RQ1.1: Given an SEE method, can our synthetic data generator help improve prediction performance over the baseline that does not use synthetic projects? When? Could it be detrimental?* Experimental results show that our synthetic projects usually have positive effect on and are rarely detrimental to the performance of SEE methods. They are particularly helpful for small and medium data set sizes for MLR 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.

• *RQ1.2: Given an SEE method, if our synthetic projects are helpful for prediction performance, why are they helpful? If they are detrimental, why are they detrimental?* The effectiveness of our synthetic projects is mainly due to the data augmentation and the robustness enhancement in the areas that data noise may injure the quality of SEE model construction. Different SEE methods have different improvement magnitude which is mainly affected by their *locality* and *globality* properties. Our synthetic projects are rarely detrimental to the baseline performance that does not use synthetic projects.

• *RQ1.3: How well does our data generator perform compared to other data generators in SEE literature?* Experimental results show that our synthetic project generator is significantly superior to or has no significant difference from Kamei et al's [93] that is the only competitor in SEE literature. Their data generator [93] probably brings no significant improvement to the baseline performance that does not use synthetic projects.

This thesis also studies the impact of training set size on prediction performance based on the *holdout* evaluation. Experimental results show SEE methods usually achieve better performance with a larger training set; the magnitude of superiority decreases on the increase of the training size. This is a by-product of answering RQ1.

7.1.2 Uncertain Effort Estimation for Noisy Data Problem

Uncertain effort estimation tackles the data noise problem of SEE and supports more informative decision making by accessing prediction risks. PIs with CLs are recommended by software estimation experts as a more reasonable representation of reality [103]. However, most SEE methods only produce point prediction. Chapters 4 and 5 introduce/propose uncertain effort estimators, answering RQ2 and RQ3 of the thesis.

Chapter 4 introduces RVM to provide PIs with CLs, answering the second research question of the thesis:

RQ2. Is there any ML approach that can provide uncertain effort estimation? How well can it perform in terms of point and uncertain prediction?

The potential of RVM in the context of SEE has been validated by answering the sub-research questions outlined in section 4.1 as follows.

• *RQ2.1: How well does RVM perform compared to other SEE methods when used as a point estimator?* Experimental results show that RVM is very competitive compared to other SEE methods, being usually ranked top two out of seven methods across 11 data sets in terms of MAE. Friedman tests detect its significant superiority to MLP and show similar performance with other SEE methods. Thus, RVM is a promising SEE method and worthwhile for further investigation.

• *RQ2.2: How to provide PIs with CLs based on RVM? How well can the PIs with CLs derived from RVM perform?* We provide the way of deriving PIs with CLs in section 4.2 and validate the PIs with two specific cases $CL_{0.6827}$ and $CL_{0.9545}$. Experimental results

show that the PIs can often achieve relatively good hit rate, but they can be too wide to be informative. Therefore, further studies should focus on providing better PIs with CLs.

Chapter 5 ensembles a set of RVMs into a unified uncertain effort estimator, answering the third research question of the thesis:

> *RQ3. Can we improve the PIs of RVM? How? How well does this method perform compared to state-of-the-art point/uncertain methods?*

The proposed SynB-RVM adopts Bootstrap resampling to produce multiple RVMs based on adapted Bootstrap training bags whose replicated training examples are replaced with their synthetic counterparts. The constructed RVMs are then incorporated into a unified uncertain estimator, based on which PIs with CLs are derived.

The performance of SynB-RVM has been validated by answering the three sub-research questions outlined in section 5.1 as follows.

• *RQ3.1: When used as a point estimator, how well can SynB-RVM perform compared with other SEE methods?* Experimental results show that SynB-RVM can either significantly outperform or have similar point prediction performance compared to other SEE methods. Specifically, SynB-RVM significantly outperforms RVM (EmpRVM), BtstrpRVM (Bagging+RVM), k-NN, SVR, MLP, RT, Bagging+RT, and Bagging+SVR with respect to at least one performance metric; it performs similarly to ATLM (EmpATLM) and BtstrpATLM (Bagging+ATLM).

• *RQ3.2. When used as an uncertain estimator, can SynB-RVM's PIs achieve adequate hit rate with narrower and more informative PIs?* In terms of hit rate, SynB-RVM usually has significantly better performance than other SEE methods except for RVM. In terms of relative width, SynB-RVM usually produces significantly better PIs with CLs than those from RVM and BtstrpRVM; it performs similarly to EmpRVM, BtstrATLM, and EmpATLM. Nevertheless, SynB-RVM has performance superiority with medium/large effect size over EmpRVM, BtstrATLM, and EmpATLM in most data sets.

• *RQ3.3. Which components of SynB-RVM contribute to the point and uncertain prediction performance improvement?* Analytical results show that the three versions of SynB-RVM perform similarly in terms of point and uncertain prediction. SynB-RVM has two key components: synthetic displacement and Bootstrap pruning. Analytical results show the effectiveness of synthetic displacement and the two components as a whole in improving the performance of RVM in terms of both point and uncertain prediction. They also suggest that synthetic displacement has a more significant impact than Bootstrap pruning on point and uncertain prediction.

Besides the main contributions in proposing and validating a novel uncertain estimator, we are the first to provide a thorough experimental comparison on uncertain SEE methods. Due to the encouraging results, SynB-RVM is likely to help project managers to make better informed decisions by accessing the project management risks.

7.1.3 Statistical Analysis for Model Sensitivity Problem

Chapter 6 investigates the sensitivity to parameter settings of SEE methods in the online scenario, answering the fourth research question of the thesis:

RQ4. To what extent do parameter settings affect the performance of SEE methods, and should we pay attention to their parameter tuning?

We propose an analysis methodology based on Cohen's d effect size [164] to investigate to what extent parameter settings affect the performance of SEE methods.

The model sensitivity problem has been tackled by answering the three sub-research questions outlined in section 6.1 as follows.

• *RQ4.1: How sensitive is an SEE method to its parameter settings in terms of average performance across time steps?* Different SEE methods have different sensitivity to the parameter settings. Specifically, RT and Bagging+RT are not very sensitive to parameter settings in terms of average performance across time steps, but parameter tuning is recommended for gaining better performance. Though MLP and Bagging+MLP can achieve

good performance, they are extremely sensitive to their parameter settings, and even to the starting points of their learning algorithms. K-NN is not very sensitive to parameter settings, and 1-NN is not recommended for SEE due to its inferior performance.

• *RQ4.2: Does the best parameter setting of an SEE method in terms of average performance across time steps perform consistently well throughout time steps in comparison to other parameter settings?* The best parameter settings can commonly achieve better performance than the default and the worst ones, though there are a few time steps where the default or even the worst parameter settings outperform the best ones. In particular, k-NN is less stable in terms of the consistency of the best parameter settings throughout time steps, as it happens more frequently that the best parameter settings perform the worst in some time steps.

• *RQ4.3: Could Bagging help to lessen the base learners' sensitivity to parameter settings?* Incorporating SEE methods into Bagging ensemble can help make the performance of the default parameter settings closer to the best ones. Practically, it would be an acceptable choice to combine MLP and RT into Bagging with their default parameter settings, when there is no time to perform parameter tuning.

7.2 Future Work

This section presents potential research directions that may further develop our proposed SEE methods in the thesis.

7.2.1 Data Generator with Guided Choice of Training Examples

In chapter 3, we have randomly chosen the training examples based on which our synthetic projects are generated. One reason for the random choice is to allow the data to speak for itself. For instance, the training examples in crowded regions that are less likely to contain large variations are more likely to be chosen, which is preferable.

Other strategy that emphasizes customers' preferences or encodes expert knowledge into the choice of the training examples can potentially further improve the prediction performance, leading to the future research question:

> *RQ1*: Is there other strategy for choosing training examples based on which synthetic projects are generated? Can this strategy further improve the effectiveness of our synthetic projects?*

This study will give us a better understanding on the effect of the choice of training examples. It may also further enhance the effectiveness of our synthetic projects.

7.2.2 Synthetic Ordinal/Categorical Feature Modelling

In chapter 3, the generation of synthetic ordinal (categorical) features are modelled by binomial (discrete uniform) distributions. This modelling makes sense due to the effectiveness of our synthetic projects. However, as discussed in section 3.2.3, our synthetic feature modelling may not fit reality perfectly. For instance, a *newly developed* software project may be more likely to be *enhanced* rather than *re-developed*; employees with *normal* expertise would be more likely to evolve with *high* rather than *low* expertise. Therefore, non-symmetric distributions would be better to model the categorical (ordinal) features, leading to the future research question:

> *RQ2*: Can we design a non-symmetric discrete distribution for being more suitable for categorical (ordinal) feature modelling? Can this further improve the effectiveness of our synthetic projects?*

This study will potentially tackle the data scarcity problem better. To this end, expert knowledge on the distribution of categorical and ordinal features is required.

7.2.3 Synthetic Effort Modelling

In section 3.2.2, synthetic effort value is determined by numerical features only. Assigning synthetic effort based on ordinal/categorical features is challenging, as it requires expert

knowledge or data analysis with a large amount of training examples. This is potentially a harder problem than SEE itself. Moreover, changing some ordinal/categorical features would increase the effort, whereas changing some others would decrease it. Altogether, this would cause small variation in the effort.

Considering ordinal/categorical features for synthetic effort assignment may further improve the effectiveness of our synthetic projects, leading to the future research question:

RQ3*: How can we encode ordinal and categorical features into synthetic effort generation? Which ordinal or categorical features are more predictive? Can this strategy enhance the effectiveness of our synthetic projects?

This study will potentially improve the prediction performance. Expert knowledge on the predictive ability of categorical/ordinal features is required.

7.2.4 Effort Noise Modelling and Non-Symmetric PIs

In chapters 4 and 5, effort noise is modelled by a Gaussian distribution, which is reasonable according to the central limit theorem [145]. However, the Gaussian noise assumption disregards the fact that effort has to be positive (see the last subsection of section 5.7.2). More studies can focus on non-symmetric right-skewed distributive modellings for effort noise and the corresponding non-symmetric PIs, leading to the future research question:

RQ4*: Can we provide a suitable non-symmetric modelling for effort noise? How can the PIs with CLs be derived? What is the performance of this noise modelling compared with RVM and SynB-RVM in terms of point and uncertain prediction performance?

This study will potentially enhance uncertain prediction performance of SynB-RVM by providing non-symmetric PIs with CLs. Possible examples for effort noise modelling include right-skewed Gaussian distribution [14] and Gamma distribution [80]. However, non-Gaussian noise modelling will destroy the analytical solution of RVM and SynB-RVM, causing considerably complicated deductions.

7.2.5 SynB-RVM Variants

In section 5.7.2, we discuss that SynB-RVM can be encoded with any base leaner that can produce probabilistic effort estimation. Specifically, the base learner can be a single method like RVM that provides probabilistic prediction itself, an ensemble of methods like BtstrpRVM that provides probabilistic prediction as a group, or even a cascade of ensembles. In this sense, our approach can be generalized as *Synthetic Bootstrap ensemble of Probabilistic Predictors*, leading to the future research question:

RQ5*: How can we encode other probabilistic SEE methods into our framework? Can they outperform SynB-RVM in terms of uncertain prediction?

This study will potentially incorporate the profits of other uncertain methods and improve uncertain prediction performance. But we need to consider the extra computational load.

7.2.6 Adapting Our Proposed Methods to Online Scenario

Our methods proposed in chapters 3, 4 and 5 are in the offline scenario that is the most typical setting in the SEE community. The offline scenario is suitable for the companies that have accumulated adequate training examples, but it is difficult especially for the start-ups that have few or even no completed software projects. Therefore, adapting our proposed methods to the online scenario will be a worthwhile future work as

RQ6*: How to adapt our methods in chapters 3, 4 and 5 to the online scenario?

This study will potentially enhance the practical usage of our methods.

7.2.7 Model Sensitivity of Our Methods in Online Scenario

In chapters 3, 4 and 5, we propose/introduce three methods to handle small and noisy data problem of SEE in the offline scenario. We have investigated reasonable amount of model parameters of our methods and their competitors, and performed fair comparisons among these SEE methods based on their best parameter settings.

Nevertheless, it would be worthwhile to further investigate the model sensitivity of our proposed methods in the online scenario using the analysis methodology proposed in chapter 6, leading to the future research question:

RQ7*: To what extent the parameter settings of our methods in chapters 3, 4, and 5 affect the prediction performance in the online scenario?

This study will provide a more thorough understanding of our methods in terms of the sensitivity to parameter settings, prompting their practical usage.

APPENDIX A

Point Effort Estimation Methods

This appendix discusses point effort estimation methods, which are related to the thesis as background knowledge and are used in the experimental comparisons against our proposed methods in this thesis.

Consider a training set of N software projects

$$\mathscr{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N, \tag{A.1}$$

where $\mathbf{x}^n = [x_1, \cdots, x_D] \in \mathbb{R}^D$ is the n^{th} training data consisting of D features, and $y^n \in \mathbb{R}^1$ is the actual effort for developing this project. As mentioned in section 1.2.1, categorical features are converted into real values, and thus can be represented as in $\mathbb{R}$. Align all training examples in line and form the matrix of training examples as

$$\mathbf{X} = [\mathbf{x}^1, \cdots, \mathbf{x}^N]^T \in \mathbb{R}^{N \times D}. \tag{A.2}$$

The matrix of training examples contains all training information that can be used to build an SEE method.

A.1 Linear Regression

Linear regression models are popularly used in the context of SEE, and is usually shown to perform well after appropriate data transformation [188, 99]. As formulated in section 1.2.1, SEE is a regression problem usually with more than one input feature, we therefore consider multivariate linear regression in the thesis, each variate corresponding to one effort feature.

A.1.1 Multivariate Linear Regression (MLR)

Multivariate Linear Regression (MLR) assumes a linear relationship between the input features and output effort as

$$y = \theta_0 + \theta_1 x_1 + \cdots \theta_d x_d + \cdots + \theta_D x_D \tag{A.3}$$

where x_d is the d^{th} feature of SEE data, $d \in \{1, \cdots, D\}$, and $\boldsymbol{\theta} = [\theta_0, \cdots, \theta_D]^T \in \mathbb{R}^{D+1}$ are the model parameters.

Conventionally, we minimize the loss function $\mathscr{L}$ to determine the optimal model parameters $\boldsymbol{\theta}^*$ by least square estimation [144] as

$$\begin{aligned} \mathscr{L} &= \sum_{n=1}^{N}(y_n - (\theta_0 + \theta_1 x_1^{(n)} + \cdots + \theta_D x_D^{(N)}))^2 \\ &= \sum_{n=1}^{N}(y_n - \boldsymbol{\theta}^T \cdot \overline{\mathbf{x}^{(n)}})^2 = (\mathbf{y} - \overline{\mathbf{X}}\boldsymbol{\theta})^T(\mathbf{y} - \overline{\mathbf{X}}\boldsymbol{\theta}), \end{aligned} \tag{A.4}$$

where $\overline{\mathbf{x}^{(n)}} = [1, x_1^{(n)}, \cdots, x_D^{(n)}]^T \in \mathbb{R}^{(D+1)}$, $\mathbf{y} = [y_1, \cdots, y_n]^T \in \mathbb{R}^N$, and $\overline{\mathbf{X}} = [\overline{\mathbf{x}^{(1)}}, \cdots, \overline{\mathbf{x}^{(n)}}]^T \in \mathbb{R}^{n\times(D+1)}$. To get the optimal $\boldsymbol{\theta}^* = [\theta_0^* \cdots, \theta_D^*]$, we need to compute the differentiation with the equations as

$$\frac{\partial \mathscr{L}}{\theta_0} = \frac{\partial \mathscr{L}}{\theta_1} = \cdots = \frac{\partial \mathscr{L}}{\theta_D} = 0, \tag{A.5}$$

where we have $D+1$ equations, corresponding to the $D+1$ unknown model parameters. In matrix-vector form, the set of equations can be represented as

$$0 = \frac{\partial \mathscr{L}}{\partial \boldsymbol{\theta}} = 2\overline{\mathbf{X}}^T(\overline{\mathbf{X}}\boldsymbol{\theta} - \mathbf{y}). \tag{A.6}$$

Deriving the matrix equation, we have the analytical solution of the MLR model as

$$\boldsymbol{\theta}^* = (\overline{\mathbf{X}}^T\overline{\mathbf{X}})^{-1}\overline{\mathbf{X}}^T\mathbf{y}. \tag{A.7}$$

Given a testing example $\mathbf{x}$ with an unknown effort value y, the prediction process proceeds as

$$\hat{y} = (\boldsymbol{\theta}^*)^T \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \tag{A.8}$$

where we hope for small error of $|y - \hat{y}|$.

MLR has been used with mixed evaluation results in SEE: some studies found that MLR was the most accurate model in their experimental studies [49, 35, 124]; some found that methods such as artificial neural networks or analogy-based approaches outperformed MLR [48, 43]; some others argued that MLR with appropriate data transformations such as logarithm of numerical features can produce suitable and more accurate prediction performance [99]. In the thesis, MLR is taken as a proper baseline for evaluating the point prediction performance of SEE methods.

A.1.2 Automatically Transformed Linear Model (ATLM)

Naively applying MLR may not be adequate since SEE data is often skewed, and appropriate transformations of the inputs and/or output are often required to form a proper MLR [99]. To this end, Whigham et al. [188] amended classical MLR by certain data preprocessing, namely Automatically Transformed Linear Model (ATLM).

ATLM is a linear model which assumes a linear relationship between the input features and output efforts after appropriate automatic transformations upon them. First, ATLM

assesses the suitability of logarithm and square-root transformations of each effort variable (inputs and output) based on the underlying distribution discovered from the training data. For each effort variable, the transformation (logarithm, square-root and none) that results in the least skewed data is applied to construct the final linear model. Skewness is measured by the b_1 metric proposed in [79]. ATLM can automatically decide when to apply what transformation and thus may be more adequate for SEE than MLR. Then, the training of ATLM would be proceeded as a normal MLR with the transformed data samples for training and prediction. In particular, MLR is a special case of ATLM when no transformations are applied.

ATLM has been shown to be a suitable baseline for comparison of SEE methods [188]. It has been shown to be comparable or superior to other SEE methods such as Pareto ensembles of artificial neural networks [135] or the hybrid ABE-PSO [95]. In the thesis, we use the *R codes* provided by the authors for the implementation of ATLM, and use *R.matlab* package [22] to configure the R implementation into the MATLAB framework.

A.1.3 Potential Issue of the Linear Models

As discussed in chapters 3 and 5, one potential issue of MLR/ATLM is that it may suffer numerical problems while giving effort estimates for some testing examples. For instance, it may produce an extremely large or even infinite effort estimate for a testing project, which is obviously impractical.

This erratic effort estimation may arise from outer-interpolating training points to predict a testing project that is isolated and very distant from any of the training projects, thus causing a very erratic prediction (e.g. very large estimated effort) and large error. The situation could be even worse if, e.g., the erratically large prediction takes place in the logarithmic effort space when using ATLM, which would be inverse-transformed back to the original effort space causing an even larger or infinite effort prediction.

The unstable prediction performance of MLR/ATLM may also arise from the scarce data problem of SEE, where MLR/ATLM may suffer from *ill-conditioned problem* when doing matrix inversion in the training process, meaning that a small error in the data can cause much larger errors in the solution. Even worse, ATLM may suffer incorrect statistic estimation in its automatic transformation mechanism for insufficient training examples.

To circumvent this numerical issue, we set up a threshold for the predicted efforts of MLR/ATLM at the value of 10^6 in the thesis. Those prediction that surpass this threshold will not take part in their performance evaluation. The assigned threshold is reasonable because the actual effort values of the investigated data sets are much smaller than it. This treatment is actually giving advantage to linear models in the performance comparison. Certain performance metrics such as mean logarithm absolute error can also alleviate or circumvent this numerical problem.

A.2 Relevance Vector Machine (RVM)

In the thesis, we briefly introduce this learning machine in the context of SEE, and deliberately omit details and derivations for easy understanding. More details can be found in chapter 7.2 of [25] and Tipping's work [177].

Relevance Vector Machine (RVM) [25, 177, 59] is a generalised linear model in terms of model parameters (not in terms of input features), and can be represented for output y given a project vector $\mathbf{x} \in \mathbb{R}^D$ as

$$y = \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) + \varepsilon, \tag{A.9}$$

where $\boldsymbol{\theta} = [\theta_0, \theta_1, \cdots, \theta_N]^T \in \mathbb{R}^{N+1}$ are model parameters, N is the number of training examples, ε is the uncertain information originated from actual effort collection, and $\boldsymbol{\phi}(\cdot) = [\phi_0(\cdot), \cdots, \phi_N(\cdot)]$ are known as *basis functions* that give linear model substantial flexibility for modelling nonlinear relation between $\mathbf{x}$ and y. In the case of RVM, each *basis function* associates with a separate training project $\mathbf{x}_n$ and measures the distance of this training example to the testing project $\mathbf{x}$. There are several choices for the basis function. In the thesis, we adopt the non-normalized Gaussian basis function:

$$\phi_n(\mathbf{x}) = \begin{cases} e^{-\frac{(\mathbf{x}-\mathbf{x}_n)^2}{2c^2}}, & n > 0 \\ 1, & n = 0 \end{cases} \tag{A.10}$$

where $\mathbf{x}_n$ is the n^{th} training example, and c is the tuning parameter that controls their spatial scale and can be determined using cross-validation method [25]. This basis function is chosen for its *locality* character for SEE data, which can be beneficial to SEE [136].

It is noteworthy that RVM's training procedure involves the calculation of the inverse of the *kernel matrix* $\boldsymbol{\Phi} = [\mathbf{1}, \boldsymbol{\phi}(\mathbf{x}_1), \cdots, \boldsymbol{\phi}(\mathbf{x}_N)]^T \in \mathbb{R}^{(N+1)\times N}$, and thus requires a training data set composed of different training examples. This is to avoid the *invertibility problem* of the data matrix when identical columns/rows exist [59, 177].

Following the Bayesian framework, RVM first introduces a zero-mean Gaussian *prior* over model parameters as $p(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \prod_{n=0}^{N} \mathcal{N}(\theta_n|0, \alpha_n^{-1})$, where $\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \cdots, \alpha_N]^T$ is a vector of $N+1$ *hyperparameters*. The prior is a belief on the SEE model before any observation and evidence is taken into account. The detailed shape of prior is governed by the hyperparameters $\boldsymbol{\alpha}$, with their most probable values iteratively estimated from the training examples $\mathscr{D}$. After the iterative procedure finishes, the training examples corresponding to non-zero model parameters are called *relevance vectors* in line with the *support vectors* of Support Vector Machine (SVM), which is a popular learning machine mainly designed for classification problem [181, 180, 182]. RVM can be considered as a Bayesian approach to SVM in the context of regression, which can provide a probabilistic prediction instead of a point estimate for a testing project.

Then, we can obtain the *posterior* of model parameters $p(\boldsymbol{\theta}|\mathscr{D})$, being a conditional probability assigned after the training examples $\mathscr{D}$ are taken into account. The posterior of the model parameters is a Gaussian distribution proportional to the product of the Gaussian *prior* $p(\boldsymbol{\theta})$ and the Gaussian *likelihood* of all training examples $p(\mathscr{D}|\boldsymbol{\theta})$, which is calculated according to Bayes' Rule:

$$p(\boldsymbol{\theta}|\mathscr{D}) = \frac{p(\boldsymbol{\theta})p(\mathscr{D}|\boldsymbol{\theta})}{p(\mathscr{D})}. \tag{A.11}$$

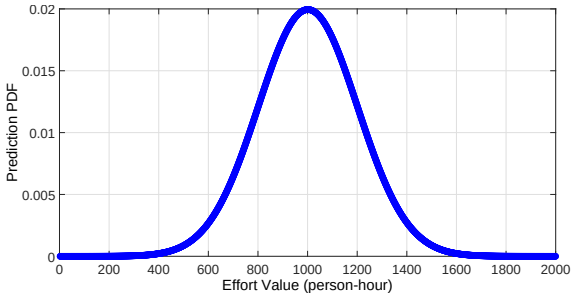


Figure A.1: Probabilistic effort estimation from RVM for a software project. The predicted effort values are Gaussian distributed with the most likely value at 1,000 person hour.

Finally, we can have a *probabilistic* (Gaussian) effort estimation for a testing project as shown in Fig. A.1. A point estimate can be easily obtained by being assigned to the mean of the Gaussian distribution (e.g. 1,000 person-hour in Fig. A.1) for being the most likely effort value.

RVM explicitly encodes the effort noise ε into its modelling, which is assumed to be Gaussian distributed as

$$\varepsilon \sim \mathcal{N}(0, \sigma^2), \tag{A.12}$$

where σ^2 can be determined automatically during the training process. As a result, the probabilistic effort estimation arises from the Gaussian assumption of the effort noise. It is noteworthy that Gaussian effort estimation relates but does not equal to Gaussian effort noise. In Bayesian terms, the assumption of Gaussian effort noise provides a *prior* knowledge for a project effort before any training sample is observed, which is the case that project effort follows σ^2-Gaussian distribution with its mean value defined by Eq. (A.9); the Gaussian prediction of a new project is a *posterior* estimate after training examples are observed and used to train the model.

Ideally, the distribution of the effort noise should be proposed by carefully investigating the residues between the real cost effort values and the collected effort values of the training examples. But this is usually impossible since the real required effort values exempted from noise are not known in reality. Among a set of probability distributions, Gaussian distribution can often match the actual noise distributions in real-world processes reasonably well, and is easy to deal with due to the well-developed mathematical theory behind it [119]. In the context of SEE, the uncertainty is assumed to originate from the Gaussian noise assumption on the effort values, which lays its foundation on *central limit theorem* [145], stating that the summation of several independent random processes tends to a normal distribution even if the original variables themselves are not normally distributed. Considering the errors/noises that generate model uncertainty as random variables, their overall effect is reasonable to be simulated by Gaussian distribution. Nevertheless, this assumption still has problem for disregarding the fact that effort values have to be positive. Better performance may be obtained with a more proper effort noise assumption, which would, however, result in much more complicated deduction when training the model of Eq. (A.9).

A key characteristic of RVM is that people introduce a separate *hyperparameter* for each model parameter θ_i instead of a single shared *hyperparamter* as in classical Bayesian linear regression model [25]. This mechanism results in *sparsity*, for which a large subset of model parameters $\boldsymbol{\theta}$ will be driven to zero with their corresponding training examples pruned [59]. This formulation of prior is a type of *Automatic Relevance Determination* (ARD) prior [177, 143]. With the mechanism of ARD, RVM can automatically choose 'relevance' projects, namely *relevance vectors*, from training examples, which can capture the major structure of the training space [177].

A.3 Support Vector Regression (SVR)

Support Vector Regression (SVR), a counterpart of Support Vector Machine (SVM) classifier [181] for regression problem, is a *kernel* approach that implements the *structural risk minimization* principle for good generalization performance [55]. It is based on a linear model with respect to model parameters $\mathbf{w} \in \mathbb{R}^{D'}$ as

$$f(\mathbf{x}) = \langle \boldsymbol{w}, \boldsymbol{\phi}(\mathbf{x}) \rangle + b, \tag{A.13}$$

where $\boldsymbol{\phi}$ is a nonlinear function that maps from the original feature space $\mathbb{R}^D$ to a higher dimensional feature space $\mathbb{R}^{D'}$, and $\langle \cdot, \cdot \rangle$ denotes the inner product.

The model parameters are determined by minimizing the ε-sensitive loss function as

$$\begin{array}{ll} \text{minimize} & \mathcal{L}_{\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*} = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{n=1}^{N} (\xi_n + \xi_n^*) \\ \text{subject to} & \begin{cases} (\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}^n) \rangle + b) - y^n \leqslant \varepsilon + \xi_n \\ y^{(n)} - (\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}^n) \rangle + b) \leqslant \varepsilon + \xi_n^* \\ \xi_n, \xi_n^* \geqslant 0. \end{cases} \end{array} \tag{A.14}$$

where $\varepsilon > 0$ measures the tolerance to the deviation between the estimated and the actual effort values, *slack variable* ξ_n measures the deviation of the estimated effort value of the n^{th} training example exceeding the actual effort value by more than ε, ξ_n^* measures the deviation dropping below the actual effort value by more than ε, and regularization term $C > 0$ measures the trade-off between the flatness of f and the total amount of intolerant deviations. As illustrated in figure A.2, the ε-sensitive loss function defines an ε-tunnel around the predicted effort values, where the errors inside the tunnel are set to be zero but the errors outside the tunnel are measured by variables ξ and ξ^*. Parameter ε and C need to be determined by cross validation method.

The learning process of SVR involves the use of *Lagrangian multipliers*, which only rely on dot products between $\{\boldsymbol{\phi}(\mathbf{x}^n)\}$. Thus, *kernel theory* can be used to avoid computing the transformation $\boldsymbol{\phi}(\mathbf{x})$ explicitly:

$$k(\mathbf{x}^m, \mathbf{x}^n) = \langle \boldsymbol{\phi}(\mathbf{x}^m), \boldsymbol{\phi}(\mathbf{x}^n) \rangle, \tag{A.15}$$

where the input data are mapped via a kernel function $k(\cdot, \cdot)$. The details of the learning algorithm can be found in [167], and its implementation is based on Sequential Minimal Optimization (SMO) algorithm [62].

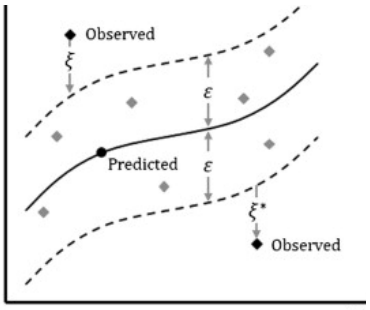


Figure A.2: SVR's model parameters [55]. The ε-sensitive loss function in Eq. (A.14) defines an ε-tunnel around the predicted effort values, where the errors inside the tunnel are zero and the errors outside the tunnel are measured by variables ξ and ξ^*.

There are several choices of kernels such as linear, Gaussian, polynomial, and Sigmoid kernels. Selecting a particular kernel and kernel parameters is usually very important and should be based on domain knowledge. In the thesis, we use linear kernel for being shown to be a better choice for SEE [146]:

$$k(\mathbf{x}^i, \mathbf{x}^j) = \langle \mathbf{x}^i, \mathbf{x}^j \rangle. \tag{A.16}$$

SVR is designed for small data problems [55], which seems suitable to SEE. However, SVR has not been popularly used in the community of SEE partially because of the contradictory conclusions drawn in previous studies [160, 146, 7]. Some studies claimed its superior performance in SEE [146, 160, 41]. For instance, experimental results in [146] showed that SVR with linear kernel significantly outperformed liner regression and artificial neural networks based on a data set of software projects from NASA. On the contrary, some studies claimed similar or inferior performance of SVR compared with artificial neural networks or MLR [7].

A.4 Analogy-Based Estimation (ABE)

Analogy-Based Estimation (ABE), also called Case-Based Reasoning (CBR), is a popular SEE approach for its simplicity and intuitive interpretation that mimics human instinctive decision making [123, 165, 122, 9, 114, 87]. It is based on the assumption that software projects that are similar in the feature space would have similar effort values [114, 94].

The typical procedures of ABE methods comprise a search for the similar projects that have been completed: (1) compute the distances between the testing example and the training examples (i.e. the completed software projects) based on the input features and some distance metric; (2) sort the distances and extract the k nearest neighbours of the testing example; (3) the mean/median effort of the chosen completed projects is returned as the effort estimate. For this reason, ABE is usually referred as k-Nearest Neighbours (k-NN) in the context of ML.

Distance measures the closeness or the difference between two software projects in the feature space, and Euclidean distance is the most commonly used for being suitable to

continuous features such as software size and duration of a project [165, 122, 35]. The value of optimal k in k-NN has been debated for decades in the SEE community [42]. Commonly, it is determined by an expert of the local organization or pre-defined rules such as cross-validation [69].

ABE was first introduced to the SEE community by Shepperd and Schofield [165]. The effort of the nearest neighbour was adopted as the estimated effort of the testing example based on Euclidean distance and standardized features. Their experimental results demonstrated the potential of ABE for SEE. Thereafter, there have been many studies investigating variable designs of ABE approaches and their performance in the context of SEE. However, the reported results have been inconsistent in terms of the prediction accuracy of ABE approaches compared to other SEE methods [75]. On the one hand, some studies claimed that ABE methods were inferior to other SEE methods [141, 35, 78]. In particular, Myrtveit et al. [141] suggested that their performance was sensitive to the experimental design; Briand et al. [35] found that ABE methods were less robust than others especially when dealing with heterogeneous data sets. There have been also some studies finding no clear winner between ABE methods and other conventional estimation methods [125]. On the other hand, more studies tend to demonstrate superior performance of ABE approaches than other SEE methods, and thus claimed that ABE could be a viable alternative to conventional estimation methods in terms of prediction accuracy and flexibility [165, 15, 123, 44, 109, 111].

There are many ABE variants for SEE mainly differing in their distance metrics, data preprocessing mechanisms, the ways of integrating the final estimated effort values, and the incorporation with other techniques such as feature selection [114, 75, 109, 111, 113]. For instance, Kocaguneli et al. [111] proposed a method to proceed feature selection and outlier pruning as a data preprocessing, to which conventional ABE was applied for SEE. Their experiments showed supportive results compared to regression trees.

In this thesis, I adopt the basic ABE approach that does not combine with other sophisticated techniques such as feature selection, for its simplicity, popularity and being fair to compare against other SEE methods. Specifically, the modified Euclidean distance is adopted which is commonly formulated in the SEE community:

$$d(\mathbf{x}^{n_1}, \mathbf{x}^{n_2}) = \sqrt{\sum_{d=1}^{D} ||x_d^{n_1} - x_d^{n_2}||_*^2}, \tag{A.17}$$

where $\mathbf{x}^{n_1}$ and $\mathbf{x}^{n_2}$ denote two software projects and

$$||x_d^{n_1} - x_d^{n_2}||_*^2 = \begin{cases} (x_d^{n_1} - x_d^{n_2})^2, & x_d \text{ is a numerical feature} \\ 1, & x_d \text{ is a categorical feature and } x_d^{n_1} \neq x_d^{n_2} \\ 0, & x_d \text{ is a categorical feature and } x_d^{n_1} = x_d^{n_2}. \end{cases} \tag{A.18}$$

Each feature is preprocessed to have zero mean and unit variance as

$$\frac{x_d - \mu_d}{\sigma_d} \tag{A.19}$$

where μ_d (σ_d) denotes the mean (STD) of the d^{th} feature. The normalization is to ensure that no individual feature has greater influence than others in deciding the effort of a predicting project. Following [111], the median of the effort values is returned as the estimate of the predicting project.

A.5 Regression Tree (RT)

Regression Tree (RT) has easy-to-understand structures that provide *if-then* rules to separate software projects with respect to their features and make effort estimation. Its rules can be easily readable by practitioners. There are several types of RT [31, 69]. Usually, binary regression trees are adopted for SEE [136], where each branching node is split into two children based on the values of a feature as illustrated in figure A.3. In the thesis, RT is implemented by function *fitrtree* in MATLAB or *REPTree* in WEKA [67].

As shown in figure A.3, each leaf node represents a subset of the training examples used to create the tree. The impact of the feature values is also considered in constructing the tree. For instance, if functional size is considered as the most important feature to determine the effort value, it would be used for the highest level split of the tree. Less important features would be used in lower level splits or even not used at all. Learning process consists of determining which features to split and based on what values of this feature. The splitting process is the same as CART [31] when implemented in MATLAB or C4.5 [151] when implemented in WEKA. The hierarchy of features can be particularly useful for SEE, as data sets frequently have many (more and less relevant) features yet insufficient training examples (scarcity problem) [136, 109, 111].

In the prediction process, the leaf node that is the most relevant to the testing example in terms of the tree rules is determined, and the training examples in this branch are used to make the effort estimation. Therefore, RT has the *locality* property, whose effort estimate is based on the projects that are most similar to the testing example with respect to the input features [136]. As SEE data sets tend to be relatively small and very heterogeneous, such approaches are likely to be more adequate and can help dealing with the heterogeneity within the data set [114, 136].

RT is among the most frequently used SEE approaches, has potential advantage, and has been shown to achieve good performance [34, 35, 78, 136, 110, 135, 34, 187, 139]. For instance, Minku et al. [136] showed that RT was more frequently among the best performed methods, and was a good choice of base learners for the ensemble approach. They also showed that Bagging with RT performed well, being frequently among the best approaches and rarely performing considerably worse than the best ones.

A.6 Artificial Neural Network (ANN)

Artificial Neural Network (ANN) is inspired by the architecture of biological neural networks, comprising simple interconnected neurons. The neurons compute a weighted sum of their inputs and generate an output by passing through an activation function. A

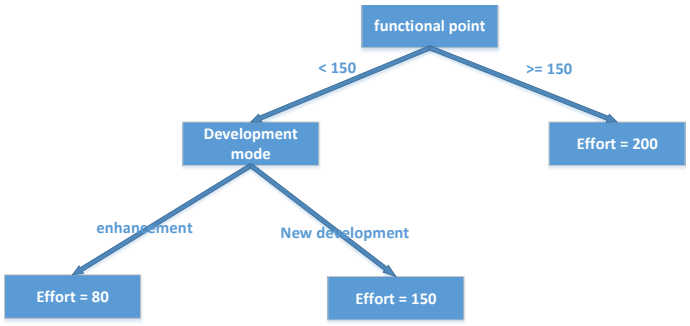


Figure A.3: An example of RT for SEE and its prediction process.

popular example of the activation function is *sigmoid*:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \in [0, 1]. \tag{A.20}$$

ANN makes no or minimal assumptions on the function being modelled from input features to effort values and the data being used for training, and can approximate any continuous function. The training process consists of adjustments to the connection weights by means of *back propagation* [154]. For more details, please refer to [24].

A few types of ANN have been used in SEE [47, 89], among which Multilayer Perceptron (MLP) is one of the most implemented form [76]. MLP consist of at least three layers of neurons, and the neurons of a layer are connected to all neurons of the next layer. MLP have not been shown to be so frequently among the best SEE approaches [136]. For instance, Dejaeger et al. [49] argued that techniques such as ordinary least squares regression based on logarithm transformed data performed in general better than MLP analysed in their study. On the other hand, many studies reported the favourably performance of MLP for SEE [48, 7, 61]. For instance, Tronto et al. [48] showed that MLP improved SEE over conventional linear models because they were not restricted to linear functions, being able to model observations that lie far from the best straight line. Earlier work also reported favourably over MLP [190]. Nevertheless, compared with RT and ABEs that are more straightforward and understandable, MLP are more like a less welcoming 'black-box' to project managers in terms of their training and prediction processes, which tells very limited information how the models proceed effort estimate.

A.7 Ensembles of Learning Methods

Lately, ensembles of learning machines have been attracting the attention of the SEE community as they can frequently improve performance over single learning machines [29, 116, 136, 8, 110]. An ensemble approach consists of a set of base models that are trained to perform the same task and then combined together with the aim of improving the final prediction performance. Ensemble approach can improve the prediction accuracy over its base learners because each of them has particular assumptions, weaknesses and strengths, which may be best suitable to different parts of the training examples [8, 193],

and can facilitate each other by patching the errors made by others [57].

It is commonly agreed that the base models should behave differently; otherwise, the overall prediction performance will not be better than the individual models [36]. Intuitively, if base models making the same mistakes are combined into an ensemble, the ensemble will take the same mistakes as the individual models and its overall performance will be no better than the individual performance. On the contrary, ensembles composed of *diverse* base models can compensate the mistakes of certain models through the correct prediction performed by other models. In the context of SEE, *diversity* refers to the prediction/errors made by the models, i.e., two models are said to be diverse if they make different errors on the same testing examples [38]. Different ensemble approaches can be seen as different ways to generate diversity among the base models.

Bootstrap **agg**regat**ing** (*Bagging*) is a typical ensemble approach designed to improve the stability and accuracy of learning machines, which can reduce the prediction error when its base models are unstable [32]. Bagging generates multiple base models trained on different Bootstrap training bags as follows. Given a training set $\mathscr{D}$ of size N, Bagging generates M new training sets $\{\mathscr{D}^{(m)}|m = 1, \cdots, M\}$, each of size N, by sampling from $\mathscr{D}$ uniformly and *with replacement*. On each new training set $\mathscr{D}^{(m)}$, one base model is trained and thus we have M trained models via Bagging. The final prediction is the average of the prediction from all base models.

Boosting [56] is another popular ensemble approach, which arranges base models in a sequential manner. Unlike Bagging, the subset creation is not random but depends on the performance of the previous models: each new subsets contains the training examples that are likely to have large prediction errors by previous base models, and thus each base model pays special attention to the training examples on which the previous method is unsuccessful. Boosting is reported to be better than Bagging in general ML, but has trouble in handling noisy data [21, 192]. Likely for this reason, Boosting is less popularly used in SEE since SEE data is highly likely to contain data noise.

In SEE literature, ensemble methods are grouped into two categories: (1) homogeneous that combines the same type of learning machines with different configurations and (2) heterogeneous that combines at least two types of learning machines [57]. The homogeneous approach is the most commonly used [76]. Recent literature in SEE tends to support the use of ensemble approaches, showing that ensembles of SEE methods usually achieve generally better results than single learning machines [76, 110, 137, 136, 110].

For instance, Braga et al. [29] claimed that Bagging could improve the performance of several base models such as RT and MLP; and Kultur et al. [116] reported that an adapted version of Bagging provided very large improvements over several single SEE methods. Moreover, Minku and Yao [137] reported that a Bagging ensemble of MLP performed similarly to RT in SEE. Later, experimental results in [136] showed that combining the power of ensembles to local approaches, such as Bagging ensembles of RT, outperformed several other SEE methods. They concluded that combining ensembles and locality is a way to tailor ensembles for SEE, and indicated that more improvements may still be achieved if additional tailoring is performed. Moreover, Kocaguneli et al. [110] proposed an ensemble scheme that outperformed single methods for SEE. Their method combines

several types of so called solo-methods (i.e. single learners and preprocessing techniques) to perform SEE. They also used Bagging and Boosting as solo-methods in their heterogeneous ensemble approach. They reported that the ensemble presented less instability than solo-methods when ranked in terms of the total number of wins, losses, and wins-losses, considering several different performance metrics and twenty data sets. These observations confirmed earlier results reported in the ensemble learning literature that ensembles generally perform better than its base models. They also reported that the ensemble methods obtained less losses than other methods.

APPENDIX B

Statistical Tests for SEE Methods

This appendix discusses statistical tests used in this thesis for validating the significance of performance difference in terms of point prediction of SEE methods.

Though most SEE studies involve comparisons among different SEE methods [89], it has been not long since statistical tests are introduced into the experimental framework of SEE from the general ML literature [136, 50, 105, 29, 156, 46]. The popular statistical tests used in the SEE community include Wilcoxon signed-rank tests for a comparison of two SEE methods across multiple data sets, and Friedman tests for a comparison of more than two methods across multiple data sets. Recent studies also emphasize the importance of considering effect size to quantify the difference in performance [164]. Note that our discussion on Wilcoxon signed-rank test and Friedman test are written with large inspiration from Demsar's paper [50].

B.1 Wilcoxon Signed-rank Test

Wilcoxon signed-ranks test is a non-parametric statistical test, and is typically used to compare two methods across multiple data sets based on their ranks. The null hypothesis (H0) states that the two methods are equivalent in terms of prediction performance. The alternative hypothesis (H1) states that they differ.

Wilcoxon signed-rank test is conducted in the following procedures [50]. (1) Let d_i be the performance superiority of the method P_2 to the method P_1 on the i^{th} out of N data set. The differences $\{d_i\}$ are ranked according to their absolute values, and average ranks are assigned in case of ties. (2) Let R^+ be the sum of ranks across the data sets where P_2 outperforms P_1 (i.e. $d_i > 0$), and R^- be the sum of ranks where P_1 outperforms P_2 (i.e. $d_i < 0$). Ranks of $d_i = 0$ are split evenly among the sums, and if the number is odd, one is ignored. This step is formulated as

$$\begin{aligned} R^+ &= \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i), \\ R^- &= \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i). \end{aligned} \tag{B.1}$$

Table B.1: Comparison of predictors P_1 vs P_2 in terms of prediction accuracy [50, table 2].

Data set	P_1	P_2	d_i	rank
D1	0.763	0.768	+0.005	3.5
D2	0.599	0.591	-0.008	7
D3	0.954	0.971	+0.017	9
D4	0.628	0.661	+0.033	12
D5	0.882	0.888	+0.006	5
D6	0.936	0.931	-0.005	3.5
D7	0.661	0.668	+0.007	6
D8	0.583	0.583	0.000	1.5
D9	0.775	0.838	+0.063	14
D10	1.000	1.000	0.000	1.5
D11	0.940	0.962	+0.022	11
D12	0.619	0.666	+0.047	13
D13	0.972	0.981	+0.009	8
D14	0.957	0.978	+0.021	10

(3) Let T be the smaller of the sums as

$$T = \min(R^+, R^-). \tag{B.2}$$

(4) For a large number of data sets, the statistics z should be distributed approximately normally, defined as

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}}. \tag{B.3}$$

Specifically, with $\alpha = 0.05$, the null-hypothesis can be rejected if z is smaller than -1.96. When the number of data sets is small, we can refer to the tables of *critical values* of Wilcoxon signed-rank test [2] to compare the performance of two methods. Most books on general statistics include a table of exact critical values for T and N up to 25 (or sometimes more) for us to refer. In practice, there is often not enough data sets investigated, therefore the tables for critical values are often required when comparing the performance of two methods.

An example of Wilcoxon signed-rank test is described in table B.1 and [50]. The null hypothesis (H0) is that the algorithms P_1 and P_2 perform equally well in terms of prediction accuracy. As shown in table B.1, the predictors performed equally on two data sets (D8 and D10) with $d_8 = d_{10} = 0$. The ranks are assigned from the lowest to the highest with respect to the absolute difference $\{d_i\}$. In particular, the equal difference of d_1 and d_6 is assigned with their average ranks. The sum of ranks for the positive and negative difference are

$$\begin{aligned} R^+ &= 3.5 + 9 + 12 + 5 + 6 + 14 + 11 + 13 + 8 + 10 + 1.5 = 93, \\ R^- &= 7 + 3.5 + 1.5 = 12. \end{aligned} \tag{B.4}$$

Thus, we can have that

$$T = \min(R^+, R^-) = 12. \tag{B.5}$$

According to the table of *critical values* of Wilcoxon signed-ranks test [2] with the confidence level $\alpha = 0.05$ and $N = 14$, the difference between methods P_1 and P_2 is significant if the smaller of the sums is equal to or less than 21. Therefore, we reject the null-hypothesis and conclude that methods P_1 and P_2 perform significantly differently.

B.2 Friedman Test

Friedman test is a non-parametric statistical test and typically used to determine whether the prediction performance of multiple methods is significantly different across multiple data sets [50]. The null hypothesis (H0) states that all the predictors are equivalent in terms of prediction accuracy/error. The alternative hypothesis (H1) states that at least one pair of the methods differ.

Friedman test is conducted in the following procedures [50]. (1) Let r_n^k be the rank of the k^{th} of K algorithms on the n^{th} out of N data sets. (2) Compute the average ranks of the methods across all data sets as

$$R_k = \frac{1}{N}\sum_{n=1}^{N} r_n^k, \quad \forall k = \{1, \cdots, K\}. \tag{B.6}$$

(3) Compute the Friedman statistic as

$$\chi_F^2 = \frac{12N}{k(k+1)}[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1^2)}{4}], \tag{B.7}$$

and it should be distributed according to χ_F^2 with $K-1$ degrees of freedom. Iman and Davenport showed that Friedman's χ_F^2 was undesirably conservative and derived a better statistic base on χ_F^2 as

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2}, \tag{B.8}$$

and it should be distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom when N and K are big enough (as a rule of a thumb, $N > 10$ and $K \geqslant 4$). For small numbers of methods and data sets, we can refer to the tables of the exact *critical values* of Friedman tests [1] to accomplish the statistical test.

An illustration of Friedman test is discussed in table B.2 and [50]. The null hypothesis (H0) states that the four methods perform statistically equally in terms of prediction accuracy. The rank r_n^k of method P_k on the n^{th} data set is in the parentheses associated to its prediction accuracy. Their average ranks $\{R_k\}$ across data sets provide a fair comparison of the methods. As shown in table B.2, methods P_2 and P_4 have the best and second best average ranks, followed by P_3 with the third average rank. P_1 has the worse average rank. Compute the statistics to judges whether the average ranks are significantly

Table B.2: Comparison of predictors P_1, P_2, P_3, and P_4 in terms of prediction accuracy. The ranks in the parentheses are used in computation of the Friedman test [50, table 6].

Data set	P_1	P_2	P_3	P_4
D1	0.763 (4)	0.768 (3)	0.771 (2)	0.798 (1)
D2	0.599 (1)	0.591 (2)	0.590 (3)	0.569 (4)
D3	0.954 (4)	0.971 (1)	0.968 (2)	0.967 (3)
D4	0.628 (4)	0.661 (1)	0.654 (3)	0.657 (2)
D5	0.882 (4)	0.888 (2)	0.886 (3)	0.898 (1)
D6	0.936 (1)	0.931 (2.5)	0.916 (4)	0.931 (2.5)
D7	0.661 (3)	0.668 (2)	0.609 (4)	0.685 (1)
D8	0.583 (2.5)	0.583 (2.5)	0.563 (4)	0.625 (1)
D9	0.775 (4)	0.838 (3)	0.866 (2)	0.875 (1)
D10	1.000 (2.5)	1.000 (2.5)	1.000 (2.5)	1.000 (2.5)
D11	0.940 (4)	0.962 (2.5)	0.965 (1)	0.962 (2.5)
D12	0.619 (3)	0.666 (2)	0.614 (4)	0.669 (1)
D13	0.972 (4)	0.981 (1)	0.975 (2)	0.975 (3)
D14	0.957 (3)	0.978 (1)	0.946 (4)	0.970 (2)
average rank	3.143	2.000	2.893	1.964

different under the null-hypothesis as

$$\begin{aligned} \chi_F^2 &= \tfrac{12\times 14}{4\times 5}[3.143^2 + 2.000^2 + 2.893^2 + 1.964^2) - \tfrac{4.5^2}{4}] = 9.28 \\ F_F &= \tfrac{13\times 9.28}{14\times 3 - 9.28} = 3.69. \end{aligned} \tag{B.9}$$

With 4 methods and 14 data sets, F_F is distributed according to the F distribution with $4-1=3$ and $(4-1)\times(14-1)=39$ degrees of freedom. The critical value of $F(3,39)$ for $\alpha = 0.05$ is 2.85 according to the table of critical values for F distribution [3]. Therefore, we reject the H0 and conclude that the four methods differ significantly.

If the null hypothesis is rejected, we can proceed with a post-hoc test, where all methods are compared against a control method, to check where the difference is. In the thesis, I adopt Holm-Bonferroni corrections to compare each SEE method against the control method for its greater power to identify the different pairs in comparison to other post-hoc statistics based on pair-wise comparisons such as Nemenyi and for its making no additional assumptions about the hypotheses tests [50]. Sometimes the Friedman test reports a significant difference but the post-hoc test fails to detect the significantly different pairs. This is mainly due to the lower power of the post-hoc test. In other words, the post-hoc test is generally weaker than the Friedman test. In this case, no other conclusions than that some methods do differ can be drawn [50].

Holm-Bonferroni post-hoc test is conducted as the procedures of Holm's [72]. (1) Compute statistics $\{z_k\}$ by comparing method P_k against the control method P_c in ranks

Table B.3: Holm-Bonferroni post-hoc correction after Friedman test with the significance level 0.05 for predictors P_1, P_2, P_3, and P_4 in terms of prediction accuracy. P_1 is chosen as the control method. Statistics $\{p_k\}$ have been sorted in increasing order [50].

$\sigma(k)$	Predictor	$z = (R_1 - R_k)/SE$	p	$\alpha/(K - \sigma(k))$
1	P_4	$(3.143 - 1.964)/0.488 = 2.416$	0.016	$0.05/(4-1) = 0.017$
2	P_2	$(3.143 - 2.000)/0.488 = 2.342$	0.019	$0.05/(4-2) = 0.025$
3	P_3	$(3.143 - 2.893)/0.488 = 0.512$	0.607	$0.05/(4-3) = 0.050$

as

$$z_k = (R_k - R_c)/\sqrt{\frac{K(K+1)}{6N}}, \forall k \in \{1, \cdots, K\}. \tag{B.10}$$

(2) Find the corresponding probabilities $\{p_k\}$ based on the statistics $\{z_k\}$ from the CDF table of Gaussian distribution. (3) Sort the $\{p_k\}$ increasingly such that $p_{\sigma(1)} \leqslant p_{\sigma(2)} \cdots p_{\sigma(K-1)}$ (suppose that $p_{\sigma(K)}$ corresponds to the control method), where $\sigma(\cdot)$ is a permutation function. (4) Compare $p_{\sigma(k)}$ with $\frac{\alpha}{K-\sigma(k)}$, where α is a significance level, such as $\alpha = 0.05$. Holm-Bonferroni's step-down procedure starts with the smallest p value. Specifically, if $p_{\sigma(1)}$ is below $\frac{\alpha}{K-1}$, the corresponding hypothesis is rejected and we are allowed to compare $p_{\sigma(2)}$ with $\frac{\alpha}{K-2}$. If the second hypothesis is rejected, the test proceeds with the third, and so on. As soon as a certain null hypothesis cannot be rejected, all the remaining hypotheses are accepted.

The procedures of Holm-Bonferroni correction is illustrated in table B.2 and [50]. Let the standard error be $SE = \sqrt{\frac{4 \cdot 5}{6 \cdot 14}} = 0.488$, and compute the statistics z_k as in table B.3. Find the corresponding probabilities $\{p_k\}$ based on those $\{z_k\}$ as shown in the table. Note that the $\{p_k\}$ have been sorted in increasing order. Compare $p_{\sigma(k)}$ with $\frac{\alpha}{K-\sigma(k)}$ and we can see that the first and the second hypotheses can be rejected, but not the third one. So we conclude that P_1 performs significantly worse than P_4 and P_2.

B.3 Effect Size

Effect size is a simple way of quantifying the difference between two methods with multiple runs [183, 46]. It can be used to justify whether there is large performance improvement over a *control method*, since the statistical significance test cannot detect the magnitude of performance difference. There are more than seventy varieties of effect sizes [12, 183, 46], among which we discuss the parametric Cohen's d and non-parametric Vargha and Delaney's A_{12} as they have been recommended in the community of SEE [121, 164].

Cohen's d is probably the most popular standardized effect size, which is based on the difference between the average performance across multiple runs (e.g. average performance across data sets) [168]. In the SEE context, if it is obvious that which of the two groups of experiments corresponds to the *control method*, the effect size d can be calculated as

$$d = \frac{|\overline{pf_{ctr}} - \overline{pf_{cmp}}|}{std_{ctr}}, \tag{B.11}$$

where $\overline{pf_{cmp}}$ and $\overline{pf_{ctr}}$ denote the average prediction performance across data sets of the competing and control methods respectively, and std_{ctr} denotes the standard deviation of the performance of the control method across data sets. If it is not obvious to decide the control method, the effect size can be calculated using the *pooled standard deviation* as

$$d = \frac{\overline{pf_1} - \overline{pf_2}}{\sqrt{\frac{std_1^2 + std_2^2}{2}}}, \tag{B.12}$$

where $\overline{pf_i}$ denotes the prediction performance of the i^{th} ($i \in \{1, 2\}$) method across data sets, and std_i denotes its corresponding standard deviation. Overall, Cohen's d effect size is measured by the difference in mean that is scaled over the standard deviation. The effect size d can be interpreted in terms of Cohen's categories [164] as: small (≈ 0.2), medium (≈ 0.5) and large (≈ 0.8).

A potential problem of Cohen's d effect size is that it is parametric and assumes the normality of the data, and thus the measure of effect size is sensitive to the violation of the assumption of normality [12, 46]. For this reason, some more robust non-parametric alternatives have been suggested [168].

The Vargha and Delaney's A_{12} is a non-parametric effect size that makes no assumptions on the underlying distribution [183, 12]. Its use has been advocated in [121], and one example of its use in software engineering can be found in [150]. Effect size A_{12} is computed as

$$A_{12} = \frac{1}{N} \cdot (\frac{R_1}{N} - \frac{N+1}{2}), \tag{B.13}$$

where N is the number of data sets, and R_1 is the statistics of the Wilcoxon rank sum test[1]. Wilcoxon rank sum test (also called Mann-Whiney U-test) is non-parametric and does not require the assumption of normal distributions [126]. The null hypothesis states that the prediction performance of methods P_1 and P_2 are sampled from continuous distributions with equal median.

In the SEE context, A_{12} measures the probability that running predictor P_1 yields better performance than running the other predictor P_2. If the two predictors are equivalent, then $A_{12} = 0.5$. This effect size is easier to interpret compared to the d family. For instance, $A_{12} = 0.7$ entails we would obtain higher results 70% of the time with predictor P_1. It is interpreted according to Vargha and Delaney's categories [183]: small ($\geqslant$0.56), medium ($\geqslant$0.64) and large ($\geqslant$ 0.71).

Though being less robust than non-parametric effect size, the parametric Cohen's d is more powerful than the non-parametric Vargha and Delaney's A_{12} when the normality assumption holds. However, the tests to check whether assumptions (e.g. normality tests) hold are not always good, and thus A_{12} would be a 'safe' option though it may be a bit weaker. In practice, people should choose the type of effect size according to their purpose.

[1]MATLAB function *ranksum* can be used to calculate this statistical test. Wilcoxon rank sum test is used to determine whether two *independent* samples are selected from populations having the same distribution; Wilcoxon signed-rank test is used for this purpose but with *dependent* samples.

APPENDIX C

Effect of Synthetic Data on Prediction Performance in Original Scale of Effort Values

To investigate the effectiveness of our synthetic project generator proposed in chapter 3, table 3.3 reports the performance comparisons in terms of MAE in the logarithm scale of effort values for being less affected by project size.

In this appendix, we report the performance comparisons in terms of MAE in the original scale of effort values. Specifically, once an SEE method provides an effort estimate, take the exponentiation of that value as the final prediction and calculate the prediction performance in terms of MAE using effort values in the original scale. Table C.1 lists the performance comparisons between pairs of *syn.SEEr* vs *bsl.SEEr* across 14 data sets in terms of MAE for small, medium, and large training set sizes. We can see that the key conclusion that syn.SEEr always performed similarly/better than bsl.SEEr remains the same.

Again, the performance of MLR/ATLM is unstable in some SEE data sets. For instance, ATLM performs extremely poorly in many of the data sets, with infinite MAE (mean MAE of 30 runs), when the training set is small. Further investigation finds that ATLM performs extremely poorly on one or two of the 30 runs. As discussed in section 3.4.1, the unstable performance of MLR/ATLM may be due to the scarcity of training examples causing the ill-conditional problem when doing matrix inversion in the training process. This problem becomes even worse when computing the prediction performance in the original scale of effort values.

[0]This appendix corresponds chapter 3 for answering RQ1.

Table C.1: Performance comparison of pairs of *syn.SEEr* vs *bsl.SEEr* across 14 data sets in terms of MAE in the original effort space for small, medium and large training set sizes. The different training set sizes refer to different *holdout* values in table 3.1. The reported values are the mean of 30 runs followed by their 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. The last two rows list Wilcoxon tests with Bonferroni correction. The overall comparison of *bsl.SEEr* vs *syn.SEEr* can be seen from *aveRank* (average rank). The first value 1 (or 0) in *Wilcoxon* row means there is (or not) significant difference, and its corresponding p-value comes the next. Significant difference is highlighted in orange (dark grey) on this row. 'Inf' indicates *infinite* error in terms of MAE, which only happens to MLR or ATLM with small training set size. Further investigation finds that only one or two out of the total 30 runs commit extremely large error.

(a) Small training set size.

Data	syn.MLR	bsl.MLR	syn.ATLM	bsl.ATLM	syn.k-NN	bsl.k-NN	syn.RVM	bsl.RVM	syn.RT	bsl.RT	syn.SVR	bsl.SVR
Maxwell	**5207.4±1329.3**	**49636.7±82397.5**	**5654.1±2005.6**	**Inf±NaN**	5244.0±868.6	5285.7±858.9	**4262.7±788.4**	**4728.6±981.5**	**5147.4±693.5**	**5335.0±749.4**	**4230.8±835.1**	**4532.9±850.9**
Cocomo81	**450.8±217.6**	**Inf±Inf**	**648.9±476.4**	**Inf±NaN**	625.5±123.3	625.2±136.5	**390.0±155.6**	**451.3±136.0**	607.4±122.8	627.1±130.1	402.5±128.2	421.2±145.8
Nasa93	**435.6±185.9**	**Inf±Inf**	**552.0±398.5**	**Inf±Inf**	470.7±75.8	478.4±73.0	**317.5±103.8**	**335.8±101.8**	**440.0±77.5**	**467.6±68.8**	339.3±86.7	331.2±69.4
Kitchenham	**2122.3±1141.1**	**3230.0±2871.6**	3850.1±4919.2	5875.8±8794.2	2277.5±317.6	2305.9±350.1	**2171.6±485.4**	**2323.6±364.8**	**2372.9±335.5**	**2413.1±249.5**	**1877.2±616.0**	**2046.8±391.3**
Albrecht	**24.7±48.5**	**Inf±Inf**	**30.1±51.7**	**Inf±NaN**	12.9±4.2	12.5±4.4	**11.0±4.6**	**12.5±5.9**	**15.0±4.8**	**16.3±3.4**	9.7±3.5	10.3±4.0
Kemerer	751.9±2829.2	2272.1±10472.1	**1001.6±3209.8**	**Inf±NaN**	132.2±43.0	140.3±38.1	132.5±50.8	130.2±46.7	151.3±40.5	154.8±32.5	114.8±41.4	118.4±42.3
Deshar	**3851.0±3064.1**	**Inf±Inf**	**4655.2±5003.8**	**Inf±Inf**	2711.9±316.0	2728.7±380.7	2579.1±402.8	2640.7±533.1	**2752.8±346.3**	**2949.0±284.4**	2324.3±264.8	2326.3±310.8
Org1	**5468.8±6301.4**	**Inf±Inf**	**29722.3±88952.6**	**Inf±NaN**	3674.6±804.8	3557.5±378.5	3560.1±365.9	3590.3±407.7	3862.3±678.6	3777.4±440.4	**3476.9±335.2**	**3481.6±384.4**
Org2	273853.8±1480233.9	293784.7±1468380.1	3452.7±2447.7	59061.8±247795.7	2284.0±896.0	2207.2±687.8	2129.6±520.1	2042.0±396.7	2403.3±1054.6	2335.8±825.1	2158.0±720.3	2186.0±874.7
Org3	1291.1±394.8	1434.8±714.0	1310.9±478.7	1494.6±770.8	**1296.9±150.4**	**1312.0±166.5**	**1284.9±255.7**	**1383.7±181.6**	**1365.0±204.5**	**1549.2±118.7**	**1158.3±200.6**	**1217.4±220.8**
Org4	**7623.3±10567.0**	**Inf±Inf**	**6742.3±5388.8**	**Inf±NaN**	4693.1±774.8	4416.2±404.2	4442.2±271.6	4411.3±257.1	4618.0±810.7	4586.8±813.0	4275.5±316.5	4426.0±717.8
Org5	1034100.0±5160676.2	Inf±Inf	**138289.8±418598.6**	**Inf±Inf**	6979.1±1234.0	7129.3±1101.8	7191.9±1707.4	7218.3±967.8	7684.0±1575.6	7509.1±1304.0	**6388.8±2096.3**	**6905.9±1988.9**
Org6	**1309619.1±6963276.4**	**Inf±Inf**	**2683972.5±9401040.3**	**Inf±Inf**	3503.4±919.3	3440.5±839.1	3468.3±817.2	3635.1±805.0	4544.8±1948.0	4631.0±1880.4	**3455.7±1500.7**	**3634.1±1081.8**
Org7	**2803187.7±14862957.4**	**Inf±Inf**	**3333716.6±15060437.3**	**Inf±Inf**	5889.4±1619.0	5529.5±812.5	5577.9±1042.1	5447.3±617.8	5759.9±1314.1	5529.5±812.5	5635.9±1240.5	5396.4±779.7
aveRank	1.00	2.00	1.00	2.00	1.50	1.50	1.29	1.71	1.36	1.64	1.14	1.86
Wilcoxon	1	0.000091	1	0.000091	0	0.669800	0	0.135254	0	0.625732	1	0.003763

(b) Medium training set size.

Data	syn.MLR	bsl.MLR	syn.ATLM	bsl.ATLM	syn.k-NN	bsl.k-NN	syn.RVM	bsl.RVM	syn.RT	bsl.RT	syn.SVR	bsl.SVR
Maxwell	**3628.6±1149.4**	**6669.2±7619.6**	**3670.4±1152.7**	**10361.5±22773.4**	4870.7±1984.5	5071.2±1972.5	**3921.5±1019.3**	**4381.8±1340.7**	4656.1±1618.0	4661.2±1786.7	3631.9±1327.4	3766.8±1302.2
Cocomo81	258.9±159.4	261.9±131.6	**295.1±172.4**	**506.3±390.2**	522.6±286.2	548.0±283.9	262.2±164.8	261.3±169.3	**445.6±236.3**	**497.0±231.5**	256.8±151.3	266.1±137.7
Nasa93	327.2±137.1	260.1±157.7	327.2±137.1	260.1±157.7	464.3±152.4	473.6±161.4	246.7±80.3	251.8±79.9	387.2±148.8	393.1±148.9	296.2±113.3	280.4±103.8
Kitchenham	1673.2±282.4	9698.5±42660.7	1723.3±383.6	10005.9±42632.8	2032.3±231.0	2038.2±247.4	**1848.3±282.6**	**2034.8±369.1**	2203.7±302.7	2200.3±290.3	1668.4±364.4	1782.4±478.7
Albrecht	**9.3±4.5**	**18.3±23.0**	**10.3±8.0**	**21.1±24.0**	9.4±4.9	9.4±4.8	8.6±3.9	8.3±3.6	**10.7±4.9**	**14.6±5.9**	8.6±4.3	8.3±3.7
Kemerer	**122.5±83.1**	**1298.4±3830.9**	**118.0±81.8**	**2472.5±7938.6**	111.3±79.3	119.1±79.5	105.5±73.8	112.8±79.4	128.1±79.6	137.9±69.1	97.3±70.7	98.9±74.2
Deshar	**2190.0±355.8**	**2586.6±846.0**	**2188.8±357.8**	**2621.1±850.9**	2436.6±217.8	2425.6±243.5	2227.7±341.5	2289.7±422.4	2570.8±276.4	2590.4±330.7	2018.9±257.9	2035.7±276.1
Org1	3411.0±456.7	3893.2±1931.0	3583.9±703.9	4677.7±4655.4	3463.9±392.2	3488.2±413.2	3552.1±408.3	3567.4±382.3	3692.4±232.8	3632.4±183.2	**3261.0±351.2**	**3451.4±411.2**
Org2	1848.9±310.5	1887.1±421.2	1846.3±318.4	1887.6±429.9	1978.0±318.1	1957.4±282.4	1880.4±361.0	1827.9±357.4	**1954.0±253.0**	**2086.4±299.4**	1817.2±333.8	1833.5±302.0
Org3	1124.0±215.7	1168.6±358.5	1128.1±219.5	1174.2±365.9	1209.1±96.1	1214.3±103.5	**1138.0±130.0**	**1239.2±160.4**	**1251.6±128.0**	**1297.5±77.2**	1081.2±163.6	1068.3±120.1
Org4	**4192.4±449.7**	**8658.1±19680.0**	**4251.6±559.6**	**9879.7±21191.7**	4209.8±435.3	4202.6±379.1	**4052.5±335.1**	**4470.0±452.3**	**4199.7±265.5**	**4374.5±258.5**	**3991.5±287.2**	**4110.5±351.2**
Org5	**7268.3±6623.5**	**23826.5±56644.7**	10483.1±12093.6	27476.3±57745.5	5988.1±1400.1	5944.9±1386.5	**5880.4±1650.5**	**6549.7±2135.5**	**6066.5±1532.7**	**6848.0±1362.1**	4424.9±1586.1	4625.8±1680.3
Org6	**11502.9±20457.8**	**90310.8±394316.3**	**35407.4±97997.7**	**167984.4±527885.4**	2894.5±783.8	2960.8±836.8	**3092.0±825.6**	**3900.1±2564.6**	**3240.8±697.8**	**3830.4±742.2**	**2886.0±1025.2**	**3179.1±1074.6**
Org7	5138.5±825.4	15477.8±58189.8	5294.1±1211.4	15549.2±58178.2	5192.4±856.9	5236.9±747.3	**5011.7±568.7**	**5684.5±1439.5**	**4959.0±993.1**	**5190.6±781.5**	4836.5±779.0	4904.0±818.5
aveRank	1.07	1.93	1.07	1.93	1.29	1.71	1.21	1.79	1.14	1.86	1.21	1.79
Wilcoxon	1	0.000670	1	0.000670	0	0.172607	1	0.016255	1	0.003763	1	0.016255

(c) Large training set size.

Data	syn.MLR	bsl.MLR	syn.ATLM	bsl.ATLM	syn.k-NN	bsl.k-NN	syn.RVM	bsl.RVM	syn.RT	bsl.RT	syn.SVR	bsl.SVR
Maxwell	4314.8±4878.2	4089.5±5153.2	4326.1±4891.4	4180.9±5174.5	6561.9±10529.7	6606.7±10506.1	3983.2±4636.6	4999.8±6782.3	**4450.5±7220.8**	**4806.7±7286.5**	3859.9±3985.7	3562.1±3844.0
Cocomo81	232.1±401.6	247.8±531.5	248.6±449.0	423.7±1360.0	326.5±454.4	387.8±485.6	136.2±220.2	172.4±279.4	**242.5±421.3**	**359.9±520.5**	183.8±340.5	206.0±375.4
Nasa93	174.8±250.6	130.5±190.6	174.8±250.6	130.5±190.6	282.6±458.5	302.3±463.6	135.5±171.4	184.6±214.6	177.9±363.7	166.1±263.9	117.2±156.3	147.2±206.8
Kitchenham	1552.2±849.6	1562.6±861.0	1483.3±771.7	1499.2±767.0	1802.0±1042.0	1817.2±1089.8	1513.3±847.2	1551.3±902.0	**1893.0±1080.8**	**1977.6±1045.2**	1412.9±705.4	1410.8±684.8
Albrecht	6.3±6.8	11.5±31.6	6.3±6.8	11.5±31.6	6.7±10.2	8.5±14.6	5.5±5.6	7.0±7.8	**6.9±6.9**	**11.7±12.0**	5.8±9.4	6.3±11.4
Kemerer	110.3±194.3	115.6±193.3	106.8±194.5	112.8±194.0	119.2±207.2	120.4±208.0	85.5±148.0	103.1±227.7	125.1±220.6	145.1±222.0	99.7±200.7	101.5±197.6
Deshar	2030.8±420.4	2114.8±442.7	2031.5±419.5	2117.0±443.7	2325.5±487.4	2377.4±456.9	2011.1±474.4	2089.2±483.2	2239.2±572.5	2321.3±707.8	2016.4±425.3	2041.7±422.8
Org1	2779.6±2475.8	2764.4±2417.1	2914.8±2447.3	3077.2±2420.1	3448.6±2740.6	3431.4±2938.9	3153.3±3000.7	3176.2±3043.7	**3378.9±2829.1**	**3933.4±2645.8**	2849.2±2521.8	2881.8±2446.5
Org2	1575.7±442.9	1582.9±486.4	1565.9±447.4	1567.1±485.5	1772.5±501.5	1835.2±489.7	1631.7±537.7	1636.4±587.9	1738.2±465.8	1684.7±510.9	1524.7±396.8	1520.6±439.7
Org3	944.6±199.9	942.4±206.3	944.7±197.2	942.4±206.3	1065.4±230.5	1078.7±216.8	950.9±206.1	964.5±220.7	998.3±222.3	976.0±235.2	931.1±193.7	929.4±200.4
Org4	3909.8±879.0	3963.8±859.0	3944.6±878.5	4046.7±917.9	4126.1±1009.8	4239.5±939.1	3753.7±880.5	3815.0±892.5	3997.0±899.0	4074.2±946.9	3838.6±837.2	3896.8±794.7
Org5	3806.3±1886.2	4671.7±3252.0	3976.3±1817.3	4733.0±3233.7	5628.3±2677.5	5783.1±3051.8	**4478.5±2463.8**	**5715.3±3091.7**	5515.4±3079.3	5719.0±3411.1	3767.8±1656.0	3364.6±1482.1
Org6	4269.9±5608.5	10318.5±12906.8	4580.1±5872.7	10343.3±12890.4	2909.3±1396.8	2838.7±1415.9	2135.7±1039.4	2006.8±781.1	2864.0±1368.8	2845.3±1401.7	2085.0±891.3	2165.6±743.8
Org7	4648.4±1676.9	4656.7±1571.9	4660.3±1696.1	4663.1±1595.4	5065.9±1561.5	4915.4±1565.3	4649.9±1709.4	4675.9±1643.7	4458.3±1427.5	4394.8±1541.3	4470.1±1448.1	4366.9±1328.7
aveRank	1.29	1.71	1.21	1.79	1.21	1.79	1.07	1.93	1.36	1.64	1.43	1.57
Wilcoxon	0	0.153076	1	0.016255	0	0.153076	1	0.000670	0	0.067627	0	0.903198

APPENDIX D

Supplementary Experiments with Separate Test Set

We do not have spare testing sets and report the best performance determined on validation sets. As explained in Chapters 4 and 5, such experimental setting is rational and consistent to previous SEE studies [109, 111, 135, 136], because SEE data sets are usually small and the evaluation would be potentially invalid if further spare for testing sets. To tackle this concern, we investigate three SEE data sets that are relatively large (i.e., Kitchenham, Org3, and Org4 according to table 2.1), for which the division for training, validating, and testing sets would be more proper. Table D.1 lists the information of their effort values.

Specifically, Chapters 5 evaluates SynB-RVM with respect to point performance (in terms of MAE, MdAE, LSD, SA) and uncertain performance (in terms of hit rate and relative width), concluding that SynB-RVM is a robust winner and rarely performs significantly worse than its competitors (see table 5.7). This appendix complements the results of tables 5.2(a), 5.3, and 5.5 when spare testing sets are used.

The evaluation procedures are as follows. (1) Randomly holdout 90% data set to be the training and validation sets, and the rest is spared as the testing set. (2) Determine the best parameter setting of this method in terms of the average MAEs across 10 times 10-fold CV based on the training and validation sets. MAE is selected for being unbiased towards over/under-estimation and for reflecting the actual error magnitude. (3) Build the model with the best parameter settings using both training and validation sets. (4) Evaluate the performance on the separate testing set. (5) Repeat steps (1)$\sim$(4) for 30 times and report the mean performance. In fact, the experimental design is the same to that of chapter 5 except that the performance is evaluated in spare testing sets.

D.1 Point Estimate With Spare Test Set

Point performance of SynB-RVM is compared to ATLM and RVM when spare testing data is used. ATLM usually performs the best among the competing methods and thus is chosen in the experiment; RVM is chosen for being the base learner of SynB-RVM. Other point estimators are excluded for being obviously inferior to SynB-RVM according

Table D.1: Mean/Median of the actual effort values in Kitchenham, Org3 and Org4. Q_1 represents the first quartile (the middle value between the smallest and the median effort values), and Q_3 represents the third quartile (the middle value between the median and the highest value of the effort).

Data Set	Mean (STD)	Median ($Q_3 - Q_1$)
Kitchenham	3113.1 (9598)	1557 (2066)
Org3	2007.1 (2665.9)	1089.5 (1610)
Org4	5970.3 (8141.3)	3520 (5117)

Table D.2: Point prediction performance of SEE methods in terms of MAE with spare testing sets. The reported values are the mean of 30 runs of 10-fold CV with their optimal parameter settings determined by validation sets. Effect size across 30 runs of each data set against the control method is computed. SynB-RVM_2Dhist is chosen as the control method as in table 5.2. Cells in green (light grey)/orange (dark grey) indicate better or worse performance against the control method with medium/large effect size.

Data Set	SynB-RVM_SpMn	SynB-RVM_1Dhist	SynB-RVM_2Dhist	RVM	ATLM
Kitchenham	1698.15	1687.50	1686.52	1725.89	1603.57
Org3	1013.48	1032.38	1042.45	1009.91	952.37
Org4	3747.95	3728.60	3710.21	3689.02	3808.66

to table 5.2. This complementary experiment aims to justify that the results in table 5.2 and the analyses in chapter 5 are valid regardless of not using spare testing sets.

Table D.2 lists the point performance in terms of MAE. For each data set, effect size A12 of each SEE method against SynB-RVM_2Dhist across the 30 runs is computed as we did in table 5.2 to quantify the performance superiority/inferiority. We can see that SynB-RVM can achieve similar point performance as in table 5.2(a), indicating the validation of the experimental results and analyses regarding point prediction in chapter 5. Specially, SynB-RVM outperforms ATLM in Kitchenham and Org4 with large effect size, while is inferior to ATLM in Org3 with large effect size; SynB-RVM outperforms RVM in Kitchenham with medium effect size, while is inferior to RVM in Org3 with large effect size. These statistical results are the same with those of table 5.2(a).

D.2 Uncertain Estimate With Spare Test Set

Uncertain performance of SynB-RVM is compared to RVM, EmpRVM, and EmpATLM when spare testing data is used. EmpRVM is chosen for usually outperforming other competing uncertain methods; RVM is chosen for being the base learner of SynB-RVM.

D.2.1 Evaluation on Hit Rate

It is not suitable to report median hit rate or to conduct statistical tests across the three data sets for evaluating hit rate as we did in table 5.3. Instead, we report the average hit rate and the percentages that succeed in hit rate across 30 runs for each data set.

Table D.3: Average hit rate across 30 runs for SynB-RVM, RVM, and EmpRVM at each CL on each data set when using spare testing sets. Hit rate is measured in Eq. (5.9). The values in the parentheses are the percentages (in 100%) of the 30 runs that succeed in hit rate. Cells in yellow (light grey) highlight methods whose mean values succeed in reaching or surpassing the corresponding hit rate.

(a) Average hit rate and its passing percentage across 30 runs on Kitchenham.

CL%	SynB-RVM_SpMn	SynB-RVM_1Dhist	SynB-RVM_2Dhist	RVM	EmpRVM
10.00	26.09(100.00)	25.59(100.00)	25.06(100.00)	16.62(100.00)	9.61(36.67)
20.00	47.83(100.00)	46.98(100.00)	46.59(100.00)	36.87(100.00)	20.46(73.33)
30.00	64.30(100.00)	63.79(100.00)	63.15(100.00)	56.00(100.00)	28.74(23.33)
40.00	73.81(100.00)	73.26(100.00)	72.94(100.00)	67.91(100.00)	38.32(13.33)
50.00	79.65(100.00)	79.39(100.00)	79.14(100.00)	76.74(100.00)	48.37(16.67)
60.00	83.59(100.00)	83.22(100.00)	82.99(100.00)	81.17(100.00)	58.90(40.00)
68.27	88.74(100.00)	88.30(100.00)	87.66(100.00)	84.64(100.00)	66.14(13.33)
70.00	89.69(100.00)	89.43(100.00)	88.97(100.00)	85.40(100.00)	67.86(3.33)
80.00	92.66(100.00)	92.62(100.00)	92.50(100.00)	90.25(100.00)	77.01(0.00)
90.00	94.07(100.00)	93.86(100.00)	93.79(100.00)	93.89(100.00)	87.22(0.00)
95.45	95.89(83.33)	95.94(83.33)	95.75(76.67)	95.24(36.67)	92.23(0.00)
99.73	96.58(0.00)	96.56(0.00)	96.54(0.00)	97.49(0.00)	96.23(0.00)

(b) Average hit rate and its passing percentage across 30 runs on Org3.

CL%	SynB-RVM_SpMn	SynB-RVM_1Dhist	SynB-RVM_2Dhist	RVM	EmpRVM
10.00	48.23(100.00)	47.26(100.00)	46.98(100.00)	28.91(100.00)	9.75(36.67)
20.00	68.33(100.00)	67.51(100.00)	67.43(100.00)	49.71(100.00)	19.32(30.00)
30.00	78.87(100.00)	78.19(100.00)	77.37(100.00)	60.99(100.00)	29.63(46.67)
40.00	83.95(100.00)	83.79(100.00)	83.72(100.00)	68.23(100.00)	38.85(33.33)
50.00	87.82(100.00)	86.93(100.00)	86.91(100.00)	75.56(100.00)	48.64(36.67)
60.00	91.54(100.00)	90.99(100.00)	90.49(100.00)	81.28(100.00)	58.77(13.33)
68.27	93.37(100.00)	93.31(100.00)	93.11(100.00)	84.09(100.00)	67.18(20.00)
70.00	93.70(100.00)	93.52(100.00)	93.40(100.00)	84.86(100.00)	68.37(16.67)
80.00	95.19(100.00)	95.00(100.00)	94.98(100.00)	88.17(100.00)	78.09(6.67)
90.00	96.81(100.00)	96.73(100.00)	96.40(100.00)	92.16(100.00)	89.26(36.67)
95.45	97.82(100.00)	97.63(100.00)	97.61(100.00)	95.33(50.00)	94.47(16.67)
99.73	98.95(6.67)	98.83(3.33)	98.70(3.33)	98.46(0.00)	98.42(0.00)

(c) Average hit rate and its passing percentage across 30 runs on Org4.

CL%	SynB-RVM_SpMn	SynB-RVM_1Dhist	SynB-RVM_2Dhist	RVM	EmpRVM
10.00	9.70(33.33)	9.67(33.33)	9.70(33.33)	10.27(53.33)	9.10(23.33)
20.00	19.21(30.00)	19.21(23.33)	19.13(20.00)	21.83(80.00)	18.33(10.00)
30.00	33.91(93.33)	33.96(93.33)	33.96(96.67)	39.73(100.00)	25.96(0.00)
40.00	49.73(100.00)	49.59(100.00)	49.54(100.00)	52.02(100.00)	34.62(0.00)
50.00	61.94(100.00)	62.27(100.00)	62.05(100.00)	63.72(100.00)	43.99(3.33)
60.00	68.96(100.00)	68.50(100.00)	68.50(100.00)	70.05(100.00)	53.99(0.00)
68.27	72.65(100.00)	72.60(100.00)	72.73(100.00)	75.38(100.00)	63.61(0.00)
70.00	73.83(100.00)	73.88(100.00)	73.96(100.00)	76.42(100.00)	64.89(0.00)
80.00	79.51(30.00)	79.62(43.33)	79.54(36.67)	81.34(90.00)	74.75(0.00)
90.00	83.74(0.00)	83.58(0.00)	83.72(0.00)	85.87(0.00)	83.80(0.00)
95.45	86.89(0.00)	86.80(0.00)	87.02(0.00)	89.02(0.00)	88.42(0.00)
99.73	93.01(0.00)	92.95(0.00)	93.25(0.00)	94.89(0.00)	95.44(0.00)

Table D.3 lists the average hit rate across 30 runs for each method and each data set. The values in parentheses are the percentages of the 30 runs that succeed in reaching the desired hit rate. We can see that RVM and all the three types of SynB-RVM can almost always succeed in hit rate; while EmpRVM usually fails. The observation is consistent to the results of table 5.3, indicating the validation of the evaluation in hit rate in chapter 5.

Table D.4: Relative width of similar hit rate of the three types of SynB-RVM, RVM, and EmpRVM. The reported values are the mean of 30 runs of 10-fold CV.

(a) Similar hit rate of the investigating uncertain SEE methods in line with table 5.4.

Data Set	B_HitR	SynB-RVM_SpMn	SynB-RVM_1Dhist	SynB-RVM_2Dhist	RVM	EmpRVM
Kitchenham	0.9623	0.9589	0.9594	0.9654	0.9524	0.9623
Org3	0.9842	0.9895	0.9883	0.9870	0.9846	0.9842
Org4	0.9295	0.9301	0.9295	0.9325	0.9489	0.9544

(b) Relative width of similar hit rate of uncertain SEE methods in line with table 5.5.

Data Set	SynB-RVM_SpMn	SynB-RVM_1Dhist	SynB-RVM_2Dhist	RVM	EmpRVM
Kitchenham	3.1054	3.0470	4.1682	3.1504	2.5938
Org3	7.4060	8.0170	8.4530	7.5141	3.9485
Org4	3.2543	3.2509	3.2522	4.5481	5.9212

Particularly, the mean hit rates of EmpRVM almost always fail, showing its significant inferiority to SynB-RVM.

D.2.2 Evaluation on Relative Width

We evaluate relative width the same procedures in section 5.4.2 except that no statistical tests are conducted due to the small number of data sets.

Table D.4(a) lists the benchmark hit rates together with the chosen hit rates for the SEE methods investigated. The chosen hit rates are obviously similar as desired for a fair comparison of their relative width regardless of none statistical test. Table D.4(b) lists the relative width in line with the chosen hit rate. We can see that the relative width is similar to that of chapter 5 (see tables 5.6 and 5.5), indicating the validation of the evaluation of uncertain prediction in chapter 5.

List of References

[1] Critical values for Friedman test. https://www.york.ac.uk/depts/maths/tables/friedman.pdf. One citation in section B.2.

[2] Critical values for the Wilcoxon signed-ranks test. http://users.stat.ufl.edu/~winner/tables/wilcox_signrank.pdf. 2 citations in sections B.1 and B.1.

[3] Upper critical values of the F distribution. https://www.itl.nist.gov/div898/handbook/eda/section3/eda3673.htm. One citation in section B.2.

[4] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering (TSE)*, SE-9(6):639–648, 1983. One citation in section 2.4.1.

[5] M. Algabri, F. Saeed, H. Mathkour, and N. Tagoug. Optimization of soft cost estimation using genetic algorithm for NASA software projects. In *National Symposium on Information Technology: Towards New Smart World*, pages 1–4, Feb 2015. One citation in section 2.3.

[6] S. Aljahdali and A. F. Sheta. Software effort estimation by tuning COCOMO model parameters using differential evolution. In *ACS/IEEE International Conference on Computer Systems and Applications*, pages 1–6, May 2010. One citation in section 2.3.

[7] S. Aljahdali, A. F. Sheta, and N. C. Debnath. 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)*, pages 1–8, 2015. 5 citations in sections 1, 3.3.3, 3.4.1, A.3, and A.6.

[8] E. Alpaydin. Techniques for combining multiple learners. In *Engineering Intelligent Systems*, pages 6–12, 1998. 3 citations in sections 5.1, 5.2, and A.7.

[9] L. Angelis and I. Stamelos. A simulation tool for efficient analogy based cost estimation. *Empirical Software Engineering (ESE)*, 5(1):35–68, 2000. 6 citations in sections 2.2.1, 3.3.3, 5.3.3, 5.3.4, 5.4.2, and A.4.

[10] L. Angelis and I. Stamelos. Reply to comments by M. Jørgensen, on the Paper: 'A Simulation Tool for Efficient Analogy Based Cost Estimation'; by L. Angelis and I. Stamelos, Published in Empirical Software Engineering, 5, 35-68 (2000). *Empirical Software Engineering (ESE)*, 7(4):377–381, 2002. One citation in section 2.2.1.

[11] A. Arcuri and G. Fraser. On parameter tuning in search based software engineering. In *Symposium on Search-Based Software Engineering (SSBSE)*, pages 33–47, Szeged, Hungary, 2011. 2 citations in sections 2.3 and 3.4.1.

[12] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *International Conference on Software Engineering (ICSE)*, pages 1–10, 2011. 4 citations in sections 3.4.1, 5.4.1, B.3, and B.3.

[13] J. Armstrong. *"The Forecasting Dictionary," in J. S. Armstrong (Ed.), Principles of Forecasting: a Handbook for Researchers and Practitioners.* Kluwer, 2001. 2 citations in sections 2.4.3 and 4.2.

[14] Adelchi Azzalini. *The Skew-Normal and Related Families.* Institute of Mathematical Statistics Monographs. Cambridge University Press, 2013. 3 citations in sections 5.2.2, 5.7.2, and 7.2.4.

[15] Mohammad Azzeh, Daniel Neagu, and Peter I. Cowling. Analogy-based software effort estimation using fuzzy numbers. *Journal of Systems and Software (JSS)*, 84(2):270 – 284, 2011. One citation in section A.4.

[16] Paula B., L. Torgo, and R. Ribeiro. SMOGN: A pre-processing approach for imbalanced regression. In *First International Workshop on Learning with Imbalanced Domains: Theory and Applications*, volume 74 of *Machine Learning Research (MLR)*, pages 36–50, ECML-PKDD, Skopje, Macedonia, 2017. One citation in section 2.1.3.

[17] A. Bakir, B. Turhan, and A. Bener. A comparative study for estimating software development effort intervals. *Software Quality Journal (SQJ)*, 19(3):537–552, 2011. One citation in section 2.2.3.

[18] S. Barua, M. M. Islam, X. Yao, and K. Murase. MWMOTE–Majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Transactions on Knowledge and Data Engineering*, 26(2):405–425, 2014. One citation in section 2.1.1.

[19] B. Baskeles, B. Turhan, and A. Bener. Software effort estimation using machine learning methods. In *International symposium on Computer and Information Sciences*, pages 1–6, 2007. 2 citations in sections 1 and 4.1.

[20] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter - Special Issue on Learning from Imbalanced Datasets*, 6(1):20–29, 2004. One citation in section 2.1.1.

[21] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 1999. One citation in section A.7.

[22] Henrik Bengtsson. 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`, 2016. 2 citations in sections 3.3.3 and A.1.2.

[23] S. Bibi, I. Stamelos, and E. Angelis. Software cost prediction with predefined interval estimates. In *Software Measurement European Forum*, pages 237–246, 2004. One citation in section 2.2.3.

[24] Christopher Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995. One citation in section A.6.

[25] Christopher Bishop. *Pattern Recgonition and Machine Learning*. Springer, 2006. 4 citations in sections 5.2.2, A.2, A.2, and A.2.

[26] B. W. Boehm. Software engineering economics. *IEEE Transactions on Software Engineering (TSE)*, 10(1):4–21, 1984. 4 citations in sections (document), 1, 2.4.1, and 2.4.

[27] D. Dennis Boos and L.A. Stefanski. *Essential Statistical Inference: Theory and Methods*. Springer Texts in Statistics. Springer New York, 2013. One citation in section 3.2.1.

[28] P. Braga and A. Oliveira. Software effort estimation using machine learning techniques with robust confidence intervals. In *International Conference on Tools with Artificial Intelligence*, pages 181–185, 2007. One citation in section 2.2.2.

[29] P. Braga, A. Oliveira, G. Ribeiro, and S. Meira. Bagging predictors for estimation of software project effort. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1595–1600, 2007. 3 citations in sections 4.3.3, A.7, and B.

[30] P. Branco. Resampling approach for regression tasks under imbalanced domains. Master's thesis, Department of Computer Science, Faculty of Science, University of Porto, 2014. Master thesis. One citation in section 2.1.3.

[31] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984. One citation in section A.5.

[32] Leo Breiman. Bagging predictors. *Journal of Machine Learning*, 24(2):123–140, 1996. 2 citations in sections 4.3.3 and A.7.

[33] L. Briand, K. Emam, and F. Bomarius. Cobra: A hybrid method for software cost estimation, benchmarking, and risk assessment. In *International Conference on Software Engineering (ICSE)*, pages 390–399, 1998. One citation in section 2.2.1.

[34] L. Briand, K. Emam, D. Surmann, I. Wieczorek, and K. D. Maxwell. An assessment and comparison of common software cost estimation modeling techniques. In *International Conference on Software Engineering (ICSE)*, pages 313–323, 1999. One citation in section A.5.

[35] L. Briand, T. Langley, and I. Wieczorek. A replicated assessment and comparison of common software cost modeling techniques. In *International Conference on Software Engineering (ICSE)*, pages 377–386, 2000. 3 citations in sections A.1.1, A.4, and A.5.

[36] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: A survey and categorisation. *Information Fusion*, 6(1):5 – 20, 2005. One citation in section A.7.

[37] M. Cartwright, M. Shepperd, and Q. Song. Dealing with missing software project data. In *International Workshop on Enterprise Networking and Computing in Healthcare Industry*, pages 154–165, 2003. 3 citations in sections 4, 4, and 3.

[38] A. Chandra and X. Yao. Ensemble learning using multi-objective evolutionary algorithms. *Journal of Mathematical Modelling and Algorithms*, 5(4):417–445, 2006. One citation in section A.7.

[39] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 6:321–357, 2002. 7 citations in sections (document), 2.1.1, 2.1.1, 1, 2.1.1, 2.1.2, and 2.1.4.

[40] Zhihao Chen, Tim Menzies, Dan Port, and Barry Boehm. Feature subset selection can improve software cost estimation accuracy. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, pages 1–6, 2005. 2 citations in sections 3.2.2 and 5.6.1.

[41] Vladimir Cherkassky and Yunqian Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113 – 126, 2004. 6 citations in sections 3.3.3, 3.3.4, 3.4.1, 5.3.3, 5.3.5, and A.3.

[42] Bodin Chinthanet, Passakorn Phannachitta, Yasutaka Kamei, Pattara Leelaprute, Arnon Rungsawang, Naoyasu Ubayashi, and Kenichi Matsumoto. A review and comparison of methods for determining the best analogies in analogy-based software effort estimation. In *Annual ACM Symposium on Applied Computing*, pages 1554–1557, 2016. One citation in section A.4.

[43] Nan-Hsing Chiu and Sun-Jen Huang. The adjusted analogy-based software effort estimation based on similarity distances. *Journal of Systems and Software (JSS)*, 80(4):628 – 640, 2007. One citation in section A.1.1.

[44] Nan-Hsing Chiu and Sun-Jen Huang. The adjusted analogy-based software effort estimation based on similarity distances. *Journal of Systems and Software (JSS)*, 80(4):628 – 640, 2007. One citation in section A.4.

[45] Sunita Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transactions on Software Engineering (TSE)*, 25(4):573–583, 1999. One citation in section 2.2.4.

[46] R. Coe. It's the effect size, stupid: What effect size is and why it is important. http://www.cem.org/attachments/ebe/ESguide.pdf, 2002. 3 citations in sections B, B.3, and B.3.

[47] Vachik S. Dave and Kamlesh Dutta. Neural network based models for software effort estimation: A review. *Artificial Intelligence Review*, 42(2):295–307, 2014. One citation in section A.6.

[48] I. F. de Barcelos Tronto, J. D. S. da Silva, and N. Sant'Anna. Comparison of artificial neural network and regression models in software effort estimation. In *International Joint Conference on Neural Networks (IJCNN)*, pages 771–776, 2007. 3 citations in sections 6.3.1, A.1.1, and A.6.

[49] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens. Data mining techniques for software effort estimation: A comparative study. *IEEE Transactions on Software Engineering (TSE)*, 38(2):375–397, 2012. 8 citations in sections 1, 1.1.1, 1.1.2, 2, 2.3, 6.3.1, A.1.1, and A.6.

[50] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research (JMLR)*, 7:1–30, 2006. 12 citations in sections (document), 3.4.1, 5.4.1, 5.5.3, B, B.1, B.1, B.2, B.2, B.2, B.3, and B.2.

[51] Kitchenham doi. https://doi.org/10.5281/zenodo.268457, 2017. One citation in section 2.4.1.

[52] Maxwell doi. https://doi.org/10.5281/zenodo.268461, 2009. One citation in section 2.4.1.

[53] Nasa93 doi. https://doi.org/10.5281/zenodo.268419, 2008. One citation in section 2.4.1.

[54] D. Drown, T. Khoshgoftaar, and N. Seliya. Evolutionary sampling and software quality modeling of high-assurance systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(5):1097–1107, 2009. One citation in section 2.1.2.

[55] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In *Neural Information Processing Systems (NIPS)*, pages 155–161, 1996. 6 citations in sections (document), 3.3.3, 5.3.3, A.3, A.2, and A.3.

[56] H. Drucker, C. Cortes, D. Jackel, Y. LeCun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994. One citation in section A.7.

[57] Mahmoud O. Elish. Assessment of voting ensemble for estimating software development effort. In *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 316–321, 2013. One citation in section A.7.

[58] Andrew Estabrooks. A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence*, pages 18–36, 2004. One citation in section 2.1.1.

[59] A. Faul and M. Tipping. Analysis of sparse Bayesian learning. In *Advances in Neural Information Processing Systems 14*, pages 383–389. MIT Press, 2001. 4 citations in sections 3.3.3, A.2, A.2, and A.2.

[60] E. C. Fieller and E. S. Pearson. Tests for rank correlation coefficients: I. *Biometrika*, 49(1-2):185–191, 1962. One citation in section 5.5.4.

[61] G.R. Finnie, G.E. Wittig, and J-M. Desharnais. A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software (JSS)*, 39(3):281 – 289, 1997. One citation in section A.6.

[62] Gary William Flake and Steve Lawrence. Efficient SVM regression training with SMO. *Machine Learning*, 46(1):271–290, 2002. One citation in section A.3.

[63] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion MMRE. *IEEE Transactions on Software Engineering*

(TSE), 29:985–995, 2003. 7 citations in sections 2.4.2, 2.4.2, 2.4.2, 2.4.2, 2.4.2, 4.3.2, and 6.1.

[64] J. Cuadrado Gallego, D. Rodriguez, M. Sicilia, M. Rubio, and A. Cresp. Software project effort estimation based on multiple parametric models generated through data clustering. *Journal of Computer Science and Technology*, 22(3), 2007. One citation in section 3.4.2.

[65] Farhad Soleimanian Gharehchopogh and Amir Pourali. A new approach based on continuous genetic algorithm in software cost estimation. *Journal of Scientific Research and Development*, 2(4):87–94, 2015. One citation in section 2.3.

[66] Katarina Grolinger, Besa Muslimi, Miriam Capretz, and Mark Benko. Eef-cas: An effort estimation framework with customizable attribute selection. *Advancements in Computing Technology*, 5(13):14–33, 2013. One citation in section 1.

[67] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009. 2 citations in sections 4.3.3 and A.5.

[68] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *Advances in Intelligent Computing*, pages 878–887, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. One citation in section 2.1.1.

[69] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001. 6 citations in sections 1.1.1, 3.3.2, 4.3.2, 5.3.2, A.4, and A.5.

[70] Haibo He, Yang Bai, E. A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, June 2008. 2 citations in sections 2.1.1 and 2.1.1.

[71] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Transaction on Knowledge and Data Engineering*, 21(9):1263–1284, 2009. One citation in section 2.1.1.

[72] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979. One citation in section B.2.

[73] Mohamed Hosni, Ali Idri, Alain Abran, and Ali Bou Nassif. On the value of parameter tuning in heterogeneous ensembles effort estimation. *Soft Computing*, 2017. One citation in section 2.3.

[74] Robert T. Hughes. Expert judgement as an estimating method. *Information and Software Technology (IST)*, 38(2):67 – 75, 1996. One citation in section 1.

[75] Ali Idri, Fatima A. Amazal, and Alain Abran. Analogy-based software development effort estimation: A systematic mapping and review. *Information and Software Technology (IST)*, 58:206 – 230, 2015. One citation in section A.4.

[76] Ali Idri, Mohamed Hosni, and Alain Abran. Systematic literature review of ensemble effort estimation. *Journal of Systems and Software (JSS)*, 118:151 – 175, 2016. 3 citations in sections 5.3.3, A.6, and A.7.

[77] ISBSG. The international software benchmarking standards group. `http://www.isbsg.org`, 2011. 5 citations in sections 2.4.1, 2.4.1, 3.3.1, 4.3.1, and 2.

[78] R. Jeffery, M. Ruhe, and I. Wieczorek. Using public domain metrics to estimate software development effort. In *International Software Metrics Symposium*, pages 16–27, 2001. 2 citations in sections A.4 and A.5.

[79] D. N. Joanes and C. A. Gill. Comparing measures of sample skewness and kurtosis. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 47(1):183–189, 1998. One citation in section A.1.2.

[80] N. L. Johnson and S. Kotz. Distributions in Statistics: Continuous Univariate Distributions. Vol. 1,2. Boston etc.: Houghton Mifflin Company. I: XIV, 300 p.; II: XIII, 306, 1970. 2 citations in sections 5.7.2 and 7.2.4.

[81] M. Jørgensen. Comments on 'A Simulation Tool for Efficient Analogy Based Cost Estimation', by L. Angelis and I. Stamelos, Published in Empirical Software Engineering, 5, 35-68 (2000). *Empirical Software Engineering (ESE)*, 7(4):375–376, 2002. One citation in section 2.2.1.

[82] M. Jorgensen. Realism in assessment of effort estimation uncertainty: It matters how you ask. *IEEE Transactions on Software Engineering (TSE)*, 30(4):209–217, 2004. One citation in section 4.1.

[83] M. Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software (JSS)*, 70(1):37 – 60, 2004. 2 citations in sections 1 and 6.1.

[84] M. Jørgensen. Evidence-based guidelines for assessment of software development cost uncertainty. *IEEE Transactions on Software Engineering (TSE)*, 32(11):942–954, 2005. 6 citations in sections 1.1.2, 2.4.3, 3.4.2, 4.1, 4.4.2, and 5.4.2.

[85] M. Jørgensen. Forecasting of software development work effort: Evidence on expert judgement and formal models. *International Journal of Forecasting*, 23(3):449 – 462, 2007. One citation in section 1.

[86] M. Jørgensen. The influence of selection bias on effort overruns in software development projects. *Information and Software Technology (IST)*, 55(9):1640–1650, 2013. One citation in section 2.4.2.

[87] M. Jørgensen, U. Indahl, and D. Sjøberg. Software effort estimation by analogy and "regression toward the mean". *Journal of Systems and Software (JSS)*, 68(3):253–262, 2003. 3 citations in sections 3.3.3, 5.3.3, and A.4.

[88] M. Jørgensen and M. Kjetil. How large are software cost overruns? a review of the 1994 CHAOS report. *Information and Software Technology (IST)*, 48(4):297–301, 2006. One citation in section 2.4.2.

[89] M. Jørgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering (TSE)*, 33(1):33–53, 2007. 4 citations in sections 1, 1.1.4, A.6, and B.

[90] M. Jørgensen and K. Sjøberg. An effort prediction interval approach based on the empirical distribution of previous estimation accuracy. *Information and Software Technology (IST)*, 45(3):123 – 136, 2003. 2 citations in sections 2.2.2 and 5.3.4.

[91] M. Jørgensen, K. H. Teigen, and K. Molokken. Better sure than safe? Overconfidence in judgement based software development effort prediction intervals. *Journal*

of Systems and Software (JSS), 70:79–93, 2004. 5 citations in sections 1.1.2, 2.4.3, 3.4.2, 4.1, and 5.4.2.

[92] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. Matsumoto. The effects of over and under sampling on fault-prone module detection. In *International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 196–204, 2007. 2 citations in sections 2.1.1 and 2.1.2.

[93] Yasutaka Kamei, Jacky Wai Keung, Akito Monden, and Ken-ichi Matsumoto. An oversampling method for analogy-based software effort estimation. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 312–314, 2008. 8 citations in sections 2.1.4, 2.5, 3.2, 3.4, 3.4.3, 3.4.3, 3.5, and 7.1.1.

[94] J. W. Keung, B. A. Kitchenham, and D. R. Jeffery. Analogy-x: Providing statistical inference to analogy-based software cost estimation. *IEEE Transactions on Software Engineering (TSE)*, 34(4):471–484, July 2008. One citation in section A.4.

[95] V. Khatibi Bardsiri, D. N. Jawawi, S. Z. Hashim, and E. Khatibi. A flexible method to estimate the software development effort based on the classification of projects and localization of comparisons. *Empirical Software Engineering (ESE)*, 19(4):857–884, 2014. One citation in section A.1.2.

[96] Thanh Tung Khuat and My Hanh Le. Optimizing parameters of software effort estimation models using directed artificial bee colony algorithm. *Informatica*, 40:427–436, 2016. One citation in section 2.3.

[97] B. Kitchenham and S. Linkman. Estimates, uncertainty, and risk. *IEEE Software*, 14(3):69–74, 1997. 2 citations in sections 1.1.3 and 4.1.

[98] B. Kitchenham, E. Mendes, and G. Travassos. Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering (TSE)*, 33(5):316 – 329, 2007. One citation in section 6.1.

[99] B. Kitchenham and Emilia Mendes. Why comparative effort prediction studies may be invalid. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, pages 4:1–4:5, 2009. 7 citations in sections 1, 3.3.3, 5.4.1, 5.6.1, A.1, A.1.1, and A.1.2.

[100] B. Kitchenham, S. L. Pfleeger, B. McColl, and S. Eagan. An empirical study of maintenance and development estimation accuracy. *Journal of Systems and Software (JSS)*, 64(1):57–77, 2002. 2 citations in sections 2.4.1 and 1.

[101] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd. What accuracy statistics really measure. *IEE Proceedings - Software Engineering*, 148(3):81–85, 2001. One citation in section 2.4.2.

[102] B. Kitchenham, L. M. Pickard, S. Linkman, and P. W. Jones. Modeling software bidding risks. *IEEE Transactions on Software Engineering (TSE)*, 29(6):542–554, 2003. 2 citations in sections 1 and 4.1.

[103] M. Klas, A. Trendowicz, A. Wickenkamp, J. Munch, N. Kikuchi, and Y. Ishigai. The use of simulation techniques for hybrid software cost estimation and risk analysis. In *Advances in computers*, volume 74 of *Software Development*, pages 115–174. Academic Press, 2008. 3 citations in sections 1.1.3, 4.1, and 7.1.2.

[104] M. Klas, A. Trendowizc, Y. Ishigai, and H. Nakao. Handling estimation uncertainty with bootstrapping: Empirical evaluation in the context of hybrid prediction methods. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 245–254, 2011. 6 citations in sections 2.2.1, 2.4.3, 4.1, 4.4.2, 5.3.4, and 5.4.2.

[105] E. Kocaguneli, A. Bener, and Y. Kultur. Combining multiple learners induced on multiple datasets for software effort prediction. In *International Symposium on Software Reliability Engineering*, Mysuru, India, 2009. One citation in section B.

[106] E. Kocaguneli, B. Cukic, and H. Lu. Predicting more from less: Synergies of learning. In *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, San Francisco, CA, USA, 2013. 3 citations in sections 1.1.1, 3.1, and 7.1.1.

[107] E. Kocaguneli, B. Cukic, T. Menzies, and H. Lu. Building a second opinion: Learning cross-company data. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, Baltimore, USA, 2013. 3 citations in sections 1.1.1, 3.1, and 7.1.1.

[108] E. Kocaguneli and T. Menzies. Software effort models should be assessed via leave-one-out validation. *Journal of Systems Software*, 86(7):1879–1890, July 2013. One citation in section 3.3.2.

[109] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering (TSE)*, 38(2):425–438, 2012. 6 citations in sections 1.1.1, 3.1, 3.3.2, A.4, A.5, and D.

[110] E Kocaguneli, T. Menzies, and J. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering (TSE)*, 38:1403âĂŞ1416, 2012. 3 citations in sections 1, A.5, and A.7.

[111] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, and R. Madachy. Active learning and effort estimation: Finding the essential content of software effort estimation data. *IEEE Transactions on Software Engineering (TSE)*, 39(8):1040–1053, 2013. 6 citations in sections 3.3.2, 3.3.3, A.4, A.4, A.5, and D.

[112] E. Kocaguneli, T. Menzies, and J. W. Keung. Kernel methods for software effort estimation. *Empirical Software Engineering (ESE)*, 18:1–24, 2013. One citation in section 2.3.

[113] E. Kocaguneli, T. Menzies, and E. Mendes. Transfer learning in effort estimation. *Empirical Software Engineering (ESE)*, 20(3):813–843, 2015. One citation in section A.4.

[114] E. Kocaguneli and T.J. Menzies. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering (TSE)*, 38(2):425–438, 2012. 4 citations in sections 3.3.3, 5.3.3, A.4, and A.5.

[115] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1137–1143, 1995. One citation in section 3.3.2.

[116] Yigit Kultur, Burak Turhan, and Ayse Basar Bener. Enna: Software effort estimation using ensemble of neural networks with associative memory. In *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 330–338, 2008. 4 citations in sections 1.1.1, 3.1, 7.1.1, and A.7.

[117] K. Langsari and R. Sarno. Optimizing COCOMO II parameters using particle swarm method. In *International Conference on Science in Information Technology*, pages 29–34, Oct 2017. One citation in section 2.3.

[118] Safae Laqrichi, Francois Marmier, Didier Gourc, and Jean Nevoux. Integrating uncertainty in software effort estimation using bootstrap based neural networks. *IFAC Symposium on Information Control Problems in Manufacturing*, 48(3):954–959, 2015. 2 citations in sections 2.2.1 and 5.3.4.

[119] Sauchi S. Lee. Regularization in skewed binary classification. *Computational Statistics*, 14(2):277–292, 1999. 2 citations in sections 2.1.1 and A.2.

[120] Sauchi S. Lee. Noisy replication in skewed binary classification. *Computational Statistics and Data Analysis*, 34(2):165–191, 2000. One citation in section 2.1.1.

[121] N. Leech and A. Onwuegbuzie. A call for greater use of non-parametric statistics. Technical report, US Department Education, 2002. 2 citations in sections B.3 and B.3.

[122] Y. Li, M. Xie, and T. Goh. A study of project selection and feature weighting for analogy based software cost estimation. *Journal of Systems and Software (JSS)*, 82(2):241 – 252, 2009. 5 citations in sections 3.2.2, 3.3.3, 5.3.3, 5.6.1, and A.4.

[123] Y. Li, M. Xie, and T. Goh. A study of the non-linear adjustment for analogy based software cost estimation. *Empirical Software Engineering (ESE)*, 14(6):603–643, 2009. One citation in section A.4.

[124] Stephen G. MacDonell and Martin Shepperd. Combining techniques to optimize effort predictions in software project management. *Journal of Systems and Software (JSS)*, 66(2):91 – 98, 2003. One citation in section A.1.1.

[125] C. Mair and M. Shepperd. The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In *International Symposium on Empirical Software Engineering*, pages 491 – 500, 2005. 3 citations in sections 3.4.1, 6.1, and A.4.

[126] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947. One citation in section B.3.

[127] K. Maxwell. *Applied Statistics for Software Managers*. Prentice Hall PTR, 2002. One citation in section 2.4.1.

[128] E. Mendes and N. Mosley. Bayesian network models for web effort prediction: A comparative study. *IEEE Transactions on Software Engineering (TSE)*, 34(6):723–737, 2008. One citation in section 2.2.4.

[129] Solomon Mensah, Jacky Keung, Michael Bosu, and Kwabena Bennin. Duplex output software effort estimation model with self-guided interpretation. *Information and Software Technology*, 94:1 – 13, 2018. One citation in section 2.2.3.

[130] Krishna R. Pryor D. Menzies, T. The SEACRAFT repository of empirical software engineering data. `https://zenodo.org/communities/seacraft`, 2017. 3 citations in sections 2.4.1, 3.3.1, and 4.3.1.

[131] T. Menzies, Z. Chen, J. Hihn, and K. Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering (TSE)*, 32(11):883–895, 2006. One citation in section 1.

[132] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering (TSE)*, 33(1):2–13, 2007. One citation in section 3.3.1.

[133] T. Menzies, R. Krishna, and D. Pryor. The PROMISE repository of empirical software engineering data. `http://openscience.us/repo.` North Carolina State University, Department of Computer Science, 2015. One citation in section 1.

[134] T. Menzies and M. Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering (ESE)*, 17:1–17, 2012. 2 citations in sections 2.3 and 6.1.

[135] L. Leandro Minku and Xin Yao. Software effort estimation as a multi-objective learning problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22, 2013. 6 citations in sections 2.4.2, 5.4.1, 6.1, A.1.2, A.5, and D.

[136] Leandro L. Minku and X. Yao. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology (IST)*, 55(8):1512–1528, 2012. 23 citations in sections 1, 1.1.1, 1.1.4, 2.3, 2.4.1, 2.4.1, 3.1, 3.3.3, 3.4.2, 4.3.3, 5.3.3, 5.6.1, 6.1, 6.2.1, 6.2.2, 6.2.3, 6.3.1, A.2, A.5, A.6, A.7, B, and D.

[137] Leandro L. Minku and Xin Yao. A principled evaluation of ensembles of learning machines for software effort estimation. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, pages 9:1–9:10, 2011. One citation in section A.7.

[138] Leandro L. Minku and Xin Yao. Can cross-company data improve performance in software effort estimation? In *International Conference on Predictive Models in Software Engineering (PROMISE)*, pages 69–78. ACM, 2012. 2 citations in sections 4 and 6.2.1.

[139] Leandro L. Minku and Xin Yao. How to make best use of cross-company data in software effort estimation? In *International Conference on Software Engineering (ICSE)*, pages 446–456, New York, NY, USA, 2014. One citation in section A.5.

[140] Michinari Momma and Kristin Bennett. A pattern search method for model selection of support vector regression. In *International Conference on Data Mining*, pages 261–274, 2002. 2 citations in sections 3.3.4 and 5.3.5.

[141] I. Myrtveit and E. Stensrud. A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Transactions on Software Engineering (TSE)*, 25(4):510–525, July 1999. One citation in section A.4.

[142] I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering (TSE)*, 31(5):380–391, 2005. 2 citations in sections 2.4.2 and 2.4.2.

[143] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer, 1996. One citation in section A.2.

[144] J. Neter, M. Kutner, C. Nachtsheim, and W. Wasserman. *Applied Linear Statistical Models*. McGraw-Hill, 1996. One citation in section A.1.1.

[145] NIST. Engineering statistics, NIST/SEMATECH e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/, 2012. 4 citations in sections 2.2.2, 5.7.2, 7.2.4, and A.2.

[146] Adriano Oliveira. Estimation of software project effort with support vector regression. *Neurocomputing*, 69(13):1749 – 1753, 2006. 5 citations in sections 3.3.3, 3.4.1, 5.3.3, A.3, and A.3.

[147] Adriano Oliveira, Petronio Braga, Ricardo Lima, and Marcio Cornelio. GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *Information and Software Technology (IST)*, 52(11):1155 – 1166, 2010. Special Section on Best Papers PROMISE 2009. One citation in section 2.3.

[148] L. Pelayo and S. Dick. Applying novel resampling strategies to software defect prediction. In *Annual Meeting of the North American Fuzzy Information Processing Society*, pages 69–72, 2007. One citation in section 2.1.2.

[149] C. Pendharkar, H. Subramanian, and A. Rodger. A probabilistic model for predicting software development effort. *IEEE Transactions on Software Engineering (TSE)*, 31(7):615 – 624, 2005. One citation in section 2.2.4.

[150] S. Poulding and J. A. Clark. Efficient software verification: Statistical testing using automated search. *IEEE Transactions on Software Engineering (TSE)*, 36(6):763–777, 2010. One citation in section B.3.

[151] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. One citation in section A.5.

[152] Prasad Reddy and CH. Hari. Software effort estimation using particle swarm optimization with inertia weight. *International Journal of Software Engineering*, 2(4):87–96, 2011. One citation in section 2.3.

[153] R. P. Ribeiro. *Utility-based Regression*. PhD thesis, Department of Computer Science, Faculty of Science, University of Porto, 2011. PhD thesis. One citation in section 2.1.3.

[154] Raul Rojas. *Neural Networks - A Systematic Introduction*. Springer-Verlag, Berlin, New-York, 1996. One citation in section A.6.

[155] Lior Rokach and Oded Maimon. *Clustering Methods*, pages 321–352. Springer US, Boston, MA, 2005. One citation in section 5.6.1.

[156] R. Rosenthal. *The Handbook of Research Synthesis*. Russell Sage Foundation, Sage, New York, 1994. One citation in section B.

[157] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987. One citation in section 5.6.1.

[158] Rohit Kumar Sachan, Ayush Nigam, Avinash Singh, Sharad Singh, Manjeet Choudhary, Avinash Tiwari, and Dharmender Singh Kushwaha. Optimizing basic COCOMO model using simplified genetic algorithm. *Procedia Computer Science*, 89:492 – 498, 2016. One citation in section 2.3.

[159] Federica Sarro, Alessio Petrozziello, and Mark Harman. Multi-objective software effort estimation. In *International Conference on Software Engineering (ICSE)*, pages 619–630, 2016. One citation in section 2.2.5.

[160] Shashank Mouli Satapathy and Santanu Kumar Rath. Use case point approach based software effort estimation using various support vector regression kernel methods. *CoRR*, abs/1401.3069, 2014. 4 citations in sections 3.3.3, 3.4.1, 5.3.3, and A.3.

[161] P. Sentas, L. Angelis, and I. Stamelos. Multinomial logistic regression applied on software productivity prediction. In *Panhellenic Conference in Information*, 2003. One citation in section 2.2.3.

[162] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris. Software productivity and effort prediction with ordinal regression. *Information and Software Technology (IST)*, 47(1):17–29, 2005. 4 citations in sections 1, 2.2.3, 2.4.1, and 1.

[163] Yeong-Seok Seo, Kyung-A Yoon, and Doo-Hwan Bae. An empirical analysis of software effort estimation with outlier elimination. In *International Conference on Predictive Models in Software Engineering (PROMISE)*, pages 25–32, 2008. One citation in section 5.6.1.

[164] M. Shepperd and S. McDonell. Evaluating prediction systems in software project estimation. *Information and Software Technology (IST)*, 54:820–827, 2012. 11 citations in sections 2.4.2, 2.4.2, 2.4.2, 3.3.2, 4.3.2, 6.1, 6.2.4, 7.1.3, B, B.3, and B.3.

[165] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering (TSE)*, 23(12):736–743, 1997. 7 citations in sections 1.1.4, 2.4.2, 2.4.2, 3.3.3, 4.3.3, 5.3.3, and A.4.

[166] Alaa F. Sheta. Estimation of the COCOMO model parameters using genetic algorithms for NASA software projects. *Journal of Computer Science*, 2(2):118–123, 2006. One citation in section 2.3.

[167] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004. One citation in section A.3.

[168] Patricia Snyder and Stephen Lawson. Evaluating results using corrected and uncorrected effect size estimates. *The Journal of Experimental Education*, 61(4):334–349, 1993. 2 citations in sections B.3 and B.3.

[169] L. Song, L. Minku, and X. Yao. The impact of parameter tuning on software effort estimation using learning machines. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, pages 9:1–9:10, Baltimore, USA, 2013. 8 citations in sections 1.1.1, 2.3, 3.3.2, 3.3.3, 3.4.1, 4.3.3, 5.3.3, and 0.

[170] L. Song, L. Minku, and X. Yao. The potential benefit of relevance vector machine to software effort estimation. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, pages 52–61, Turin, Italy, 2014. 8 citations in sections 3.3.2, 3.3.3, 3.4.2, 0, 5.3.4, 5.4.2, 6.1, and 6.2.1.

[171] L. Song, L. Minku, and X. Yao. A novel automated approach for software effort estimation based on data augmentation. In *ACM Symposium on the Foundations of Software Engineering (FSE)*, Lake Buena Vista, Florida, USA, 2018. 3 citations in sections 0, 6.1, and 6.2.1.

[172] L. Song, L. Minku, and X. Yao. Software effort interval prediction via Bayesian inference and synthetic Bootstrap resampling. Submitted to ACM Transactions on Software Engineering and Methodology (TOSEM'18), 2018. One citation in section 0.

[173] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. A general software defect-proneness prediction framework. *IEEE Transaction on Software Engineering (TSE)*, 37(3):356–370, 2011. One citation in section 3.3.1.

[174] I. Stamelos and L. Angelis. Managing uncertainty in project portfolio cost estimation. *Information and Software Technology*, 43:759–768, 2001. 2 citations in sections 1.1.2 and 4.1.

[175] I. Stamelos, L. Angelis, P. Dimou, and E. Sakellaris. On the use of Bayesian belief network for prediction of software productivity. *Information and Software Technology (IST)*, 45(1):51–60, 2003. One citation in section 2.2.4.

[176] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *International Conference on Software Engineering (ICSE)*, pages 321–332, May 2016. One citation in section 2.3.

[177] M. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning*, 1:211–244, 2001. 6 citations in sections 3.3.3, 5.2.1, 5.6.3, A.2, A.2, and A.2.

[178] Luis Torgo, Paula Branco, Rita P. Ribeiro, and Bernhard Pfahringer. Resampling strategies for regression. *Expert Systems*, 32(3):465–476, 2015. One citation in section 2.1.3.

[179] Luis Torgo, Rita P. Ribeiro, Bernhard Pfahringer, and Paula Branco. SMOTE for regression. In *Progress in Artificial Intelligence*, pages 378–389, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. One citation in section 2.1.3.

[180] V. Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982. One citation in section A.2.

[181] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. 2 citations in sections A.2 and A.3.

[182] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998. 2 citations in sections 1.1.1 and A.2.

[183] Andras Vargha and Harold D. Delaney. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 6 2000. 6 citations in sections 3.4.1, 5.4.1, 6.2.4, B.3, B.3, and B.3.

[184] George Wadsworth and Joseph Bryan. *Introduction to Probability and Random Variables*. A Wiley publication in mathematical statistics. McGraw-Hill, 1960. One citation in section 3.2.1.

[185] Shuo Wang and Xin Yao. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, 62:434–443, 2013. One citation in section 2.1.2.

[186] Gary M. Weiss. Mining with rarity: A unifying framework. *SIGKDD Explor. Newsl.*, 6(1):7–19, 2004. 2 citations in sections 2.1.1 and 2.1.2.

[187] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology (IST)*, 54(1):41–59, 2012. 10 citations in sections 1, 1.1.2, 1.1.4, 2, 3.3.3, 4.1, 5.3.3, 6.1, 6.2.1, and A.5.

[188] Peter A. Whigham, Caitlin A. Owen, and Stephen G. Macdonell. A baseline model for software effort estimation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(3):20:1–20:11, 2015. 6 citations in sections 3.3.3, 5.3.3, 5.3.4, 5.4.1, A.1, and A.1.2.

[189] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. 2 citations in sections 3.4.1 and 5.5.3.

[190] Gerhard Wittig and Gavin Finnie. Estimating software development effort with connectionist models. *Information and Software Technology (IST)*, 39:469 – 476, 1997. One citation in section A.6.

[191] B. Wlodzimierz. *The Normal Distribution: Characterizations with Applications*. Springer-Verlag, 1995. 3 citations in sections 4.2, 5.2.2, and 5.2.2.

[192] H. Zhao and Sudha Ram. Constrained cascade generalization of decision trees. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):727–739, 2004. One citation in section A.7.

[193] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 1st edition, 2012. 3 citations in sections 5.1, 5.2, and A.7.