

UNIVERSIDAD DE COSTA RICA  
SISTEMA DE ESTUDIOS DE POSGRADO

EVALUATING AN AUTOMATED PROCEDURE OF MACHINE  
LEARNING PARAMETER TUNING FOR SOFTWARE EFFORT  
ESTIMATION

Tesis sometido a la consideración de la Comisión del  
Programa de Estudios de Posgrado en Computación  
e Informática para optar al grado y título de  
Maestría Académica en Computación e Informática

LEONARDO VILLALOBOS ARIAS

Ciudad Universitaria Rodrigo Facio, Costa Rica

2021

# Dedicatoria

A mi madre, quien a través de todos estos años me ha brindado su amor y apoyo incondicional, y me ha motivado a buscar la excelencia.

A mi padre, quien me ha enseñado sobre las cosas más importantes de la vida, y siempre ha estado ahí para escuchar mis problemas y darme consejo en los tiempos más difíciles.

A mi hermano menor, quien ha sido mi compañero en los buenos y los malos momentos, y quien ha trabajado como la voz de la razón que muchas veces me hace falta.

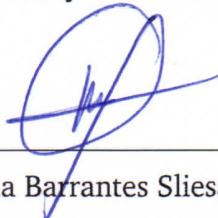
# Agradecimientos

Un agradecimiento sincero a todos los que han contribuido, directa o indirectamente, a este trabajo.

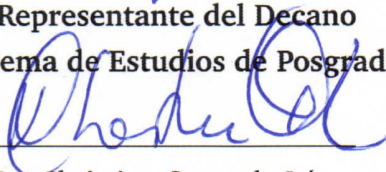
Agradezco de manera especial a mi tutor, el Dr. Christian Quesada López, por su increíble dedicación a este proyecto y su guía en el mundo académico. Deseo agradecer también a mis asesores de tesis, el Dr. José Guevara Coto, la Dra. Alexandra Martínez Porras y el Dr. Marcelo Jenkins Coronas, por su ayuda durante la elaboración de esta investigación.

Esta investigación fue desarrollada en el marco del proyecto de investigación No. 834-B8-A27, “*Evaluación empírica de una metodología para la automatización de la medición del tamaño funcional del software*”, que fue apoyado conjuntamente por el *Centro de Investigaciones en Tecnologías de la Información y Comunicación (CITIC)* y la *Escuela de Ciencias de la Computación e Informática (ECCI)* y de la Universidad de Costa Rica (UCR).

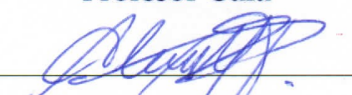
Esta tesis fue aceptada por la Comisión del Programa de Estudios de Posgrado en Computación e Informática de la Universidad de Costa Rica, como requisito parcial para optar al grado y título de Maestría Académica en Computación e Informática.



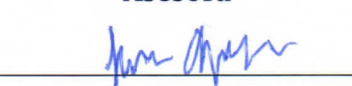
Dra. Gabriela Barrantes Sliesarieva  
**Representante del Decano  
Sistema de Estudios de Posgrado**



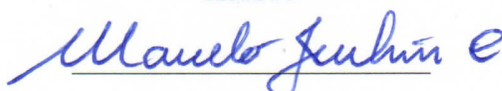
Dr. Christian Quesada López  
**Profesor Guía**



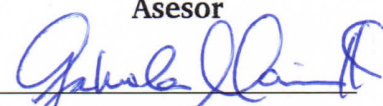
Dra. Alexandra Martínez Porras  
**Asesora**



Dr. José Guevara Coto  
**Asesor**



Dr. Marcelo Jenkins Coronas  
**Asesor**



Dra. Gabriela Marín Raventós  
**Directora del Programa  
Posgrado en Computación e Informática**



Leonardo Villalobos Arias  
**Sustentante**



# Table of contents

Dedicatoria . . . . .	ii
Agradecimientos . . . . .	iii
Hoja de aprobación . . . . .	iv
Table of contents . . . . .	ix
Resumen . . . . .	x
Abstract . . . . .	xi
List of tables . . . . .	xiii
List of figures . . . . .	xv
List of acronyms . . . . .	xvi
<b>1 Introduction</b>	<b>1</b>
1.1 Objectives . . . . .	4
1.2 Methodology . . . . .	5
1.3 Contributions and products . . . . .	6
1.4 Document structure . . . . .	7
<b>2 Background</b>	<b>9</b>
2.1 Data science in software engineering . . . . .	9
2.2 Search based software engineering . . . . .	10
2.3 Software effort estimation . . . . .	12
2.3.1 SEE process . . . . .	13
2.3.2 SEE datasets . . . . .	14
2.4 SEE evaluation frameworks . . . . .	17
2.4.1 Baseline SEE frameworks . . . . .	21
2.5 Machine learning . . . . .	23

2.5.1	Data transformation techniques . . . . .	27
2.5.2	Feature selection techniques . . . . .	29
2.5.3	Clustering techniques . . . . .	32
2.5.4	Validation approaches . . . . .	33
2.5.5	Machine learning algorithms . . . . .	35
2.5.6	Hyper-parameter tuning approaches . . . . .	38
2.6	Empirical methodologies in software engineering . . . . .	44
2.6.1	Design science methodology for information systems and software engineering . . . . .	44
2.6.2	Empirical software engineering . . . . .	46
2.6.3	Systematic mapping studies . . . . .	47
2.6.4	Controlled experiments . . . . .	49
2.7	Software engineering methodologies . . . . .	52
<b>3</b>	<b>Hyper-parameter tuning for machine learning software effort estimation: a systematic literature mapping</b>	<b>54</b>
3.1	Study design . . . . .	54
3.1.1	Research questions . . . . .	55
3.1.2	Control studies . . . . .	55
3.1.3	Search strategy . . . . .	55
3.1.4	Inclusion and exclusion criteria . . . . .	56
3.1.5	Selection process . . . . .	57
3.1.6	Quality assessment . . . . .	57
3.1.7	Data extraction and analysis . . . . .	58
3.2	Results . . . . .	60
3.2.1	RQ1: Hyper-parameter tuning approaches used in machine learning SEE . . . . .	61
3.2.2	RQ2: Datasets used in hyper-parameter tuning machine learning SEE . . . . .	75
3.2.3	RQ3: Performance metrics of hyper-parameter tuning approaches used in machine learning SEE . . . . .	80
3.2.4	Discussion . . . . .	87

3.2.5	Conclusions of the mapping study . . . . .	96
<b>4</b>	<b>ChimeraHPT: an automated machine learning hyper-parameter tuning framework</b>	<b>98</b>
4.1	Pre-processing . . . . .	99
4.2	Model training and evaluation . . . . .	102
4.2.1	Framework configuration . . . . .	102
4.2.2	Evaluation loop . . . . .	106
4.2.3	Machine learning techniques . . . . .	110
4.3	Statistical analysis . . . . .	113
4.3.1	Model evaluation . . . . .	115
4.3.2	Model verification . . . . .	116
4.4	Summary . . . . .	116
<b>5</b>	<b>Evaluation of the effectiveness of the automated hyper-parameter tuning procedure: a series of quasi experiments</b>	<b>118</b>
5.1	Quasi experiment 1: Evaluation of grid and random search for support vector regression . . . . .	119
5.1.1	Study summary . . . . .	119
5.1.2	Main results . . . . .	120
5.2	Quasi experiment 2: Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation . . . . .	121
5.2.1	Study summary . . . . .	121
5.2.2	Main results . . . . .	122
5.3	Quasi experiment 3: Hyper-parameter Tuning using Genetic Algorithms for Software Effort Estimation . . . . .	123
5.3.1	Study summary . . . . .	123
5.3.2	Main results . . . . .	123
5.4	Quasi experiment 4: Multi-objective Hyper-parameter Tuning for Software Effort Estimation . . . . .	124
5.4.1	Study summary . . . . .	124
5.4.2	Main results . . . . .	125

5.5	Quasi experiment 5: Comparative study of Random Search Hyper-Parameter Tuning for Software Effort Estimation . . . . .	126
5.5.1	Study summary . . . . .	126
5.5.2	Main results . . . . .	127
5.6	General findings and discussion . . . . .	128
<b>6</b>	<b>Conclusion</b>	<b>131</b>
6.1	Summary of results . . . . .	131
6.1.1	SO1: Characterization of hyper-parameter tuning approaches for machine learning . . . . .	131
6.1.2	SO2: Automation of a hyper-parameter tuning procedure for machine learning . . . . .	133
6.1.3	SO3: Evaluation of the effectiveness of the automated hyper-parameter tuning procedure for machine learning . . . . .	135
6.2	Contributions . . . . .	139
6.3	Future work . . . . .	139
<b>A</b>	<b>Hyper-parameter tuning techniques in SEE</b>	<b>142</b>
<b>B</b>	<b>ChimeraHPT list of technique and parameters</b>	<b>157</b>
<b>C</b>	<b>Paper 1: Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: un mapeo de literatura</b>	<b>163</b>
<b>D</b>	<b>Paper 2: Evaluation of Grid and Random Search for Support Vector Regression</b>	<b>180</b>
<b>E</b>	<b>Paper 3: Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation</b>	<b>192</b>
<b>F</b>	<b>Paper 4: Hyper-parameter Tuning using Genetic Algorithms for Software Effort Estimation</b>	<b>204</b>
<b>G</b>	<b>Paper 5: Multi-objective Hyper-parameter Tuning for Software Effort Estimation</b>	<b>212</b>

<b>H Paper 6: Comparative study of Random Search Hyper-Parameter Tuning for Software Effort Estimation</b>	<b>227</b>
<b>Bibliography</b>	<b>236</b>

# Resumen

Los algoritmos de aprendizaje automático han sido utilizados para crear modelos con mayor precisión para la estimación del esfuerzo del desarrollo de software. Sin embargo, estos algoritmos son sensibles a factores, incluyendo la selección de hiper parámetros. Para reducir esto, se han investigado recientemente algoritmos de ajuste automático de hiper parámetros. Es necesario evaluar la efectividad de estos algoritmos en el contexto de estimación de esfuerzo. Estas evaluaciones podrían ayudar a entender qué hiper parámetros se pueden ajustar para mejorar los modelos, y en qué contextos esto ayuda el rendimiento de los modelos.

El objetivo de este trabajo es desarrollar un procedimiento automatizado para el ajuste de hiper parámetros para algoritmos de aprendizaje automático aplicados a la estimación de esfuerzo del desarrollo de software. La metodología seguida en este trabajo consta de realizar un estudio de mapeo sistemático para caracterizar los algoritmos de ajuste existentes, desarrollar el procedimiento automatizado, y conducir cuasi experimentos controlados para evaluar este procedimiento.

Mediante el mapeo sistemático descubrimos que la literatura en estimación de esfuerzo ha favorecido el uso de la búsqueda en cuadrícula. Los resultados obtenidos en nuestros cuasi experimentos demostraron que algoritmos de estimación no-exhaustivos son viables para la estimación de esfuerzo. Estos resultados indican que evaluar aleatoriamente 60 hiper parámetros puede ser tan efectivo como la búsqueda en cuadrícula, y que muchos de los métodos usados en el estado del arte son solo más efectivos que esta búsqueda aleatoria en 6% de los escenarios. Recomendamos el uso de la búsqueda aleatoria, algoritmos genéticos y similares, y la búsqueda tabú y armónica.

# Abstract

Software effort estimation requires accurate prediction models. Machine learning algorithms have been used to create more accurate estimation models. However, these algorithms are sensitive to factors such as the choice of hyper-parameters. To reduce this sensitivity, automated approaches for hyper-parameter tuning have been recently investigated. There is a need for further research on the effectiveness of such approaches in the context of software effort estimation. These evaluations could help understand which hyper-parameter settings can be adjusted to improve model accuracy, and in which specific contexts tuning can benefit model performance.

The goal of this work is to develop an automated procedure for machine learning hyper-parameter tuning in the context of software effort estimation. The automated procedure builds and evaluates software effort estimation models to determine the most accurate evaluation schemes. The methodology followed in this work consists of first performing a systematic mapping study to characterize existing hyper-parameter tuning approaches in software effort estimation, developing the procedure to automate the evaluation of hyper-parameter tuning, and conducting controlled quasi experiments to evaluate the automated procedure.

From the systematic literature mapping we discovered that effort estimation literature has favored the use of grid search. The results we obtained in our quasi experiments demonstrated that fast, less exhaustive tuners were viable in place of grid search. These results indicate that randomly evaluating 60 hyper-parameters can be as good as grid search, and that multiple state-of-the-art tuners were only more effective than this random search in 6% of the evaluated dataset-model combinations. We endorse random search, genetic algorithms, flash, differential evolution, and tabu and harmony search as effective tuners.

# List of Tables

1	List of acronyms . . . . .	xvi
2.1	SEE datasets and their qualities. . . . .	16
2.2	SEE frameworks. . . . .	19
3.1	PICO clusters and search string. . . . .	56
3.2	Data extraction fields. . . . .	58
3.3	Hyper-parameter tuning approaches and related SEE studies. . . . .	62
3.4	Study focus and related SEE studies. . . . .	64
5	Machine learning algorithms and related SEE studies. . . . .	65
3.6	Cross-validation approaches and related SEE studies. . . . .	70
3.7	Data transformations and related SEE studies. . . . .	72
3.8	Feature selection and related SEE studies. . . . .	74
9	SEE datasets and related SEE studies. . . . .	75
10	Evaluation metrics and related SEE studies. . . . .	81
11	Analysis techniques and related SEE studies. . . . .	85
12	Challenge categories and related SEE studies. . . . .	88
4.1	Categories, techniques, parameters and possible values for the FW file. . . . .	103
4.2	Parameters and possible values for the DS file. . . . .	104
4.3	Parameters and possible values for the DT file. . . . .	104
4.4	Libraries used in for the construction of ChimeraHPT. . . . .	110
4.5	Machine learning algorithms supported in ChimeraHPT. . . . .	111
4.6	Hyper-parameter tuners supported in ChimeraHPT. . . . .	112
4.7	Cross-validation approaches supported in ChimeraHPT. . . . .	112



4.8	Data transformations supported in ChimeraHPT. . . . .	112
4.9	Feature selectors supported in ChimeraHPT. . . . .	113
4.10	Evaluation metrics supported in ChimeraHPT. . . . .	114
4.11	Baseline estimators supported in ChimeraHPT. . . . .	114
5.1	Quasi experiments of the third specific objective. . . . .	119
1	List of selected studies. . . . .	142
2	Quality assessment per paper. . . . .	148
3	Machine learning techniques by sub-type and related SEE studies. . . .	152
1	Categories, techniques, parameters and possible values for the AS file.	157
2	Categories, techniques, parameters and possible values for the LA file.	158
3	Categories, techniques, parameters and possible values for the PT file.	160

# List of Figures

1.1	Design of the research project. . . . .	5
1.2	Summary of research methodology. . . . .	7
2.1	Software effort estimation process phases. . . . .	13
2.2	Simple machine learning framework. . . . .	25
2.3	Summary of the design science approach. . . . .	45
2.4	Systematic literature review and mapping process. . . . .	48
2.5	Complete empirical experiment process. . . . .	51
2.6	Incremental development process. . . . .	53
3.1	Systematic mapping study steps and results. . . . .	57
3.2	Classification scheme for extracted fields. . . . .	59
3.3	Hyper-parameter tuning approaches and usage through time. . . . .	63
3.4	Categorization of techniques of the evaluation scheme and usage through time. . . . .	65
3.5	Machine learning algorithms and usage through time. . . . .	67
3.6	Cross-validation approaches and usage through time. . . . .	71
3.7	Data transformation approaches and usage through time. . . . .	73
3.8	Feature selection approaches and usage through time. . . . .	74
3.9	Dataset origins and usage through time. . . . .	78
3.10	PROMISE and ISBSG datasets and usage through time. . . . .	79
3.11	Evaluation metrics and usage through time. . . . .	83
3.12	Reported challenges through time. . . . .	90
4.1	Tasks of the ChimeraHTP procedure. . . . .	99

4.2	Tasks and activities of the ChimeraHTP procedure. . . . .	100
4.3	Hyper-parameter tuning in ChimeraHPT. . . . .	109

# List of acronyms

Table 1: List of acronyms

1 + 1:	One plus one genetic algorithm
ANN:	Artificial neural network
ANOVA:	Analysis of Variance
BL:	Baseline
BO:	Bayesian optimization
CART:	Classification and regression trees
CC:	Cross-company
CGA:	Compact genetic algorithm
CV:	Cross-validation
DC:	Design cycle
DE:	Differential Evolution
DQ:	Design questions
DS:	Dataset
DT:	Data transformation
EC:	Empirical cycle
EM:	Evaluation metrics
ESE:	Empirical software engineering
FP:	Function points
FS:	Feature selection
GA:	Genetic algorithm

Continued on next page

Table 1: List of acronyms (Continued)

GS:	Grid search
HS:	Harmony search
ISBSG:	International Software Benchmarking Standards Group
KNN:	K-nearest neighbors
KLOC:	Kilo lines of code
KQ:	Knowledge questions
LA :	(Machine) Learning algorithm
LOC:	Lines of code
MAE:	Mean absolute error
MAR:	Mean absolute residual
MBRE:	Mean balanced relative error
MdAE:	Median absolute error
MdAR:	Median absolute residual
MLP:	Multi-layer perceptron
ML:	Machine learning
MLSEE:	Machine learning software effort estimation
MdMRE:	Median Magnitude of Relative Error
MMRE:	Mean Magnitude of Relative Error
MSE:	Mean squared error
PICO:	Population, Intervention, Comparison, Outcome
PROMISE:	Predictive Models in Software Engineering
PSO:	Particle swarm optimization
RQ:	Research question
RS:	Random search
SA:	Standardized accuracy
SBSE:	Search based software engineering
SE:	Software Engineering

Continued on next page

Table 1: List of acronyms (Continued)

SEE:	Software effort estimation
SLR:	Systematic literature review
SMS:	Systematic mapping study
SO:	Specific objective
SVM:	Support vector machines
SVR:	Support vector regression
WC:	Within-company



UNIVERSIDAD DE  
COSTA RICA

SEP Sistema de  
Estudios de Posgrado

**Autorización para digitalización y comunicación pública de Trabajos Finales de Graduación del Sistema de Estudios de Posgrado en el Repositorio Institucional de la Universidad de Costa Rica.**

Yo, Leonardo Villalobos Arias, con cédula de identidad 4 0225 0296, en mi condición de autor del TFG titulado Evaluating an Automated Procedure of Machine Learning Parameter Tuning for Software Effort Estimation

Autorizo a la Universidad de Costa Rica para digitalizar y hacer divulgación pública de forma gratuita de dicho TFG a través del Repositorio Institucional u otro medio electrónico, para ser puesto a disposición del público según lo que establezca el Sistema de Estudios de Posgrado. SI  NO \*

\*En caso de la negativa favor indicar el tiempo de restricción: \_\_\_\_\_ año (s).

Este Trabajo Final de Graduación será publicado en formato PDF, o en el formato que en el momento se establezca, de tal forma que el acceso al mismo sea libre, con el fin de permitir la consulta e impresión, pero no su modificación.

Manifiesto que mi Trabajo Final de Graduación fue debidamente subido al sistema digital Kerwá y su contenido corresponde al documento original que sirvió para la obtención de mi título, y que su información no infringe ni violenta ningún derecho a terceros. El TFG además cuenta con el visto bueno de mi Director (a) de Tesis o Tutor (a) y cumplió con lo establecido en la revisión del Formato por parte del Sistema de Estudios de Posgrado.

FIRMA ESTUDIANTE

Nota: El presente documento constituye una declaración jurada, cuyos alcances aseguran a la Universidad, que su contenido sea tomado como cierto. Su importancia radica en que permite abreviar procedimientos administrativos, y al mismo tiempo genera una responsabilidad legal para que quien declare contrario a la verdad de lo que manifiesta, puede como consecuencia, enfrentar un proceso penal por delito de perjurio, tipificado en el artículo 318 de nuestro Código Penal. Lo anterior implica que el estudiante se vea forzado a realizar su mayor esfuerzo para que no sólo incluya información veraz en la Licencia de Publicación, sino que también realice diligentemente la gestión de subir el documento correcto en la plataforma digital Kerwá.

# Chapter 1

## Introduction

Software development deals with limited resources [1], and must carefully allocate assets such as development personnel and budget to ensure the success of a project. For this reason, a key problem in software engineering is the estimation of software cost and effort [1], which is known as software effort estimation (SEE). Software effort estimation techniques have practical applications in budgeting, risk analysis, project planning, and improvement analysis [2]. Hence, the accuracy of software effort estimates is of vital importance. Overestimating the effort could lead to rejecting a potentially beneficial project, and the loss of a strategic opportunity. Underestimating the effort, on the other hand, could result in accepting a project that would fail to achieve its expected payoff [3].

There are two main approaches for software effort estimation: expert judgment and engineering [2, 4, 5]. Expert judgment provides estimates based on practitioners' expertise [2]. While this approach is useful in the absence of empirical data, it is difficult to evaluate due to its normally informal and undocumented nature [5]. The engineering approach builds estimation models using information from past projects [5]. Traditionally, Software effort estimation has employed parametric models (those expressed as a mathematical equation) such as COCOMO, SLIM, and PRICE\_S [2]. However, these models have been reported to yield poor estimations as size or complexity of software grows [2, 6].

The use of machine learning (ML) algorithms in software effort estimation has been an active research area from the late 1970s to date [7, 6, 8, 9, 10]. Based on historical project data, machine learning algorithms generate and adjust (train) an estimation model [11]. A model is constructed from an evaluation scheme, i.e., combination of dataset, validation approach, data pre-processing, feature selection



technique, machine learning algorithm, and hyper-parameter tuning approach. This model is then trained to minimize the prediction error on existing data, while remaining general enough to predict new, unseen data [11]. The main application of such models is to aid experts in performing or revising their effort estimations, thus improving the decision-making process of software projects [12].

The use of ML algorithms in SEE comes with the challenge of sensitivity to multiple factors: input data, data transformation, feature selection, validation approach, and hyper-parameter values [13, 14]. The impact of factors on the prediction accuracy of effort estimation models has been extensively studied, with somewhat contradictory results [15, 16, 17, 18, 19, 13]. Many of these studies grant no access to their experimental artifacts (source code), and lack reporting detail on their experimental design, especially on the factors that affect the estimation models [13]. Because of this, the results obtained by one study may be impossible to replicate in another. This phenomenon is known as conclusion instability [13]. This instability can be measured through the prediction stability: the variance in the prediction accuracy [20].

One key factor that can affect software effort estimation models is the hyper-parameter configuration of the machine learning algorithm [14, 21]. These Hyper-parameters are values that must be set before the model is trained, as they affect its construction [12, 21]. Examples of hyper-parameters include the amount of hidden layers  $a$  in a multilayer perceptron, and the kernel type in a support vector machine. Software effort estimation literature uses the term *parameter* instead of *hyper-parameter* when referring to selection or tuning (i.e., parameter tuning instead of hyper-parameter tuning) [8, 14, 22, 23, 24, 25]. Many studies of ML in SEE do not report the hyper-parameter settings, or do not use a hyper-parameter tuning procedure [13, 26]. Generally, hyper-parameters are set to their default values, or determined through an iterative, manual process [21].

Hyper-parameter tuning approaches (also known as parameter optimization) are search algorithms that seek the hyper-parameter settings for a ML model that minimize the estimation or prediction error [27]. These approaches can automate manual tuning and achieve similar performance [21]. Previous works have reported that hyper-parameter tuning improves the accuracy of effort estimation [14]. Moreover, tuning can reduce conclusion instability by improving the prediction stability with respect to default parameter models [28]. Examples of tuning approaches are grid search, random search [29], particle swarm optimization [30] and genetic algorithms [31]. These tuners train and evaluate models with different hyper-parameter configurations, and choose the one with highest accuracy [27]. This requires signif-

icant time and processing power [21], thus in practice, an organization would use only the best tuning approach to improve their estimations [27, 21].

A software effort estimation procedure is necessary to determine the best hyper-parameter tuning approach for a particular dataset and model. A SEE procedure is a detailed series of tasks and sub-activities used in accordance with a measurement method, to obtain an effort estimate [32]. A framework is required to automate the otherwise manual activities of this procedure. A SEE framework is the implementation of a procedure, automating the process of building and evaluating machine learning models [33].

The software engineering literature has defined multiple procedures and frameworks. The first studies proposed the use of data statistics, such as the mean or median, as a comparison baseline against estimation models [34, 16]. Afterwards, the first notion of an evaluation framework was proposed [35, 36, 37]: a method that allows the comparison of two different ML algorithms, favoring the use of public datasets, validation techniques, and configuration evaluation techniques. Later, this notion was extended by adding comparisons for each evaluation scheme, and introducing a training-testing-validation split of the datasets [38]. These ideas were brought into the SEE field [39] and better analysis techniques were introduced to increment the conclusion stability of SEE studies [40].

As SEE frameworks were being developed, baseline frameworks emerged. These frameworks included baseline techniques or metrics that allow comparison of results from different studies, even when using different procedures. Examples of these are the  $p_0$  predictor and the standardized accuracy metric [41, 42], the minimum interval of equivalence metric [43], and the individual absolute residuals [44]. With the advent of these baselines, usage of biased metrics such as the mean magnitude of relative error (MMRE) [45] was discouraged [46, 47, 48].

Lately, empirical studies have begun to include hyper-parameter tuning as part of their frameworks. Previous research has evaluated the impact of hyper-parameter tuning in the accuracy of effort estimation models, and suggested that tuning should be included in future frameworks [14]. More recent studies in software effort estimation have employed hyper-parameter tuning for online estimation [12], evaluated tabu search [49] and bee's algorithm for tuning [50], and proposed fast and efficient approaches such as dodge [51, 52], flash [53, 54], and differential evolution [55, 56]. However, only a few studies in software effort estimation have compared hyper-parameter tuning approaches for machine learning [57, 58, 24]. This

thesis thus evaluates the effectiveness of some of the existing hyper-parameter tuning approaches for machine learning algorithms to estimate software effort.

## 1.1 Objectives

The **main goal** of the thesis was *to develop an automated procedure for machine learning hyper-parameter tuning in the context of software effort estimation*. To meet this general objective, three specific objectives were proposed in the context of software effort estimation:

- S01. To characterize hyper-parameter tuning approaches for machine learning.
- S02. To automate a hyper-parameter tuning procedure for machine learning.
- S03. To evaluate the effectiveness of the automated hyper-parameter tuning procedure for machine learning.

The automated procedure supports the selection of techniques that comprise the evaluation scheme: data sets (DS), validation approaches (CV), data pre-processing transformations (DT), feature selections (FS), machine learning algorithms (LA), and hyper-parameter tuning (PT). The set of evaluation schemes to evaluate was constructed as every possible combination of all selected techniques. Each scheme was evaluated by constructing, training, and evaluating an effort estimation model. In this process, the hyper-parameter tuning algorithm adjusted and selected the hyper-parameter values of the machine learning model.

To evaluate the automated procedure, we employed 9 benchmark datasets from the software effort estimation literature [59] and 4 subsets from the ISBSG 2018 release 1 repository [60]. Such datasets included information about past software engineering projects, and contained data of the company and the project size. The procedure conducted a fair comparison of each hyper-parameter tuning approach with respect to their effectiveness to improve the prediction accuracy and stability of machine learning algorithms. Our automated procedure is potentially useful for software engineering stakeholders. With its use, such professionals can determine the best tuners and models for their company data, regardless of the characteristics of their projects.

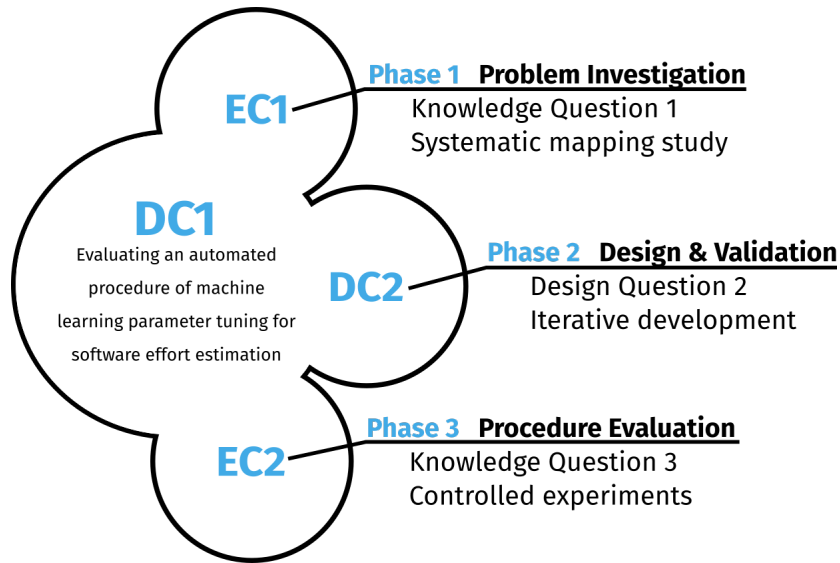


Figure 1.1: Design of the research project.

## 1.2 Methodology

The development of this research and thesis document followed the design science methodology for information systems and software engineering [61]. Design science proposes a framework of design and investigation of research artifacts in the context of software engineering. Design science uses a combination of design and empirical cycles to propose design or research questions. Each question was answered at the end of the completion of its respective cycle [61]. In the context of this research, the research artifact was the automated procedure of machine learning hyper-parameter tuning for software effort estimation.

The design of this research was composed by a set of design cycles (DC) and empirical cycles (EC), as shown in figure 1.1. A main design cycle covered the main objective of the thesis. The cycle consisted of three phases, each consisting of a sub-cycle and covering a specific objective of the thesis.

To guide this research, the following design questions (DQ) and knowledge questions (KQ) were defined:

**KQ1:** What are the characteristics of machine learning hyper-parameter tuning approaches for software effort estimation?

**DQ2:** How to design an automated parameter tuning procedure for machine learning software effort estimation?

KQ3: What is the effectiveness of the automated parameter tuning procedure for machine learning software effort estimation?

The first phase covered knowledge question 1 and the first specific objective (SO1), in which machine learning hyper-parameter tuning approaches in SEE were characterized. To accomplish this, an empirical cycle was executed to conduct a systematic mapping study [62].

The second phase covered design question 2 and the second specific objective (SO2), in which the automated hyper-parameter tuning procedure for machine learning was designed, implemented and validated. To accomplish this, a design cycle was executed using the iterative development methodology [63].

The third phase covered knowledge question 3 and the third specific objective (SO3), in which effectiveness of the automated hyper-parameter tuning procedure was evaluated. In this evaluation, the procedure determined the impact of the hyper-parameter tuning approaches on the estimation accuracy of effort estimation models. To accomplish this, an empirical cycle, consisting of controlled quasi experiments, was executed [64].

The completion of the activities related to these three phases resulted in different products that comprise this thesis document. Figure 1.2 summarizes the objectives, method, and products of each phase of the research.

The methodology followed in each of the three phases is covered on its specific chapter of the thesis document. Chapter 3 presents the methodology for phase one to conduct a systematic mapping study. Chapter 5 contains the methodology for phase three, presenting a design for each of the conducted controlled quasi experiments.

### 1.3 Contributions and products

The following are the contributions and products of this research:

- The report of a systematic mapping study of the existing hyper-parameter tuning approaches; additionally covering the machine learning algorithms, datasets, data pre-processing, feature selection, and validation approaches used in conjunction with hyper-parameter tuning.
- The design and implementation of an automated hyper-parameter tuning machine learning procedure for software effort estimation.

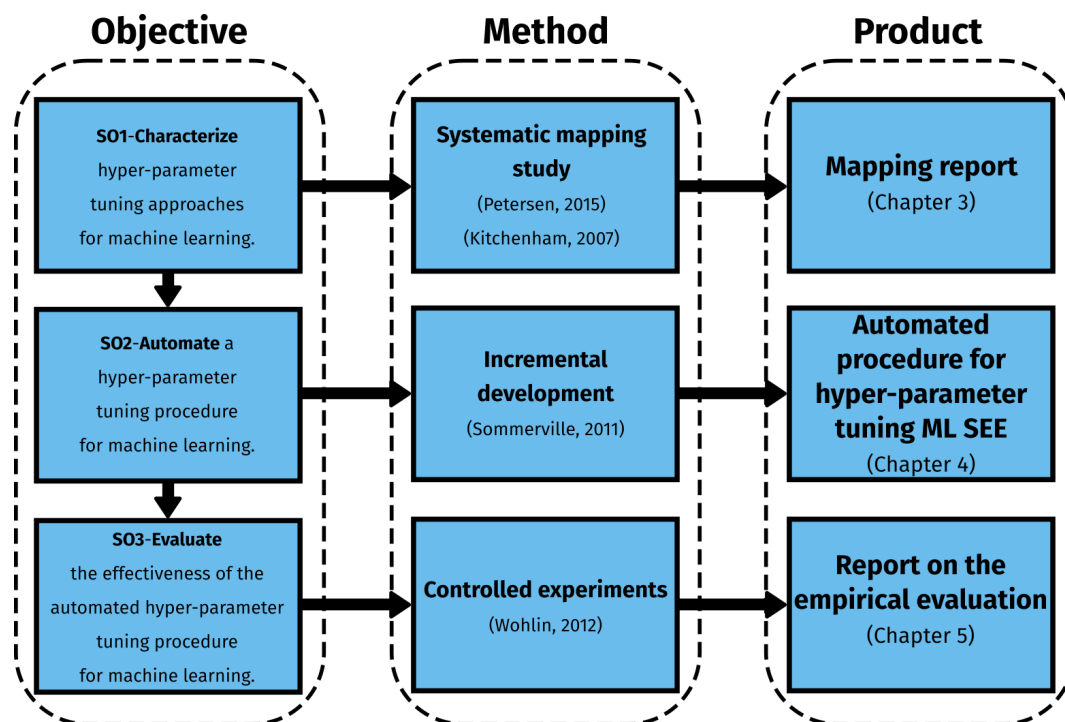


Figure 1.2: Summary of research methodology.

- An empirical evaluation of the hyper-parameter tuning procedure in terms of accuracy and stability, and a set of the hyper-parameter tuning approaches that provide the best improvement of prediction accuracy for each particular dataset and machine learning algorithm.

## 1.4 Document structure

This thesis document is structured as follows: Chapter 2 introduces the concepts of software effort estimation, hyper-parameter tuning and machine learning, and empirical research methodologies. Chapter 3 presents the mapping study on hyper-parameter tuning approaches in SEE, showing the reported hyper-parameter tuners, machine learning algorithms, datasets, evaluation metrics and challenges. Chapter 4 details the automated hyper-parameter tuning procedure for machine learning, as well as the implemented framework to support this procedure. Chapter 5 presents the design and results of the series of quasi experiments that evaluated the automated hyper-parameter tuning procedure. These studies covered the following areas: effect of tuning on the accuracy and stability of SEE models, genetic and bio-inspired tuning, multi-objective hyper-parameter tuning, and random search vs. state-of-the-art

tuning. Lastly, chapter 6 summarizes the results of this research and shows possible venues for future work.

# Chapter 2

## Background

This chapter presents the background concepts related to software effort estimation and machine learning. Section 2.1 contains the foundation and role of data science in software engineering. Section 2.2 contains a basic explanation of the search-based software engineering research field. Section 2.3 contains the foundations of software effort estimation. Section 2.4 contains the general process of machine learning evaluation frameworks for software effort estimation. Section 2.5 contains the principles of machine learning. Sections 2.6 and 2.7 respectively contain the foundations of the empirical methodologies and software engineering methodologies that will be used in the thesis.

### 2.1 Data science in software engineering

The software engineering process naturally produces a large amount of data, in the form of source code, bug reports, test cases, and user stories, among others. Research in software engineering attempts to build new knowledge, based on this information [65]. Software engineering data can come from different sources, collected using different methods, be structured or unstructured, have varying degrees of noise, and be objective or subjective. This variability introduces the need for data analysis for software engineering [65].

Software analytics uses data-driven approaches to enable software practitioners to perform data exploration and analysis, in order to obtain insightful and actionable information for completing various tasks around software systems, software users, and the software development process [66, 67, 65]. Data analysis in software engineering should provide a solution to a recurrent problem in the field, while being accessible



(automatable, re-usable, and actionable) to software engineering practitioners [65].

Software analytics is relevant and used in many research areas of software engineering. Some of these include repository mining, software testing and fault detection, defect prediction, software analytics, service analysis, security analysis, and software effort estimation. In the case of software effort estimation, data science techniques—such as machine learning—are used to perform predictive analysis of project effort. In order to minimize project costs, a practitioner could use historical project data to predict costs of future development projects. In this scenario, prediction techniques such as machine learning aid the data analysis tasks, which would ultimately to a quantifiable benefit to the practitioner—the reduction development costs due to minimization of estimation risk [65, 68]. In addition, search-based software engineering techniques can be combined with data science approaches. For instance, a machine learning technique can be combined with a search-based hyper-parameter tuning approach [51].

## 2.2 Search based software engineering

Search based software engineering (SBSE) is a research field inside software engineering, which applies search based optimization techniques to SE problems. In other words, SBSE reformulates SE problems to search problems [69, 68, 70, 71]. Search problems in this contexts are problems in which an optimal (or near-optimal) solution is sought in a search space composed of candidate solutions [71]. A search function guides this process, differentiating which is best between two candidate solutions. In addition, SBSE techniques work with limited resources—i.e. computation time—, and thus usually do not search within the entirety of this sub-space [70]. Techniques such as random search, hill climbing, simulated annealing, nature-based search algorithms (such as cuckoo search, artificial bee colonies, and particle swarm optimization), and genetic algorithms comprise the field of SBSE [70, 72].

The search-based software engineering approach and its techniques can be applied to software effort estimation—particularly modeling [73] and hyper-parameter tuning [51]. SBSE approaches can be used to estimate effort, select relevant features for a dataset, and adapt models to new data, and select hyper-parameter values for models. For effort estimation, genetic programming approaches can be used to estimate software effort. One of the first authors [73] to do so is Dolado [74, 75], who employed genetic algorithms to find accurate functions to predict effort. SBSE can

aid in online effort estimation, which considers the change of data over time. Effort estimation is a changing problem, as the company that performs the estimation will change over time. An approach such as Bayesian evolutionary algorithms may aid in this task [73]. For feature selection, SBSE approaches this as a multi-objective optimization problem, in which the amount of features of the input data is minimal, while maximizing estimation accuracy. Other factors, such as the cost of collecting a particular feature, can be considered in a multi-objective feature selection [73]. Kirsopp et al. [76] used a hill climbing approach to select features for Case Based Reasoning systems to estimate software effort. Lastly, for hyper-parameter tuning, SBSE algorithms can be used to search for good hyper-parameter values for machine learning algorithms applied to effort estimation, using approaches such as genetic algorithms [31] or particle swarm optimization [77].

One of the main barriers for search based algorithms (or machine learning in the context of this proposal) is tuning. Search algorithms have a series of hyper-parameters that need to be set before execution, and the value of these impact the performance of the approach [78]. The use of hyper-parameter tuning is a trade-off problem in itself. On one hand, default hyper-parameter values from the literature can function adequately well in some cases, but not quite for larger amounts of data [79]. Thus, it would be beneficial to perform a hyper-parameter tuning process. On the other hand, hyper-parameter tuning may become very expensive, but the improvements that are produced by it may not be so good [78]. Based on this, it is important that hyper-parameter tuning approaches act based on the allocated search budget [79, 78].

Recently, a novel field of empirical software engineering has emerged: DUO (Data mining Using/Used-by Optimizers) [51]. These research field combines the strength of data mining approaches and optimization algorithms, which function as an “advisor” that aids the tasks of a data analyst, and that can aid construction of better predictive models. Similarly, data mining and optimization approaches complement each other. Optimization—such as hyper-parameter tuning—can be used to improve predictive accuracy of data miners. In turn, data mining approaches can be used to mitigate the problem of diminishing returns that optimization approaches face—dividing work using clustering, for instance. Because of this, data mining approaches that employ no optimization have been considered deprecated [51], and previous work should be revisited and reevaluated.

## 2.3 Software effort estimation

Software effort estimation (SEE) can be defined as the process of predicting or estimating, from base characteristics of a software development project, the effort required for its completion, usually measured in man-hours [1, 2, 5]. Estimation is not a process of coming up with a number that a team must commit to, but rather a process to aid decision making [5].

Two approaches exist for effort estimation: the expert judgment approach and the engineering approach [2, 4, 5]. The expert approach is the most used process in industry, even though it is more subjective and is less measurable (and thus, is more difficult to evaluate) than the engineering or model approach. Expert judgment is highly dependent on the expertise on the people that participate on the process [5]. In addition, it is difficult to measure how effective this type of evaluation really is. While a successful project may seem to respect its allocated budget, it is possible that said project may do so at stake of not meeting all requirements [5].

The second approach to software effort estimation is the engineering approach [5], also referred to as the model or mathematical approach [2, 4]. Instead of relying in expert knowledge, this estimation method is based on information on past projects and careful management and usage of this information. One of the most important characteristics of this approach is understanding that not all existing models apply to every situation, but rather it is imperative to understand the context and constraints of the available information and select one or more applicable models based on these [5]. One analysis technique is to look at the existing historical data, one variable at a time, and proposing a potential model that explains its relation with effort [5]. The engineering approach can employ two types of prediction technique: parametric models, and learning models [2]. Parametric models are based on mathematical functions, and learning models include regression methods and machine learning algorithms [2].

Software effort estimation is a process that is performed through the life cycle of a project, instead of only at the project planning. At the beginning of a software development project, most information available is either obtained on past projects, or documentation from the current project. As development progresses, more empirical information on the project—such as time and requirement completion—is known. This is a phenomenon in effort estimation known as the cone of uncertainty [5]. Accuracy of the effort estimation of a project could improve along time, as new infor-

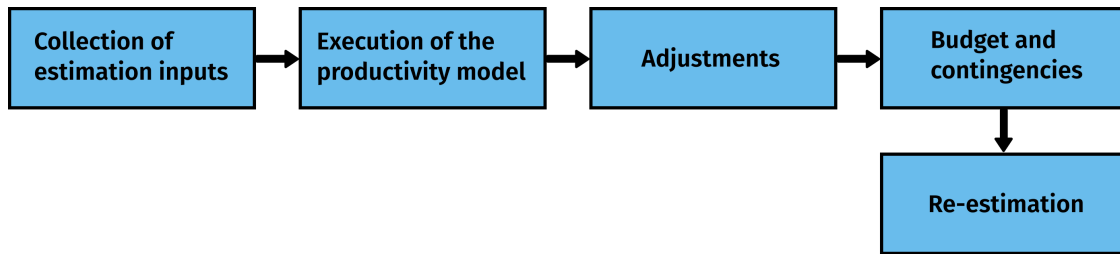


Figure 2.1: Software effort estimation process phases. Adapted from [5].

mation is available as time goes on. Some uncertainty remains in the prediction, as the interaction between variables is not fully known.

### 2.3.1 SEE process

The general effort estimation process has 5 phases [5], as shown in figure 2.1: (1) collection of input data, (2) productivity model prediction, (3) model adjustment, (4) budget decision, and (5) re-estimation.

The first phase of the estimation process deals with the collection of input data. These often deal with those resources, processes, and products associated with the project that could potentially drive the cost of the project.

The second phase of the estimation process relates to the productivity model. A productivity model simulates the development process, based on the input data, and generates an estimate of the required effort or resources. This estimate may be either a single value, or an estimation interval with certain degree of confidence.

The third phase of the estimation process is the adjustment process. This refers adjustment of the model, and the entire effort estimation process, to accommodate for novel information. Through the early estimation process, new information and constraints—originally not in the historical data—are discovered.

The fourth phase of the estimation process is the budget decision. This refers to selecting a value or set of values from the estimation process, so that the project is (or is not) carried out based on this estimate. This process is performed manually, with expert judgment.

The fifth phase of the estimation process is the re-estimation. The software effort estimation process is, by nature, iterative, as new and more pertinent information is through time. The re-estimation process is similar to the adjustment process, as it comprises the incorporation of new estimation into the productivity model to generate a more accurate estimation.

This thesis focuses on the second step of the estimation process: the productivity model. Through this work, it will be referred as a prediction model. While traditionally productivity models were often mathematical equations, more recent research has shifted into machine learning models. Section 2.5 covers the bases of machine learning and its process.

### 2.3.2 SEE datasets

Data is the cornerstone of empirical software engineering research and practice [80]. Data drives the estimation process, as it is used to produce effort prediction models. Software effort estimation datasets contain metrics about completed software engineering projects, such as size, effort, duration, programming language, type of business, and so forth [80]. Software development companies often use their own project data to perform SEE, but often do not share or publish their own datasets. Because of this, a small amount of datasets are available for research, and of which some are widely used in software effort estimation research.

There are several datasets that are commonly used in software effort estimation [81, 82, 83, 84, 85, 80]. These datasets are available in either the PROMISE repository<sup>1</sup>, or the International Software Benchmarking Standards Group (ISBSG)<sup>2</sup>. Table 2.1 shows the names, timeliness, amount of projects and features, and some of the units of important features of these datasets. The timeliness column shows the completion dates of the projects in the dataset. Datasets with an asterisk (\*) in their timeliness denote that no date information is available, and instead dataset publication date is used. The Proj. and Feat. columns show the amount of software projects (rows) and the metrics recorded for each project (columns), respectively. The FP approach column shows the type of function points used to estimate software size. The size unit column shows estimated or delivered size of the final product, measured in either lines of code (LOC), thousand lines of code (KLOC), or amount of files. The effort unit column shows the amount of time required (in either hours or months of work). The type column shows whether the dataset contains information belonging to a single company (WC, within-company), or multiple companies (CC, cross-company).

While unique, all datasets share some common features. All datasets share the target variable, effort, which is also the feature that is most frequented in primary

<sup>1</sup><http://promise.site.uottawa.ca/SERepository/>

<sup>2</sup><https://www.isbsg.org/>

studies [86]. The other dataset variables are then used to perform estimation of the target feature. Other important variables for effort estimation include (adjusted) functional size, development type, language type, development platform, team size, organization type, programming language, among others [87]. These variables describe the amount of work required to complete the software project, and play an important role in estimation problems. Other types of variables cover information about the company or developing team.

There are some limitations regarding these datasets. Some are comprised of projects of multiple companies (cross-company, CC), as opposed to one company (Within-company, WC; or single company, SC). These have the problem of having more outliers, and the data is more heterogeneous, as projects with similar characteristics may have completely different effort values. Some solutions to this problem include partitioning of the data into sub-datasets, or elimination of extreme values altogether [86]. Because of these limitations, datasets are seldom used as they are, and instead transformations or selection of data is performed.

Studies have reached conflicting results on the effect of CC studies against WC studies [88, 16, 19]. In some cases, prediction is worse for CC studies. For other cases, there is benefit—or it is indifferent—in using CC data instead of only WC data. In general terms, usage of CC data for estimation is a trade-off between number of data instances, and heterogeneity of the data points. Thus, research on the amount of CC projects to use in software effort estimation is an ongoing research area [89, 12].

The most important advantage of these public datasets is that they diminish data collection costs, which are expensive process for organizations [86]. These benefits however, do not come with a series of challenges. The main problem of these can be generally described as the variance within different projects, which generates variance bias and conclusion instability [13]. For instance, CC datasets have two main problems: heterogeneity of data (as it is provided from multiple, different sources), and the amount of missing values [86]. A traditional effort estimation approach—such as a formula-based estimation model—may become too inaccurate given these conditions. Machine learning approaches are able to work around these problems (for instance, clustering and missing value imputations can address CC issues), and provide more accurate estimations of effort.

Table 2.1: SEE datasets and their qualities. Based on Bosu and Macdonnel [80], and Shepperd and Schofield[82].

Dataset	Timeliness	Proj.	Feat.	FP approach	Size unit	Effort unit	Type
Albrecht	1974–1979	24	8	Albrecht FP	No	Hours	WC
China	2011*	499	19	Albrecht FP	No	Hours	?
Cocomo81	1981*	63	19	No	LOC	Months	?
Desharnais	1982–1988	81	12	FP	No	Hours	WC
Finnish	1997*	38	9	FP	No	Hours	CC
ISBSG10	2010	37	94	IFPUG 4+, NESMA	LOC	Hours	CC
ISBSG16	1989–2015	7518	264	IFPUG, NESMA, MARK II, COS- MIC	LOC	Hours	CC
ISBSG18	1989–2016	8261	252	IFPUG, NESMA, MARK II, COS- MIC, others	LOC	Hours	CC
Kemerer	1981–1985	15	8	No	KLOC	Months	WC
Kitchenham	1994–1988	145	10	FP	No	Hours	WC
Maxwell	1993	62	27	FP	No	Hours	WC
Miyazaki94	1994*	48	9	No	KLOC	Months	CC
NASA93	1971–1987	93	24	No	LOC	Months	CC
SDR	2000	12	25	No	LOC	Months	CC
Telecom	1997*	18	4	No	Files	Months	WC

## 2.4 SEE evaluation frameworks

In software effort estimation, many factors affect the accuracy of a prediction model—as well as conclusion stability of studies—, which can be generalized into two larger factors: bias and variance [13]. Variance refers to the dispersion or distance of the different model predictions, which would in turn cause conclusion instability across studies. Bias refers to the distance from the predicted values to the real effort values, which would also cause conclusion instability across studies.

Conclusion stability due to variance is mainly related to two possible sources: variance from sampling, and variance from the validation scheme [13]. Variance from sampling is related to the origin of the model input data, particularly when dealing with the same model across two or more different datasets. An effort estimation dataset is a non-random sample from a population of software projects, usually tied to a specific organization and using the same collection method. Variance from the validation scheme comes to the noise that has its origin in randomly splitting a dataset for training and validation. If all data is used for training, it is possible that the model accuracy is overestimated. Using different data for training the model and assessing its accuracy lessen this overestimation. However, randomly splitting the dataset into training and validation could assign «good» cases into one group and «bad» cases into the other, affecting the accuracy.

Conclusion stability due to bias is mostly related to human or experimental factors, such as the experimenter, language, sampling, and verification [13]. For instance, a particular researcher that proposes a novel estimation model could be biased to showcase its model as the best model of all studied. Another instance of a source of bias could be introduced by the accuracy metric.

In order to reduce both bias and variance, multiple machine learning evaluation frameworks have been proposed in effort estimation [35, 36, 37, 38, 39, 41, 40, 26, 43, 42, 89, 44, 90, 33, 12]. The general objective of the frameworks presented by these studies is to reduce conclusion instability and guarantee replicability of the study. In the case of instability by variance, these frameworks apply different combinations of data pre-processing and use validation schemes to mitigate effects of randomness and high data set variance. For instability by bias, these frameworks propose comparison of multiple techniques across different datasets. SEE evaluation frameworks use multiple machine learning processes to build and evaluate models under different conditions. In general terms, these machine learning processes can



be summarized in 6 elements: datasets, validation approach, data pre-processing (which covers transformation, clustering, and missing value imputation), feature (or attribute) selection, machine learning algorithms (and their hyper-parameters), and hyper-parameter tuning approach for these algorithms. In the beginning, these frameworks were proposed for defect prediction, the same principles were applied in effort estimation.

Table 2.2 shows some the existing SEE evaluation frameworks in the literature. For each framework, the parts of the machine learning process are shown: datasets (DS), data transformation (DT), clustering (CL), missing value imputation (MV), attribute selection (AS), and machine learning algorithm (LA). The amount of studied techniques are shown for each of these categories. Sub-partitions or sub-techniques are not counted. For the validation approach (CV), the table shows if the study uses train-test split, train-test-validation split, Leave-one-out cross-validation, k-Fold cross-validation,  $M \times N$ -way cross-validation, or an online learning scheme. For parameter tuning (PT), the table shows if no tuning is done, if tuning is done manually, or if an automated hyper-parameter tuning approach is used. For baseline techniques (BL), the table shows whether a study uses or not one such technique. For evaluation metrics (EM), the table shows whether a study uses classification metrics—such as recall, precision, or area under a ROC curve—, relative error based metrics, or absolute error based metrics. In the case of usage of more than one type of metrics, the main type of metric is denoted (i.e. the metrics used for tuning or comparison of techniques).

One of the first proposals of a framework was done by Menzies et al. [35], in the area of defect prediction. This study presents the notion of a baseline experiment—a way to make two different studies comparable, by using the same data set. In their proposed framework, they employ  $M \times N$ -cross validation, all the combinations of two data transformation (filter) techniques, one attribute selection, and three learning algorithms. In addition, they employ as metrics probability of error detection ( $pd$ ) and probability of false alarm ( $pf$ ).

Zhang and Zhang [36] commented on the framework by Menzies et al., addressing that the metrics they used in their proposal—probability of detection, and probability of false alarm—are not satisfactory enough metrics. Instead, they recommend to use the precision and recall metrics for defect prediction, as  $pd$  and  $pf$  are influenced by class imbalance. Menzies et al. responded to Zhang and Zhang’s comment [37] by “agreeing to disagree” with their assessment. They present scenarios in which high precision and recall do not necessarily result in good predictions. In general terms, their argument is that precision and recall can be inversely proportional—increasing

Table 2.2: SEE frameworks.

Framework	DS	CV	DT	CL	MV	AS	LA	PT	BL	EM
Menzies et al., 2006[35]	10	$M \times N$	2	0	0	1	3	No	No	Classification
Song et al., 2010[38]	2	Train-test-validation	2	0	0	2	3	No	No	Classification
Mittas and Angelis, 2012[39]	6	K-fold	0	0	0	0	11	No	No	Absolute error
Keung et al., 2013[40]	11	Leave-one-out	7	0	0	3	9	No	No	Relative error
Song et al., 2013[14]	3	Online learning	0	0	0	0	4	Auto	No	Absolute error
Minku et al., 2015[91]	1	Train-test	0	0	0	0	3	Manual	Yes	Absolute error
Minku et al., 2017[89]	3	K-fold	0	3	0	0	2	Auto	Yes	Absolute error
Murillo-Morera et al., 2017[90]	1	$M \times N$	8	0	0	5	15	No	Yes	Absolute error
Minku, 2019[12]	1	Online learning	0	6	0	0	10	Auto	Yes	Absolute error
Quesada-López et al., 2019[33]	1	$M \times N$	3	0	0	5	15	Manual	Yes	Absolute error

one decreases the other—and in some cases it is better to sacrifice precision to achieve high recall.

An extension to the benchmark framework by Menzies et al. is presented by Song et al. [38]. Their contribution is twofold: (1) their framework draws conclusions for each machine learning scheme (combination of data transformation, attribute selection, and learning algorithm) instead of individually for each learning algorithm, and (2) their framework uses a two-step process to measure the “true”, unbiased performance of the best model. The two-step process of the proposed framework involves splitting data into a training and a validation set. The first step is the same as the framework by Menzies et al., using only the training set. The second step is the validation of the best model. This step re-trains the best model using all of the training set, and predicts the data of the validation set, measuring model performance. The

idea is to simulate a real situation, in which the selected scheme is used against new, unseen data. One important conclusion of the study was that there was no best universal scheme that overshadows all the others, but rather different schemes work on different data.

An evaluation framework for software cost (effort) estimation was proposed by Mittas and Angelis [39]. Their framework is different from Menzies et al. in two main aspects: (1) they only consider learning algorithms, and (2) they perform  $k$ -fold cross validation instead of  $M \times N$ -cross validation. Their main contribution is not in the framework steps, but rather in the analysis of the results of the obtained framework. In their work, they employ clustering of techniques, using Tukey tests and the Scott-Knott algorithm. Their results suggest that it may be better to search for a set of best estimation models, instead of a single best model.

Keung et al. [40] apply their previous knowledge of machine learning in defect prediction [35] for effort estimation. They propose an evaluation framework for effort estimation to try to reach conclusion stability. In their framework, they evaluate combinations of data transformation techniques and learning algorithms across different datasets. In a manner similar to Mittas and Angelis, they use Wilcoxon-signed rank and the win, tie, loss algorithm to rank the different combinations. The study concludes that stability can be reached.

Song et al. [14] perform a study on the impact of hyper-parameter tuning of different machine learning algorithms. In their study, they train and evaluate four different machine learning algorithms using all possible combinations of hyper-parameter values defined across a range. This technique would be later known as grid search. To measure the impact of tuning these hyper-parameter values, the study compares the best setting against the worst and default settings. One of the main conclusions of the study is that the value of tuning depends on the technique, as some algorithms are more sensitive to hyper-parameter values than others.

Minku et al. approach SEE research from an application perspective. On 2015, Minku et al. [91] used the Dycom approach to adapt cross-company data to make within-company predictions. The objective of this approach is to minimize the amount of WC data necessary to build a model, as this data is expensive to gather for most organizations. In 2017, Minku et al. [89] continue this research trend by adapting the Dycom model to use an online learning approach, and to include data clustering to focus on CC data that is most similar to the WC data. In 2019, Minku [12] incorporates automated hyper-parameter tuning into the Dycom approach, which is the first

proposal of hyper-parameter tuning procedure for online models in software effort estimation.

Murillo-Morera et al. [90] employ a genetic algorithm to automatically find the best machine learning schemes: combinations of data pre-processing, attribute selection, and learning algorithms. This genetic approach is compared against an exhaustive framework, and the proposal outperforms this baseline. Based on this study, Quesada-López et al. [33] perform an exhaustive evaluation of these previous schemes on the ISBSG dataset, using different models for projects measured in IFPUG FPA and COSMIC FFP, and performing manual hyper-parameter tuning for the learning algorithms. In their study, they compare complete models against simplified models using functional size in terms of their accuracy. Both studies reach similar results regarding the learning algorithms: regression-based techniques tend to perform well for the majority of scenarios.

Optimally, results obtained by these frameworks should be comparable. To achieve this, baseline metrics can be incorporated into these frameworks, allowing comparison across different studies.

### 2.4.1 Baseline SEE frameworks

A recurring problem in SEE literature using machine learning is that—across all SEE studies—there is no optimal technique, but rather each study obtains different conclusions; even ones that are contradictory with previous studies, depending in the dataset that is used [18]. Thus, a fair, unbiased way to compare different prediction systems for SEE was necessary, even across different machine learning processes [41]. Several studies have proposed evaluation frameworks that employ a baseline, which allows the comparison of results across different SEE studies.

Initially, effort estimation studies that used baselines employed statistical metrics of the predicted variable. For instance, Jørgensen [34] employed as a baseline the mean productivity multiplied by estimated size. In other study, Mendes and Kitchenham employ the sample median as a baseline [16].

One of the first SEE frameworks to propose a baseline based on an estimation technique was done by Shepperd and MacDonnel [41]. In their proposal, they suggest using random guessing—from all of the target metrics, taking one value randomly—as a baseline estimator  $p_0$ . For all of the evaluated schemes  $p_i$ , an accuracy metric such as the mean absolute residual (MAR, also known as mean absolute error or MAE) is

collected. This accuracy metric will depend on the data set that was used, and could vary between studies. Shepperd and MacDonnel suggest using the same accuracy metric on the baseline predictor  $p_0$  to create a baseline metric. The study proposes two baseline metrics: standardized accuracy (SA) and an adaptation of Glass's effect size ( $\Delta$ ). Standardized accuracy is a ratio of how much any estimator  $p_i$  is better than randomly guessing values of the dataset. Because an obtained SA value could be due to factors other than the estimators, Shepperd and MacDonnel suggest using the effect size  $\Delta$  as a metric of how significant is the relationship (difference) between the estimator  $p_i$  and the baseline estimator  $p_0$ .

Based on Shepperd and MacDonnel, Langdon et al. [42] propose an extension to the baseline estimation metric  $MAR_{p_0}$ , used to calculate SA. In their work, they argument that the baseline estimator  $p_0$  is biased, as it has a tendency of randomly encountering the right answer by chance and thus overestimates. While this would be no problem in large data sets, as  $MAR_{p_0}$  would converge into the mean value, effort estimation data sets are often small. Langdon et al. propose using a deterministic version of  $MAR_{p_0}$  when the data set has less than these 2000 observations (to produce an unbiased, denoised baseline), and Shepperd and MacDonnel's proposed  $MAR_{p_0}$  when the data set has enough observations. This version of the Standardized Accuracy would become one of the most used metrics in the later SEE framework proposals, and the basis for following baseline proposals. For example, since the year 2015 this metric has been employed by Minku et al. [91, 89, 12], Murillo-Morera et al. [90], Quesada-López et al. [33], among others.

Dolado et al. [43] propose another baseline metric, based on Shepperd and MacDonnel's SA. The novel metric is based on the concept of hypothesis testing, particularly intervals of equivalence. They define the minimum interval of equivalence (MIE) as the interval in which difference of the values of a metric (such as SA) of two predictors would need to fall into, such that they can be considered equal (i.e. having the same accuracy). Based on this, they propose the MIEratio: a metric that measures how far apart is an estimator from the baseline prediction  $p_0$ . While SA measures how many times is the predictor better than the baseline, the MIEratio measures how close is the prediction to the real values.

Lavazza and Morasca [44] propose their own baseline metric, based on Shepperd and MacDonnel [41] and on Langdon et al. [42] In contrast to the standardized accuracy, which is calculated from the average errors of two estimators, the individual absolute residuals (IARA) is a metric that compares paired predictions, such as those between the prediction model and the baseline model, and measures the probability

of them having different performance.

Usage of these metrics would allow two independent studies to be compared, assuming they employ the same data and baseline. This is an improvement over the previous state of the art, in which the way to compare two models was through a replication study. In addition, currently there are baseline metrics for both single point estimates and interval estimates.

## 2.5 Machine learning

This section introduces the concept and theoretical foundations of machine learning, ending with some of the open problems and considerations.

Machine learning is, in essence, programming computers so that they are able to “learn” from data inputted to them [11]. In the context of ML, learning refers to converting experience-based information into knowledge, a process called generalization [92]. There are two machine learning approaches for generalization: instance-based learning and model-based learning [92]. Instance-based learning functions by memorization of the data, and generalizes this data to new cases by using a similarity measure. Model-based learning builds an intermediate representation of the data: a model. This model is used in conjunction with a set of prediction rules to generalize to new instances. Machine learning algorithms cannot purely rely on information; and instead a balance between domain knowledge and information is necessary to construct a learner that generalizes to both known and unseen data [11].

Learning problems cover a wide domain, but can be generally classified by 4 dimensions: interaction, role, teaching, and protocol [11]. The interaction dimension refers to the interaction between the learner and the environment from which it learns. There are two types of interaction: supervised learning and unsupervised learning. Supervised learning is performed with a specific goal or task, while unsupervised learning is performed in an exploratory manner to summarize or group data.

The role dimension refers to the role played by the learner in the process [11]. Similarly, there are two roles: active and passive. An active learner directly interacts with the environment at the time of training. Particularly, active participation involves altering factors in the environment and measuring changes in the relevant variables. As such, active learners are closer in process to an experimental design. In contrast, a passive learner does not influence the environment, and limits itself to only observing.

In empirical research, passive learning is closer to a case study.

The teaching dimension refers to the entity that oversees or guides the learning process [11]. In general terms, there are two types of teaching: passive teacher, and adversarial teacher. A passive teacher presents data as-is, without any alterations. In the case of machine learning, such data is usually obtained from a random process. An adversarial teacher follows an iterative process, in which the teacher adds noise in order to “mislead” the learner. The objective of this type of teacher is to perform in a worst-case scenario basis, so that the learner is able to differentiate even in the most difficult cases.

The protocol dimension refers to the timing of the learning [11]. Again, two types of learning protocols exist: online and batch. Online learning can be better described as a ‘real-time’ exercise, in which the learner is presented information, and has only a limited amount of time to process it and generate knowledge. This knowledge is then validated and the cycle repeats itself. A batch learner, in contrast, has a larger amount of training data, as well as more time to process it before generating knowledge.

While machine learning can cover any combination of these four learning dimensions, it favors itself more closely to passive role learning using a batch learning protocol [11]. So, in general, most machine learning problems are either classified as supervised learning or unsupervised learning. Supervised learning is the main approach in the context of software effort estimation.

Most machine learning process can be described with a simple learning framework. Four elements comprise this framework [11]: input, output, model, and measures (or metrics). Figure 2.2 shows an example of this framework. The learning process starts with training the learner with the input data so that it is able to generate an output that describes the data correctly. This description is usually referred to as a rule that maps the input data to a specific feature or value: a prediction. To assess the effectiveness of this description, measures of success are used to determine the degree of errors in the prediction.

The input of the learning model refers to the data that the model has available to make its prediction [11]. The training data is comprised by a set of instances: cases of the problem at hand being studied. Each instance is defined as a vector of features. Each feature represents a characteristic or attribute of the instance. In general terms, there are two types of features: numerical and categorical. Numerical features correspond to a numerical value, either discrete or finite and optionally within a defined range. Categorical features correspond to non-numerical labels that function

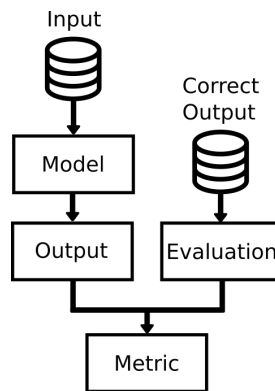


Figure 2.2: Simple machine learning framework [11].

to group data. The input data is often seen as a matrix, in which rows correspond to the instances and columns to the features. In supervised learning, one or more such features are chosen as the target feature(s), which the model aims to describe or predict in terms of the rest.

The output of the learning model refers to a prediction or classification rule that is the result of the model training [11]. The goal of the training process is to obtain this output, so that it be used to predict the target features for new data. Usually, the predicted features are hard to obtain, or can only be obtained at the end of a process.

The learning model proper is a representation or hypothesis on how does the studied phenomena functions, particularly how the features explain or relate to the target features [11]. At the beginning of the learning process, the model starts as an empty or with a random hypothesis. The learning process modifies and refines this model so it better explains the relation between features. Because the data obtained is subject to some degree of randomness (or noise), it can be represented using a statistical model. A learning model thus attempts to infer or represent this statistical model.

The measures of success are a metric of how effective is the prediction produced by the trained model [11]. The error of a classifier is the probability that it does not predict the correct value (either categorical or numerical) of a new instance.

A machine learning model produces as its output a prediction of one or more target features, based on input information [11]. Depending on the type of the target feature, a machine learning problem may be classified as either a classification problem or a regression problem. The type of the task limits the applicable problems, as there are classification and regression models.

A classification task deals with determining a categorical feature of a new instance.



As its name implies, classification tasks deals with problems such as categorizing or labeling new data. A regression task deals with predicting a numerical feature of a new instance. The machine learning model creates a pattern or function that related the input variables with the target variable.

**Insufficient quantity of training data** Training a machine learning algorithm requires a large amount of instances. Many real world problems, such as natural language processing, are subject to complex and intricate interactions of factors. As such, these phenomena can only be explained with complex models or rules. However, the data used to train a model may not be representative enough of the real-world problem [93]. No matter how powerful the machine learning algorithm is, it will not generalize well enough to unseen cases if the amount of data is too small with respect to the complexity of the real-problem. Thus, it is important to balance the amount of resources used to develop a machine learning approach against the amount of resources used to gather new data [93, 92].

**Overfitting and underfitting** A machine learning model may acquire an excellent performance in the training data, but it also might be unsuitable to new data. This phenomenon is called overfitting, and it happens when the model is overly trained with a noisy or small dataset [92]. For instance, a model trained with noisy data may detect patterns on uninformative features; thus drawing relations or conclusions that do not generalize well to real, unseen data. Some solutions to this problem include reducing the amount of features used to train, increasing the amount of training data, or correcting the noise in the training data. Another approach is to select a simpler model; for example using a linear regression instead of a polynomial regression.

Similarly, underfitting is a phenomenon that happens when the model is too simple to explain the underlying structure of the data [92]. Solutions to this problem include selecting a more complex model, or by “releasing” the constraints of the model, such as increasing the value of a hyper-parameter.

**Testing and validation** The only way to ‘truly’ evaluate a ML model is to try to generalize unseen cases, using this model to predict new data. However, releasing a model with unknown performance is a risk. Instead, the model should be first evaluated (and adjusted, if necessary). One approach to do this is to split the data into two sets: training and validation [92]. As its name implies, the data is trained using the training set. The validation set functions as this unseen, novel data that

can be used to evaluate the model. This approach emulates the release of the model into the world to evaluate its performance, but without the risk of using an untested model in a real environment. This can be used to detect overfitting: when training error is low, but testing error is high. Similarly, underfitting can be detected when model performance is bad in the training set.

**The no-free-lunch theorem** The no-free-lunch theorem states that there is no universal learning model that fits all problems or scenarios [11]. Formally, the theorem states that, for each learner, there is at least one task in which that learner fails. That task can be, however, accomplished by another learner. Thus, different learning models function for different problems or different contexts. The no-free-lunch theorem is related to another issue named the bias-complexity tradeoff [11]. This tradeoff deals with the benefits of using no prior knowledge against using prior knowledge. Using too much domain knowledge in a machine learning problem biases the learner (as it potentially misses important feature relationships) and could result in underfitting. Using no domain knowledge would make the learner find any explanation for the target feature, which could result in overfitting.

The following subsections introduce each of the components used in the application of machine learning. Section 2.5.1 contains data transformations that are commonly applied to datasets. Section 2.5.2 introduces attribute (feature) selection techniques. Section 2.5.3 shows the concept clustering and how can it be applied to machine learning. Section 2.5.4 covers some of the validation approaches used to maintain conclusion stability. Section 2.5.5 details some of the most used machine learning algorithms in the literature.

### 2.5.1 Data transformation techniques

Data transformation (also known as feature manipulation or normalization) are simple transformations applied to each one of the features of a dataset, with the objective of decreasing approximation errors of a particular model [11]. Data transformations are also applied so that the data satisfies requisites of learners, most notably the constraint of following a normal distribution. Another reason for applying data transformation is to transform each feature to the same scale. However, in some cases transformation might actually decrease performance. Thus, the transformation that should be applied to the data will depend on the context and the model being used. Examples of data transformations are centering, unit range, standardization, clipping,

sigmoidal, and logarithmic.

**Centering:** Centering transforms each feature so its mean is 0, which is achieved by subtracting the mean to each value:  $y_i = y_i - \bar{y}$ , being  $\bar{y}$  the mean of feature  $y$ .

**Unit range:** The unit range transformation, also known as normalization or min-maxing, bounds a feature to a numeric interval. For example, the unit range 0 to 1, and unit range -1 to 1 changes each feature to either of the ranges  $[0, 1]$  or  $[-1, 1]$ . The first transformation is achieved with the formula  $y_i = (y_i - y_{min}) / (y_{max} - y_{min})$ , while the second one is achieved with  $y_i = 2 * (y_i - y_{min}) / (y_{max} - y_{min}) - 1$ .

**Standardization:** Standardization converts each feature into a standard normal distribution by applying the transformation  $y_i = (y_i - \bar{y}) / s_y$ , being  $\bar{y}$  the mean of feature  $y$  and  $s_y$  the standard deviation of feature  $y$ .

**Clipping:** Clipping functions similarly to the minimum or maximum feature. For each feature, the user specifies a maximum value  $b$ . Each instance with a value higher than  $b$  is set to  $b$ :  $y_i = \max(y_i, b)$ . The same transformation can be applied for a minimum value  $c$ :  $y_i = \min(y_i, c)$ .

**Sigmoidal:** The sigmoidal transformation applies the sigmoid function to each feature, which functions as a “soft” version of clipping:  $y_i = (1 + \exp(-by_i))^{-1}$ , having  $b$  as an user hyper-parameter.

**Logarithmic:** The logarithmic transformation applies the natural logarithm function to each feature:  $y_i = \log(b + y_i)$ , being  $b$  an user hyper-parameter. This transformation is used to transform data that has an exponential pattern to instead resemble a normal distribution. Moreover, the logarithm spreads out data points that are too close together.

**One-hot encoding:** Also known as one-of encoding, this transformation converts a categorical feature with  $n$  unique values into  $n$  binary (numerical) features, each unique value associated with a column. For each project, the column corresponding to the original value is set to 1 and the rest to 0.

**Binary encoding:** Similarly to one-hot encoding, the binary encoding transformation converts a categorical feature into a set of binary features. However, binary encoding reduces the amount of new columns by encoding each unique value

into a binary number. A categorical feature with  $n$  unique values is converted into  $\lceil \log_2(n) \rceil$  binary features.

**Binning:** The binning transformation converts a numerical feature into a category, by dividing the feature in ranges and assigning values. For example, features above certain threshold can be classified as “high”, and those below as “low”.

**Box Cox:** The Box Cox transformation shapes numerical features to resemble a normal distribution. The features are transformed using the function  $y' = (y^\lambda - 1)/\lambda$ , where  $\lambda$  is an adjustment factor that has to be found for each feature. Values of  $\lambda$  can range from -5 to 5. If  $\lambda = 0$ , then the transformation  $y' = \ln(y)$  is used.

**K-means clustering:** The K-means clustering algorithm is presented and explained in section 2.5.3. While clustering algorithms can be used to divide a dataset, they can also function to generate a new feature. In the case of using K-means clustering as a data transformation, each project is assigned a number or identified of the cluster they belong to.

## 2.5.2 Feature selection techniques

Not all features on a dataset are relevant or informative to the machine learning task. Using a dataset with only a minimal subset of features brings benefits such as a reduction in the required training time and memory, and reducing the possibility of overfitting. Because less information is used to build the model, there is some impact to the accuracy of the model.

Feature (or attribute) selection is an optimization problem that attempts to find a minimal subset of features that have good performance compared to a model using the full dataset [11]. Moreover, reducing or removing the non-informative features can make it easier for the machine learning algorithm to identify the underlying pattern in the data [92]. One approach for feature selection is to try all possible subsets of features, train the model, and evaluate which one has the best performance. However, this exhaustive search is often not computable in reasonable time. Thus, feature selection methods have been developed to search for a good (although not optimal) feature subset.

There are three general approaches for attribute selection: filters, wrappers, and embedded [94, 95]. Filter methods select a subset of all features, based only on

properties of the data. In most cases, each feature is evaluated using a score, and only those features with high scores are selected. Advantages of filter methods include their computational speed and simplicity, and good scaling to higher-dimensionality datasets. In addition, filter methods need only to be performed once, and then the new dataset can be used in multiple models.

In contrast, wrapper methods work in conjunction with the learning model. Wrapper techniques rely on the selection of a subset of data, training the model with said subset, and evaluating the model; repeating with different subsets until a solution is found. In other words, wrappers are filters that use the model accuracy as their training function. Wrapper methods have the advantage of taking into consideration the interactions of the data and the model, as well as interaction between features. Thus, wrappers could potentially achieve a subset that fits better for each individual model, rather than one that fits “good” for all models. However, wrappers require more computational resources and have the problem of overfitting. Examples of attribute selection techniques include Pearson’s correlation, forward selection, and backward elimination.

Embedded methods combine the advantages of both filters and wrappers by performing feature selection in conjunction with training of the model [94]. Embedded methods retain the benefits of being able to find the best feature set for the machine learning algorithm of wrappers, while also retaining the higher speeds of filters. Examples of embedded methods are tree-based machine learning models like Classification and Regression Trees (CART), as these models perform feature discrimination as part of their training process.

**Exhaustive Search:** The most basic approach to feature selection is exhaustive search, in which every possible combination of features (i.e. power set) is tried. A model is built using each combination, and the one that maximizes a metric (or minimizes a loss function) is selected. The approach is exhaustive, as there are  $2^n$  combinations for  $n$  features.

**Variance threshold:** Variance thresholding is a basic feature selection approach, based on the idea that low variance features could be less useful than high variance features [96]. A feature with a single value (univariate) is not useful to predict effort, while a feature with a wider variety of values may aid in the identification of cases in which effort may be high, low, or in between. The variance threshold method calculates the variance of each feature, and then drops all those whose variance is under the threshold.

**Pearson’s correlation:** An example of a filter technique is the Pearson’s correlation coefficient [11]. This metric assigns a score to each feature based on how well it fits a linear relationship with the target feature. Most features would, however, not follow a linear relationship with the target variable alone; but could potentially do so in conjunction with other features. The technique can take this into consideration and also evaluates how well each feature along with other features fit a linear relationship with the target variable.

**Forward selection:** One example of a wrapper technique is forward selection [11]. Forward selection starts having a set with no features. Then, for each not included feature, it builds a model with the current set and that feature, and evaluates its performance. The feature which improves the performance the most is added to the set. This process is then repeated until a stop condition is reached—usually number of features.

**Backward elimination:** Another example of a wrapper technique is backward elimination, which is a greedy search algorithm [11]. Backward elimination functions by having the entire subset as the starting point, and removing one feature at a time. The subset with the best performance is then selected, and the algorithm repeats itself recursively, until a stop condition is reached.

**Genetic algorithm:** Genetic algorithms can be adapted for feature selection. In such cases, a chromosome contains a binary digit for each feature, which represents if the feature is included or excluded. The genetic algorithm then finds the feature combinations that maximize some function (i.e. accuracy).

**RReliefF:** Relief for regression, known as RReliefF, is a feature selection algorithm that identifies and considers the interaction between features. To accomplish this, the algorithm randomly selects data instances and determines the “closest” neighbors through a similarity function. For each feature, the values of the instance and their neighbors are compared. If the values are similar, the weight of the feature increases, and the weight decreases if they are similar. Only those features with their final weight above a threshold are selected.

**Particle Swarm Optimization:** The particle swarm optimization is presented and explained in section 2.5.6. Similar to clustering approaches, a set of features can be represented using binary encoding and PSO can be used to find an optimal set of features.

**Principal Component Analysis:** More than a feature selection algorithm, principal component analysis functions as a way to represent the data using less features. The technique works by transforming the data to a new coordinate system. The most meaningful “vector” (the one in which data varies the most) is used as the first dimension. The second and further dimensions are similarly obtained by “removing” the previous dimensions from the data and determining the vector for which the data has the most variance. This can be repeated an amount of times equal to the number of features. In general terms, the first features hold more information (variance) than the last ones, and thus less PCA features can be used.

### 2.5.3 Clustering techniques

Exploratory data analysis is an exercise performed previous to any machine learning application. The objective of such analysis is to become better acquainted with the data and its properties, in order to take better decisions on how to approach this data. One method of exploratory analysis is clustering, which consists of grouping a set of data instances—projects, in SEE—such that similar instances are in the same group, and different instances in different groups [11]. Most clustering algorithms are based on the concept of ‘distance’ between data points. This distance represents the notion that similar projects are ‘closer together’ and different projects are ‘farther away’. These distances can be defined mathematically, using functions. Examples of these are the Euclidean, Manhattan, and Minkowsky distance.

Clustering can be used as a part of the machine learning process, post exploratory analysis. One such utility of clustering is to approach the problem of heterogeneity in the data. In the case of effort estimation, data heterogeneity can come from differences between project instances. For example, if project data is taken from two different companies, A and B, they may follow certain patterns or tendencies. It is possible that the new company for which the estimation is being performed, C, has conditions more similar to A. A prediction model built using both data from companies A and B would induce prediction errors due to the differences between companies B and C; and it would be possible that a model using only data from company A would have better performance. A solution to this issue consist of employing clustering to split data into two separate sub sets, train two separate models with each sub set, and test which model offers better accuracy [89]. Some examples of clustering techniques are linkage-based clustering, k-means clustering, and spectral clustering.

**Linkage-based clustering:** Linkage-based algorithms are the intuitive approach of clustering. Each data point starts as its own cluster, and iteratively the closest clusters are joined together. Each round the amount of clusters decreases, and the amount of data points per cluster increases. If the algorithm continued unbounded, it would reach one cluster which contains all of the data points. To avoid this, a condition to stop the algorithm (such as number of clusters) must be determined.

**K-means clustering:** K-means clustering is an approach to split a set of data into  $k$  different groups, so that some cost (i.e. distance) is minimized. Given that this problem is NP-hard, there are several algorithms that approximate a solution. One such algorithm consists of creating  $k$  cluster centroids randomly in the data space, and cyclically repeat 1) assigning each data point to the closest cluster and 2) re-setting each centroid as the mean of all data points in the cluster. This is repeated until a certain number of iterations, or until no changes occur.

**Spectral clustering:** Another approach for clustering problems is using a graph notation: each vertex representing a data point, and each edge containing the distance between a pair of vertices. With this representation, a clustering problem is reduced into finding a partition of the graph that minimizes within-group weights and maximizes between-group weights. A similarity graph—higher weights indicating more similar instances—can be used for the same purpose, reversing the goals of the partition algorithm. This approach is known as spectral clustering.

## 2.5.4 Validation approaches

A machine learning task can result in multiple models, all of which can have very similar performance metrics. Nonetheless, we are interested in selecting the one that consistently has the highest generalization capabilities. In addition, we can increase the generalization and performance of the model, as well as reducing overfitting, by adjusting the hyper-parameters of the algorithm [14, 21]. However, tuning of these hyper-parameters results in selecting the best values for the particular data, and may not generalize well to unseen data.

A solution to this problem is through use of a validation approach. The validation approach—also known as cross-validation—relies on evaluating the trained model using new, unseen data. Because we are working with a bounded dataset, the vali-



validation approach splits the dataset into two sets: the training set and the validation set. After defining these subsets, all models are trained using the training set, and their performance is measured using the validation set. Some examples of validation techniques are hold-out set, k-fold cross validation, and train-validation-test split.

One additional problem may arise when training a machine learning model. The data used to train the model may not be representative of the real world: a problem known as data mismatch [92]. This problem can be detected by using data that is as representative as possible for the validation set. However, there are now two possible explanations for high accuracy on the training set and low on the validation set: either overfitting or data mismatch. A solution to this problem is to use another split of the training set, called the development set [92]. If the model does not fit to the development set, the problem is due to overfitting, as both the training and development set have the same data origin. If the model does not fit to the validation set, the problem is due to data mismatch, as the validation set uses real-world data.

**Hold-out set:** The hold-out set validation scheme, also known as hold-out validation, is the simplest way to perform validation. This is the same scheme described in general for validation: the dataset is randomly partitioned into two parts: training and validation. The training set is inputted to the model, while the validation set is held out of the learning process. The model is then used to predict, using the information in the validation set. The predicted values are then compared to the true values in the dataset. The validation set can also be referred to as training set.

**K-fold cross validation:** The k-fold cross validation approach is used to make a more “efficient” use of the available data. The original dataset is split into  $k$  subsets (folds) of similar size. One fold is taken as the validation set, and the remainders are taken as the training set. The same training and evaluation process as hold-out set is then applied, resulting in one set accuracy metrics. This process is then repeated with the remaining  $k - 1$  folds, each acting as the validation set once. This would result in  $k$  sets of accuracy metrics, which are averaged to obtain the final accuracy metrics. The special case in which  $k$  equals the number of instances in the dataset is called leave-one-out cross validation.

**Train-development-validation split:** The train-validation-test split approach, as its name suggests, splits the data into three sets with those names. The first two sets, training and development, are used to train and select the best model among

all possible options while avoiding the problem of over and underfitting. Once the best model is selected, its ‘true’ accuracy is evaluated using the validation set. This final evaluation mitigates the risk of data mismatch, as well as offering an estimate of its performance in unseen data.

**Bootstrapping:** Bootstrapping is a cross-validation approach that consists of re-sampling the dataset with replacement [97]. For a dataset with  $n$  projects, out-of-sample bootstrapping samples  $n$  projects with replacement, meaning that a project can be selected more than once. The sampled projects are then assigned to the training set, and those projects that were not sampled conform the testing set. On average, the test set will contain 36.8% of the total projects. This procedure can be repeated multiple times, each time with a different sample of training projects. Repeated bootstrapping has the advantage of determining confidence intervals of an estimator, as well as being more stable and least biased than other cross-validation methods [98].

**Online:** While not strictly a cross-validation approach, online effort estimation can be viewed and treated as such. Online effort estimation works by simulating a real estimation environment, in which a development entity has an initial set of projects which are used to predict effort for a new project. When this new project is completed, it can join the project database and work to predict future projects.

### 2.5.5 Machine learning algorithms

The central part of the machine learning process is the machine learning algorithm or model. A machine learning model has a base algorithm, which has a base hypothesis or formula to estimate effort, and an adjustment or training algorithm. The specific model instance is a set of weights or model parameters related to this formula. Estimation of effort is produced by combining the input data and model parameters into this formula. The training process of the model uses the algorithm to perform predictions on the training data, and adjusts the model parameters of the model to minimize the prediction error [11]. Machine learning algorithms can be customized further by setting values known as hyper-parameters, which adjust the process of training the model [21]. Examples of machine learning models are linear models, k-nearest neighbors, decision trees, support vector machines, and neural networks.

**Linear models:** Linear models—also known as linear predictors—are one of the most

useful families of hypothesis classes [11]. Many of the most complex learning algorithms are based on linear models, or use linear models as a part of themselves. Linear models assume there is a function that maps the features to the target feature. Such function can be linear, polynomial, exponential/logarithmic, or take any form. For this reason, multiple linear models exist. These techniques, based on the input data, try to determine the optimal values for the underlying function that best predict the target variable. For example, in the case of a linear regression, the technique models the problem as a linear function:  $y_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_n x_{ni}$ , being  $x_1, \dots, x_n$  the features of the data. The linear regression model will find the values for the variables  $\beta_0, \beta_1, \dots, \beta_n$  that minimize the prediction error. Other types of linear models include polynomial regression and logistic regressions.

**K-nearest neighbors:** The k-nearest neighbors (kNN) model functions on the basis that “things that look alike must be alike” [11]. This algorithm forms a dimensional feature space, each instance of the dataset being a point inside this space. When a new instance is given to the model, the model searches for one or more points in space that are the closest to the new point. The model then predicts the value of the new instance by combining the values of the existing points, usually using an average, median, or other function. A kNN model can be built using different definitions of distance, the most common being euclidean distance—the distance between two points in space. The number of instances  $k$  that the model uses to create the estimation can also be configured.

**Decision trees:** The decision tree model functions by the divide-and-conquer strategy [11]. A tree model functions by recursively splitting a dataset, so that homogeneous—in terms of the target feature—groups are formed. A tree model starts at its root node with the base dataset. The learner algorithm finds the feature that best splits the dataset according to the target feature—for instance, if the value to predict is a category, the split will try to make a group for each value. Two new subsets are formed: each becoming a child node of the current node. This definition is applied recursively until either all nodes become homogeneous, or some stopping criteria is reached.

**Support vector machines:** Support vector machines (SVM) are a type of model useful in high dimensional feature spaces [11]. This technique searches for a boundary that splits the data based on the target feature. In the case of categories, for example, the support vector machine will try to adjust this boundary

so it splits data by their category. The technique obtains its name because it searches for a set of vectors, the support vectors, that separate the data. Because not all problems are linearly separable, modern SVM implement a higher dimension transformation known as the kernel method, to have additional dimensions to find this separation margin.

**Neural networks:** The artificial neural network model is a machine learning technique inspired by the neurons of the brain [11]. A neural network is a graph whose edges correspond to neurons, and edges to connections between neurons. Each neuron has multiple inputs and one output. As an input, the neuron receives the output of other neurons, and outputs a signal. In addition, the neuron has a weight for each of its inputs, and a threshold value for its output. This signal is often computed as the weighted sum of the inputs of the neuron. If this signal exceeds the threshold value, the signal is sent; and if not, no signal is sent. Neural networks are trained by adjusting this set of weights, using an algorithm known as the backpropagation algorithm.

**Case Based Reasoning:** Case Based Reasoning is a family of techniques that determine the effort value of a project based on similarities with past projects. One such approach is analogy based estimation, in which the project(s) most similar (using a distance function) to the current are selected, and their effort value is combined (by averaging or using other function) to give an estimate.

**Naive Bayes:** A Bayesian model or Naive Bayes model is a probabilistic approach for machine learning. The model functions by determining the probability of the predicted value based on the value of the features.

**Ensembles:** Each machine learning technique has advantages and disadvantages, and functions differently depending on the characteristics of the dataset. Ensemble approaches try to offset this by using multiple machine learning models and combining their resulting effort values. An ensemble may be heterogeneous, in which each of their inner models are different, or homogeneous, meaning the ensemble uses the same model, but with different input data or hyper-parameters.

**Bagging:** Bootstrap **aggregating** is a specific type of homogeneous ensemble that tries to improve the stability of a base machine learning model. Bagging performs  $n$  bootstrap samplings of the original dataset and trains a models for each of them.

**Stacking:** Stacking is a specific type of ensemble that uses an additional machine learning algorithm as an aggregation function. After training the members of the ensemble, the second model is trained based on the predictions of these ensembles to predict the actual effort. Thus, the prediction method usually assigns weights to each of the models, depending on how accurate their results are.

## 2.5.6 Hyper-parameter tuning approaches

A machine learning model is given a set of data to train. This training consists of adjusting the internal values (parameters) of a machine learning model so that a metric, usually prediction error, is minimized. In contrast, hyper-parameters are values of the training algorithm (not the model) that must be set before the training begins, and remain constant through the training process [21, 92]. In a typical machine learning process, the researcher manually determines the value of these hyper-parameters, trains the model, and verifies that the prediction accuracy is satisfactory. If it is not, the process is repeated using a different set of hyper-parameters. When performed manually, this process is labor intensive and can take hundreds of iterations [21]. Examples of hyper-parameters include the amount of instances  $k$  for the k-Nearest Neighbors algorithm, the amount of layers in an Artificial Neural Network, and the kernel in a Support Vector Machine.

Researchers have proposed automatic selection methods for hyper-parameter values, to make machine learning algorithms more accessible [21]. These methods—also known as (hyper-)parameter tuning approaches—have the goal of quickly finding an effective combination of hyper-parameter values that maximizes accuracy or other metric. Hyper-parameter tuning approaches automatize the iterative process while requiring no skill set in machine learning, and can achieve equally good or better results than manual tuning [27, 99, 100].

Luo classifies hyper-parameter tuning approaches into two categories: independent of previous machine learning problems, and dependent on expert knowledge [21]. Independent methods iterate over a series of possible hyper-parameter configurations and selects the one which results in better accuracy. This involves building and training a model using these hyper-parameters. Dependent methods instead use information on previous machine learning problems (including algorithm, hyper-parameter values, data properties, and accuracy) to predict the best hyper-parameter configuration. This, in turn, is a machine learning problem of its own; but doesn't require

repeatedly training models with different settings.

Model parameters are adjusted in the training process automatically, but hyper-parameters can be manually adjusted or automatically tuned. As such, this thesis will use the terms hyper-parameters and parameters interchangeably, referring always to the hyper-parameters of the model. In addition, this thesis focuses solely on independent methods, to make the proposed methodology more accessible to software engineering practitioners, requiring less knowledge of machine learning. Examples of hyper-parameter tuning approaches used in SEE literature include grid search, random search, particle swarm optimization [21], genetic algorithms, Tabu search, supervised online tuning, Caret tuning, and Dodge.

**Grid search:** The grid search technique is an exhaustive approach of hyper-parameter tuning [29]. This technique requires the definition of a series of values for each hyper-parameter. Based on this, the method forms the search space by assembling each combination of values of these hyper-parameters.

**Random search:** Random search is a method based on grid search, which searches on a subset of the search space instead of all possible combinations. This technique provides a level of accuracy improvement similar to grid search, with the benefit of not being as computationally costly [29].

**Particle swarm optimization:** Particle swarm optimization (PSO) is an algorithm that simulates the behavior of birds flocking to find food in an area [77]. In PSO, each single solution (hyper-parameter values) is a particle in the search space. A fitness function is calculated for all of these particles, to determine which is closest to the optimal solution. After determining which is the closest, the remainder particles “fly” closer to the optimal solution. This process is repeated until a stop condition is reached, either quality of the solution, or number of iterations.

**Genetic algorithms:** Genetic algorithms (GA) are a global optimization method based on the theory of natural selection. GA generates an initial set of random individuals, called chromosomes. The ‘fitness’ of each individual is evaluated—usually by a metric or fitness function. A percentage of the most apt individuals is kept, and the remainder is discarded. A process of reproduction is then applied to the surviving individuals. A set of new individuals is generated by applying crossover and mutation operators to the current population. This process is repeated for several generations, or until a satisfactory solution is found [101].

In SEE research, Oliviera et al. [102] employ genetic algorithms to perform simultaneous feature selection and hyper-parameter tuning for machine learning algorithms. Their chromosome design is divided into two parts: the hyper-parameter values of the machine learning technique (as numerical or text values), and the input features of the dataset that will be used (as binary values).

**Compact genetic algorithm:** The compact genetic algorithm (CGA) was proposed as a less memory-intensive variation of the traditional GA [103]. Instead of storing all individuals of a population, the population itself is represented as a probabilistic vector from which the new individuals are sampled. In return, the best individuals of each generation modify the population vector, making the distribution more likely to contain similar individuals. While CGA offers less usage of memory, traditional GA can achieve better results when the user has knowledge of the problem space.

**1+1 Genetic algorithm:** One of the potential problems when employing a genetic algorithm approach is setting its parameter values [104]. In the case of tuning this goes twofold, as it would be necessary to configure the parameters of the optimizer, as well as the base model hyper-parameters. The one plus one (1+1) genetic algorithm simulates a chain of individuals, starting from a single value. The first individual generates a mutant offspring, and the best of the two values is conserved. This process is repeated, one individual at a time (hence its name). 1+1 is a self-adaptive method: instead of relying on a set value for mutation rate, the mutation rate used in each iteration is randomized, sampled from a distribution [105].

**Bayesian optimization:** Bayesian optimization (BO) is an iterative algorithm that determines the next point to be explored using the information obtained by previous hyper-parameters [99]. As an optimization method, BO finds a set of values that minimize an error function. BO employs a probabilistic model to predict the performance of future values using all available information. Advanced BO models balance the choice between exploration and exploitation when valuing the next point, where exploration refers to evaluating points away from the visited space, and exploitation to visit the most promising values as predicted by the model [106, 107].

**Tabu search:** Tabu search is a search algorithm proposed by Glover in order to overcome shortcomings of local search [108]. Tabu search requires four precon-



ditions: 1) the representation of possible solutions, 2) the definition of local transformations (i.e. moves) to explore neighbor solutions, 3) an objective function, and 4) the definition of the Tabu list size, and aspiration and termination criteria. The algorithm starts by exploring an initial solution and neighboring solutions, using the transformations. The current solutions are evaluated and the best of these is used in the next iteration to search for promising neighboring solutions. To avoid repetition and encourage diversity of solutions, recently visited solutions are marked as ‘taboo’ and are stored in a Tabu list. This is repeated until a stop condition is found, which usually is a certain amount of iterations. In software effort estimation, Coraza et al. have performed two studies [49, 22] on the impact of Tabu search for support vector regression.

**Harmony search:** Harmony search (HS) is a nature-based heuristic algorithm that mimics the search for musical harmony [109]. HS works in “improvisation sessions”, in which the algorithm determines a random set of hyper-parameters and evaluates their performance. A harmony memory keeps track of the best sessions encountered, and uses their values for future improvisations. To avoid local optima, a pitch adjustment mechanism may be introduced. Similar to the mutation rate of a genetic algorithm, the pitch adjustment rate determines with some probability if each selected value is shifted up or down.

**Supervised online tuning:** Minku [12] proposed the first hyper-parameter tuning procedure for online software effort estimation. The procedure accounts for changes in the project data as new project information is recorded. The online tuning approach uses a technique similar in essence to grid search: all possible hyper-parameter settings are used to build and evaluate an initial set of models. However, instead of keeping the best solution, all model instances are maintained. When new within-company projects are added to the database, all models are retrained with the new data, and the model with the best predictive performance for the new WC projects is chosen to estimate new projects. This instance is used to estimate effort of new projects, until new training data is received and the process is repeated.

**Dodge:** Agrawal et al. [52] propose the DODGE approach to as a simple approach that generates learners with accurate predictions, lowering the order of magnitude of runtime by avoiding redundant hyper-parameter settings. The algorithm is based on the concept of avoiding hyper-parameter settings whose performance score falls within  $\epsilon$  of previously explored tuners. Dodge employs



a weighted tree to store the explored hyper-parameters. This tree is initialized with  $N_1$  randomly sampled hyper-parameter values. These are evaluated, and those which fall within  $\epsilon$  of other settings are deprecated (their weight lowers) while the others are endorsed (their weight increases). For  $N_2$  iterations, the algorithm then explores mutations of those settings with higher weights. These mutations are basic operations on the existing hyper-parameters, such as increasing or decreasing a numerical parameter using a random value, or selecting the best categorical value of previously explored settings.

**Flash:** Flash is a sequential model-based optimization algorithm, similar to Bayesian optimization [54]. Such approaches are useful when a problem is unknown in nature, and consists of (1) analyzing the known information about the problem, and (2) selecting the next step using that information. Flash employs machine learning techniques, a classification and regression tree, to (1) model the behavior of the explored hyper-parameter space as well as to predict the values of new points. To determine which points are worth exploring next, Flash employs (2) random sampling of possible points in the search space, predicting their performance with the behavior model. Flash learns about the optimization problem and nature of the search space as it explores more values.

**Differential Evolution:** Differential evolution (DE) is a simple, yet effective evolutionary algorithm [110]. Initially, the algorithm evaluates random individuals from a population, and determines a “frontier” that includes the most fit individuals. Each iteration, DE generates a new individual from three randomly sampled members of the frontier, by mutating and combining their hyper-parameter values. The new individual is added to the frontier if its fitness is better than at least one of the individuals of the frontier.

**Hyperband:** The hyper-bandit (hyperband) algorithm balances fine- and coarse-grained exploration of a space. Hyperband is based on the concept of “halving a space”, exploring a set amount of evenly-distanced values and discarding the worst half [111]. The method divides the allocated exploration budget  $B$  into  $n$  pieces, evenly allocating the budget ( $B/n$ ). As there is a trade off of using a larger  $n$  (many configurations with less training) or a small  $n$  (few configurations with more training), hyperband uses successive halving to select the most promising  $n$  values.

**Beam search:** Beam search is originally an graph search algorithm that can be used

to prune nodes [112]. Based on the current position, the algorithm determines the most promising vertex by using breadth-first search, with a limited amount of explored edges per iteration. This technique has been adapted for hyper-parameter tuning by representing each possible model as a vertex in a graph, and each edge as a transformation [113].

**Bees algorithm:** The bees algorithm is a bio-inspired method that simulates the behavior of foraging bees. The algorithm uses a number of scout bees in the search space, and selects a subset of sizes (space around the scouts) to send additional bees to explore, which may find better or worse conditions. A subset of these bees is selected, and again additional bees are sent to explore nearby places. This process is repeated for an amount of iterations, a search budget is exhausted, or a threshold fitness is achieved.

**Hill climbing:** Hill climbing is a term for optimization techniques that start from an initial solution, and perform changes to that solution that improve fitness, until the solution cannot be improved. One such example in hyper-parameter tuning is Expectation-Maximization [114], which tunes a single hyper-parameter by starting from the lowest possible value. A model is constructed using the following value, and if the fitness increases, this process is repeated. The algorithm ends when increasing the value does not cause an increase in fitness.

**Immune algorithm:** The immune algorithm for hyper-parameter tuning [115] is a bio-inspired algorithm that represents candidate solutions as antibodies. An initial random set of antibodies is generated, and their affinity (similarity) is calculated against other antibodies and the antigen (target function). The antibodies with the highest affinity with other antibodies are removed to avoid duplicate solutions. Lastly, the antigens with the highest affinity with the antigens are proliferated with mutation. This process is then repeated for a set number of iterations.

**Satin bowerbird optimizer:** The satin bowerbird optimizer is a hyper-parameter tuning algorithm proposed for SEE [116], and it was designed for adaptive neuro-fuzzy inference system models. The algorithm is based on the way male satin bower birds attract females. An initial population of male birds is generated as random hyper-parameter values. The probability (fitness) of each male is calculated, and one is selected as the elite. The algorithm then performs a series of cycles until a stop condition is met. Inside the cycles, a new individual is created

for the population by combining them with other members. Every new and old individual is then mutated, and their fitness is re-calculated. At the end of each cycle, the best half of the population is kept, the remainders are discarded, and the elite solution is updated.

## **2.6 Empirical methodologies in software engineering**

This section presents the design science methodology (section 2.6.1) and the two empirical methodologies that will be used in this thesis: systematic mapping study (section 2.6.3), and controlled experiments (section 2.6.4).

### **2.6.1 Design science methodology for information systems and software engineering**

The design science methodology for information systems and software engineering was followed for the development of this thesis. Design science is a research methodology that focuses on the design and investigation of artifacts in a particular context [61]. Design science is composed of two parts, design and investigation, which correspond to the two research problems of design science, design problems and knowledge questions. Design problems are the necessity of a change in the real world, which require a design that satisfies some stakeholder goals. Knowledge questions are the necessity of knowledge of the current state of the world. A research problem on software engineering is a set of both design problems and knowledge questions.

To solve a research problem, design science employs two types of cycle: design cycles and empirical cycles. Design cycles propose artifacts that satisfy design problems, and empirical cycles propose answers that satisfy the knowledge questions. The design cycle is composed of three parts: problem investigation, treatment design and treatment validation. The procedure of such cycle is to find a potential point of improvement, propose a solution artifact, and evaluate how does it aid the problem.

In contrast, the empirical cycle is composed of five parts: research problem analysis, research and inference design, validation of design, and data analysis. The typical execution is to find and define the knowledge problem, propose how will this problem be solved, validate this solution, execute the proposed solution, and evaluate the knowledge obtained.

Figure 2.3 summarizes the design science approach and its general framework.

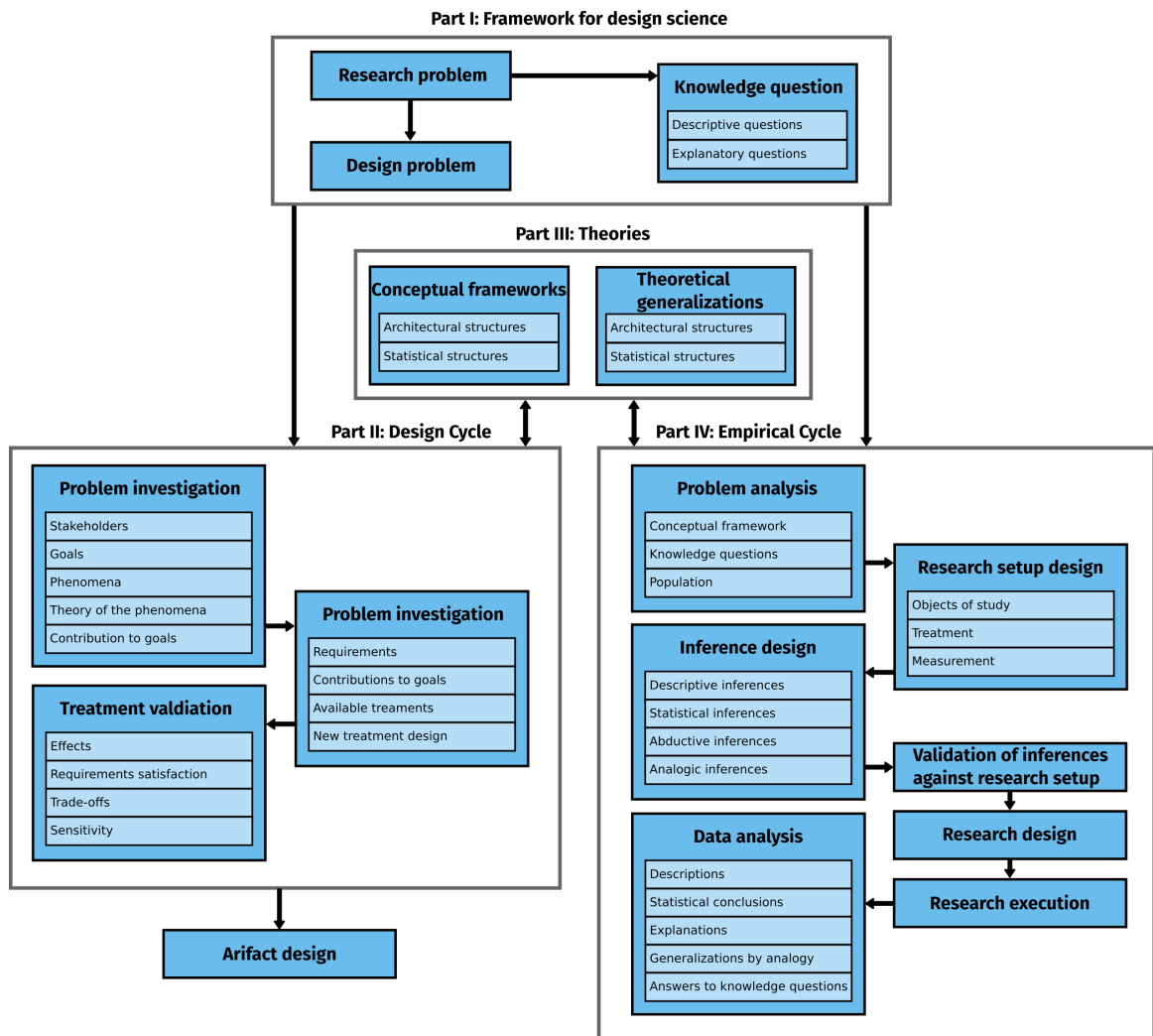


Figure 2.3: Summary of the design science approach. Adapted from Wieringa [61].

When a research problem is defined (Part I), the design problems are approached using the design cycle (Part II), and the knowledge questions are approached using the empirical cycle (Part IV). Both cycles are aided by background theories, and in turn both cycles can add to these theories (Part III).

In this thesis, we will approach the design problem of an implementation of a hyper-parameter tuning framework, and the knowledge problem of the efficacy assessment of the existing hyper-parameter tuning approaches. We will thus employ design cycles to implement this framework, and empirical cycles to evaluate the hyper-parameter tuning approaches using this framework.

## 2.6.2 Empirical software engineering

Research in software engineering can be performed by taking four methodical approaches: scientific, engineering, empirical, and analytical [117]. Empirical software engineering can be defined as a research method in which a model is proposed and evaluated through empirical studies—studies based on evidence.

Empirical software engineering divides research into primary studies and secondary studies. Primary studies are empirical studies that aggregate evidence to software engineering literature, while secondary studies compile the evidence of primary studies [64].

Secondary studies can be divided into systematic literature reviews and systematic literature mappings [64]. A systematic literature review is a study that collects and synthesises empirical evidence, guided by research questions that are focused on the outcomes of the primary studies. Systematic literature mappings are secondary studies that are similar in form to literature reviews, but the scope of the research questions is more general and oriented to determine the state of the art. We select the systematic literature mapping as the strategy for a study that is part of this thesis, as our objective is to unveil the state of the art in the existing hyper-parameter tuning approaches and other machine learning techniques.

Primary studies can be further divided into surveys, case studies, and experiments; depending on their level of control [64]. Surveys are an empirical study that is used to take a snapshot of a situation, and can be used, for example, to determine the effects of a new technique or tool. Case studies are conducted in a real environment to investigate a particular phenomenon, and is usually applied in industrial contexts. Experiments are an empirical strategy that is used when control over a situation is

desired, at the expense of some degree of realism. We select the experiment as the strategy of a study that is part of this thesis, as our objective is to compare hyperparameter tuning approaches under different circumstances. Such study requires high degree of control over the factors that affect the performance of these methods.

### 2.6.3 Systematic mapping studies

Systematic mapping studies are empirical studies that are designed to give an overview of a research area through classification and counting of research contributions [62]. They involve searching the literature in order to know what topics have been covered in the literature, and where the literature has been published.

Petersen et al.'s guidelines for conducting systematic literature mappings describe how to conduct a mapping process for empirical software engineering—covering the search, study selection, analysis, presentation of data, and validity evaluation, among others. While systematic literature reviews are intended to synthesize evidence for a specific research area, systematic literature mappings aim for structuring and summary of a broader research area [62]. Literature surveys and mapping overlap in protocol, but differ in the reach of their objectives and conclusions.

Kitchenham et al.'s guidelines for systematic literature reviews are the predecessor to Petersen et al.'s guidelines, and specify how to conduct a more in-depth review. In terms of steps and parts of the protocol, both guidelines offer the same content, but differ in depth of this content. Petersen et al.'s guidelines are built into the foundation of Kitchenham et al.'s guidelines, and thus we reference them in the mapping process.

The systematic study guidelines follow a three step process: (1) planning, (2) conducting, and (3) reporting the mapping process. Figure 2.4 summarizes this process.

**Planning** The mapping planning comprises the election of all decisions relevant for conducting the mapping, and results in a mapping protocol. The process starts with identifying a need—an open question or topic about a research area—and selecting an adequate scope to approach this need. After this identification, one or more pertinent research questions are presented. These questions will drive the mapping process, and the aim of the process should culminate in answering these questions. Following this, a strategy or protocol defines the strategy for identifying the relevant studies, assessing their quality, extracting relevant information, synthesizing the results, and evaluating potential threats to validity. Lastly, the protocol is validated among peers—to assess that the process is

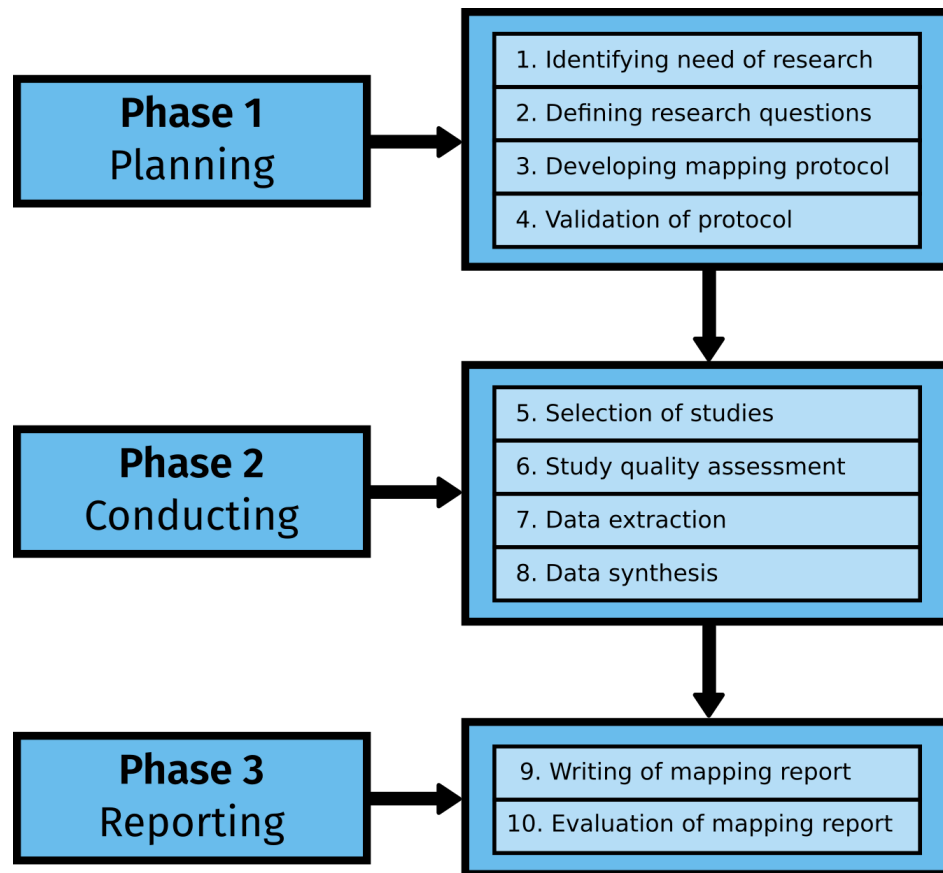


Figure 2.4: Systematic literature review and mapping process. Adapted from Kitchenham et al. [118].

unbiased and answers properly to the research questions.

**Conducting** The conduction of the mapping follows the process specified in the protocol to obtain the results that should satisfy the need for the mapping. Following the strategy defined in the protocol, the first step is the identification of those studies relevant to answer the research questions. The identified studies are then evaluated and ranked according to their quality and relevance regarding the mapping. A researcher may opt to exclude studies based on this criteria. Once identified and selected, the information that is relevant to the mapping is extracted from the studies. As this information should be described in a synthesized form, an analysis and presentation process is performed—usually with the aid of visualization. The mapping process is iterative: newfound information in the conduction may require changes in the planning, which would then require another iteration of the process.

**Reporting** The mapping reporting deals with the presentation of the mapping protocol and results in an orderly manner. This process culminates in a mapping report or research paper. One such way of presenting the results of a mapping would be: introduction, related work, research method, results, and discussion. Lastly, the report is evaluated among peers, usually aided with a checklist of all the content that the review should cover.

#### 2.6.4 Controlled experiments

A controlled experiment in software engineering is an empirical enquiry that manipulates one factor or variable of the studied setting. Based in randomization, different treatments are applied to or by different subjects, while keeping other variables constant, and measuring the effects on outcome variables. In human-oriented experiments, humans apply different treatments to objects, while in technology-oriented experiments, different technical treatments are applied to different objects [64].

Wohlin et al.'s [64] book on software experimentation describes how to conduct experiments (and quasi-experiments) for empirical software engineering. Compared to other types of empirical study, experiments are performed in controlled environments to compare the effect of different factors—or combinations of those—have an effect on a measurable outcome. Experiments have more control on both the execution and measurement of the experiment when compared to other types of empirical study, albeit they demand a higher investigation cost. A quasi experiment is an exper-



imental process in which subjects cannot be randomly assigned treatments. Empirical experiments describe how does a set of variables influence a particular phenomenon. A series of hypotheses on the influence of these variables is established at the beginning of the experiment, and the purpose of the experiment is to gather evidence to support or reject these hypotheses.

The experimental process consists of five activities: (1) scoping, (2) planning, (3) orientation, (4) analysis and interpretation, and (5) presentation and package. Similarly to iterative development, the experimental process is not a one-pass process. Rather, its nature is iterative, as newfound evidence may go against or require changes in the experimental design. Figure 2.5 shows the empirical experiment process.

**Scoping** The scoping activity defines the objectives and goals that will drive the experiment. The objective of this activity is defining a framework that defines: what is studied (objective), the intention of the study (purpose), which effect is studied (quality focus), from whose view is it studied (perspective), and where is the study conducted (purpose). In addition, a set of hypothesis may be proposed at this point.

**Planning** The planning activity lays out the foundation of the experiment. This activity produces an experimental design: a document that defines the details of what will be studied, how will the study be performed, and how does the experiment reduce bias or undesired effects in order to produce solid, conclusive evidence. A typical experimental design covers: context selection, hypothesis formulation, variables selection, selection of subjects, choice of design type, instrumentation, and validity evaluation.

**Operation** The operation activity consists in carrying out the experiment, particularly the process up to the recollection of data. It consists of three parts: preparation, execution, and data validation. The preparation step consists of preparing everything needed for the collection of data, including study subjects and instruments. The execution step deals with the collection of the data, and generally follows the design of the experiment. The data validation step ensures that the collected data is correct and provides a valid picture of the experiment.

**Analysis and interpretation** The analysis and interpretation activity has two objectives: describing the collected data, and drawing conclusions from said data. The activity is similarly comprised by descriptive statistics, data set reduction,

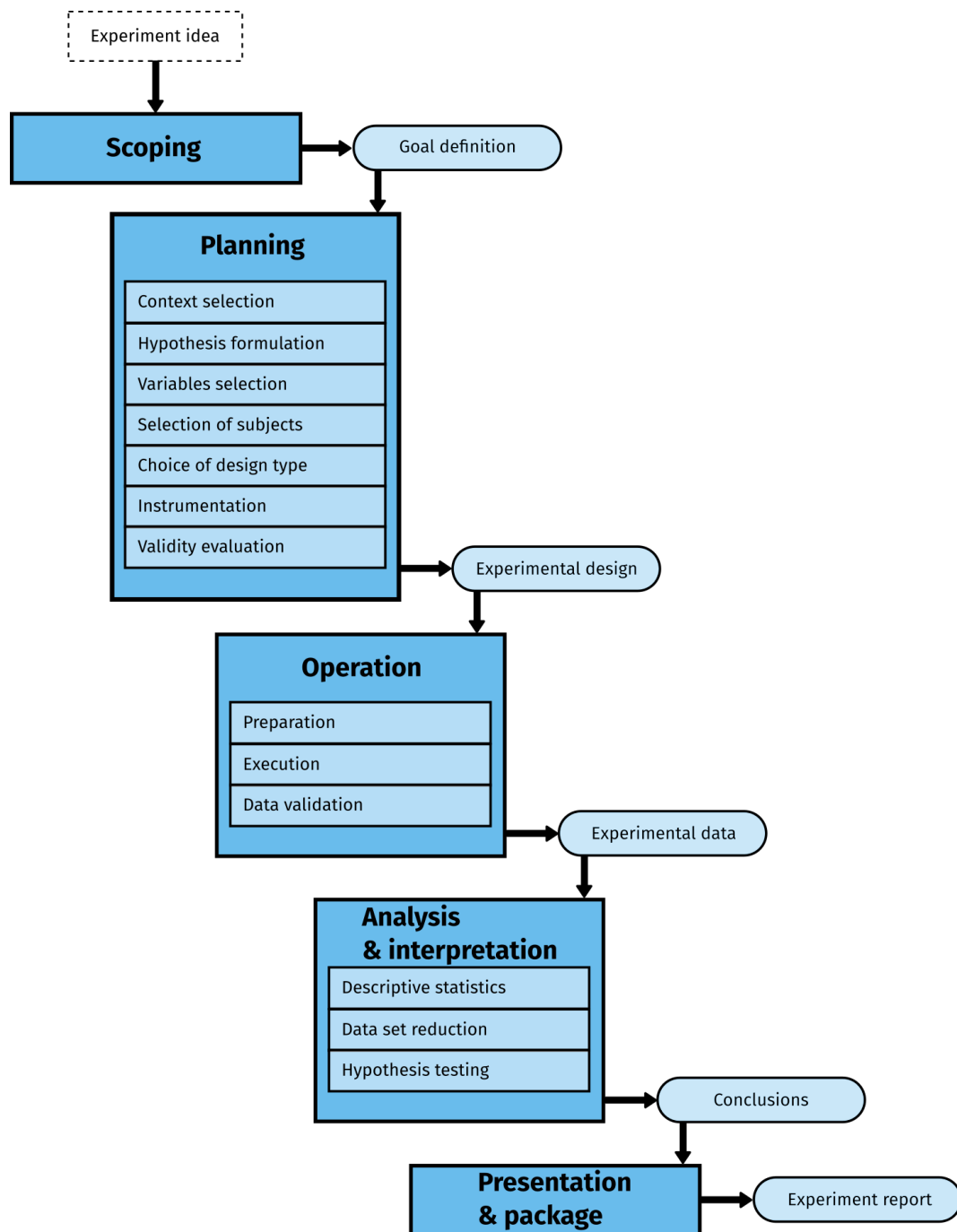


Figure 2.5: Complete empirical experiment process. Adapted from Wohlin et al. [64].

and hypothesis testing. A general overview and visualization of the data is first performed by using descriptive statistics, so the researcher can better understand the nature of this information and what can be concluded. After this process, it is usually concluded that only a subset of information is relevant for the hypothesis testing. Thus, a data set reduction process is performed, in which either features or data points are removed from the data set. Once this is performed, hypothesis testing can be performed. In this process, inferential statistics are used to test a series of hypotheses, mostly those described in the experimental design. One important aspect of this process is interpretation and reasoning on the results of this hypothesis testing, and how do these results impact future research.

**Presentation and package** The presentation and package activity of the process deals with presenting the results of the experiments, either in a publication paper, a report, presentation, or other medium. In addition to the results of the experiment, a report should also clearly present the experimental design to aid future replication.

## 2.7 Software engineering methodologies

A software process is a set of detailed activities that lead to the production of a software system [63]. All software processes contain four fundamental activities: specification, development, validation, and evolution. Because there are different types of software, there is no universal software process. Instead, multiple software processes are defined, and one should be selected accordingly to the nature of the software to be produced [63].

Sommerville's definition of incremental software development explain the process of a software development project that is performed in multiple iterations that are comprised by a waterfall-like process [63]. Each incremental iteration is comprised of specification, development and validation activities; with the goal of delivering a new version of the system. The incremental model is better for software with constantly changing requirements, which is apt for development parallel with the execution of the systematic literature mapping.

There are three activities performed in the incremental software engineering process: (1) specification, (2) design and implementation, and (3) validation. Figure 2.6 shows the incremental development process and its relation to these three activities.



Figure 2.6: Incremental development process. Adapted from Sommerville [63].

**Specification** Software specification—also known as requirements engineering—is the process of understanding and defining what new functionality or characteristics are required from the system. The general procedure for a specification activity is to collect, document and validate a set of requirements.

**Design and implementation** Software design and implementation is an activity that consists of two sub-activities: design and implementation. The software design is a description of the structure of the software to be implemented. The software implementation process converts this design into an usable system. These two activities may be preformed sequentially or interleaved.

**Validation** Software verification and validation is the process that demonstrates that the system both conforms its specification, and meets the expectations of the stakeholders. Software validation can be applied to both product and process. Systems should be tested on different levels: unit, integration, system, and acceptance. In the case of incremental development, new functionality should be tested as it is implemented.

## Chapter 3

# Hyper-parameter tuning for machine learning software effort estimation: a systematic literature mapping

This chapter presents the results of a systematic mapping study to address the first specific objective of this thesis: to characterize the existing machine learning hyper-parameter tuning approaches in the context of software effort estimation.

### 3.1 Study design

The systematic mapping study was conducted using the guidelines by Petersen et al. [62] and Kitchenham and Charters [119], as described in section 2.6.3. The objective of the mapping study was to characterize machine learning hyper-parameter tuning approaches for software effort estimation. We analyze them in terms of cross-validation approaches, data transformations, feature selectors, machine learning algorithms, parameter tuning approaches, datasets, evaluation metrics, and analysis techniques.

A preliminary version of this study [120] was published as a paper at the 4th International Conference on Information Technology & Systems (ICITS'21). This article is available on full text form on appendix C.

### 3.1.1 Research questions

Research questions of a mapping study aim to structure the current knowledge in the research area, and show the distribution of research papers [62]. These questions are answered by classifying the selected primary studies [62]. We proposed the following research questions:

**RQ1** Which approaches are used in machine learning hyper-parameter tuning for SEE?

To answer this question, we characterized the most used cross-validation approaches, data transformations, feature selectors, machine learning algorithms, and hyper-parameter tuning approaches in hyper-parameter tuning machine learning SEE.

**RQ2** What datasets are used in machine learning hyper-parameter tuning for SEE?

To answer this question, we identified the datasets that were used to evaluate hyper-parameter tuning machine learning SEE approaches.

**RQ3** What performance metrics are used in machine learning hyper-parameter tuning approaches for SEE?

To answer this question, we identified the most used evaluation metrics and analysis procedures to evaluate the performance of SEE approaches and determine the impact of hyper-parameter tuning.

### 3.1.2 Control studies

We defined a set of 15 control papers [8, 12, 14, 22, 24, 49, 50, 57, 102, 121, 122, 123, 124, 125, 126] to construct and validate the study design, as well as the results of the search process. These studies comprise primary evaluations of SEE that use one or more hyper-parameter tuning approaches. The studies were used to perform a preliminary version of the study design, aiding in the creation and validation of the current protocol.

### 3.1.3 Search strategy

The search strategy of a systematic literature mapping is a group of activities that allows selection of research that is relevant to answer the proposed research questions.

In our case, this strategy was built based on the research questions and the control studies. This study used an automated search process and inclusion and exclusion criteria to find the relevant primary articles.

To guide the automated search process, a PICO analysis was performed based on the keywords of the control studies [119]. Based on this analysis, we proposed a search string for the automated search. Table 3.1, shows the four PICO clusters and the search string.

Table 3.1: PICO clusters and search string.

Population	Software effort estimation, software cost estimation
Intervention	Machine learning, machine learning schemes, parameter tuning, parameter optimization
Comparison	–
Outcome	Data sets, cross-validation approaches, data transformation, feature selection, machine learning algorithms, parameter tuning approaches, evaluation metrics, results, challenges
Search string	("software" AND ("effort estimation" OR "effort prediction" OR "cost estimation" OR "cost prediction" ) AND ("tuning" OR "optim*" OR "setting*" OR "combinat*" OR "ensemble*" OR "scheme*"))

The automated search was performed in the following databases: Scopus, IEEE Xplore, Web of Science, and ScienceDirect. The string was adapted to the format of each database. Furthermore, results for Scopus were limited to those belonging to computer science or engineering.

### 3.1.4 Inclusion and exclusion criteria

The inclusion and exclusion (I/E) criteria determined which of the studies identified by the automated search process were relevant for this study. The I/E process evaluated each of the identified papers based on their title, abstract and keywords. If these elements did not provide enough information to decide whether to include or exclude a paper, this process was performed on the full paper text.

To be included in the mapping study, a paper must have met all of the inclusion criteria and none of the exclusion criteria. We defined four inclusion criteria: The paper is a primary study (I1), in the field of SEE (I2), that uses machine learning for SEE (I3), and uses hyper-parameter tuning (I4). We decided to exclude those papers that are not in English (E1), and whose full text is not available (E2).

The inclusion and exclusion criteria resulted in a total of 79 papers. The main

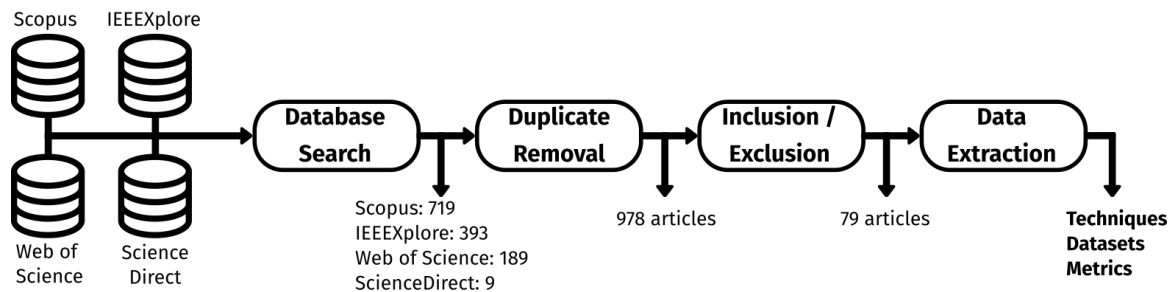


Figure 3.1: Systematic mapping study steps and results.

reason for the exclusion of studies in the process was the (I4) criterion, as the majority of papers did not use hyper-parameter tuning approaches; using default or hand-picked parameters, or not reporting how were hyper-parameters were selected.

### 3.1.5 Selection process

Figure 3.1 shows the steps and results of the systematic mapping process. The automated search process retrieved a total of 1,310 papers from the four databases. After performing an automated removal of duplicates (based on author, title, and year), the amount of studies was reduced to 978. The inclusion and exclusion criteria further dwindled this number to a total of 79 papers. Table A.1 in appendix A shows the complete list of articles, including their year, title, authors and publication venue. Each paper was assigned an ID consisting of the letter S plus a number (e.x. paper S1).

### 3.1.6 Quality assessment

As part of the recommended systematic literature mapping process, we evaluated the quality of the selected articles to determine their level of detail in reporting. We set five quality criteria in the form of questions: (Q1) Does the study report its goal or main objective? (Q2) Does the study report research questions? (Q3) Does the study report the datasets that were used? (Q4) Did the study measure accuracy with an unbiased metric? (Q5) Did the study analyze the obtained results? Each question was answered with using a Yes, Partial, or No scale; which granted 1, 0.5, and 0 points respectively. The total score was the sum of the scores of the five criteria, for a maximum possible score of 5 points. We did not use the quality score to exclude studies but it is a reference for researchers interested in the reporting quality of studies in SEE.



The distribution of the quality scores for the 79 papers ranged from 2 to 5, with a median score of 3.0, a mean score of 3.3, and a standard deviation of 0.94. This indicates that the studies have an acceptable level of quality to answer the research questions. The average lowest scores were on Q2 and Q4, which indicates that many studies did not report their research questions nor used unbiased metrics. Table A.2 in appendix A shows the individual and total scores per paper.

### 3.1.7 Data extraction and analysis

For each research question, the relevant information to answer it was manually extracted from the selected articles. Table 3.2 shows the categories and fields that were extracted from each paper, mapped to each research question. The data we obtained from each extraction field was numerical or categorical. Numerical data corresponds mainly to the results obtained per study, and the parameter values for the machine learning techniques. Categorical data corresponds to the names of the obtained techniques, datasets, evaluation metrics, and challenges. Figure 3.2 shows a classification scheme for the cross-validation approach, data transformation, feature selection, parameter tuning, dataset and evaluation metrics. These scheme was built using the keywording technique described by Petersen et al. [62] applied to the control studies. The studies were extracted in chronological order, from most recent to least recent.

Table 3.2: Data extraction fields.

RQ	Fields
General	Study type, main objective, research questions, study focus, online/offline estimation, type of estimation (CC/WC), experimental design, threats to validity, future work, main results
RQ1	Cross-validation approach, data transformations, feature selectors, machine learning algorithms, parameter tuning approaches; for each: sub-types, parameters, sub-techniques
RQ2	Datasets, origin, CC
RQ3	Evaluation metrics (category and sub-category), data analysis techniques

The following strategies were applied for analysis and synthesis of the extracted information. The information of each extracted element was tabulated into the data extraction form. An analysis process was performed on each category on elements to respond each research question.

To answer research question 1, we classified, grouped, and counted the techniques of the machine learning parameter tuning process that we encountered for each pa-

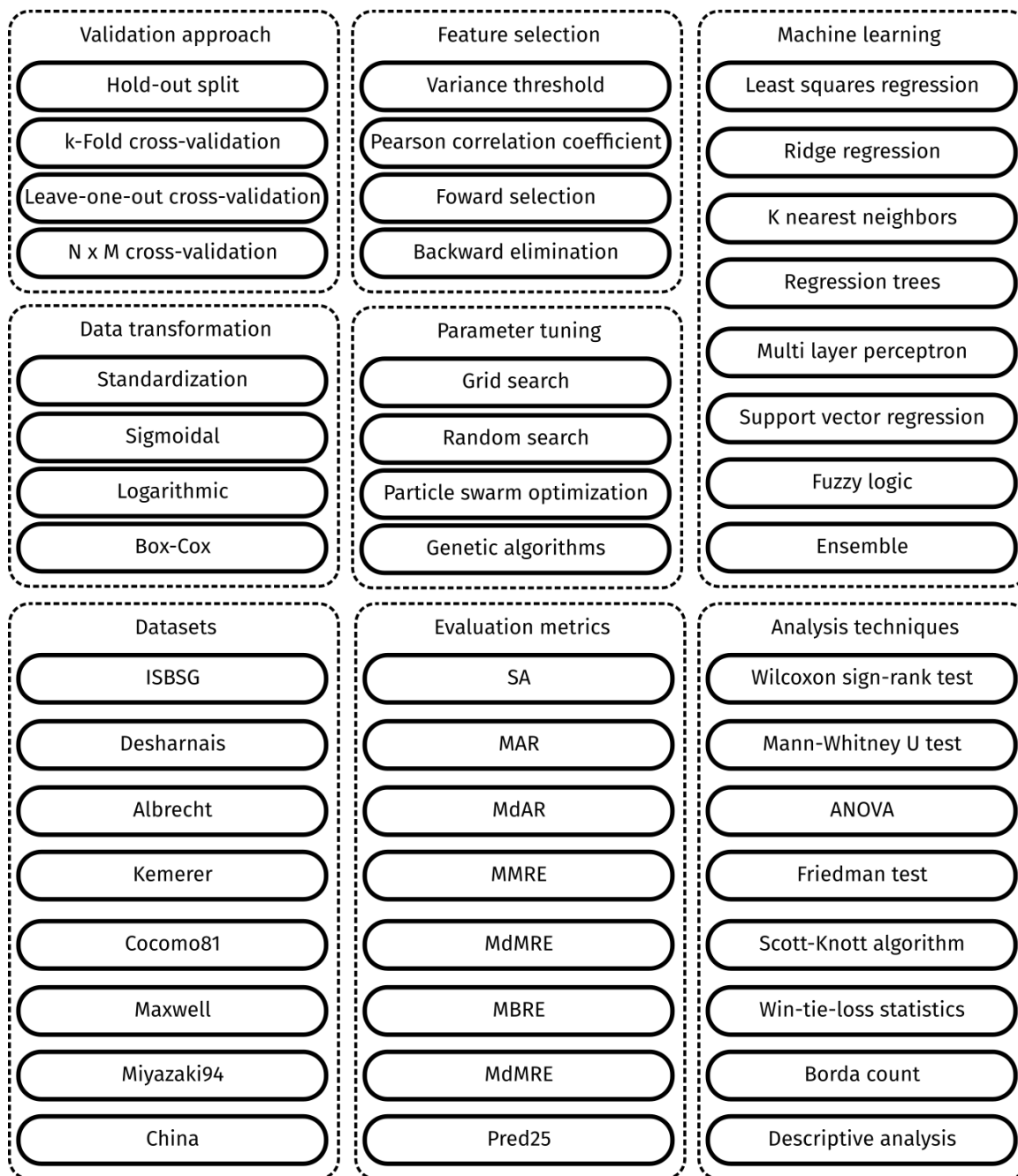


Figure 3.2: Classification scheme for extracted fields.

per, based on the keywording technique [62]. These techniques were clustered into families (i.e. multilayer perceptrons and recurrent neural networks both count as a neural network) so that we could present a general overview of machine learning techniques on SEE. We performed a descriptive analysis of the techniques found and the frequency of their use in the included papers. In addition, complemented this analysis with bubble and other types of frequency plots.

To answer research question 2, we counted the datasets that each paper used to build and evaluate models to estimate software effort. We counted splits or partitions of datasets as their original set. We performed a descriptive analysis of the datasets found and the frequency of their use in the included papers and complemented this analysis with bubble and other types of frequency plots.

To answer research question 3, we grouped and counted the evaluation or performance metrics used to assess effort estimation models. We grouped these techniques by their base formulas. For example, the metrics mean absolute error (MAR) and median absolute error (MdAR) were categorized as *absolute error* metrics. Moreover, we grouped and counted the analysis techniques used to process the metrics collected by SEE studies. Such techniques comprised descriptive analysis and statistical tests. We performed a descriptive analysis of the evaluation metrics and analysis techniques that were found, and their frequency in the included papers, complimenting this analysis with bubble and other types of frequency plots.

Potential threats to validity include the threat of missing relevant studies due to the automated search process, researcher bias in the inclusion/exclusion and data extraction process, and the generalizability of results to the broad machine learning SEE literature.

## 3.2 Results

Next we present the results of the systematic mapping study. Section 3.2.1 presents the hyper-parameter tuning and machine learning techniques reported in the studies. Section 3.2.2 shows the SEE datasets. Section 3.2.3 presents the evaluation metrics and analysis techniques reported in the studies. Appendix A contains tables with additional information than those shown in this chapter. The extraction form and results of this literature mapping can be found online<sup>1</sup>.

---

<sup>1</sup><http://tiny.cc/hpt-slm>

### 3.2.1 RQ1: Hyper-parameter tuning approaches used in machine learning SEE

#### 3.2.1.1 Hyper-parameter tuning

We encountered 12 different hyper-parameter tuning approaches in machine learning SEE literature. Table 3.3 shows the hyper-parameter tuning approaches and their related studies, and figure 3.3 shows the use of these approaches through the years. The most used technique was grid search, with a total of 60 studies. Many of these studies did not explicitly report usage of this technique, but instead reported using all possible parameter combinations. Thus, we counted such cases as using the grid search technique. The second most used technique was genetic algorithms, with a total of 14 studies. Particle swarm optimization (PSO) was the third most used technique, with 6 studies. The tabu search approach was researched in three studies, and random search was covered in two papers. Lastly, the beam search, bee’s algorithm, hill climbing, immune algorithm, online supervised tuning, and satin bowerbird optimization approaches were studied in only one paper. One study reported utilizing k-fold as a hyper-parameter tuning approach. even though this technique was usually employed for cross-validation.

This distribution shows that SEE studies favor exhaustive approaches such as grid search. Grid search is an effective method that can find the optimal hyper-parameter combination from a pre-defined grid. Grid search has two limitations: 1) it relies on a pre-defined search grid, and 2) it can be computationally expensive depending on the size of the grid. This is not a problem for models like analogy-based estimation which often relied on three categorical parameters. On the other hand, a model like neural network may not be viable to be grid-searched, as it relies on a combination of categorical and numerical attributes. Thus, one possible explanation for the increased use of grid search could be a relative simplicity of the hyper-parameter search space. Moreover, many hyper-parameter tuning studies may employ the approach as a baseline for comparison with other heuristic tuners. From figure 3.3 we see that, while grid search and genetic algorithms were predominantly used up until the year 2012, from there onward SEE studies have started to explore alternative tuners.

We analyzed the “study focus” of the 79 studies in order to determine *how* are these tuners used. We categorized the studies under two dimensions regarding their focus: object focus and evaluation focus. The **object focus** dimension determines which part of the machine learning process is being evaluated. For instance, one

Table 3.3: Hyper-parameter tuning approaches and related SEE studies.

Type	N	Studies
Grid Search	60	S12, S20, S30, S33, S51, S41, S40, S44, S55, S66, S1, S3, S4, S5, S6, S8, S10, S11, S14, S15, S17, S18, S19, S22, S23, S24, S26, S27, S28, S29, S31, S32, S34, S35, S36, S39, S38, S42, S43, S46, S47, S48, S49, S50, S52, S56, S57, S58, S59, S60, S62, S63, S65, S68, S69, S70, S73, S75, S78, S79
Genetic Algorithm	14	S37, S53, S66, S18, S42, S45, S54, S67, S70, S71, S72, S74, S76, S77
Particle Swarm Optimization	6	S12, S37, S2, S13, S25, S31
Tabu Search	3	S40, S64, S2
Random Search	2	S40, S21
Beam Search	1	S79
Bee's Algorithm	1	S61
Hill Climbing	1	S7
Immune Algorithm	1	S68
K-Fold Cross-Validation	1	S16
Online Supervised Tuning Procedure	1	S7
Satin Bowerbird Optimization	1	S25

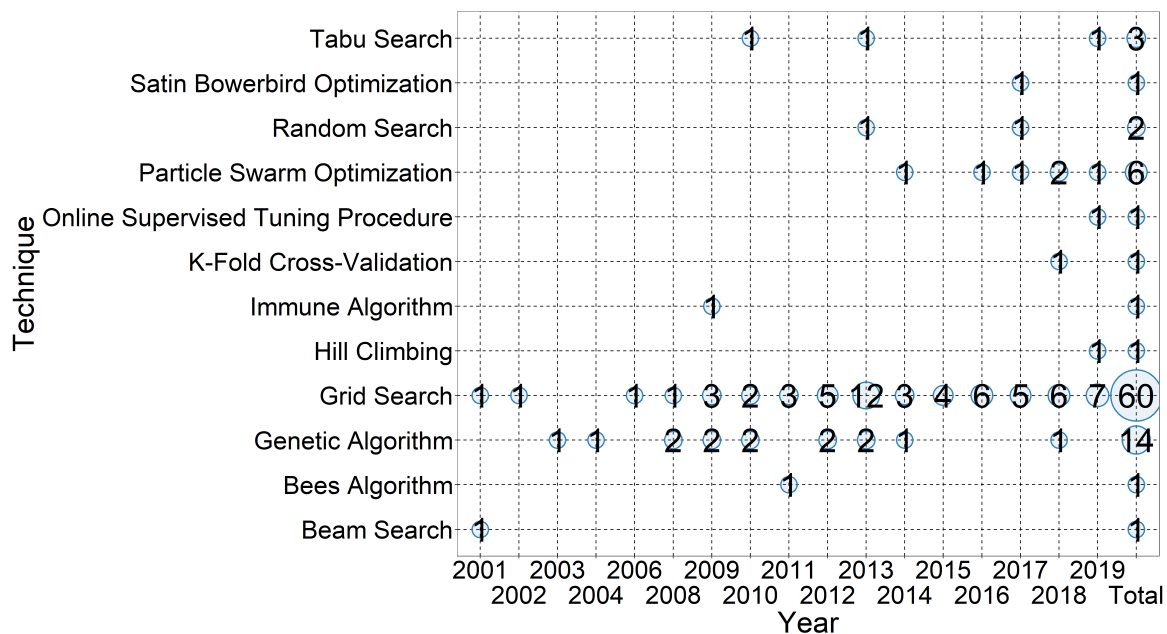


Figure 3.3: Hyper-parameter tuning approaches and usage through time.

study may focus on the comparison of a tuned against an untuned model, while another may compare two models that use tuning. We considered the first study to be tuning focused, and the second to be machine learning focused. The **evaluation focus** dimension determines whether the study is performing a benchmark (evaluation of multiple state-of-the-art) or a proposal (evaluation of a novel technique against some state-of-the-art). Table 3.4 shows the 10 unique focus combinations we identified and papers with that focus.

The majority (67 out of 79, or 85%) of the selected studies research on two objects: machine learning algorithms and hyper-parameter tuning. Only 10 studies had other types of technique as an object focus, including feature selection, data pre-processing, and clustering. Two studies undertook a multi-technique approach and tried to determine the best combination of learning algorithms, parameter tuning, pre-processing, and feature selection. Out of the 79 studies, 53 (67%) performed benchmarks of existing techniques and 26 (33%) proposed and evaluated novel techniques.

Regarding the use of tuning in SEE, the distribution of study focuses has several implications. First, SEE studies primarily utilize tuning to evaluate other parts of the machine learning process (often learning algorithms). This indicates that hyper-parameter tuning has been considered as part of the machine learning process, and these studies compared their techniques on their optimal configurations. However,

Table 3.4: Study focus and related SEE studies.

Focus	N	Studies
Learning algorithm - Benchmark	28	S1, S3, S6, S8, S9, S12, S14, S17, S24, S27, S28, S30, S36, S41, S42, S43, S47, S48, S49, S52, S55, S56, S58, S61, S62, S63, S67, S73
Parameter tuning - Benchmark	18	S11, S18, S23, S26, S29, S31, S33, S37, S38, S39, S40, S44, S45, S46, S51, S64, S65, S66
Learning algorithm - Proposal	17	S2, S10, S16, S19, S22, S25, S32, S35, S53, S57, S59, S60, S70, S71, S76, S77, S78
Parameter tuning - Proposal	4	S7, S68, S72, S74
Feature selection - Benchmark	3	S4, S5, S20
Data pre-processing - Benchmark	2	S34, S50
Data pre-processing - Proposal	2	S13, S75
Feature selection - Proposal	2	S69, S79
Multiple - Benchmark	2	S15, S54
Clustering - Proposal	1	S21

this also shows that relatively few studies (22 out of 79, 28%) have evaluated the *impact* of hyper-parameter tuning in SEE. Moreover, out of the 22 tuning-focused studies, 10 use only grid search (S11, S23, S26, S29, S33, S38, S39, S46, S51, S65), 5 use grid search and at least one other tuner (S18, S31, S40, S44, S66), and 7 use non-grid search tuners (S7, S37, S45, S64, S68, S72, S74). Thus, only a small fraction of studies have compared the effect of a particular tuning against an exhaustive baseline like grid search. We recommend future SEE researchers to evaluate their proposed tuning approaches against a baseline tuner, as well as to utilize other tuning approaches besides grid search.

In addition to hyper-parameter tuning approaches, SEE studies employed other techniques of the evaluation scheme: machine learning algorithms, feature selection, data pre-processing, and cross-validation approaches. Figure 3.4 shows the distribution of usage of these approaches through the years. All papers used at least one machine learning algorithm, and cross-validation approach. Only one paper did not employ hyper-parameter tuning, but they reported that it will be done in a follow-up study. For data pre-processing, we encountered a total of 44 studies reported using at least one pre-processing method. Only 24 papers used feature selection methods.

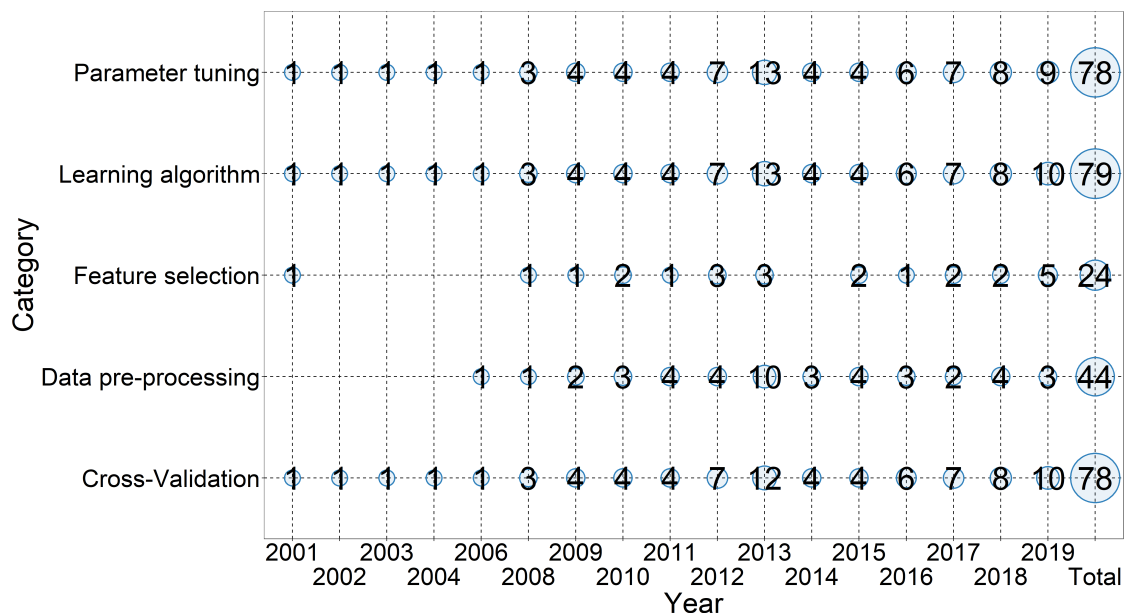


Figure 3.4: Categorization of techniques of the evaluation scheme and usage through time.

### 3.2.1.2 Machine learning algorithms

Table 3.5 shows the machine learning algorithms used across the studies, and figure 3.5 shows the use of these techniques through the years. In total, 21 different types of machine learning algorithm were encountered. These techniques were categorized by their general type of algorithm. For example, linear regression and ridge regression both count as a regression-type technique. The complete detail of techniques by sub-type, including supporting studies and counts, is available on Appendix A, in table A.3.

Table 3.5: Machine learning algorithms and related SEE studies.

Type	N	Studies
Neural Network	48	S3, S5, S6, S8, S9, S10, S11, S12, S15, S17, S18, S19, S20, S22, S23, S25, S27, S29, S30, S32, S33, S34, S36, S41, S42, S46, S47, S49, S51, S53, S54, S55, S58, S59, S60, S61, S63, S66, S67, S69, S70, S71, S72, S74, S76, S77, S78, S79

Continued on next page



Table 3.5: Machine learning algorithms and related SEE studies. (Continued)

Regression Tree	38	S1, S3, S5, S6, S7, S8, S10, S11, S12, S14, S15, S18, S19, S20, S22, S23, S25, S28, S29, S30, S32, S34, S35, S41, S43, S46, S47, S51, S55, S57, S59, S60, S61, S62, S63, S66, S69, S74
Regression	36	S2, S3, S6, S7, S8, S9, S10, S11, S14, S15, S17, S19, S22, S23, S25, S28, S29, S32, S35, S40, S41, S43, S49, S50, S55, S58, S59, S60, S61, S64, S65, S71, S74, S75, S78, S79
Support Vector Regression	33	S3, S5, S6, S8, S9, S10, S11, S12, S15, S16, S17, S18, S19, S20, S22, S28, S30, S31, S38, S39, S40, S42, S53, S54, S55, S60, S64, S66, S67, S68, S69, S71, S72
Case Based Reasoning	32	S4, S8, S13, S18, S21, S22, S24, S26, S28, S29, S32, S34, S35, S37, S40, S43, S44, S45, S47, S48, S50, S52, S55, S56, S59, S61, S62, S64, S65, S69, S74, S79
Ensemble	17	S4, S5, S6, S12, S14, S20, S22, S30, S41, S42, S46, S47, S49, S52, S57, S59, S63
K Nearest Neighbors	17	S3, S5, S6, S9, S10, S12, S14, S15, S20, S22, S23, S28, S30, S41, S46, S51, S73
Bagging	12	S7, S8, S10, S22, S46, S47, S51, S53, S63, S67, S71, S72
Random Forest	8	S1, S3, S6, S8, S11, S18, S22, S28
Boosting	6	S3, S6, S8, S11, S22, S60
Stacking	4	S6, S8, S28, S62
Bayesian Model	3	S6, S22, S28
Mean	3	S22, S40, S64
Median	3	S7, S40, S64
Rule Based Estimation	2	S22, S46
Gaussian Process	1	S22

Continued on next page

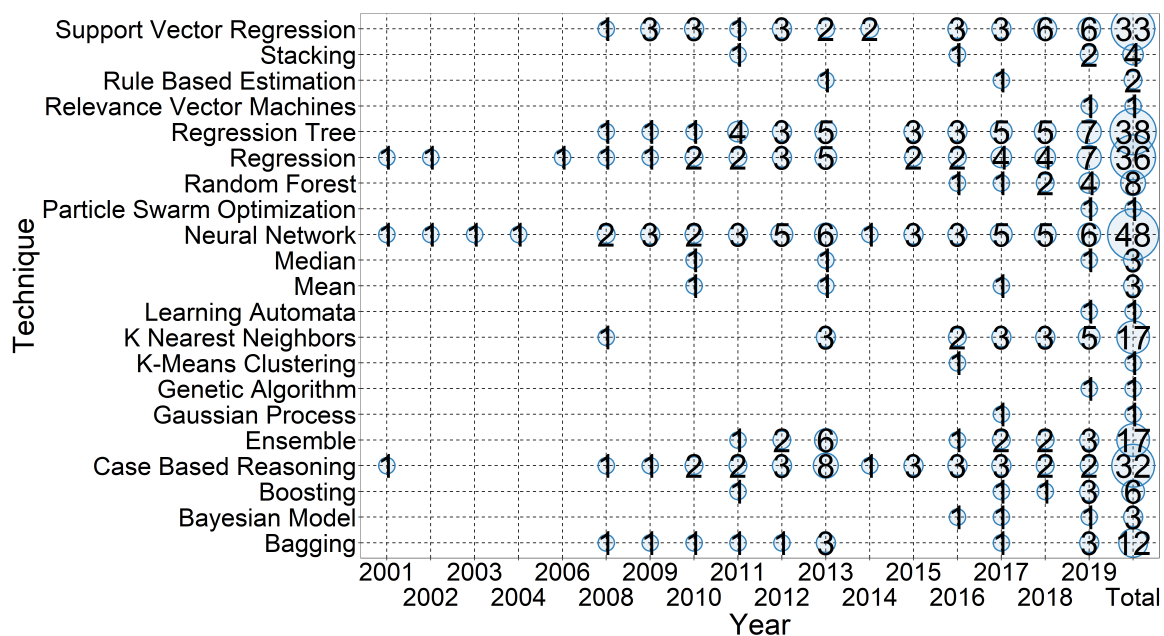


Figure 3.5: Machine learning algorithms and usage through time.

Table 3.5: Machine learning algorithms and related SEE studies. (Continued)

Genetic Algorithm	1	S2
K-Means Clustering	1	S28
Learning Automata	1	S2
Particle Swarm Optimization	1	S2
Relevance Vector Machines	1	S10

The most used technique were neural networks, with a total of 48 studies. Of these, the most used type of network were multi-layer perceptrons with 21 studies (S3, S5, S8, S9, S10, S12, S18, S19, S20, S22, S29, S30, S42, S46, S47, S51, S54, S55, S61, S63, S66). Other reported types of neural networks were radial basis function networks (13 studies, S15, S19, S22, S29, S46, S47, S54, S55, S60, S63, S69, S71, S72), general regression neural networks (S11), GMDH polynomial neural network (4 studies, S11, S76, S77, S78), morphological-rank-linear perceptrons (3 studies, S19, S53, S67), self-organized neuro-fuzzy networks (3 studies, S76, S77, S78), adaptative neuro-fuzzy interface system (2 studies, S25, S42), neuro-fuzzy networks (2 studies, S76, S77), particle swarm optimization neural network (2

studies, S23, S27), probabilistic neural networks (S11), cascade-correlation neural network (S11), general regression neural network (S11), genetic algorithm particle swarm optimization neural network (S27), genetic algorithm neural network (S36), multilayer dilation-erosion-linear perceptron (S19), recurrent neural network (S54), and error back propagation networks (S33). Neural network models have been popular through all the years of SEE research. The second most used technique were regression trees, with a total of 38 studies. The regression tree category covers techniques such as classification and regression trees (16 studies, S3, S8, S15, S23, S25, S29, S32, S34, S35, S41, S43, S55, S59, S60, S69, S74), M5P trees (4 studies, S18, S22, S30, S55, S61, S66), M5 trees (2 studies, S22, S55), REP trees (4 studies, S22, S46, S57, S63), C4.5 (S28), chaid decision tree (S28), J48 (S22), M5Prime (S30), and J48 trees (S22). While regression trees were not used in the first years of SEE tuning research, their use has seen an increase in the latest years, with a maximum of 7 studies in 2019. Regression approaches were employed in 36 studies, in many cases as a base for comparison, and were the third most used technique. The regression algorithms that used in more than one study were ordinary least-squares regression (15 studies, S2, S3, S6, S8, S9, S14, S15, S19, S22, S35, S41, S50, S55, S60, S71), step-wise regression (12 studies, S23, S25, S29, S32, S35, S41, S43, S59, S61, S64, S65, S74), multiple regression (10 studies, S17, S23, S25, S29, S32, S43, S49, S58, S59, S75), ridge regression (4 studies, S3, S6, S8, S55), logistic regression (3 studies, S19, S22, S28), least median squares regression (2 studies, S22, S55), and morphological-rank-linear filter regression (2 studies, S60, S71). Similar to neural networks, regression models have been frequently used in SEE.

Regarding the remaining single-model techniques, support vector regression (SVR) was used in 33 studies, making it the fourth most used model. Similar to CART, SVR models were not used in the initial tuning studies, but rose in popularity through the years. Case-based reasoning was employed in 32 studies, of which 21 cases were variants of analogy-based estimation (S4, S8, S13, S18, S21, S24, S26, S29, S32, S35, S43, S44, S45, S47, S48, S50, S56, S59, S62, S69, S74). The peak use of case based reasoning was in 2013, and it has not been as used as other techniques in the more recent years. The  $k$  nearest neighbors technique was used in a total of 17 studies, with varying values of the parameter  $k$ , mainly used from the year 2013 onward. Three studies used the Bayesian model, and 2 employed rule-based estimation. Techniques that were only used in one study are Gaussian process, genetic algorithms,  $k$ -means clustering, learning automata, particle swarm optimization, and relevance vector machines. Lastly, 4 studies employed statistics of the dataset, such as mean and median,

as baseline estimators.

Different types of combination models were studied in SEE literature. Of these, ensemble methods were the most researched, with a total of 17 studies. Of these, 10 studies employed heterogeneous ensembles (S5, S6, S12, S14, S20, S22, S30, S41, S42, S46), 3 used homogeneous ensembles (S4, S42, S49), and 8 studies used one or more types of specialized ensemble (S6, S14, S22, S46, S47, S59, S57, S63). Ensembles saw peak use in 2013, but have been used ever since. Other combination techniques include bagging (12 studies), random forests (8 studies), boosting (6 studies), and stacking (4 studies). One noteworthy observation of these techniques is that hyper-parameter tuning has to account for the parameters of the individual techniques. This was often performed in a two-step process: a first of hyper-parameter tuning was performed to select the best parameters for the base models, and afterwards the combined techniques were built using models with these parameters. If the combined technique had parameters, such as the aggregation function in ensembles, a second iteration of hyper-parameter tuning was performed.

With the exception of regression-type techniques, the most used techniques in SEE share one common trait: they have more than three hyper-parameters that could require tuning. This applies two-fold for combination techniques, as they must configure both the parameters of the “inner” techniques as well as the “outer” parameters of the ensemble itself. Moreover, the most used technique was neural networks, which has many hyper-parameters, including: learning rate, momentum, amount of hidden layers, activation functions (per layer), amount of hidden neurons (per layer), batch size, training epochs, solver. Assuming a study configures all of these hyper-parameters and using only 5 values (using a fixed size for the per-layer ones), an exhaustive tuner like grid search would explore  $5^8 = 390,625$  hyper-parameter combinations. Assuming each model requires 1 minute to train, grid search would require 6,510 hours, or 271 days (almost three quarters of a year) to find an optimal hyper-parameter combination. Thus, future research could evaluate non-grid search methods to explore if they are viable (equally or near-equally efficient) as a less time-consuming alternative.

### 3.2.1.3 Cross-validation

Table 3.6 shows the cross-validation approaches used across the studies, and figure 3.6 shows the use of these techniques through the years. In total, 5 different types of approaches were encountered. In some cases, studies employed different

Table 3.6: Cross-validation approaches and related SEE studies.

Type	N	Studies
Leave-One-Out Cross-Validation	38	S4, S5, S8, S11, S12, S13, S18, S19, S20, S21, S22, S23, S24, S26, S27, S29, S30, S35, S36, S37, S40, S41, S44, S49, S50, S53, S55, S56, S60, S62, S64, S66, S71, S72, S73, S76, S77, S78
K-Fold Cross-Validation	28	S1, S3, S10, S11, S13, S14, S15, S16, S17, S18, S19, S25, S27, S28, S31, S32, S38, S39, S40, S43, S50, S54, S59, S61, S62, S65, S66, S74
Hold-out split	27	S1, S2, S3, S9, S17, S18, S33, S34, S38, S42, S45, S46, S47, S52, S54, S55, S58, S63, S64, S66, S67, S68, S69, S70, S72, S75, S79
Online	4	S7, S45, S51, S57
Blocked Cross-Validation	1	S6

validation approaches for different datasets. For example, S4 used leave-one-out for datasets under 60 observations, and 10-fold cross validation otherwise. Cross-validation approaches are only applicable to offline effort estimation. Online estimation studies instead simulate a real estimation scenario by ordering the available projects by date. We have labeled this technique as “online”.

The most used cross-validation was the leave-one-out approach, with a total of 38 studies. Leave-one-out has been consistently used through the years in SEE research. The second most used approach was k-fold cross-validation, used in 28 studies. Of these, 15 studies used 10-fold or repeated 10-fold (S1, S3, S10, S13, S14, S15, S17, S18, S19, S28, S31, S32, S40, S50, S66), 8 studies used 3-fold (S18, S19, S43, S59, S61, S62, S65, S74), and 5 studies used 5-fold (S11, S38, S39). While not used in the very first years, studies have constantly employed k-fold as an alternative to leave-one-out. The hold-out split approach was used in 27 studies. Of these 10 employed a train-test split (S1, S2, S3, S9, S17, S18, S33, S64, S68, S79), 8 used a repeated train-test split (S45, S46, S47, S55, S63, S66, S72, S75), 5 used a train-validation-test split (S34, S52, S67, S69, S70), and one used repeated train-validation-test split (S58). In all cases, the amount of projects used for the test set varied between 10% and 50%, except in S69 where the test set encompassed 80% of the dataset. The amount of projects used for the validation set varied between 10% and 33%. Similar to k-fold, hold out split has been employed constantly in SEE literature, surpassing both leave-one-out and k-fold in several years. Lastly, 4 studies

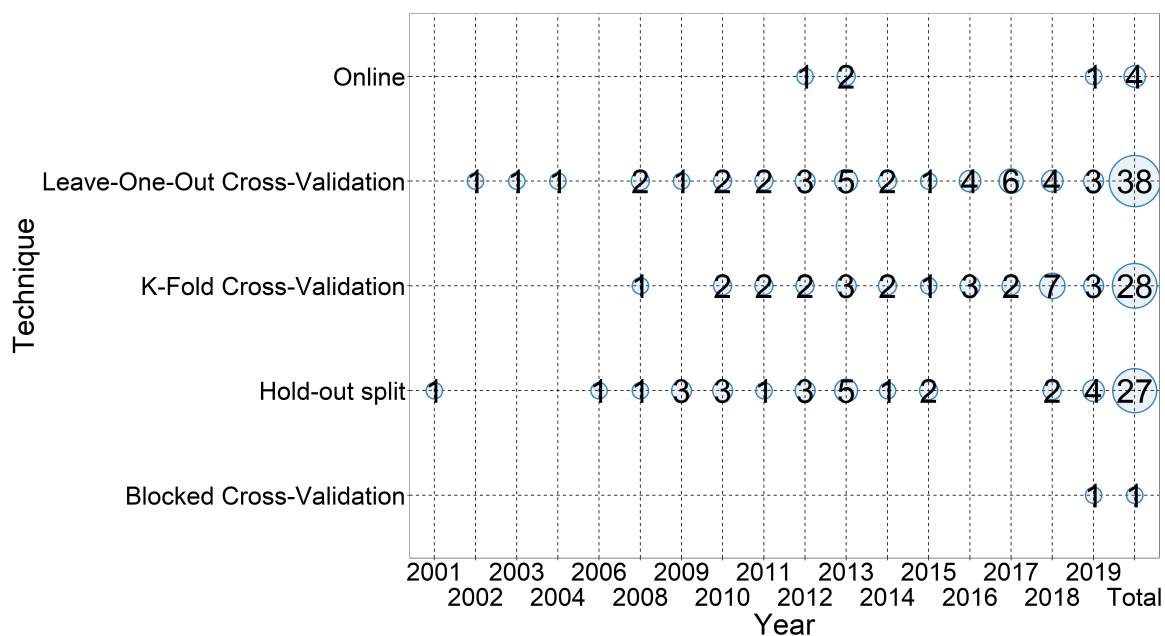


Figure 3.6: Cross-validation approaches and usage through time.

used the online approach for effort estimation, and one study employed the blocked cross-validation approach.

There is no predominant cross-validation approach from this distribution. There is a trend in the amount of cross-validation iterations, as many of these studies employ techniques with multiple iterations. Leave-one-out, k-fold, repeated hold-outs, and online cross-validations all have at least two iterations, and as many as the size of the dataset. This trend shows that SEE has shifted to searching for *stability* in their results: instead of gathering results from one train-test round, these studies measured the effectiveness of their models under different circumstances.

#### 3.2.1.4 Data transformation

Table 3.8 shows the data transformation approaches used across the studies, and figure 3.7 shows the use of these techniques through the years. In total, 10 different techniques were encountered.

The most used data transformation was the unit range  $[0, 1]$  transformation, used by 25 studies. This technique was often reported as normalization. We have chosen the unit range  $[0, 1]$  name to avoid confusion with the standardization technique. Unit range  $[0, 1]$  saw predominant use in the earlier SEE research years, but more recent studies have employed alternative approaches. The second most used technique

Table 3.7: Data transformations and related SEE studies.

Type	N	Studies
Unit Range [0,1]	25	S8, S15, S17, S19, S25, S26, S31, S32, S33, S34, S37, S38, S39, S41, S43, S45, S46, S53, S60, S62, S67, S69, S71, S74, S75
Logarithm	13	S8, S14, S15, S17, S35, S40, S41, S47, S50, S55, S58, S61, S64
Principal Component Analysis	5	S3, S8, S27, S41, S42
Standardization	5	S3, S10, S14, S15, S49
One-hot Encoding	4	S3, S10, S35, S55
Binning	2	S3, S66
BoxCox	2	S15, S55
Unit Range [-1,1]	2	S34, S58
Binary Encoding	1	S49
K-Means Clustering	1	S55

was the logarithmic transformation with a total of 13 papers. Logarithm transformation saw peak use between 2010 and 2013. After that, it was relative unused until 2018. The third most used technique was a tie between principal component analysis and standardization, with 5 studies each. Both are techniques that were not employed before 2013 in tuning SEE studies. Standardization has been mostly used in 2018 and 2019, and it indicates that the technique may be rising in popularity. One-hot encoding (also called one-of-k representation) was applied in 4 studies. The BoxCox, binning, and unit range [-1, 1] techniques were used in 2 studies. Lastly, binary encoding and k-means clustering were used in one study each.

The trend in data transformations for SEE focuses mainly on numerical features, as unit range [0, 1], unit range [-1, 1], BoxCox, and logarithm apply only to numbers. Moreover, principal component analysis, one-hot encoding allow and binary encoding allow the conversion of categorical features into numerical. This could imply that, for SEE, numerical features are better predictors of effort.

### 3.2.1.5 Feature selection

Table ?? shows the feature selection approaches used across the studies, and figure 3.8 shows the use of these techniques through the years. In total, 10 different techniques were encountered.

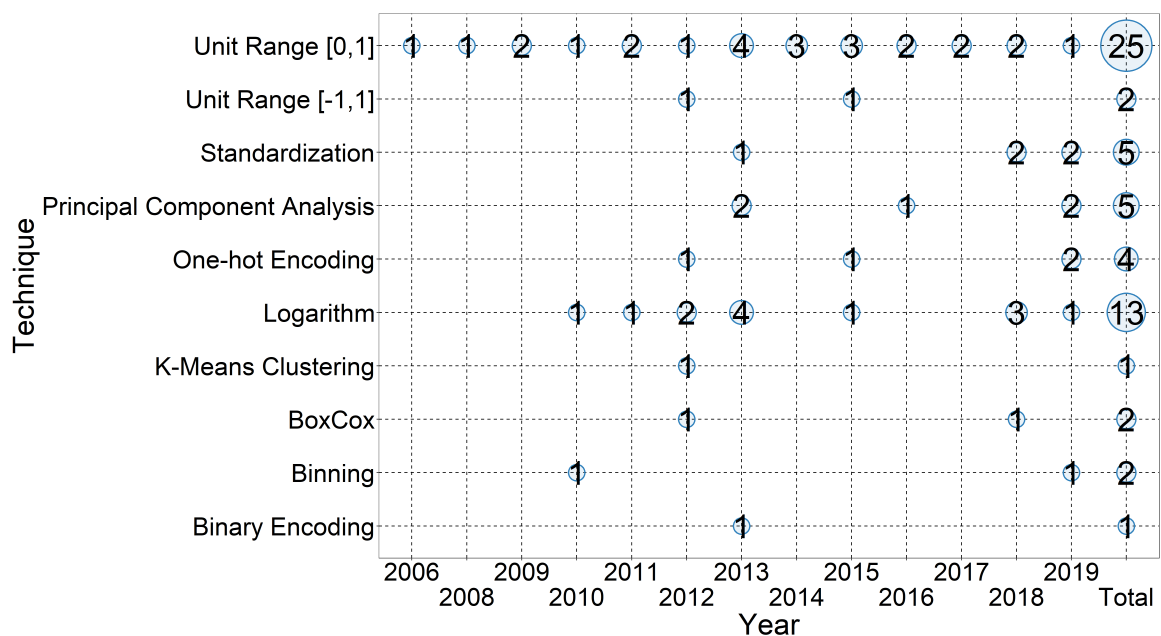


Figure 3.7: Data transformation approaches and usage through time.

The most used technique were correlation based feature selections, with a total of 10 studies. Correlation based feature selection encompasses techniques as sequential forward selection (S15, S24, S34, S41, S58), sequential backward selection (S15, S34, S58), best-first (S4, S5), k best univariate (S6), greedy stepwise search (S63), stepwise variable selection (S24). The trend in use in correlation-based FS started from 2011, and has increased since 2017. Genetic algorithm feature selection was used in 7 studies. Conversely, genetic algorithms were mainly used in the earlier years of the covered studies. Feature selection based on Pearson correlation was employed in 4 studies, 3 of which were in 2019. Similarly, RReliefF based feature selection was used in 3 studies, 2 of which were also in 2019. Lastly, 2 studies used (filter) backward feature elimination. Techniques used in only one study were case set selection, exhaustive search, particle swarm optimization, principal component analysis, and regression. Feature selection techniques can be employed as filters or wrappers. Eleven studies employed filter approaches (S4, S5, S6, S9, S15, S20, S24, S35, S40, S41, S63), and eight used the more computationally expensive wrappers (S8, S15, S18, S34, S53, S55, S66, S67).

Feature selection is often not used in conjunction with hyper-parameter tuning, as 24 out of the 78 studies that use tuning. This could be due to the higher computational requirements for some of these feature selectors, especially wrapper-type techniques. Another possible explanation for the relatively low amount of feature



Table 3.8: Feature selection and related SEE studies.

Type	N	Studies
Correlation Based Feature Selection	10	S4, S5, S6, S15, S20, S24, S34, S41, S58, S63
Genetic Algorithm	7	S18, S42, S53, S66, S67, S69, S72
Pearson correlation	4	S4, S5, S9, S40
RReliefF Based Feature Selection	3	S4, S5, S20
Backward Feature Elimination	2	S8, S55
Case Set Selection	1	S79
Exhaustive Search	1	S35
Particle Swarm Optimization	1	S31
Principal Component Analysis	1	S24
Regression	1	S41

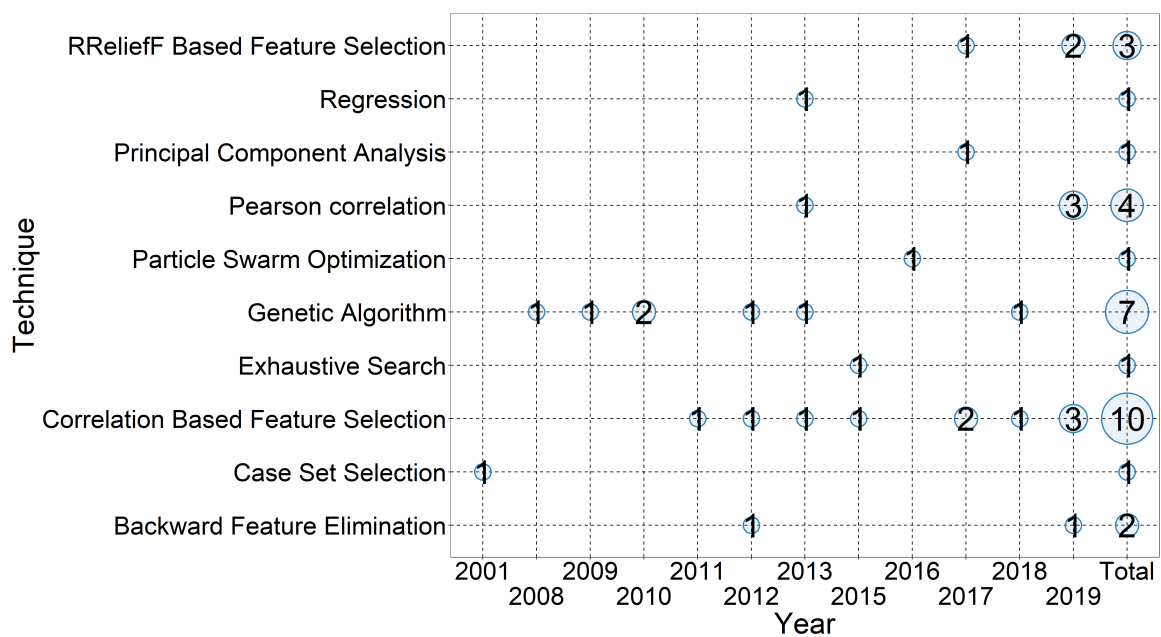


Figure 3.8: Feature selection approaches and usage through time.

selection studies is that some of the most used machine learning algorithms, such as neural networks and regression trees, innately perform feature selection. Feature selection remains an useful technique for some algorithms, like regression and case-based reasoning. In addition, the use of feature selection saw a spike on 2019. This could indicate that future studies could start considering using feature selectors.

### 3.2.2 RQ2: Datasets used in hyper-parameter tuning machine learning SEE

We identified 47 unique datasets that have been used in the literature of software effort estimation. We grouped these datasets into eight categories in accordance to their availability or repository: PROMISE, ISBSG, open, Tukutuku, unidentified, artificial, private, and IBM. Figure 3.9 shows the use of these dataset origins over the years. The PROMISE category refers to the datasets publicly available in the PROMISE data repository. Since 2008, datasets from the PROMISE repository have seen dominant use in SEE studies. The ISBSG category refers to the dataset distributed by the International Software Benchmarking Standards Group. Although this dataset is not publicly available for free, there were some free versions of the dataset available in the PROMISE repository, and has had a similar use trend to PROMISE. Similarly, the Tukutuku category refers to the dataset of the same name that was publicly available. The open category refers to other datasets that are publicly available, but not on the PROMISE or Tukutuku repositories. Tukutuku has had two visible use periods: 2010–2013, and 2018–2019. The private category refers to other datasets that are not publicly available, or are not available for free. Lastly, IBM datasets refer to private datasets that belong to the IBM company. The unidentified category refers to datasets of which the authors do not give sufficient information to trace back to an origin. The artificial datasets refer to datasets artificially generated, generally through probability distributions. Table 3.9 shows the datasets grouped by their origin and their related studies. Figure 3.10 shows the trend of usage of the PROMISE and ISBSG datasets, the two most used origins, through the years.

Table 3.9: SEE datasets and related SEE studies.

Origin	Dataset	N	Studies
--------	---------	---	---------

Continued on next page

Table 3.9: SEE datasets and related SEE studies. (Continued)

PROMISE	Desharnais	47	S12, S20, S40, S44, S53, S55, S61, S66, S2, S4, S5, S8, S9, S13, S14, S18, S19, S21, S22, S23, S24, S26, S27, S29, S31, S34, S35, S42, S45, S46, S47, S48, S50, S52, S56, S58, S59, S62, S63, S65, S67, S68, S69, S70, S72, S73, S74
	Cocomo81	35	S12, S20, S37, S44, S53, S55, S61, S66, S1, S2, S4, S5, S8, S10, S13, S14, S16, S18, S19, S21, S23, S24, S26, S27, S35, S36, S42, S46, S47, S48, S57, S62, S63, S67, S70
	Albrecht	26	S12, S20, S30, S37, S40, S44, S53, S61, S66, S4, S5, S8, S13, S18, S19, S22, S24, S25, S26, S27, S35, S42, S48, S62, S69, S70
	Kemerer	22	S12, S20, S37, S40, S44, S53, S61, S66, S8, S13, S14, S16, S18, S19, S21, S24, S25, S26, S35, S48, S62, S70
	Maxwell	20	S33, S37, S40, S44, S51, S55, S61, S8, S10, S14, S21, S22, S23, S24, S26, S35, S42, S48, S59, S62
	CocomoNasa2	15	S37, S44, S61, S2, S8, S10, S17, S21, S22, S24, S46, S47, S48, S57, S63
	China	14	S12, S20, S33, S37, S40, S4, S5, S13, S21, S24, S26, S35, S48, S62
	Miyazaki94	14	S12, S20, S30, S40, S44, S4, S5, S8, S13, S14, S24, S42, S48, S52
	Telecom	8	S37, S40, S44, S61, S14, S26, S35, S48
	CocomoSDR	7	S44, S8, S22, S24, S46, S47, S63
	CocomoNasa	4	S55, S2, S22, S57
	Kitchenham	4	S51, S10, S34, S48
	USP05	3	S55, S34, S48

Continued on next page

Table 3.9: SEE datasets and related SEE studies. (Continued)

ISBSG	ISBSG R10	13	S7, S51, S61, S10, S26, S34, S35, S45, S46, S47, S57, S62, S63
	ISBSG R8	7	S12, S1, S4, S5, S13, S18, S50
	ISBSG R11	6	S55, S3, S25, S29, S32, S43
	ISBSG R?	2	S22, S54
	ISBSG R9	2	S58, S70
Open	NasaBailey	14	S53, S66, S19, S26, S27, S35, S60, S63, S65, S71, S72, S76, S77, S78
	Finnish	4	S40, S44, S8, S24
	KotenGray	4	S53, S66, S8, S19
	CF	2	S43, S56
	Costagliola05	1	S39
	Jodpimai18	1	S15
	OUTS	1	S74
	QUES	1	S74
	Ziauddin12	1	S38
Tukutuku	Tukutuku	5	S40, S41, S64, S1, S18
Unidentified	Unidentified	2	S16, S75
	Bank	1	S50
	JunLee01	1	S79
	Stock	1	S50
	Zakrani18	1	S11
Artificial	Moderate	2	S29, S69
	Severe	2	S29, S69
	CoKem	1	S70
	DeshRand0	1	S73
	DeshRand1	1	S73
Private	ESA	1	S55

Continued on next page

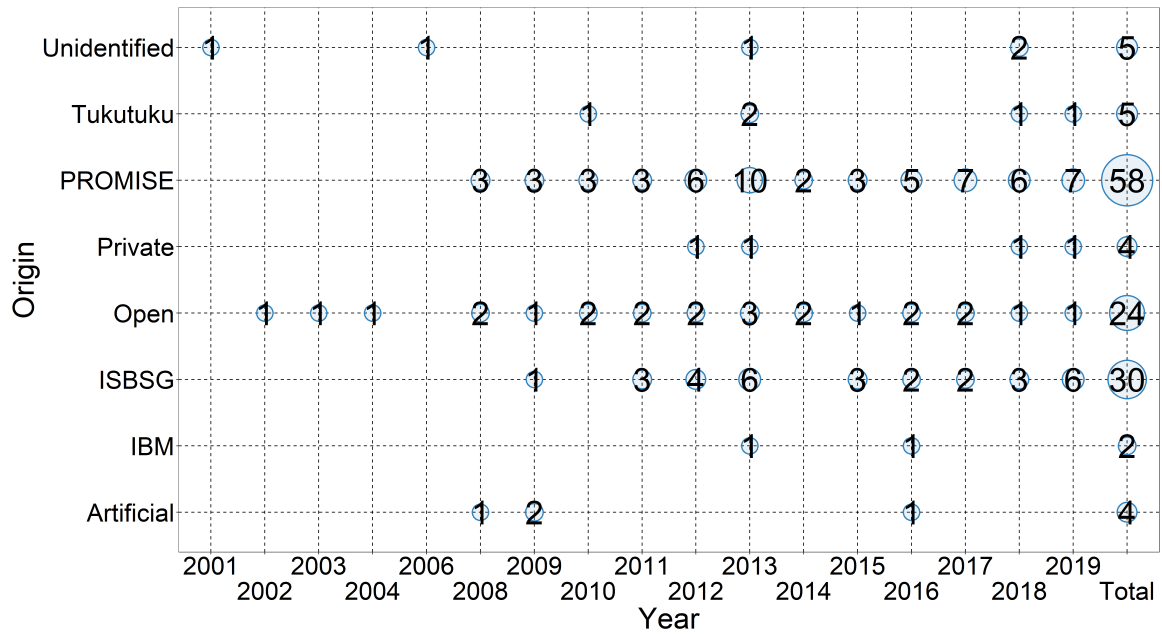


Figure 3.9: Dataset origins and usage through time.

Table 3.9: SEE datasets and related SEE studies. (Continued)

	Euroclear	1	S55
	Experience	1	S55
	IT University	1	S6
	IVR	1	S16
	Pai13	1	S49
IBM	DPS	1	S43
	RQM	1	S28
	RTC	1	S28

We identified a total of 13 PROMISE datasets, and a total of 58 studies that use them. The most used dataset was Desharnais, with a total of 47 studies, and saw peak use in 2019. The second most used dataset is Cocomo81 with a total of 35 studies. The Albrecht dataset was used in 26 studies, and the Kemerer dataset in 22 studies. Both sets were not widely used in the first years, but saw an upward trend starting from the year 2013. The Maxwell dataset is used in 20 different SEE studies, following an use pattern similar to Albrecht and Kemerer. Other PROMISE datasets employed in SEE include CocomoNasa2, China, Miyazaki94, Telecom, CocomoSDR,

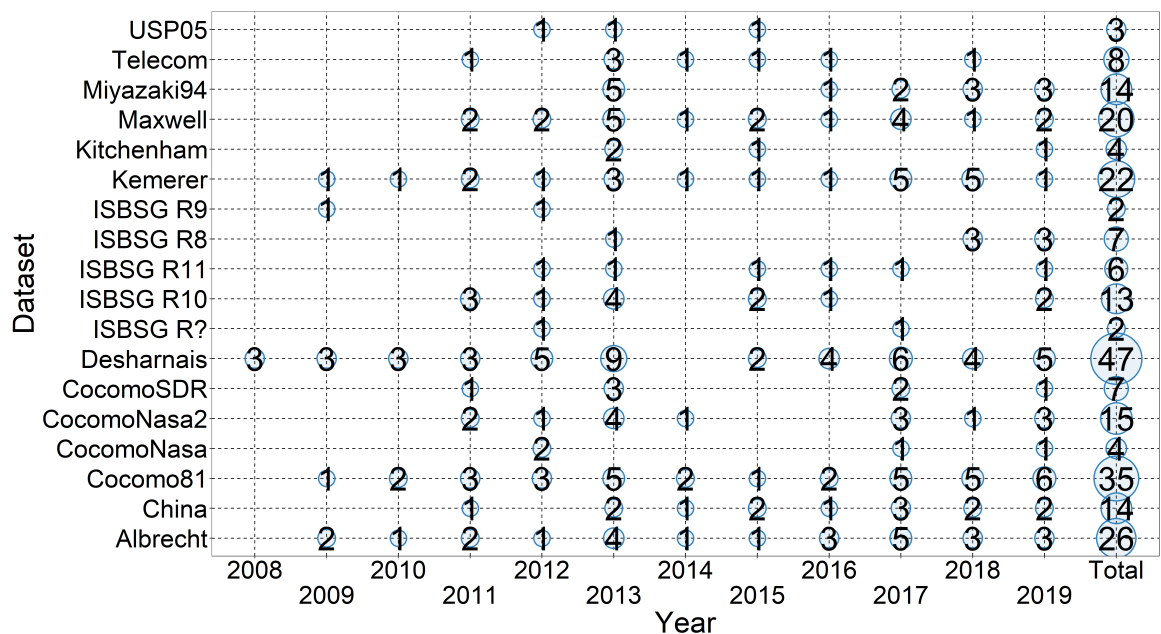


Figure 3.10: PROMISE and ISBSG datasets and usage through time.

CocomoNasa, Kitchenham, and USP05. The CocomoNasa2, China, Cocomo91, and Miyazaki94 sets have seen frequent use through the years of SEE research. On the other hand, CocomoNasa, Telecom, CocomoSDR, Kitchenham, USP05 have fallen in relative disuse, although recent studies (>2017) have covered all of them but USP05.

Four different versions of the ISBSG dataset were identified in the literature, which were used in 30 different studies. The most used version is the ISBSG release 10 dataset, being used in 13 studies. The ISBSG release 8 was used in 7 studies, the release 11 was used in 2 studies, and the release 9 was used in 2 studies as well. Lastly, two studies reported they used the ISBSG dataset, although they do not report the release number, nor traits that allowed its identification. Somewhat counter intuitively, the oldest ISBSG dataset (R8) has seen more use in recent years than the ISBSG R10 and R11, perhaps due to availability. The ISBSG R10 still remains the most used overall, but saw the peak of its use in the 2011–2016 period.

Nine different studies used open datasets not belonging to the PROMISE repository. These correspond to the NasaBailey, Finnish, KotenGray, CF, Costagliola05, Jodpimai18, OUTS, QUES, and Ziauddin12 datasets. Similarly, the Tukutuku dataset has seen use in 5 different studies. Artificial datasets were generated and used in 4 studies (S29, S69, S70, S73): Modarate and Severe, which were generated using arbitrary random distributions; and CoKek, DeshRand0 and DeshRand1, which were generated using the feature distributions from the Cocomo and Kemerer, Desharnais,

and Desharnais datasets respectively. Regarding private datasets, four studies (S6, S16, S49, S55) employed six different private datasets: ESA, Experience, Euroclear, IT University, IVR, and Pai13. One study employed a dataset obtained from IT projects of an university (S6), and one study reported used a dataset named IVR (S16). Two studies employed private datasets which originate from the IBM company. Lastly, five datasets of unknown origin were encountered.

The literature in SEE favors datasets from two particular repositories: PROMISE and ISBSG. Moreover, the four most used dataset origins, PROMISE, ISBSG, Open and Tukuruku, have publicly available datasets. While this is positive for purposes of open data and replicability of studies, these datasets contain data from old development projects. As shown in figure 3.10, the “most recent” datasets were introduced in 2013. It is also possible these datasets were used in even older studies, as this mapping contemplates only hyper-parameter tuning. Software development practices have changed in this 8 year (at minimum) period, and it is possible that estimations produced with these datasets may not be as accurate for newer development projects.

### **3.2.3 RQ3: Performance metrics of hyper-parameter tuning approaches used in machine learning SEE**

We identified 41 unique evaluation metrics that have been used in SEE literature. These have been grouped into categories, in accordance with their base metric. We have identified 10 different base metrics: relative error, absolute error, log error, square error, prediction error, error, correlation, accuracy, interval, and meta metric. The metrics were classified into a taxonomy of three levels. The first level is the base metric. The second level is used to differentiate functions that are aggregated on the base metric. For example, one relative error metric may use the relative error, while another may use the magnitude of the relative error. Lastly, the third level adds the statistical function used to calculate the concrete value for a metric. For example, the MdIBRE metric employs a base metric of relative error, an aggregate metric of inverse balance relative error, and a statistical function of median. Table 3.10 shows the taxonomy of the existing metrics, as well as the studies that employ them. Figure 3.11 shows the trend of usage of the base metrics through the years.

Table 3.10: Evaluation metrics and related SEE studies.

Level 1	Level 2	Metric	N	Studies
Relative Error	Balanced Relative Error	MBRE	13	S12, S20, S37, S41, S4, S5, S6, S8, S15, S26, S34, S48, S50
		MdBRE	3	S5, S15, S34
Estimate Magnitude of Relative Error		MEMRE	6	S40, S41, S64, S8, S48, S50
		MdEMRE	2	S64, S50
Inverse Balanced Relative Error		MIBRE	9	S12, S20, S37, S41, S4, S5, S15, S26, S48
		MdIBRE	3	S20, S5, S15
Magnitude of Relative Error		MMRE	50	S40, S41, S53, S61, S64, S66, S1, S2, S11, S14, S16, S18, S19, S21, S22, S25, S27, S29, S32, S35, S36, S39, S38, S42, S43, S45, S46, S47, S48, S49, S50, S52, S56, S58, S59, S60, S62, S63, S65, S67, S68, S69, S71, S72, S73, S74, S76, S77, S78, S79
		PRED(L)	43	S12, S20, S30, S40, S41, S44, S53, S55, S61, S64, S66, S1, S4, S5, S11, S14, S17, S18, S19, S23, S25, S27, S29, S32, S34, S35, S36, S42, S45, S46, S47, S50, S52, S58, S59, S60, S62, S63, S65, S67, S68, S69, S72
		MdMRE	21	S41, S44, S55, S61, S64, S1, S11, S14, S18, S23, S29, S45, S47, S49, S50, S52, S62, S63, S65, S69, S74
		EF	4	S42, S60, S67, S71
		Epsilon	1	S73

Continued on next page



Table 3.10: Evaluation metrics and related SEE studies. (Continued)

		Rratio	1	S75
	Relative Error	MRE	4	S29, S58, S70, S75
		RSD	2	S8, S24
		VRE	1	S75
Absolute Error	Absolute Error	MAE	24	S12, S20, S33, S40, S41, S44, S51, S3, S4, S5, S6, S8, S10, S24, S32, S34, S35, S47, S48, S56, S57, S62, S65, S75
		SA	13	S7, S12, S20, S30, S37, S4, S5, S8, S10, S13, S26, S46, S57
		Delta	11	S12, S20, S30, S37, S4, S5, S13, S26, S46, S47, S57
		MdAE	7	S20, S40, S66, S5, S10, S24, S47
		PRED	5	S39, S38, S43, S71, S74
		D	1	S51
		SdAE	1	S66
		SumAE	1	S66
		VAE	1	S75
	Absolute Percentage Error	MAPE	2	S33, S31
Log Error	Log Error	LSD	10	S12, S20, S37, S4, S5, S8, S10, S24, S26, S47
Square Error	Square Error	RMSE	4	S33, S6, S16, S39
		NRMS	3	S39, S58, S70
		MSE	2	S33, S39
Prediction Error	Prediction Error	AIC	1	S54
		MDL	1	S54

Continued on next page

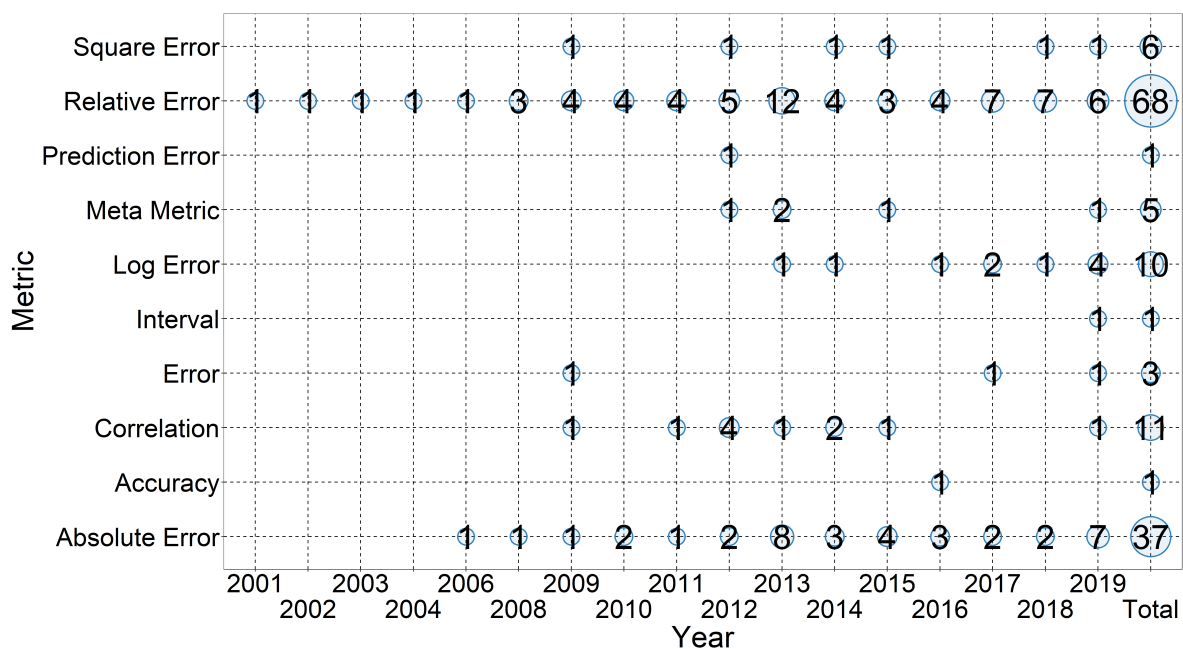


Figure 3.11: Evaluation metrics and usage through time.

Table 3.10: Evaluation metrics and related SEE studies. (Continued)

Error	Error	ME	1	S3
		SD	1	S24
	Error Tendency	Sign	1	S70
Correlation	Correlation	R2	5	S9, S35, S39, S38, S50
		R	4	S54, S58, S63, S70
		Spearman	2	S55, S56
Accuracy	Accuracy	Acc	1	S28
Interval	Hit Rate	Hit Rate	1	S10
	Interval Width	rWidth	1	S10
Meta Metric	Improvement	imp_ratio	5	S10, S34, S43, S52, S59

Of the 41 discovered metrics, 34 are based on the prediction error: the difference between the predicted and actual effort values. Metrics based on relative error (RE) were used in 68 studies, making them by far the most used type of metric. Relative error metrics comprise metric that use the difference between predicted and actual

effort values divided by an adjustment factor (usually either effort value). Five aggregate metrics were encountered for the base metric: magnitude of relative error (MRE) with 61 studies; balanced relative error (BRE) with 13 studies; inverse balanced relative error (IBRE) with 9 studies; estimate magnitude of relative error (EBRE) with 6 studies, and relative error with 6 study. Of these, the most used metric was the mean MRE (MMRE), followed by the PRED(L) metric and the median MRE (MdMRE). Relative error metrics have seen healthy use through the years on SEE tuning research, and still are the predilection of many SEE researchers.

Metrics based on absolute error (AE, also known as absolute residual) were used in 37 studies. Two aggregate metrics were encountered for the base metric: absolute error (AE) with 36 studies and absolute percentage error (APE) with 2 studies. Similar with RE metrics, the most used metric was the mean absolute error (MAE). Absolute error metrics were not popular in the 2001–2012 range, but they saw a boost in their use since 2013 and onward. Many SEE researchers have chosen to report using both relative error and absolute error metrics. Metrics based on the logarithmic error were used in 10 studies, with the only metric being the logarithmic standard deviation (LSD). Log error metrics were introduced in 2013, and have seen steady use until recently. Square error metrics were used in 6 studies. One study employed “prediction error”, a combination of training error plus a complexity factor related to neural network models. Lastly, 3 studies employed metrics based on the “pure” error: mean error, standard deviation of the error, and the direction or sign of the error.

Seven metrics do not use the prediction error as base, but instead rely on other properties of the predicted and actual values. Of these, three are metrics based on correlation coefficients, and were used in 11 studies. Accuracy metrics apply only when the target feature was categorical, which was the case in one study. Instead of providing a single “point” data about the effectivity of a model, interval metrics determine a reliable interval in for the predicted values. Lastly, meta metrics refer to a metric which is built on another metric. We encountered only one meta metric: the improvement ratio.

In addition to reporting results in terms of evaluation metrics, SEE studies used analysis techniques to draw conclusions from the results. We identified three broad types of analysis techniques in the surveyed papers: descriptive analysis, statistical tests, and ranking methods. Table 3.11 shows the existing analysis techniques in SEE literature and the studies that employ them.

Table 3.11: Analysis techniques and related SEE studies.

Type	Technique	N	Studies
Descriptive Analysis	Descriptive Analysis	77	S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S20, S21, S22, S30, S33, S37, S40, S41, S44, S51, S53, S55, S61, S64, S66, S24, S25, S26, S27, S28, S29, S31, S32, S34, S35, S39, S38, S42, S43, S45, S46, S47, S48, S49, S50, S52, S54, S56, S57, S58, S59, S60, S62, S63, S65, S67, S68, S69, S70, S71, S72, S73, S74, S75, S76, S77, S78, S79
Statistical Tests	Wilcoxon sign-rank test	24	S6, S10, S13, S14, S15, S17, S21, S40, S51, S55, S64, S25, S26, S32, S34, S35, S46, S47, S50, S57, S58, S62, S63, S65
	Friedman's test	9	S6, S10, S19, S22, S55, S46, S47, S50, S57
	Mann-Whitney U test	5	S1, S18, S61, S66, S58
	T-test	4	S55, S27, S52, S63
	Effect Size	3	S6, S7, S10
	Kolmogorov-Smirnov test	3	S4, S20, S26
	Correlation	2	S10, S63
	Tukey test	2	S19, S22
	ANOVA	1	S34
	Brunner Test	1	S24
Permutation tests	1	S56	
Ranking Method	Win-tie-loss	11	S8, S14, S15, S37, S41, S44, S24, S26, S34, S35, S46
	Scott-Knott	7	S4, S5, S7, S12, S20, S30, S41
	Borda count	4	S4, S5, S12, S20

Continued on next page

Table 3.11: Analysis techniques and related SEE studies. (Continued)

Friedman ranking	3	S46, S47, S57
Worse	1	S63

The descriptive analysis approach is used by all studies but two. This technique consists of textually describing the results obtained by each studies technique, usually aided by tables or figures that present the evaluation metrics. The descriptive analysis also offers insights on the most successful techniques, as well as the scenarios in which the studied techniques may or may not function adequately. This technique is mostly employed as a complement to statistical tests and ranking methods, but there are also studies that rely purely on descriptive analysis to draw conclusions.

The statistical tests approach is concerned on determining whether the studied techniques have a large enough difference in performance to be considered different. This is often performed by studies that focus on a particular approach. These studies evaluate the main approach, as well as some previously existing techniques as baselines. Afterwards, these statistical tests are used to confirm if the main approach had different performance than the baselines. Studies that instead compare multiple techniques can also employ statistical tests, but also tend to use ranking methods. We identified 11 different statistical tests: Wilcoxon sign-rank test, Friedman’s test, Mann-Whitney U test, t-test, effect size, Kolmogorov-Smirnov test, correlation, Tukey test, ANOVA, Brunner test, and permutation tests.

The ranking method approach is concerned with determining a rank or ordering of the machine learning approaches evaluated in a study. These ranking methods are based on statistical tests, whose results are aggregated to determine either an order relationship, equivalence groups, or a new metric altogether (primarily “win” counts). We encountered 5 unique ranking methods. The Win-tie-loss technique was used in 11 studies, and was the most used ranking method. Win-tie-loss compares two different techniques or treatments by counting how many wins, ties, and losses does a technique have across multiple iterations. The original win-tie-loss technique employs the Friedman test to determine each case. If the test determines a significant difference, the algorithm adds one point to the winner and subtracts one point from the loser. Otherwise, it is counted as a tie for both sides. The Scott-Knott algorithm was the second most used ranking method, with a total of 7 studies. Scott-Knott uses hierarchical clustering and the analysis of variance test to group and rank tech-

niques in equivalence clusters. This is thus used to determine a technique or group of techniques with the best accuracy. However, this approach can only analyze one evaluation metrics at a time. Three other counting methods were employed: Borda count with 4 studies, Friedman ranking with 3 studies, and worse with one study. Similar to win-tie-loss, these counting approaches rely on determining how many “wins” does a technique have across multiple iterations. The counting approaches over rankings (like Scott-Knott) is that they can be used to combine the results of different evaluation metrics.

SEE studies have favored the use of metrics that rely on the prediction error. Moreover, the majority of studies employ a metric that is a variant of relative error. Many researchers have also favored the use of absolute error metrics as an unbiased alternative. Going forward, it is possible that studies start reporting both RE and AE-type metrics. Thirty-five studies (S2, S3, S9, S11, S16, S23, S28, S29, S31, S33, S36, S38, S39, S42, S43, S45, S48, S49, S53, S54, S59, S60, S67, S68, S69, S70, S71, S72, S73, S74, S75, S76, S77, S78, S79) have rejected the use of statistical and ranking analysis techniques in favor of descriptive analysis. This is a problem for the reliability of the obtained results, as the conclusions many of these studies have reached are not backed from a statistical standpoint.

### 3.2.4 Discussion

In this section, we discuss the results obtained in the mapping study and the implications of these for research of software effort estimation. Section 3.2.4.1 summarizes some of the challenges reported by the identified studies. Section 3.2.4.2 discusses some of the open issues and possible venues of research based on the results and the challenges obtained in this mapping study. Section 3.2.5 concludes this literature mapping.

#### 3.2.4.1 Challenges

To complement the results obtained from the systematic mapping study, we discuss the current state of research in hyper-parameter tuning in SEE. To meet this end, we identified the challenges reported by each paper and categorize them based on the keywording approach [62]. We identified 28 unique categories of challenges in the literature. Table 3.12 shows the identified challenge categories as well as the studies that employ them, and figure 3.12 shows the trend of studies that report these

challenges over time.

Table 3.12: Challenge categories and related SEE studies.

Category	N	Studies
Risks of inaccurate estimation	26	S1, S3, S9, S10, S11, S13, S15, S17, S19, S22, S25, S26, S27, S29, S42, S46, S47, S49, S50, S54, S60, S63, S67, S70, S71, S72
Limitations of a technique	23	S12, S20, S40, S55, S1, S2, S4, S6, S10, S13, S16, S17, S18, S19, S24, S31, S32, S35, S45, S69, S75, S78, S79
Selection of parameters	21	S7, S12, S20, S33, S37, S51, S40, S61, S66, S64, S4, S11, S13, S16, S21, S26, S35, S45, S62, S63, S74
No free lunch	20	S12, S20, S30, S41, S61, S1, S2, S5, S9, S14, S15, S16, S18, S22, S24, S26, S42, S52, S63, S70
Lack of research	16	S30, S51, S44, S8, S10, S11, S13, S15, S16, S17, S18, S34, S50, S58, S63, S74
Quality of collected data	14	S7, S4, S6, S13, S16, S18, S24, S50, S59, S62, S69, S73, S75, S78
Prediction stability	13	S12, S20, S37, S51, S41, S61, S66, S8, S17, S18, S24, S26, S46
Computational Cost	12	S40, S44, S53, S14, S26, S32, S34, S45, S48, S62, S67, S69
Reporting quality of studies	9	S51, S41, S55, S2, S24, S46, S57, S58, S63
Adoption by the industry	7	S55, S53, S66, S2, S3, S16, S19
Accuracy of effort estimates	6	S14, S16, S22, S32, S57, S78
Cost of data collection	5	S7, S44, S53, S47, S58
Evaluation metrics	5	S55, S66, S24, S35, S73

Continued on next page

Table 3.12: Challenge categories and related SEE studies. (Continued)

Comparability of studies	3	S41, S55, S18
Insufficient data	3	S24, S35, S57
Lack of tuning approaches	3	S7, S12, S41
Online scenario	3	S7, S51, S57
Complexity of effort estimation	2	S70, S76
Generalizability of results	2	S55, S9
Interpretability of models	2	S56, S78
Lack of automation	2	S46, S47
Model obsolescence	2	S7, S57
Prediction uncertainty	2	S10, S28
Estimation through the life cycle	1	S15
Lack of techniques	1	S16
Overcomplexity	1	S17
Risks of point estimation	1	S10

The type of challenge most reported in the surveyed studies were the potential risks of inaccurate effort estimation, with 26 studies. These risks are related with the over- and under-estimation of effort, which can potentially cause loss of profit. This challenge was reported often at the beginning of a paper, as a justification of the research and use of machine learning as a more viable alternative to traditional SEE approaches. There are two periods in which this challenge was reported: 2008–2013, and 2016–2019. Both periods had, on their ending year, a spike of studies that reported this challenge. The reason that this challenge is still being reported, as well as the reasons SEE research is still active, is perhaps because the quality of effort estimates is not satisfactory for real-life purposes. However, this is contrasted by the null amount of practical case studies on real life effort estimation that we identified as part of this mapping.

The limitations of a technique challenge has been mentioned in 23 different stud-



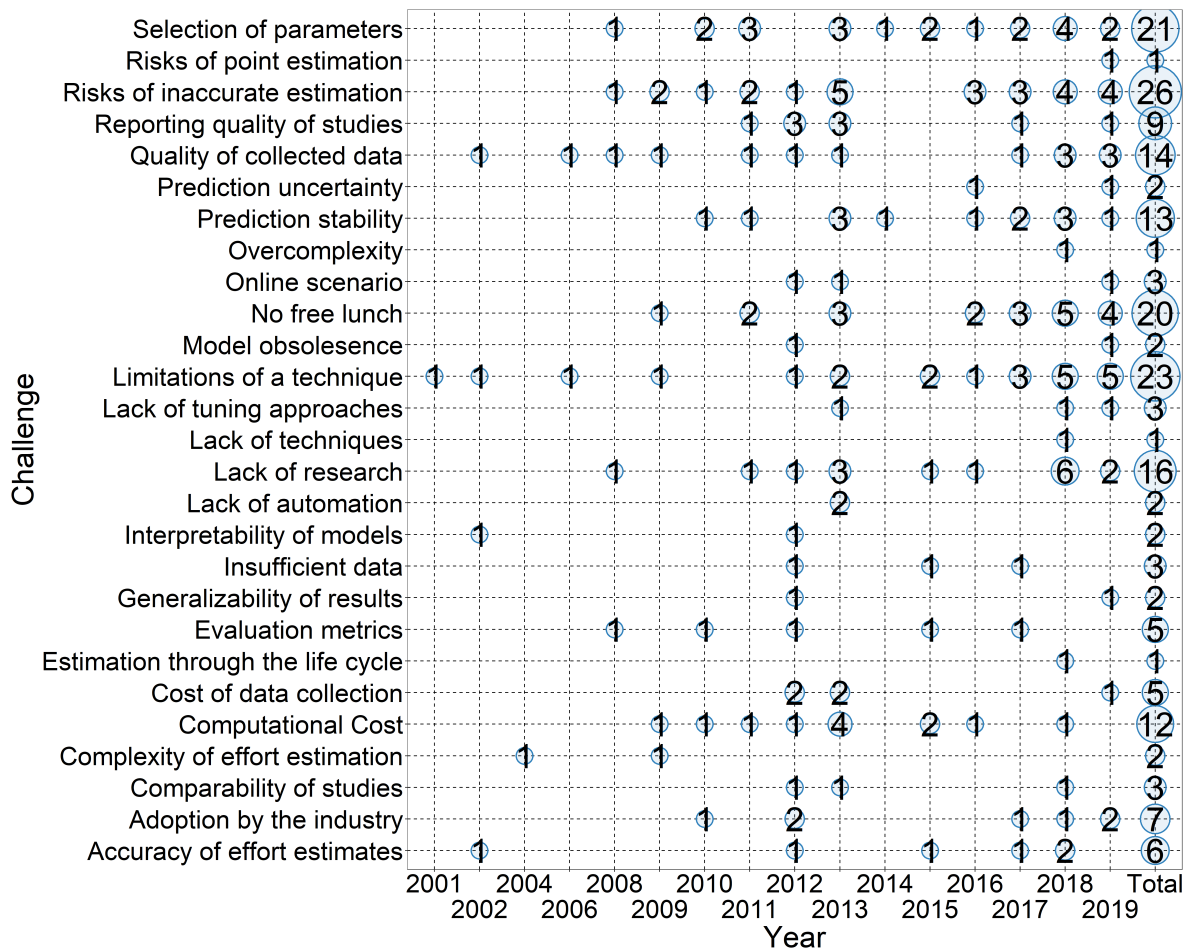


Figure 3.12: Reported challenges through time.

ies, and was the second most mentioned challenge. As its name implies, this type of challenge covers limitations about a specific machine learning technique. For example, S40 describes that one drawback of the grid search technique is that the search is always performed in the same (coarse grained) points, without taking into account the dataset to guide the search. While it has been consistently reported through the years, the amount of reports rose in the 2018 and 2019 years. This could perhaps indicate that studies have shifted their focus to addressing limitations of existing estimation techniques.

Challenges about the complexity in the selection of parameters was reported in 21 studies. This challenge is the main concern of hyper-parameter tuning for software effort estimation, and has been continuously reported across the years. As its name implies, it deals with the difficulty of manually selecting parameter values for machine learning approaches. This is often the focus of the researches studies, as they employ hyper-parameter tuning techniques to address this challenge. Starting from 2008, the challenge has been steadily reported over the years. This could indicate that, despite the amount of tuning studies performed, hyper-parameter tuning remains as a complex task that can be improved on.

The no free lunch theorem has been mentioned (although often not using this name) in 20 studies. The theorem states that there is no best machine learning technique for all existing problems, but rather there are certain techniques that are better for certain types of problem [11]. These studies have addressed this problem by performing comparisons of existing machine learning approaches, although for a limited amount of datasets. This challenge is related to the generalizability of results obtained from these studies to a real estimation scenario. This challenge has been especially reported in the 2016-2019 period. This could indicate that SEE studies could shift from single-technique evaluates to more comprehensive ones: using more effort estimation methods, more data, and more robust analysis.

The lack of research challenge has been reported in 16 studies. This challenge describes that there is a need of research studies for a particular technique or knowledge area, such as ensemble effort estimation (S30), evaluation of the impact of tuning (S51), and use of kernel methods (S44). In many cases, these challenges are reported as a justification of the research presented in the paper. Lack of research has been especially reported in the latest years, and it is a good indicator that research on hyper-parameter tuning for SEE has future work.

The quality of collected data challenge has been reported in 14 studies. These

challenge deals with the characteristics of the datasets used in SEE literature, such as heterogeneity of the software projects (S7, S16), missing values in the dataset (S4, S13), outliers (S4, S16, S18), and the accuracy of the recorded information (S6). The quality of the collected data has been a concern for SEE since its initial years, but has been a point of worry especially for the more recent studies.

The prediction stability challenge has been reported in 13 studies. This challenge has two aspects: 1) the accuracy machine learning algorithms depends on several factors (datasets, data transformation, feature selection, hyper-parameters), and 2) previous studies have reached different results with respect to the performance of the same machine learning approaches, arguably due to these factors. This challenge goes in hand with selection of parameters, as hyper-parameters are one of the factors that affect estimation, and thus stability. Initially, SEE studies focused on identified the technique with the highest accuracy. However, this challenge has been reported more and more by recent studies, as the research area matures and experimental protocols become more settled, determining that stable results are important.

A total of 12 studies report on the computational cost challenge, which describes the high dimensionality of the search space for some approaches, such as feature selectors (S44, S53), machine learning algorithms (S44), hyper-parameter values (S40), or overall amount of techniques (S14). The peak year for this challenge was 2013, and it has been reported less since then. One possibility is that computation cost becomes less of a worry as the computational power of researchers and practitioners increase.

Challenges regarding reporting quality of studies, which were reported on 9 studies, explained problems with the completeness of the reported experimental protocols in previous SEE studies. Many SEE studies let details out that difficult the replicability of results. These details include hyper-parameter tuning ranges, pre-processing applied to the dataset, specific metric calculations, among others. Such difficulty in performing a replication and lack of reporting detail could potentially make a researcher doubt the generalizability of the reported results. Contrary to prediction stability, as the research area matures the perceived quality of results could go up, as protocols and reporting procedures become more settled.

The adoption by the industry challenge has been reported in 7 studies. These challenges reported that many projects failed in the effort estimation task and resulted in delay or overbudget (S53, S66), as well as the preference of expert estimation over machine learning approaches (S55). This challenge was first reported in the 2010–

2012 period, but it has seen a rise in the most recent years. Perhaps somewhat related to potential risks of inaccurate estimation, SEE researchers are realizing that there is a lack of connection with practitioners.

Seventeen challenges have been reported in six or less studies. Accuracy of effort estimates describes that the methods in the literature are unable to provide satisfactory effort estimates. Cost of data collection explains the elevated cost of collecting project data to construct new effort estimation datasets. This challenge is related to the quality of collected data challenge, and helps explain the perceived lack of high quality datasets. It was especially reported in 2012 and 2013, indicating that there may have been an effort on those years to collect more SEE data. The evaluation metrics challenge describes that there is no clear consensus on which metrics to use (S55), or describe limitations or biases of some of the metrics used in literature, like MMRE (S66). Related to this is the challenge of comparability of studies, which reports on the difficulty, or outright impossibility, of comparing results of previous studies. One of the reasons for this was use of different metrics, but lack of reporting detail also hindered comparisons. Similar to the elevated cost of data collection, the insufficient data challenge states that the small amount of SEE datasets, as well as the amount of projects inside them, can reduce credibility of the results obtained in the area. Lack of tuning approaches was reported for online estimation (S7), ensemble estimation (S12), and interpretable methods (S41). Online scenario relates that existing studies do not take into account that effort estimation applied to real-life projects is a online problem. Other challenges include complexity of effort estimation, generalizability of results, interpretability of models, lack of automation, model obsolescence, prediction uncertainty, estimation through the life cycle, lack of machine learning techniques, overcomplexity of existing approaches, and potential risks of point estimation (in contrast with interval estimation).

#### 3.2.4.2 Open issues

Based on the results obtained in the literature mapping, as well as the identified challenges, we discuss some open issues in hyper-parameter tuning SEE literature:

**Use of tuning vs. research of tuning** There is a need to compare the existing hyper-parameter tuning approaches. Out of the 79 covered studies, only 4 focused on the evaluation of multiple hyper-parameter tuning approaches (tuning benchmark), and thus evaluated models using different tuners. The most used hyper-parameter tuning

technique was the grid search, which is an exhaustive technique that can require large computational resources, and requires manually determining a search space. With the exception of genetic algorithms, other existing tuning approaches have been used in six or less studies. Our recommendation is to empirically compare existing hyper-parameter tuning approaches, in order to determine a set of techniques that can effectively select the amount of hyper-parameters using less resources.

**“Look before you leap”** There is a particular paradox in the use of tuning for SEE, as determined by the reported challenges. Several studies have warned about the low amount of data of SEE datasets, as well as the low quality in said data. However, the most used hyper-parameter tuning technique is still the most resource consuming: grid search. Moreover, 12 studies reported on the higher costs of some of this machine learning techniques, of which tuning would only exacerbate. Given such a “simple” data and an elevated cost of tuning: is it really worth it to use grid search? We recommend SEE researchers to first analyze the properties and complexity of the dataset before using any tuning approach, and deciding whether to employ exhaustive approaches, fast tuning, or no tuning at all.

**Observations on machine learning approaches** We have identified a total of 21 broad machine learning approaches in this study. The three of the four most used approaches—regression trees, support vector regressions, and neural networks—have also the largest amount of parameters. This result reflects the research interests of hyper-parameter tuning studies in SEE: enhancing the accuracy of machine learning models through appropriate hyper-parameters. However, this could also indicate that these techniques have a better potential use in the broader areas of effort estimation. One potential avenue for research are combined methods such as ensembles or baggings, which have more hyper-parameters than single techniques.

**Feature selection and tuning** Recent studies in hyper-parameter tuning have not employed feature selection approaches, even though these approaches can further improve the accuracy of effort estimates. One possibility for lack of research in these approaches are the size of the datasets in SEE literature, which is comprised by few projects and even fewer features. Additionally, the more ‘complex’ machine learning techniques like support vector machines and neural networks may not be as reliant on the selected features as regression models. In this line, one possible solution is to employ a hybrid approach, similarly to S53 and S66, in which hyper-parameter tun-

ing and feature selection is performed by the same algorithm. Another direction for research is to compare ‘complex’ machine learning techniques and hyper-parameter tuning using feature selection.

**Tried-and-tested datasets** SEE research has focused on a particular body of datasets, mostly comprised by those that belong to the PROMISE and ISBSG repositories. The latest inclusion of a new free dataset was in 2013, with the Miyazaki94 dataset (which contains projects from 1994). Effort estimation has not seen a new publicly used dataset in more than 7 years. This is understandable, do to the elevated costs of data collection of software development projects. Ideally, one solution to this problem, as well as aiding with the adoption of these approaches, is to perform research in the industrial context. The academy could perhaps collaborate with software development practitioners to apply the existing approaches to real-world scenarios, improving quality of effort estimates. In exchange, researches could gain access to novel project data. Another possible solution is to use publicly available software repository data from sites such as github.

**Recommended metrics** Absolute error and relative error based metrics have seen widespread use in SEE research. Although critics have been made against the magnitude of relative error metrics [41], their use in the studies has prevailed. However, this use has been in conjunction with absolute error metrics. Studies continue to use relative error metrics as a means to have compatibility with previous results in SEE literature. On the other hand, baseline metrics like standardized accuracy have not been adopted by a considerable amount of studies, even though they allow for better comparability of results[41]. Our recommendation for researchers it to report their results with the standardized accuracy in conjunction with their preferred metrics.

**Online estimation** Results obtained in this mapping have shed some light on the open problems in hyper-parameter tuning for effort estimation. The area faces many challenges, such as instability of results, lack of parameter tuning approaches for certain areas, elevated cost of data collection, problems in the adoption of the proposed approaches, limited generalization of results, model obsolescence, among others. Compared to offline estimation scenarios, online estimation has been less researched. Of the 79 covered papers, only 4 research online SEE. Research in this type of estimation could help researchers see things from a different perspective, and find solutions or alternatives to some of these problems. For instance, online studies could

develop novel approaches to perform hyper-parameter tuning, alleviate or aid with model obsolescence, provide better stability. In addition, online approaches could be more compatible with real estimation scenarios, which would benefit adoption of these techniques. Moreover, as mentioned previously, collaboration in the industry could result in new project data available.

### 3.2.5 Conclusions of the mapping study

This systematic literature mapping study characterized the machine learning hyper-parameter tuning approaches used in software effort estimation. The mapping study identified 79 primary studies that utilize hyper-parameter tuning for machine learning software effort estimation.

To answer our first research question, we identified and classified hyper-parameter tuning approaches into 12 categories. Our results showed that grid search was the most used tuning approach, even though it is the most exhaustive and computationally costly. The majority (56 out of 79) of studies utilized tuning as a way to bolster the accuracy of SEE models, and only 22 studies evaluated the impact of tuning. We identified 21 machine learning algorithms, 5 cross-validation approaches, 10 data transformation techniques, and 10 feature selection approaches. Neural networks, regression trees, regression models, and support vector machines were the top 4 most used models in the surveyed studies. The most used cross-validation approach was leave-one-out. For data transformations, the predominant technique was unit range[0, 1]. Only 24 papers employed feature selection approaches, and the most used approach was correlation-based feature selection.

To answer our second research question, we identified and classified the datasets by their repository and availability. We identified 8 origins for SEE datasets. The most common repositories in the primary studies were PROMISE with 58 papers, ISBSG with 30 papers, and (other) open data with 24 papers. The most used PROMISE datasets were Desharnais, Albrecht, Kemerer, and Maxwell. For ISBSG, the most used version was ISBSG release 10. The literature in SEE favored the use of open data, increasing the replicability of the studies.

To answer our third research question, we identified and classified the evaluation metrics used by the primary studies by the base metric (type of error, correlation, etc) used. We identified 10 base metrics, and determined that SEE research favored the use of relative error and absolute error metrics. Out of 79, a total of 68 used relative error metrics. Of these, the most used were mean magnitude of relative error (MMRE)

and  $\text{Pred}(L)$ ; even though relative error metrics are known to be biased [41]. Only 37 studies employed absolute error metrics, with the mean absolute error being the most common. In addition to this, we identified and grouped the analysis techniques that these studies employed to compare the SEE models based on these metrics. We determined that almost every study (77 out of 79) utilized descriptive analysis to draw conclusions. We identified 11 statistics tests and 5 ranking methods in SEE. Many of the studies in SEE have reported their results using a biased metric, and drawing conclusions that are not backed by a statistical analysis.

We identified and categorized the challenges reported by the primary studies, in order to gain insights the state of research of hyper-parameter tuning machine learning SEE. Our analysis identified 28 categories of challenges, with the most often reported being risks of inaccurate effort estimates, limitations of individual techniques, complexity in the selection of hyper-parameters, and the no free lunch theorem (there is no overall best SEE model). Our discussion of these topics covered some open issues in the literature, including the way SEE studies have used tuning, the complexity of the datasets versus the complexity of the tuners, the types and parameters of the most used models, the relationship between feature selection and tuners, the current SEE datasets, the case of the most used metrics, and the lack of research on online estimation.

The state of hyper-parameter tuning research in SEE is currently active, as interest over tuning has kept steady over the years. All of the surveyed papers employed hyper-parameter tuning, although to varying degrees, which demonstrates that these researchers understand the value of tuning. There is a lack of evaluation on the effectivity of hyper-parameter tuning approaches (against each other), as many studies just employ grid search. Due to the quality of SEE dataset and the properties of the most used datasets, different hyper-parameter tuning approaches could achieve better or worse results. For future tuning studies, we recommend to utilize at least three hyper-parameter tuning approaches for comparison: default hyper-parameters, grid search, and any other non-exhaustive approach.



## Chapter 4

# ChimeraHPT: an automated machine learning hyper-parameter tuning framework

This chapter presents the results of an iterative development process to address the second specific objective of this thesis: to automate a hyper-parameter tuning procedure for machine learning in the context of software effort estimation (SEE). We built a procedure and framework, called ChimeraHPT, that supports machine learning and hyper-parameter tuning techniques used by SEE researchers, as identified in the literature mapping presented in chapter 3. Currently ChimeraHPT supports 40 total techniques across 5 categories: 15 hyper-parameter tuners, 8 machine learning models, 7 cross-validators, 2 data transformations, and 8 feature selectors.

ChimeraHPT is a procedure and framework mainly built on the scikit-learn<sup>1</sup> library for the Python programming language, focusing on the task of tuning and evaluating prediction models. ChimeraHPT borrows hyper-parameter tuning technique implementations from 8 additional machine learning libraries and repositories (see table 4.4), adapting them to the scikit-learn interface. We implement data analysis and statistical testing procedures for the analysis of the evaluation metrics of the constructed SEE models using the R programming language.

The ChimeraHPT procedure supports three tasks, as shown in figure 4.1: (1) pre-processing, (2) model training and evaluation, and (3) statistical analysis. Figure 4.2 shows the supported activities for each task, as well as the structure and process flow of the framework. The data pre-processing task comprises a preliminary analysis of a

---

<sup>1</sup><https://scikit-learn.org>

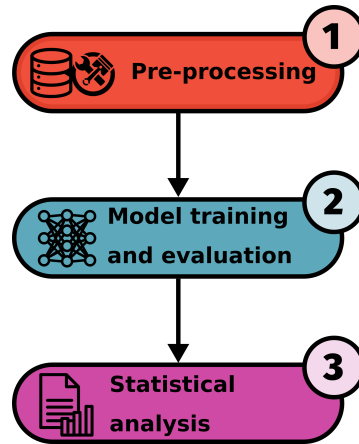


Figure 4.1: Tasks of the ChimeraHTP procedure.

dataset and its characteristics, as well as its adaptation to serve as input data for a machine learning model (Section 4.1). The model training and evaluation task involves training machine learning algorithms, their combinations with hyper-parameter tuning approaches (plus cross-validators, data transformations, and feature selectors), and evaluation of their predictive performance (Section 4.2). The statistical analysis task uses metrics collected from the previous task to draw conclusions and determine the most effective techniques for the studied data (Section 4.3). ChimeraHPT focuses on automation of model training and evaluation, and provides support of the data pre-processing and statistical analysis tasks to help researchers and practitioners in SEE to help the models learn from their data, and to determine which models are better for predicting effort for their data. The contents of this chapter are summarized on section 4.4.

## 4.1 Pre-processing

The pre-processing task analyzes a particular dataset and adapts it to be used as input data for the machine learning process. Many software effort estimation datasets contain problems such as missing data, outlier instances, redundant or unrelated features, and class imbalance (among others). In such cases, the ChimeraHPT framework selects and transforms the data to eliminate these problems and improve the model predictive performance. The framework supports three pre-processing activities, as shown on the first part of figure 4.2.

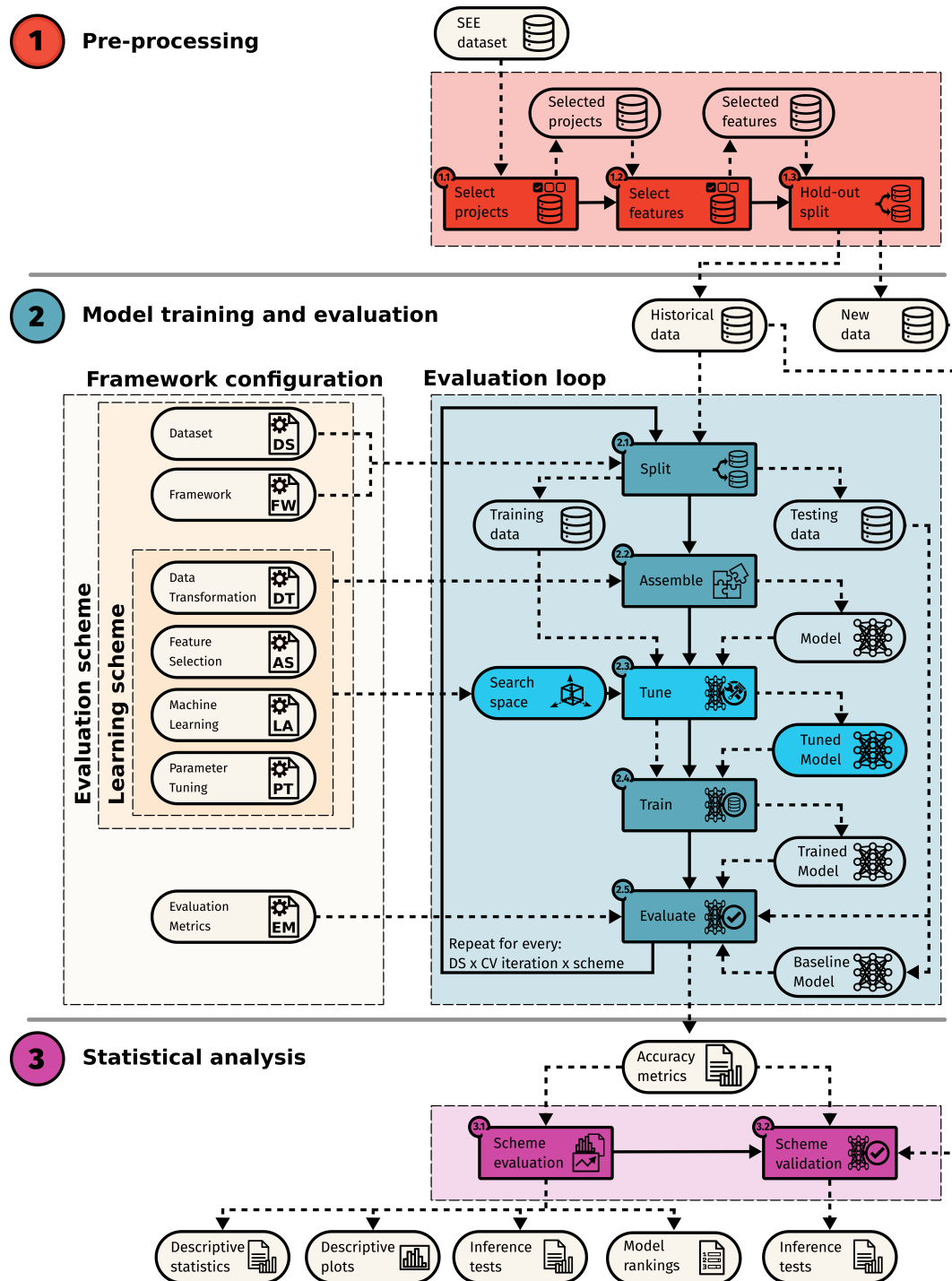


Figure 4.2: Tasks and activities of the ChimeraHTP procedure.

**Select projects** The select project activity consists of choosing a subset of all the instances (rows) of the original dataset to be used in the final project data. The selection process is done by defining selection criteria, which consists of features and a series of values. For example, if a researcher wants to only use projects with IFPUG 4+ type function points, they would define the criteria: “function point type == IFPUG 4+.” If the focus is on large development projects, the criteria “functional size  $\geq$  250” can be used. The subset of projects that meets all criteria is selected and used in the next activity.

**Select features** The select features activity consist of choosing a subset of all the features (columns) of the selected projects. Similarly to select projects, this activity is done by defining selection criteria. These criteria include manually determining the subset of features to use, excluding a type of feature (numerical or categorical), determining a threshold for missing values (i.e. all features with less than 25% missing values), and excluding low variance or one-value features. The subset of features that meets all criteria is selected and used in the next activity.

**Hold-out split** The hold-out split activity is optional and involves selecting (usually randomly) a portion of the data for use on the statistical analysis task. As explained in chapter 2, machine learning models can encounter two problems: overfitting and data mismatch. To overcome these problems, it is necessary to perform two validation approaches. ChimeraHPT is designed to perform the validation for overfitting in the second task (Section 4.2), and the validation for data mismatch on the third task (Section 4.3). The hold-out split activity takes a subset of the selected data (referred to as New data) to be used in the validation for data mismatch. The remainder data (referred to as Historical data) is used as the input data for the model training and evaluation activity. The percentage of data to be held out, as well as the selection mechanism (randomly or newest data) can be configured.

The data pre-processing task is manual, and requires constant human intervention to accurately determine the best criteria that maximizes prediction accuracy. Moreover, it is an iterative process, as a framework user can come back to this task if their models do not describe the data correctly. Because of these reasons, ChimeraHPT does not automate this task. We implemented Python scripts to perform this task for our controlled experiments, which include selection criteria for both projects and features, as well as code for the hold-out split activity.

## 4.2 Model training and evaluation

The model training and evaluation task assesses machine learning techniques applied to an estimation task. This evaluation is an iterative process; the same procedure is used to evaluate different techniques, on several datasets. As such, the evaluation task is comprised by an **evaluation loop**, which covers the assembling, training, and evaluation of a single model on a single dataset partition; repeated for each dataset and partition. Such model is built from a *learning scheme*, a combination of: data transformation, feature selection, machine learning algorithm, and hyper-parameter tuner. The combination of this learning scheme, dataset and cross-validation method is called the *evaluation scheme*. Parallel to this, the **framework configuration** component allows the modification of the techniques evaluated in this loop. The framework is configured using a set of text files, which contain the techniques and their parameters. The techniques specified in these files are combined to create all evaluation schemes. As shown on the second part of figure 4.2, the ChimeraHPT framework supports a framework configuration component (left) and implements the main evaluation loop (right). Section 4.2.1 covers the framework configuration, and section 4.2.2 covers the evaluation loop.

### 4.2.1 Framework configuration

The ChimeraHPT framework allows configuration of the techniques, datasets, and metrics that are used in the evaluation loop. This configuration is done through a series of 7 files: framework (FW), dataset (DS), data transformation (DT), attribute selection (AS), learning algorithm (LA), parameter tuning (PT), and evaluation metric (EM). These files must be placed on the `config` directory. Listing 4.1 shows the syntax for the FW file, and table 4.1 lists the supported parameters and values.

**FW – Framework** This file deals with the main configurations related to the evaluation loop. This includes the cross-validation approach to be used and its parameters. For example, the user can select k-fold cross-validation and specify the parameter k with a value of 10. The framework file requires the specification of a baseline model, which is used in the calculation of baseline metrics. Lastly, the target metrics for multi-objective tuning are specified in this file. The following is an example of the FW file:

Table 4.1: Categories, techniques, parameters and possible values for the FW file.

Cat.	Technique	Parameter	Type	Description	Ex.
CV	traintestsplit	train_size	float	The percentage of projects that are assigned to the training set	0.3
	repeatedtraintest	n_splits	int	Amount of random partitions to generate	10
		train_size	float	The percentage of projects that are assigned to the training set	0.3
	kfold	n_splits	int	Parameter k, the amount of folds to partition the data into	5
	repeatedkfold	n_splits	int	Parameter k, the amount of folds to partition the data into	3
		n_repeats	int	Amount of times the splitting is repeated	20
	leaveoneout	-	-	-	-
	bootstrap	n_bootstraps	int	Amount of partitions to generate	100
	loader	group	int	If specified, use the index files with that numeral ending	1, 2, 3
		offset	int	Amount of iterations to skip	5
Baseline	None	-	-	-	-
	marp0	-	-	-	-
	mdarp0	-	-	-	-
	median	-	-	-	-
	marp0loo	-	-	-	-
Multiobj	-	metrics	str	Name of the metrics to calculate multi-objective metrics	MAR, MMRE
		reference	float	Reference point (usually minimum) for each metric	0, 0

Listing 4.1: Syntax for the FW file.

Category: Technique  
 -parameter, type: value

**DS – dataset** This file specifies which datasets are used in the evaluation loop. The name of each dataset and its target feature must be specified in this configuration. The dataset must have the exact name and be stored in the `data` directory. In addition, small pre-processings can be specified in the file, such as which features to exclude, which rows to drop, or which symbol refers to missing values. At least one dataset must be specified. Listing 4.2 shows the syntax for the DS file, and table 4.2 lists the supported parameters and values.

Listing 4.2: Syntax for the DS file.

```
dataset.csv:
-parameter1: value
-parameter2: value1, value2, value3, ...
```

Table 4.2: Parameters and possible values for the DS file.

Parameter	Description	Possible values	Example
predict	Name of the target feature	Numerical feature name	effort
exclude	Name of features to remove from the dataset	Feature names, separated by comma	id, loc
drop	Index of rows to remove from the dataset	Row indexes, separated by comma	41, 47, 58
delimiter	Delimiter character of csv file	String, surrounded by double quotes	","
na_values	String that represents missing values in file	String, surrounded by double quotes	"NA"
as_dummies	Apply dummy encoding to categorical features	True or False	True
quotechar	Delimiter character for strings	String, surrounded by double quotes	""

Table 4.3: Parameters and possible values for the DT file.

Technique	Parameter	Type	Description	Example
None	-	-	-	-
Log	-	-	-	-
Norm	-	-	-	-

**DT – data transformation** This file specifies which data transformation techniques are used to assemble the model. If the technique allows for hyper-parameters, those can be specified either as single set values, or as a list of values for tuning. Currently, only one data transformation can be used per model, and they cannot be combined. If the file is left blank, no data transformation is assumed. Listing 4.3 shows the syntax for the DT file, and table 4.3 lists the supported parameters and values.

Listing 4.3: Syntax for the DT file.

```

technique :
-parameter1 , type : value
-parameter2 , type : value1 , value2 , value3 , ...
-parameter3 , type : [ initial , final , increase ]
-parameter4 , type : [ initial , final , increase , formula ]
-parameter5 , type : [ range1 ]; [ range2 ]; value1 , value2

```

**AS – feature selection** This file specifies which feature selection techniques are used to assemble the model. If the technique allows for hyper-parameters, those can be specified either as single set values, or as a list of values for tuning. If the file is left blank, no feature selection is assumed. Listing 4.4 shows the syntax for the AS file. Table B.1 on appendix B lists the supported techniques, parameters and values.

Listing 4.4: Syntax for the AS file.

```
technique :
-parameter1 , type : value
-parameter2 , type : value1 , value2 , value3 , ...
-parameter3 , type : [ initial , final , increase ]
-parameter4 , type : [ initial , final , increase , formula ]
-parameter5 , type : [ range1 ] ; [ range2 ] ; value1 , value2
```

**LA – learning algorithm** This file specifies which machine learning techniques are used to assemble the model. If the technique allows for hyper-parameters, those can be specified either as single set values, or as a list of values for tuning. In addition, lists of values can be specified as specific sequences, incremental sequences, or a combination of these. At least one machine learning technique must be specified. Listing 4.5 shows the syntax for the LA file. Table B.2 on appendix B lists the supported techniques, parameters and values.

Listing 4.5: Syntax for the LA file.

```
technique :
-parameter1 , type : value
-parameter2 , type : value1 , value2 , value3 , ...
-parameter3 , type : [ initial , final , increase ]
-parameter4 , type : [ initial , final , increase , formula ]
-parameter5 , type : [ range1 ] ; [ range2 ] ; value1 , value2
```

**PT – parameter tuning** This file specifies which hyper-parameter tuning approaches are used to find optimal parameter values for the estimation model. If the approach allows for hyper-parameters, those can be specified either as single set values. Universal configurations include the amount of cross-validation iterations, scoring technique (single- or multi-objective), and amount of parallel jobs. If the file is left blank, default hyper-parameters are used, except if a single value is specified in the LA file.



Listing 4.6 shows the syntax for the PT file. Table B.3 on appendix B lists the supported techniques, parameters and values.

Listing 4.6: Syntax for the PT file.

```
technique :
-parameter , type : value
```

**EM – evaluation metrics** This file specifies which metrics are calculated and saved based on the predicted values of each model. At least one metric must be specified. The list of supported evaluation metrics is found on section 4.2.3, in table 4.10.

## 4.2.2 Evaluation loop

The framework implements an evaluation loop to assemble, train, and evaluate machine learning models. The evaluation loop is comprised by the following 5 activities: 1) Split, 2) Assemble, 3) Tune, 4) Train, and 5) Evaluate. The techniques, configurations and hyper-parameters specified in the framework configuration (section 4.2.1) are loaded and used in the evaluation loop.

### 4.2.2.1 Split

The first activity is to partition the dataset(s) into training and testing data. This is done in two steps: 1) dataset loading, and 2) dataset partitioning. The result of this activity is a pair of training and testing sets, which are used in the following activities. If more than one dataset is used, or if the cross-validation approach generates more than one partition, then more than one training-testing pair is generated. The remaining activities of the evaluation loop are then performed with each generated training-testing pair.

**Dataset loading** The first step of the split activity is to load the dataset from the data repository of the framework. If pre-processing was specified in the DS file (see section 4.2.1), the specified transformations are applied in this step. If more than one dataset was included in the DS file, the dataset partitioning is individually done for each.

**Dataset partitioning** The second step from the split activity is to partition the dataset into one or more training-testing set pairs. This activity employs the cross-validation approach that was specified in the FW settings file. The result of this process is one or more pairs of train-test sets, depending on the selected cross-validation. For example, train-test split performs only one split, while leave-one-out cross-validation performs  $n$  splits, where  $n$  is the amount of projects (rows) in the dataset.

#### 4.2.2.2 Assemble

The second activity is to assemble the machine learning model. As stated previously, the model is built from a learning scheme, the combination of: data transformation, feature selection, machine learning algorithm, and hyper-parameter tuner. The result of this activity is an untrained machine learning model. The techniques that comprise the learning scheme are loaded from their respective files: DT, AS, LA, and PT. Because more than one technique can be specified in each file, more than one model can be generated. Currently, ChimeraHPT generates one model for each possible combination of techniques. For example, if 3 DTs, 2 AS, 4 MLA and 2 PT are specified, then the framework generates  $3 \times 2 \times 4 \times 2 = 48$  models. If more than one model is generated, then the remaining activities are repeated for each model.

The model is comprised by one of each type of the following techniques: data transformation, feature selection, and machine learning algorithm. The hyper-parameter tuning approach is logically considered part of the learning scheme, but in implementation the tuner works over this model instead of being part of it. Moreover, once the optimal parameters are found in the tune activity, the tuning approach is not used in the train and evaluate activities.

As part of the model assembly, a pre-processing component is added to the model. This component “overrides” the data transformation technique. This pre-processing is comprised by: 1) type selection, 2) missing-value imputation, and 3) data transformation. Type selection divides the input data by numerical and categorical features, so that different pre-processing is applied to each. Missing value imputation overwrites missing values using information derived from the available data. For numerical features, K-nearest neighbor imputation is used; and for categorical features, a new category (missing) is used. This is done because many of the scikit-learn methods are not implemented to handle missing values. Lastly, the data transformation step applies the data transformation approach from the current learning scheme to

the numerical features. The categorical features are instead converted using one-hot encoding, but no data transformation is applied to them. Thus, the final dataset is comprised only by numerical features. Similar to missing values, many methods of the scikit-learn only work on numerical features.

#### 4.2.2.3 Tune

The third activity is to tune the model hyper-parameters. This is done in two steps: 1) determine the search space, and 2) tuning. The result of this step is a tuned model.

**Determine the search space** Before tuning is performed, the framework determines the search space of each technique: data transformation, feature selection, and machine learning model. This is done by identifying the hyper-parameters from their respective configuration file that have more than one specified value. The hyper-parameters with only one possible value are fixed, and don't form part of the tuning process. The search spaces of each technique are combined into a joint search space. This way, the hyper-parameter tuner finds optimal values for the three techniques at the same time.

**Tuning** After the search space is determined, the hyper-parameter tuning approach then uses the model and search space to find optimal hyper-parameter values. As shown in figure 4.3, tuning typically consists of (1) determining the next point of exploration based on information of the search space, and (2) training and evaluating the model using the selected parameters on the current training set. Because this process requires training and evaluating a model for each exploration point, the tuning approach employs a cross-validation approach to split the training set into the sub-training and validation sets. The model is trained using the selected parameters on the sub-training set, and it is used to predict the values for the validation set. The target metrics are then calculated from the predicted and actual values. The best exploration point in the searched space is selected, and the hyper-parameters of this point are used in the train activity. Note that the hyper-parameters are optimized for the training set, and the performance may be different in the evaluation activity.

Hyper-parameter tuners in ChimeraHPT can be configured to optimize a single metric, or to perform multi-objective optimization. The advantage of utilizing multi-objective optimization is that the framework searches for hyper-parameters that potentially achieve good results in multiple metrics, resulting in an overall more compre-

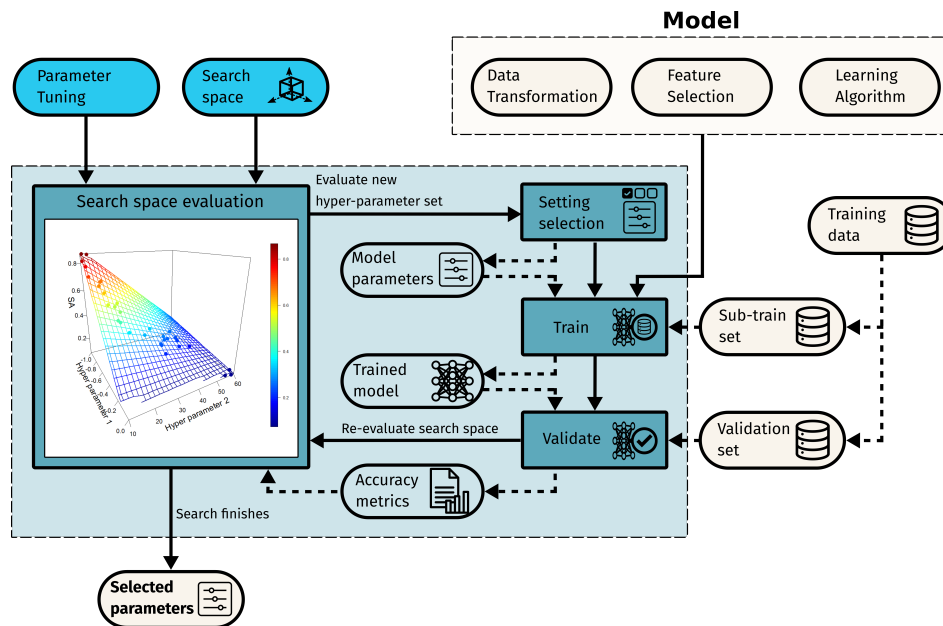


Figure 4.3: Hyper-parameter tuning in ChimeraHPT.

hensive model. Moreover, multi-objective optimization provides solutions that offer trade-offs between two or more metrics, and the final decision on which model to employ can. This is done through the PT file (see section 4.2.1). The cross-validation, target metrics, and multi-objective optimization can be configured in these settings as well.

The results of the hyper-parameter optimization are recorded. Each of the explored hyper-parameters as well as their metrics are stored in a hyper-parameter tuning results file.

#### 4.2.2.4 Train

The fourth activity is to train the tuned model. After optimal hyper-parameter values are found, the model is trained one last time using these hyper-parameters, and using as input the complete training set. This activity results on the final estimation model, which is used in the evaluate activity.

#### 4.2.2.5 Evaluate

The fifth activity is to evaluate the estimation model. This is accomplished by having the model predict the values of the testing set (without access to the correct values). The evaluation metrics are then calculated from the predicted and actual values. The

Table 4.4: Libraries used in for the construction of ChimeraHPT.

Library	URL	Purpose
scikit-learn	<a href="https://scikit-learn.org/stable/">https://scikit-learn.org/stable/</a>	Base library of the framework.
OIL	<a href="https://github.com/arennax/effort_oil_2019">https://github.com/arennax/effort_oil_2019</a>	Hyper-parameter tuning techniques.
Dodge	<a href="https://github.com/amritbhanu/Dodge">https://github.com/amritbhanu/Dodge</a>	Hyper-parameter tuning techniques.
genetic algorithm	<a href="https://pypi.org/project/geneticalgorithm/">https://pypi.org/project/geneticalgorithm/</a>	Hyper-parameter tuning techniques.
Solid	<a href="https://100.github.io/Solid/">https://100.github.io/Solid/</a>	Hyper-parameter tuning techniques.
hyperband	<a href="https://github.com/zygmuntz/hyperband">https://github.com/zygmuntz/hyperband</a>	Hyper-parameter tuning techniques.
Nevergrad	<a href="https://github.com/FacebookResearch/Nevergrad">https://github.com/FacebookResearch/Nevergrad</a>	Hyper-parameter tuning techniques.
BorutaPy	<a href="https://github.com/scikit-learn-contrib/boruta_py">https://github.com/scikit-learn-contrib/boruta_py</a>	Feature selection techniques.
Intrinsic Dimension	<a href="https://github.com/XueqiYang/intrinsic_dimension">https://github.com/XueqiYang/intrinsic_dimension</a>	Intrinsic dimensionality calculator for datasets.

specific evaluation metrics can be configured in the EM file (see section 4.2.1). These metrics are recorded in a results file, along with the techniques that comprise the current evaluation scheme: the learning scheme, the dataset, the cross-validation approach, and the selected hyper-parameters. In the case of multi-objective optimization each viable solution (i.e. every point in the pareto front) is recorded in a separate file, and the main file reports the average values for each metric.

### 4.2.3 Machine learning techniques

The ChimeraHPT framework combines different libraries to provide a wide selection of machine learning techniques from the SEE literature. Table 4.4 shows the repositories from which the framework includes, extends, or adapts techniques. The following subsections cover the list of available techniques per type. New techniques can be included in the framework by altering the corresponding “database” file. The explanation of the implemented techniques is available in chapter 2.

#### 4.2.3.1 Machine learning algorithms

Table 4.5 shows the supported machine learning algorithms, their libraries, and important hyper-parameters. Additional regression algorithms can be included, and we

Table 4.5: Machine learning algorithms supported in ChimeraHPT.

Technique	Library	Main hyper-parameters
K-Nearest Neighbors	scikit-learn	n_neighbors, weights, p
Multi-Layer Perceptron	scikit-learn	n_layers, n_hidden, activation, solver, batch_size, learning_rate, max_iter, alpha
Linear Regression	scikit-learn	fit_intercept, normalize
Ridge Regression	scikit-learn	fit_intercept, normalize, alpha, solver, max_iter
Support Vector Regression	scikit-learn	kernel, degree, gamma, C, epsilon
CART	scikit-learn	max_depth, min_samples_split, min_samples_leaf, max_features, min_impurity_decrease
Bagging	scikit-learn	n_estimators, max_samples, max_features, bootstrap
Mean	scikit-learn	-

only provided a base selection. Each technique supports hyper-parametrization as described in the scikit-learn documentation.

#### 4.2.3.2 Hyper-parameter tuning

Table 4.6 shows the supported hyper-parameter tuners, their libraries, and important configurations. In addition to the supported parameters, each technique supports 3 universal parameters: scoring (target metric to optimize, more than one for multi-objective optimization), cv (number of k-fold cross-validation folds), and n\_jobs (number of threads to use).

#### 4.2.3.3 Cross-validation

Table 4.7 shows the supported cross-validation approaches, their libraries, and important parameters.

The Loader method is an utility cross-validation that loads pre-fabricated cross-validation partitions. This permits to run the same CV approach and partitions in different computers.

#### 4.2.3.4 Data transformation

We adapted data transformations from the numpy library to be compatible with the scikit-learn pipeline. Thus, any numerical function that supports numpy notation can be used as a data transformation. Table 4.8 shows the supported data transformations, their libraries, and important parameters.

Table 4.6: Hyper-parameter tuners supported in ChimeraHPT.

Technique	Library	Main hyper-parameters
Default	ChimeraHPT	-
Grid Search	scikit-learn	-
Random Search	scikit-learn	n_iter
Random Range Search	ChimeraHPT	n_iter
Differential Evolution	OIL	mutation_rate, crossover_rate, population_size, iterations
Flash	OIL	budget, population_size, initial_size
Dodge	Dodge	epsilon, initial_size, population_size
Genetic Algorithm	genetic algorithm	max_num_iteration, population_size, mutation_probability, elit_ratio, crossover_probability, parents_portion, crossover_type, max_iteration_without_improv
Compact Genetic Algorithm	Nevergrad	budget
1+1 Genetic Algorithm	Nevergrad	budget
Particle Swarm Optimization	Nevergrad	budget, popsize
Bayesian Optimization	Nevergrad	budget
Tabu Search	Solid	tabu_size, max_steps, neighborhood_size
Harmony Search	Solid	memory_size, memory_considering_rate, pitch_adjustment_rate, fret_width, max_steps
Hyperband	hyperband	budget, eta

Table 4.7: Cross-validation approaches supported in ChimeraHPT.

Technique	Library	Main hyper-parameters
Train Test Split	scikit-learn	train_size
Repeated Train Test Split	scikit-learn	train_size, n_splits
K Fold	scikit-learn	n_splits
Repeated K Fold	scikit-learn	n_splits, n_repeats
Leave One Out	scikit-learn	-
Bootstrap	ChimeraHPT	n_bootstraps
Loader	ChimeraHPT	group, offset

Table 4.8: Data transformations supported in ChimeraHPT.

Technique	Library	Main hyper-parameters
None	scikit-learn	-
Logarithm	scikit-learn	-
Standardization	scikit-learn	-

Table 4.9: Feature selectors supported in ChimeraHPT.

Technique	Library	Type	Main hyper-parameters
None	scikit-learn	-	
Variance Threshold	scikit-learn	Filter	threshold
Correlation Percentile	scikit-learn	Filter	percentile
K Best	scikit-learn	Filter	score_func, k
False Positive Rate	scikit-learn	Filter	score_func, alpha
False Discovery Rate	scikit-learn	Filter	score_func, alpha
Family-Wise Error Rate	scikit-learn	Filter	score_func, alpha
Recursive Feature Elimination	scikit-learn	Wrapper	n_features_to_select, step
Random Forest	BorutaPY	Filter	n_estimators

#### 4.2.3.5 Feature selection

Table 4.9 shows the supported feature selectors, their libraries, its type between filter and wrapper, and important parameters.

#### 4.2.3.6 Evaluation Metrics

Table 4.10 shows the supported evaluation metrics, a brief explanation and formulas. In the presented formulas,  $y$  are the real effort values,  $\hat{y}$  are the effort values estimated by the model, and  $n$  is the amount of estimations (and real values). For hypervolume,  $m$  is the amount of specified metrics in the FW file,  $metric_i$  is the  $i$ -th specified metric, and  $reference_i$  is the  $i$ -th reference point.

#### 4.2.3.7 Baselines

Table 4.11 shows the supported set of baseline estimators for the calculation of baseline techniques: standardized accuracy and  $\Delta$ .

## 4.3 Statistical analysis

The statistical analysis task covers the analysis and verification of the evaluation metrics obtained as part of the model training and evaluation task. The objective of this task is to draw conclusions on the effectiveness of the evaluated techniques relative to the studied datasets. We identified two statistical analysis activities, as shown on the third part of figure 4.2: model evaluation, and model verification.



Table 4.10: Evaluation metrics supported in ChimeraHPT.

Metric	Description	Formula
MMRE	Mean error as a percentage of original effort	$(1/n) \sum_{i=1}^n  y_i - \hat{y}_i /y_i$
MMRE	Median error as a percentage of original effort	$median( y_i - \hat{y}_i /y_i, i = 1, \dots, n)$
Pred(L)	Percentage of predictions with relative error below a user-defined threshold, L	$count( y_i - \hat{y}_i /y_i \leq L, i = 1, \dots, n)$
MAR/MAE	Mean absolute error	$(1/n) \sum_{i=1}^n  y_i - \hat{y}_i $
MdAR/MdAE	Median absolute error	$median( y_i - \hat{y}_i , i = 1, \dots, n)$
SdAR/SdAE	Standard deviation of the absolute error	$sd( y_i - \hat{y}_i , i = 1, \dots, n)$
SA	Standardized accuracy, percentage of improvement over a baseline estimator $p_0$	$1 - MAR/MAR_{p_0}$
$\Delta$	Effect size of SA, how large is the difference between the estimator and baseline $p_0$	$(MAR - MAR_{p_0})/SdAR_{p_0}$
MBRE	Mean error as a percentage of the minimum between original effort and predicted effort	$(1/n) \sum_{i=1}^n  y_i - \hat{y}_i /\min(y_i, \hat{y}_i)$
MIBRE	Mean error as a percentage of the maximum between original effort and predicted effort	$(1/n) \sum_{i=1}^n  y_i - \hat{y}_i /\max(y_i, \hat{y}_i)$
SCC	Spearman's correlation coefficient between actual and predicted values	$covariance(y, \hat{y})/(sd(y) \cdot sd(\hat{y}))$
Hypervolume	Hyper-volume of the $m$ metrics specified in the FW file	$\prod_{i=1}^m metric_i(y, \hat{y}) - reference_i$

Table 4.11: Baseline estimators supported in ChimeraHPT.

Name	Reference	Description
$MAR_{p_0}$	[43]	Estimates effort as a random guess of the values in the testing set, repeated 1000 times. After this, calculate the mean absolute error of the estimates.
$MdAR_{p_0}$	[12]	Estimates effort as a random guess of the values in the testing set, repeated 1000 times. After this, calculate the median absolute error of the estimates.
<i>Median</i>	-	Estimates effort as the median of the dataset. After this, calculate the mean absolute error of the estimates.
$MAR_{p_0LOO}$	[43]	Estimates effort as a random guess of the values in the training and testing set, repeated 1000 times. After this, calculate the mean absolute error of the estimates. Use only for leave-one-out cross-validation.

### 4.3.1 Model evaluation

The model evaluation activity consists of employing statistical analysis to draw conclusions from the metrics collected for each of the models. This includes descriptive statistics to describe the distribution of metrics, statistical inference to determine relationships (such as equality) between the studied techniques, and plotting to visualize such distributions or relationships.

The model evaluation activity is manual, as the analysis highly depends on the studied datasets, the particular questions of the user, the techniques used, the selected metrics, and the depth of the analysis. Because of these reasons, ChimeraHPT does not automate this task. We implemented R scripts to perform this task for our controlled experiments. These scripts analyze the obtained results using plotting, statistical tests, and ranking methods.

We recommend the following procedures when analyzing hyper-parameter tuning approaches, which were obtained from the experience of conducting the experiments:

- Plotting the distribution of metric to check they lie in the expected range. For example, checking that  $SA$  is between 0 and 1 and  $MAR$  is greater than 0.
- Box-plotting or describing the main statistics of each metric (median, quantiles, inter-quantile range) for each model.
- Calculating metric improvement with respect to default hyper-parameters for each studied tuning approach and model, and checking that the difference is positive.
- Performing a statistical test (Wilcoxon signed-rank test) to determine whether tuning improves accuracy ( $SA$ ,  $MAR$ ) with respect to default parameters.
- Use Scott-Knott analysis to rank each of the studied models per dataset. Determine which models predict more accurately.
- Use Scott-Knott analysis on the normalized rankings of the previous analysis to give an overall score to the studied models.
- Use the win-tie-loss algorithm based on a statistical test (Wilcoxon signed-rank test) to compare two types of model and determine which scenarios are better. For example, comparing single- vs. multi-objective optimization.

### 4.3.2 Model verification

The model verification activity consists of reviewing that the metrics obtained in the model training and evaluation task are accurate, and not affected by problems such as data mismatch and overfitting. This is accomplished by evaluating the New data set that was reserved on the data pre-processing task.

This activity is currently supported as part of the model training and evaluation. This is accomplished by using a the loader cross-validation approach, which uses the Historical data as training and the New data as testing. For this task, we employed the Kolmogorov-Smirnov test as a goodness-of-fit test between the obtained evaluation metrics from the model training and evaluation activity, and the accuracy metric obtained from this activity.

## 4.4 Summary

We proposed the ChimeraHPT framework as an automated hyper-parameter tuning machine learning evaluation procedure. ChimeraHPT allows for construction and evaluation of SEE models, but can be extended to other domains. The framework supports three main tasks, with varying degrees of automation: data pre-processing, model training and evaluation, and statistical analysis. The framework can be configured with respect to: the machine learning techniques, the metrics to use, the ranges of hyper-parameters to tune, among others.

Considerations for extensions of this framework include some of the following:

- Further automation of the data pre-processing and statistical analysis tasks.
- Customization of the learning scheme. For example, building models that include multiple data transformations or feature selectors.
- Addition of clustering algorithms, allowing models to be trained according to different categories. For example, having different models to estimate different functional size categories.
- Support for ensemble models and customization of their inner techniques.
- Adding statistical analysis during the evaluation loop, giving constant feedback about the results during the execution.

- Estimating the duration or complexity of an execution based on the hyper-parameter search space, total techniques to analyze, dataset and cross-validation approach.
- Providing support for hyper-parameter tuning of hyper-parameter tuning approaches.

## Chapter 5

# Evaluation of the effectiveness of the automated hyper-parameter tuning procedure: a series of quasi experiments

This chapter presents the results of a series of quasi experiments to address the third specific objective of this thesis: to evaluate the effectiveness of the automated hyper-parameter tuning procedure for machine learning in the context of software effort estimation (SEE).

In total, five quasi experiments were carried on as part of this research. The quasi experiments were designed and executed based on the guidelines by Wohlin et al. [64]. This chapter presents the quasi experiments as a collection of conference papers, of which the candidate was the first author and main contributor. Table 5.1 shows the five quasi experiments, their reference, the related section of this chapter, the appendix which contains the full text publication, the motivation behind the study, and its additions (in terms of experimental design) with respect to the previous quasi experiment.

The following sections summarize the design and findings of each of the quasi experiments. The respective publications are available as appendixes to the thesis document. Section 5.1 contains an evaluation of the grid and random search techniques. The study in section 5.2 aggregates to the previous evaluation the dodge tuner and the classification and regression trees model. Section 5.3 relates the evaluation of additional nature-based hyper-parameter tuners, including genetic algorithms. Sec-

Table 5.1: Quasi experiments of the third specific objective.

Title	Ref.	Sec.	Apx.	Motivation	Additions
Evaluation of Grid and Random Search for Support Vector Regression	[127]	5.1	D	To evaluate the impact of hyper-parameter tuning applied to support vector regression for SEE, and to compare the performance of grid search against random search.	Dataset: 4 sub-sets of the ISBSG R18. Data transformation: Log. Feature selection: variance threshold, correlation percentile. Learning algorithms: SVR, ridge. Parameter tuning: grid search, random search.
Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation	[128]	5.2	E	To evaluate the impact of hyper-parameter tuning applied to classification and regression trees for SEE, to compare the performance of dodge against grid and random search, and to determine which of the studied models is better for SEE.	Learning algorithms: CART. Parameter tuning: Dodge.
Hyper-parameter Tuning using Genetic Algorithms for Software Effort Estimation	[129]	5.3	F	To evaluate the impact of genetic and bio-inspired algorithms for hyper-parameter tuning with respect to grid and random search.	Parameter tuning: genetic algorithm, compact GA, 1+1 GA, harmony search, tabu search.
Multi-objective Hyper-parameter Tuning for Software Effort Estimation	[130]	5.4	G	To evaluate the impact of multi-objective optimization for hyper-parameter tuning with respect to single-objective optimization, and to determine which of the studied models is better for SEE.	Parameter tuning: flash, differential evolution, particle swarm optimization, Bayesian optimization, hyperband. A multi-objective variant for each tuner.
Comparative study of Random Search Hyper-Parameter Tuning for Software Effort Estimation	[131]	5.5	H	To compare the impact of state-of-the-art tuners in SEE against random search in ISBSG and PROMISE datasets.	Dataset: albrecht, china, desharnais, finnish, isbsg10, kemerer, kitchenham, maxwell, miyazaki94.

tion 5.4 presents the evaluation of hyper-parameter tuners adapted for multi-objective optimization, targeting both accuracy and stability. Section 5.5 presents a comprehensive evaluation of state-of-the-art hyper-parameter tuners against random search. Section 5.6 summarizes the results of all quasi experiments and draws trends and patterns from these findings.

## 5.1 Quasi experiment 1: Evaluation of grid and random search for support vector regression

### 5.1.1 Study summary

The first quasi experiment [127] investigates the random search hyper-parameter tuning algorithm applied to software effort estimation, particularly in its ability to

improve the accuracy and stability of support vector regression and ridge regression models. Random search is compared to the exhaustive, more expensive, and widely used grid search algorithm. To perform this evaluation, the ISBSG R18 dataset was used to train and evaluate the models. For data transformations, we compared those models built using no transformation against the log transformation. We also used two feature selectors, one based on selection by correlation, and the other based on selection by variance. The models were evaluated on two metrics: standardized accuracy as a measure for accuracy, and the standard deviation of the absolute error as a measure of stability. We employed the Scott-Knott [132] procedure to produce a ranking of these models for each dataset.

Directions for future work were highlighted in the study. The first one was the extension of this quasi experiment to include additional machine learning algorithms, such as CART, and additional hyper-parameter tuners, such as Dodge, genetic algorithms, and other search methods. The second one was to replicate the study on other SEE datasets, such as PROMISE datasets. The third line for future work was the extension of this study to cover other types of data transformation and feature selection approaches. Lastly, the fourth proposal was to further research the properties of the ISBSG R18 dataset.

### 5.1.2 Main results

This quasi experiment confirmed previous findings in SEE literature with respect to the effect of hyper-parameter tuning on effort estimation models. Hyper-parameter tuning improves both the accuracy and stability of SVR models. Our results were also in line with previous accuracy values reported in SEE literature using tuning, confirming that our techniques are on the same level as the state of the art.

Across the evaluated datasets, random search was able to increase the standardized accuracy of the SVR model with log transformation to up to 0.227. This indicates that tuning made the model 22.7% more accurate with respect to randomly guessing effort values. Random search also was able to improve the model stability, bringing it to a minimum ratio of 0.84. This can be interpreted as having 16% less standard deviation of the prediction error than default hyper-parameters.

In addition, our results showed that the random search technique was able to obtain results that are similar to grid search tuned models. In all cases, the Scott-Knott algorithm determined that SVR models tuned with grid search were equivalent to their random search tuned counterpart. The largest median difference between

techniques was less than 0.05 SA for accuracy, and less than 5% improvement in the case of stability. However, random search required far less evaluations compared to grid search. Grid search explored a total of 4,128 hyper-parameter values for each SVR model. Random search randomly sampled 60 of these values, and still obtained results that are on the same line. These results show that a less exhaustive approach like random search is a viable alternative for hyper-parameter tuning.

## 5.2 Quasi experiment 2: Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation

### 5.2.1 Study summary

The second quasi experiment [128] extends our first one by further evaluating the dodge hyper-parameter tuning algorithm alongside random and grid search. Moreover, the study includes classification and regression trees (CART) as an additional model to evaluate tuning.

The remainder of the experimental design is similar to our previous study. We employed the ISBSG R18 dataset. A total of 40 models were evaluated, product of our two 2 transformations (none and log), 3 feature selectors (none, correlation percentile, 3 models (CART, SVR, ridge), and 4 tuners (default, random, grid, dodge). Because feature selection had no effect on CART and SVR, those combinations were discarded, and 40 models remained. We measured their accuracy in terms of standardized accuracy and their stability on the standard deviation of absolute error.

We performed multiple additions to the experimental design with respect to our previous publication. Firstly, we added the classification and regression tree as a machine learning algorithm, in order to determine the effect of tuning for this technique. Next, we utilized the novel dodge hyper-parameter tuning approach, and compared its performance against grid and random searches. Lastly, to extend our analysis, we added a second Scott-Knott evaluation based on the results of the first Scott-Knott groups. By doing this we are giving a general ranking of each technique, based on their ranking for each dataset.

Directions for future work were highlighted in the study. The first one was to extend the study for other SEE datasets, such as PROMISE datasets. The second one



was the extension of the quasi experiment to include additional machine learning algorithms, hyper-parameter tuners, data transformation and feature selection approaches. The third line for future work was a proposal to research the effect of adjusting the parameters of the hyper-parameter approaches, such as the  $\epsilon$  hyper-parameter in dodge.

### 5.2.2 Main results

Hyper-parameter tuning was able to improve the accuracy and stability of both CART and SVR models. For CART models, tuning improved accuracy by a maximum of 0.153 SA with respect to default hyper-parameters. This signifies a decrease of 15.3% of prediction error with respect to randomly guessing. Tuning also reduced the stability ratio to 0.819: a 18.1% decrease in standard deviation of the absolute error with respect to default hyper-parameters. For SVR, tuning improved SA by 0.227 and accuracy to a ratio of 0.829. In general terms, tuning had a larger increase of accuracy in SVR than CART. However, tuning had a larger improvement of stability (lower stability ratio) in CART than SVR.

Regarding absolute results, we found that the combination of log, SVR and tuning provided the greatest accuracy across the 40 evaluated models. These models consistently achieved first ranking in the Scott-Knott analysis across the 4 datasets. However, the secondary Scott-Knott analysis determined that tuned SVR and CART had statistically equal performance, and were categorized in the first group. The second group consisted of untuned SVR and CART. The third and last group consisted of all ridge techniques, tuned and untuned. By this analysis we determined that the accuracy of tuned CART and SVR is better than their untuned counterparts; and the accuracy of all CART and SVR models was overall better than any ridge method. While accuracy of tuned CART models was slightly lower, they had better stability than SVR models, and they provide a trade-off between the two metrics with good accuracy.

Similar to our previous study, we found that dodge provides improvements in CART and SVR that are comparable to grid and random search. SVR with no data transformation was the exception, and it ranked 1 group below other tuned SVR and CART models. The advantage to dodge was that, similar to random search, it required less explored hyper-parameter values than grid search.

## 5.3 Quasi experiment 3: Hyper-parameter Tuning using Genetic Algorithms for Software Effort Estimation

### 5.3.1 Study summary

The third quasi experiment [129] further extends the scope of hyper-parameter tuning by incorporating techniques that have been used in SEE literature as well as outside SEE. We focused on optimization techniques inspired in nature. The new evaluated tuners include three variants of genetic algorithms: classic genetic algorithm, compact genetic algorithm, and the 1+1 genetic algorithm. In addition, we include the harmony search and tabu search tuners.

We performed multiple additions to the experimental design with respect to our previous publication. Firstly, the study included 5 additional hyper-parameter tuning algorithms: GA, compact GA, and the 1+1 GA, and tabu and harmony searches. The analysis of the study thus focused on the comparison between these tuners and our previously evaluated tuners (grid, random, dodge). This study also introduced standardized stability, a metric based of standardized accuracy, as a measure for model stability.

Directions for future work were highlighted in the study. Firstly, our intent was to next extend the study for other datasets used in SEE. We also intended to evaluate further machine learning algorithms and their improvements in accuracy and stability when tuned. Lastly, we proposed the evaluation of tuning the parameters of the hyper-parameter tuning approaches.

### 5.3.2 Main results

Following the trend of our previous research, the novel tuning algorithms produced SVR and CART with accuracy and stability scores similar or better to our baseline grid and random searches. Genetic algorithms provided a maximum increase of 0.21 standardized accuracy (SA) and 0.23 standardized stability (SD). Compact genetic achieved up to 0.21 SA and 0.22 SD. 1+1 GA achieved up to 0.21 SA and 0.14 SD. Harmony achieved 0.20 SA and 0.14 SD. Lastly, Tabu achieved up to 0.21 increase in SA and 0.13 in SD. For comparison, grid and random search obtained up to 0.20 SA

and 0.13 SD.

Our analysis determined that tuned Log+SVR, CART, and Log+CART have the best accuracy of all studied models. However, we also determined that tuned SVR without the log transformation performs similarly to untuned SVR. This indicates that data pre-processing techniques can also impact the degree in which tuning can improve a model.

Regarding the question of the best evaluated model, the most effective combinations for obtaining high accuracy was Log+SVR with either traditional, compact, or 1+1 genetic algorithms. These CGA and GA combinations obtained the top ranking across all datasets, while also maintaining a level of stability comparable with the other tuners. The 1+1 tuned Log+SVR model ranked first in the COSMIC BFC set, and second in the remaining three. From our previous studies, this is the first time we have identified a method that has been able to break parity with the baseline grid search method.

## 5.4 Quasi experiment 4: Multi-objective Hyper-parameter Tuning for Software Effort Estimation

### 5.4.1 Study summary

The fourth quasi experiment [130] evaluates the effectiveness of multi-objective optimization applied to our hyper-parameter tuners. We implement multi-objective versions of all the tuners we have evaluated previously, plus the new tuners included in this study: flash, differential evolution, particle swarm optimization, Bayesian optimization, and hyperband. Instead of finding one best solution, multi-objective tuners present multiple, equally viable hyper-parameters that offer a trade-off between accuracy and stability. Our quasi experiment evaluates whether these tuners can offer accuracy and stability that is equal or better than their single objective counterparts.

Regarding analysis of results, we performed two evaluation activities. The first was to perform a ranking of all single-objective tuned techniques based on their accuracy and stability, similar to our previous quasi experiments. The second activity performed a one on one comparison of each multi-objective tuned model against its single-objective optimized counterpart, using the Wilcoxon signed-rank statistical test. To better analyze these results, we used the win-tie-loss algorithm to determine

in which scenarios was multi-objective tuning effective, and which models were not improved.

We performed multiple additions to the experimental design with respect to our previous publication. Mainly, we implemented multi-objective variants for each of the previously utilized and new hyper-parameter tuners. Such variants finds equally viable solutions that offer trade-offs between the optimized metrics. In addition, the study included 5 additional hyper-parameter tuning algorithms: differential evolution, flash, particle swarm optimization, Bayesian optimization, and hyperband. Compared to previous quasi experiments, we extended our analysis of results by using the win-tie-loss method as a means to compare multi-objective tuning, and determine whether it improves, maintains, or worsens stability and accuracy of the produced effort estimates.

Future work discussed in the study included the exploration of additional SEE datasets, machine learning algorithms, and data transformations. Also, similar to previous studies, we have discussed the possible value of exploring the effect of different configurations for the parameters of the hyper-parameter tuners are modified.

## 5.4.2 Main results

Regarding single-objective tuners, the Scott-Knott analysis determined that the top performing group in terms of accuracy is comprised by tuned Log+SVR models, except those using grid and random search. Particularly, traditional and compact genetic algorithms, as well as differential evolution and flash resulted in top-ranking models in the four studied datasets. For accuracy of Log+SVR, Flash obtained a maximum 0.537 SA; compact genetic, 0.534 SA; differential evolution, 0.534 SA; and genetic algorithm, 0.532 SA. These four methods also resulted in the top-ranking stability when performing this analysis on the standardized stability. For Flash, a SD of 0.450 was achieved; for genetic, 0.440 SD; for differential evolution, 0.430 SD; and for compact genetic, 0.426 SD. In general terms, this indicates that these models are up to 53% more accurate and 45% more stable than random guessing models. The impact of tuning (with respect to default hyper-parameters) was an increase in 0.2 SA, and 0.234 SD. Tuning was more effective in datasets that utilized full functional points, and models were more accurate on IFPUG 4+ datasets.

One important difference between the result of this and the previous study was on the Scott-Knott ranking on the accuracy of the models. Previously, the Scott-Knott method determined that dodge, tabu, and harmony search were not different

to random and grid search tuned models. In this study, for Log+SVR models, the analysis determined that these three methods have a higher increase on accuracy than grid and random searches. By adding more techniques to the analysis, the amount of Scott-Knott groups increased, and so did the gap (in terms of groups) between the best tuners and the baseline tuners.

The effect of multi-objective optimization depended on the base model. In the case of Log+SVR, the best model for single-objective optimization, resulted in more stable models at the cost of lowering their accuracy. Conversely, the best 4 tuners saw a decrease in effectiveness when switched to a multi-objective target. Results for CART and Log+CART were more consistent, showing that multi-objective optimization rarely decreased either stability or accuracy.

When analyzing the effect of multi-objective optimization on each tuner, the difference between techniques is less apparent. The four tuners with the best improvements in accuracy (CGA, DE, FS, and GA) were the ones that saw the greater loss in accuracy with multi-objective optimization; with only one win and up to 22 losses against their single-objective counterparts. For stability, the 4 tuners achieved between 6 and 12 wins, and between 2 and 7 losses. For the other tuners, results were more favorable, as they improved their stability in up to 20 cases.

## **5.5 Quasi experiment 5: Comparative study of Random Search Hyper-Parameter Tuning for Software Effort Estimation**

### **5.5.1 Study summary**

The fifth quasi experiment [131] compared many of the state-of-the-art hyper-parameter tuners against the fast, simple random search. We employed the previous 13 (12 without random search) tuners from our previous studies. However, we included 9 additional datasets from the PROMISE data repository. This quasi experiment evaluated whether these tuners can offer accuracy and stability that is equal or better than random search, which is arguably the simplest tuning approach. Our aim was to find out if (and in which scenarios) the investment in a more complex (in terms of implementation) tuning algorithm pays off.

Regarding analysis of results, we performed two comparisons using the win-tie-

loss algorithm and the Wilcoxon sign-rank test. The first round of analysis compared the obtained results against default hyper-parameters to determine in which datasets or models each tuner was effective. The second analysis compared random search against the other 12 tuners. In addition, we utilized the Scott-Knott algorithm to determine whether certain learning schemes were more effective for certain datasets.

We performed multiple additions to the experimental design with respect to our previous publication. The main change in terms of experiment design was the addition of the 9 PROMISE datasets into the experimental design: albrecht, china, deshar-nais, finnish, isbgs10, kemerer, kitchenham, maxwell, and miyazaki94. Our design however did not account for multi-objective optimization due to the computational cost of such techniques. The study changed the approach we took to the comparison of techniques. Instead of mainly relying the Scott-Knott approach, we used the Wilcoxon sign-rank test (with Holm-Bonferroni corrections) to determine, for each individual tuner, if it provided improvements with respect to default hyper-parameters and random search. As a criteria to what “improvement” is, we determined that a tuner must significantly improve accuracy and maintain or improve stability to be considered more efficient.

Future work discussed in the study included the extension of this study for further machine learning algorithms and data transformation, as well as to additional datasets in SEE. We discussed the possibility of extending this study using different “budgets” for each tuner, to further determine the pay-off of tuning.

### 5.5.2 Main results

The results on the default vs. tuned comparison revealed that no tuner was effective (improved accuracy while maintaining or improving stability) in every dataset and model. Results depended mostly on the model and dataset, and not so much on the particular hyper-parameter tuning approach. However, we determined that tuning was generally more effective on datasets with low intrinsic dimensionality. That is, tuning had a greater effect when the data was simple.

Regarding the random vs. tuned comparison, we determined that no tuner was able to outperform random search in all cases. The inverse likewise is true, and each tuner had at least one scenario in which it had better performance than random. However, in 6% of the studied scenarios (combinations of dataset and model), the 12 state-of-the-art tuners could not outperform random search. If the optimal tuner could be chosen for each dataset  $\times$  model combination, tuning would then

outperform random search in 27% of scenarios. This means that, for the remainder of scenarios, the random search hyper-parameter tuner found a solution that was equally good or better than the other tuners. The compact genetic algorithm, genetic algorithm, tabu search and hyperband approaches outperformed random search in 10 or more scenarios.

No learning scheme was dominant over all others. That is, each model had cases in which they performed better or worse. In the case of accuracy, the technique that ranked first in largest amount of scenarios was Flash tuning for SVR with logarithm transformation, which obtained an average mean standardized accuracy of 0.48 across all datasets. We also observed that ridge regression models could perform well in scenarios that SVR could not, and vice versa. In terms of stability, hyperband tuning for ridge regression with logarithm transformation ranked first in the largest amount of scenarios.

## 5.6 General findings and discussion

The results of the five quasi experiments showed that hyper-parameter tuning has its place in software effort estimation from the perspective of SEE researchers and practitioners. The first and second quasi experiments showed that tuning improve the predictive performance of SVR and CART models. This was done for the ISBSG 2018 dataset, which contains over 8,000 projects and 252 features. Our fifth quasi experiment also showed that tuning can improve the performance of ridge models in PROMISE datasets. However, this fifth quasi experiment also demonstrated that tuning does not work in all cases, and highly depends on the dataset and model. While tuning always improved performance of SVR and CART in ISBSG datasets, this was not the case for PROMISE. Inversely, tuning improved the performance of ridge regression models in PROMISE, but it did not in ISBSG.

Our second main observation is that, while tuning can be effective in many scenarios, different hyper-parameter tuners may provide similar model improvements. The results shown in the fifth quasi experiment showed that other tuners achieve similar improvements to what random search could provide in 94% of scenarios. In quasi experiments 1 and 2, the results showed that the exhaustive grid search algorithm was as effective as random search for SVR and CART. If we add to this that grid search explored over 4,000 hyper-parameter values for each model, whilst random search only did 60; the investment (in terms of computational resources spent) does not pay

off for the exhaustive grid search technique. While the non-grid tuners in quasi experiment 5 similarly explored around 60 hyper-parameters, they obtained equivalent results while being (arguably) more complex in implementation. Thus, depending on the model and dataset, researchers and practitioners do not need to heavily invest into exhaustively determining which tuner is the best. Instead, they can perform small tests with random search and one or two additional methods (particularly genetic algorithms) to assess which of these tuners best fits their data.

As a counterpoint, there were particular scenarios in which the choice of tuner was important. For instance, our quasi experiments showed that genetic algorithm, compact genetic algorithms, differential evolution, flash, tabu search and hyperband can provide improvements in accuracy for logarithm SVR that are better than random search. Moreover, Log SVR built using these tuners was in the top of the accuracy rankings in quasi experiments 4 and 5. This was also the case in quasi experiment 3 with the genetic algorithm and compact genetic algorithm tuners. Note that this was not the case for all tuners, and certain methods like Bayesian optimization hardly outperformed random search. The practical lesson here is to evaluate the impact of a group of tuners in practical applications, but to not go overboard. For example, a selection of random search, one genetic algorithm, and one heuristic method like flash can be sufficient.

Regarding multi-objective optimization, we determined that it mostly helps to cover the “weakness” of a model. For the most accurate models, SVR, it increased their stability at the cost of their accuracy. For CART models, it increased both metrics and made them closer in performance to SVR. In the case of ridge regression, it similarly increased stability at the cost of accuracy in many scenarios. We would not recommend the use of multi-objective tuning in ridge, however, as these models already offer low accuracy. More studies using multi-objective optimization are necessary to determine its benefits. We would recommend the use of multi-objective optimization for future SEE studies, but it may not be suitable for use in practical effort estimation, following the results of our quasi experiments.

As a general rule, we recommend researchers to look into their data before choosing a hyper-parameter tuner. If the data in hand is low dimensional, a tuner like random search could be as effective as any other tuner. If the dataset was instead more complex, then it may be worthwhile to evaluate other tuning approaches. Of our researched approaches, we recommend the use of those inspired in evolution: genetic algorithm, compact genetic algorithm, and differential evolution. We also endorse the use of one heuristic method: tabu search, hyperband, or harmony search.



We are also interested in the performance of model-based optimizers like flash, which employ a prediction model to determine promising hyper-parameter values. There is still room for research of hyper-parameter tuning for software effort estimation, especially from the practical, real life estimation perspective.

# Chapter 6

## Conclusion

This section concludes this thesis document by discussing the results obtained and their implications. Section 6.1 summarizes the results, addresses each specific objective, and closes with some of the learned lessons. Section 6.2 lists the contributions of this research. Section 6.3 highlights open questions that can direct future work in software effort estimation research.

### 6.1 Summary of results

This research compiled the state-of-the-art in hyper-parameter tuning for software effort estimation, developed an automated procedure for machine learning hyper-parameter tuning in the context of software effort estimation, and generated empirical evidence on the impact of hyper-parameter tuning on software effort estimation.

#### 6.1.1 SO1: Characterization of hyper-parameter tuning approaches for machine learning

We performed a systematic literature mapping study to address the first specific objective of the thesis: “to characterize hyper-parameter tuning approaches for machine learning”. The protocol and results of this study were presented in chapter 3.

The systematic literature mapping study identified 79 primary studies in software effort estimation that employed hyper-parameter tuning approaches on machine learning techniques.

We identified 12 unique hyper-parameter tuning approaches. Grid search was

used by the majority (60) studies, sometimes as a baseline but often as the only tuner. For machine learning algorithms, the mapping identified 21 methods that included neural networks, various types of regression, decision trees, support vector machines, case based reasoning, k-nearest neighbors, and combined models like ensembles, bagging, boosting and stacking. While there was no predominant model, we observed that tuning studies had a preference for models with 4 or more hyper-parameters. The study also identified 5 cross-validation approaches, 10 data transformations and 10 feature selectors. Every study employed at least one cross-validation approach, with the preferred method being leave-one-out. The most common data transformation was to limit numerical features to the [0, 1] range, and the majority of studies did not employ feature selection.

We identified 47 datasets and categorized them by their origin. The datasets were categorized in a total of 8 origins: PROMISE, ISBSG, open, tukutuku, unidentified, artificial, private, and IBM. The identified studies had a preference for open repositories (PROMISE, ISBSG, open, tukutuku), and mainly utilized datasets from the PROMISE and ISBSG repositories. The most used PROMISE datasets were deshar-nais, cocomo81, albrecht, kemerer, and maxwell. The most used version of the ISBSG dataset was release 8.

We identified 41 unique metrics used to evaluate SEE model performance, which were categorized depending on the type of error used in their formula. We identified 10 metric categories, of which relative error metrics were the most common. However, previous literature has reported that these types of metrics are biased, thus dissuading their use [41]. Instead, researchers have endorsed the use of metrics based on absolute error and baseline predictors. In addition, the studies utilized 17 different analysis techniques, classified as descriptive analysis, statistical tests and ranking methods. Thirty-five studies opted to simply describe their results and draw observations, instead of relying on a statistical method. We encourage future effort estimation studies to employ statistical analysis to improve their conclusion stability.

We grouped the challenges reported in the primary studies into 28 categories. Frequently reported problems in SEE tuning studies included the potential risks of inaccurate effort estimation, specific limitations of the studied machine learning techniques, complexity in the selection of hyper-parameters, the lack of a dominating machine learning model (no free lunch theorem), and lack of research for certain types of techniques like ensembles or online tuning.

From the design and execution of the search strategy, we discovered that many

effort estimation studies did not report the use of hyper-parameter tuning in their abstract. This hindered the search process, as the full text of many papers had to be reviewed to determine if the paper employed hyper-parameter tuning. We ask future effort estimation studies to report the types of techniques they employ in their abstract, and to use standard terminology to make effort estimation literature more accessible.

From this objective, we learned that hyper-parameter tuning studies often focus on improving the accuracy of the base model, and rarely focus on evaluating the impact of tuning. As previous studies have determined that tuning has a positive impact in model accuracy [14, 52, 25, 12], this indicates that the SEE community is aware of the benefits of tuning. We also determined that studies perform tuning on “worthwhile” techniques, meaning those that have multiple hyper-parameters, or those with values that are difficult to manually determine.

The main takeaway is that SEE researchers must expand their horizons and perform more practical case studies, in collaboration with software developers. Effort estimation datasets are outdated, and novel techniques are not reaching the industry. Collaborative studies benefit both parties, as researchers further validate their methods on new data and practitioners can improve their effort estimation practices. Research on software effort estimation has stayed too long in the lab, and our techniques deserve some sunlight.

### **6.1.2 SO2: Automation of a hyper-parameter tuning procedure for machine learning**

We implemented a hyper-parameter tuning framework to address the second specific objective of the thesis: “to automate a hyper-parameter tuning procedure for machine learning”. Chapter 4 documents the automated procedure and its supported techniques.

The framework supports three tasks: pre-processing, model training and evaluation, and statistical analysis. Moreover, the framework automates the model training and evaluation task.

The pre-processing task was divided into three activities: select projects, select features, and hold-out split. Pre-processing is performed to eliminate or lessen the effect of problems such as missing data, outliers, unrelated or redundant features, and class imbalance. This process is manual, and requires constant human intervention

and judgment. The framework does not automate this task, but it provides general guidelines in the form of the three indicated activities.

The model training and evaluation task was automated with two components: the framework configuration and the evaluation loop. The framework configuration component allows selection of techniques of the learning scheme (data transformation, feature selection, machine learning, hyper-parameter tuning), datasets, cross-validation, and evaluation metrics that will be used in the evaluation loop. The selected techniques are combined into learning schemes, which are then used to build models. The configuration component is handled through 7 files, each listing the name of the techniques or datasets, and their hyper-parameter or other options. Currently, the framework supports 15 hyper-parameter tuners, 8 machine learning models, 7 cross-validators, 2 data transformations, and 8 feature selectors.

The evaluation loop of the model training and evaluation task assembles, trains, and evaluates machine learning models based on each evaluation scheme: the combinations of dataset, cross-validation, and learning scheme in the framework configuration. The input data is divided by the cross-validation approach, resulting in one or more train-test splits. After this, a model is assembled as a combination of three techniques: data transformation, feature selection, and machine learning. Input data for this model is transformed, selected, and then used to train the machine learner. The assembled model is then optimized using the hyper-parameter tuning algorithm. The tuner combines the hyper-parameter search spaces of the transformation, selection and learner and finds the best possible value that results in the most accurate predictions. Moreover, the tuner can be configured to optimize multiple metrics (accuracy and stability). When the best hyper-parameters are found, the model is trained with the complete training set. Lastly, the trained model is evaluated on the test set, and the evaluation metrics for the evaluation scheme are recorded.

The statistical analysis task was divided into two activities: model evaluation and model verification. The model evaluation activity utilizes statistical analysis to draw conclusions from the recorded metrics. This statistical analysis can determine if the model is producing estimates in the expected ranges, assess whether tuning has an advantage over default parameters, compare the accuracy of two models, rank the models according to their performance, and conclude if there is a “best” model. Because model evaluation is a manual activity that depends on the type of study, the framework does not provide automation. The model verification activity consists of reviewing that the metrics obtained in the model training and evaluation task are correct, and not affected by problems such as data mismatch and overfitting. The

model training and evaluation task can gather new evaluation metrics by using the historical and new sets to evaluate the learning schemes on unseen data. The new metrics can be compared with the original metrics to determine if there are problems in the original evaluation.

From the literature mapping we determined that there is no standard machine learning process for SEE, and the actual use of techniques will depend on the design of the study. Our framework automates what we believe would be a standard procedure, but researchers or practitioners with a more specialized type of study may have to modify the framework to suit their needs.

From the implementation of this framework, we learned that there is good support for hyper-parameter tuning in Python, as there are multiple supporting libraries and implementations for each algorithm. However, many of the tuners were not compatible between each other, as each had their unique “interface”. For example, some tuners required as input a fitness function, while others require a model and a function that returns a metric. This framework was an effort in standardizing some of the existing techniques to the scikit-learn standards. On a similar topic, there are many machine learning libraries for Python and other programming languages, each with their own focuses, benefits and disadvantages. We selected scikit-learn as our “base” library as it provides a wide selection of machine learning techniques. If someone were to take a similar endeavor of implementing a hyper-parameter tuning (or general machine learning) framework, our recommendation would be to similarly select a base library or technology, and to adapt existing or new methods into the library interface. This does not only make the framework more standardized and extendable, but potentially makes its use easier for those familiar with the base library.

### **6.1.3 SO3: Evaluation of the effectiveness of the automated hyper-parameter tuning procedure for machine learning**

We performed a set of controlled quasi experiments to address the third specific objective of the thesis: “to evaluate the effectiveness of the automated hyper-parameter tuning procedure for machine learning”. The protocol and results of these studies were presented in chapter 5. A total of five quasi experiments were executed, with each study adding an increment over the previous one, either evaluating more techniques or doing a different analysis.

The first quasi experiment evaluated grid and random search for support vector

regression. We researched the effect of these two tuners on 4 subsets of the ISBSG R18 dataset. We constructed support vector regression and ridge regression models with and without log transformation; using 2 feature selectors for ridge regression; and using default hyper-parameters, grid search and random search for tuning. The results of the quasi experiment determined that random search obtained results that were in line with the more exhaustive grid search algorithm, with the largest difference being 0.05 standardized accuracy. Grid search was configured to evaluate 4,128 hyper-parameters, while random search did only 60. We thus concluded that random search was a viable, faster and less costly alternative to grid search for support vector regression in the context of effort estimation.

The second quasi experiment evaluated grid search, random search, and dodge hyper-parameter tuning for classification and regression trees. We extended the design of our previous quasi experiment to add CART as a model (no feature selection), and dodge as a tuning algorithm. We determined that hyper-parameter tuned CART outperformed its default parameter counterpart. However, our results showed that the best accuracy was achieved by log-transformed, tuned SVR models. CART models were generally less accurate than SVR, but instead offered a higher stability.

The third quasi experiment evaluated genetic and nature-based hyper-parameter tuning techniques for SEE. Five new tuners were added to our previous quasi experiment: genetic algorithm, 1+1 GA, compact GA, tabu search, and harmony search. This study also proposed our own variant of standardized accuracy for measuring stability. The study determined that traditional and compact genetic algorithms were more effective than grid and random search for tuning Log+SVR models, as they ranked top in accuracy across all datasets.

The fourth quasi experiment evaluated multi-objective optimization for hyper-parameter tuning. We implemented multi-objective variations of each tuner, which optimize both accuracy and stability of the model. The quasi experiment compared the performance of the tuners from the third quasi experiment against their multi-objective counterparts. Moreover, 5 additional tuners were implemented, for both single- and multi-objective optimization: flash, differential evolution, particle swarm optimization, bayesian optimization, and hyperband. The effect of multi-objective optimization depended on the base model, but it mainly offered a trade-off between stability and accuracy. For highly accurate models like Log+SVR, multi-objective tuning decreased their accuracy and increased their stability. For CART models, multi-objective optimization offered similar levels of stability, rarely decreasing it, while increasing accuracy in some cases.

The fifth quasi experiment evaluated state-of-the-art tuners against random search. This study extended the protocol of the fourth quasi experiment by adding 9 datasets from the PROMISE repository, although multi-objective optimization was not considered. For each model and dataset, the state-of-the-art tuners were compared against random search in terms of their effectivity. A model was considered effective if it could achieve better accuracy than its random search tuned counterpart, while maintaining or improving stability. The quasi experiment determined that, while tuners were almost always more effective than default hyper-parameters, they were only more effective than random search in 6% of the studied dataset  $\times$  model combinations. However, if the “optimal” tuner (i.e. the best tuner for a particular model and dataset) was selected, this probability raised to 27%.

From this objective, we learned that tuning is effective for software effort estimation. In each of the first four quasi experiments, default hyper-parameter models had consistently worse performance than tuned models. In our fifth study there were some scenarios in which tuning generated models with the same predictive accuracy than default parameters. However, we analyzed that tuning had more effect in datasets with small intrinsic dimensionality, which tends to be the case for effort estimation. Future research should employ hyper-parameter tuning to further improve the predictive accuracy of their effort estimation models.

There is work to be done regarding research in hyper-parameter tuning for software effort estimation. Our quasi experiments considered a large amount of datasets and hyper-parameter tuners, but the amount of machine learning algorithms, data transformations, feature selectors was relatively low. Moreover, the quasi experiments were by no means exhaustive, and still many tuners and datasets from SEE literature were not covered.

The most important lesson with respect to the use of tuning was emphasized in the fifth quasi experiment: “look before you leap”. Utilize a model and optimizer that are congruent with the size and complexity of your data. For example, for a dataset like *kemerer* with less than 30 data instances, an untuned CART model was as effective as CART with tuning. In this case, tuning was not necessary at all. For a dataset with more than 800 instances like *ISBSG*, the peak accuracy was only achieved by tuned Log+SVR, and the use of tuning is justified.

We recommend future studies to stop using grid search. The results of the literature mapping showed that 60 out of the 79 percent studies, or 76%, used grid search. However, in our five quasi experiments we concluded that the other hyper-



parameter tuning approaches were equally or more effective than random search to improve model accuracy. In addition, grid search explored substantially more hyper-parameters (over 4,000) than random search and other tuners (60). In cases that the hyper-parameter search space is large, grid search can be effectively replaced with a simple approach like random search.

Hyper-parameter tuning is only as important as the other types of tricks and techniques in machine learning. A tuner cannot improve performance if the quality or quantity of the original data is low, and pre-processing of datasets may be more effective in this case. We evidenced this phenomenon with the Log transformation on SVR: tuning had very low impact in SVR, but substantially improved the results of Log+SVR.

For future studies and applications of hyper-parameter tuning, we would firstly not recommend to perform an evaluation as exhaustive as ours. To evaluate 13 hyper-parameter tuners can be more than necessary for the majority of practical applications. In the case of our quasi experiments, such evaluation required months of computational time. We would thus recommend a small evaluation of tuning using between 3 and 4 methods: random search as a baseline, one genetic algorithm, one heuristic or model-based tuner, and optionally one other method. We found particularly good results for the traditional genetic algorithm, the compact genetic algorithm, flash, differential evolution, tabu search, and hyperband. Moreover, it is equally important to select an appropriate search space for these techniques; as the tuner cannot find the optimal solution if it has no access to that part of the space. On the other hand, a search space that is too wide may difficult or lengthen the search process.

Lastly, hyper-parameter tuning is as important to the machine learning process as other types of techniques and data transformations. A tuner cannot improve performance if the quality or quantity of the original data is low, and pre-processing of datasets may be more effective in this case. We evidenced this phenomenon with the Log transformation on SVR: tuning had very low impact in SVR, but substantially improved the results of Log+SVR. Thus, a combination of pre-processing, data transformations, missing value treatment, feature and project selection, appropriate machine learning technique selection, and hyper-parameter tuning are more effective at generating accurate and stable effort estimation models than using only one of these.

## 6.2 Contributions

This research provided the following contributions:

- A characterization, including categorization, of the existing machine learning techniques, datasets, and metrics used for hyper-parameter tuning SEE; as well as a report of the challenges reported in the state-of-the-art of hyper-parameter tuning SEE.
- A procedure for hyper-parameter tuning SEE that defines three tasks and sub-activities, and a framework for hyper-parameter tuning SEE that automates the model training and evaluation task of the procedure, and provides support for the pre-processing and statistical analysis tasks.
- Empirical evaluations of random and grid search hyper-parameter tuning for support vector regression and classification and regression trees, genetic and nature-inspired hyper-parameter tuning approaches, multi-objective optimization for tuning, and state-of-the-art tuning against random search in the context of SEE.

## 6.3 Future work

We have presented a systematic literature mapping that covers the existing machine learning techniques, datasets and metrics used in hyper-parameter tuning SEE. An extension of this study could cover the hyper-parameters ranges or values that were tuned in these studies. Novel tuning studies can use this information to select their hyper-parameter search space. Instead of analyzing each technique individually, future literature studies could analyze the use of technique combinations to determine research trends, and to help future studies to select effective technique combinations. One last possible venue of future work for literature studies is the comparison of the reported accuracy of SEE models. This can provide a general guideline of the expected metric values for the machine learning techniques, depending on the context. Researchers and practitioners could utilize these values to validate whether their models are within the previously reported accuracy values, or if they can take steps to improve the accuracy of their models. Another potential benefit of this type of comparison is that it can compare results of different studies if their experiment designs are similar.

The procedure proposed in this research describes three tasks, with our framework automating the model training and evaluation task and supporting the other two. Future work on this framework could automatize the pre-processing and statistical testing tasks. While human intervention is necessary, some level of automation can make these tasks shorter. For automation of pre-processing, the framework could use a rule-based approach to select or filter projects and features. An example of such automation is: the rule “missing values < 0.25 and type == numerical” indicates the framework to select only numerical features with less than 25% missing values, and would filter the current dataset with those properties. For automation of the statistical testing, the framework could provide a suite of pre-made statistical tests, plots, and ranking methods. Another potential improvement to the framework is a “scheme” configuration that permits to vary the order and techniques of the learning scheme. Currently the framework creates a model comprised of transformation+selection+model. A scheme configuration could be used to change this process to suit the users needs. For example, an user may want to perform tuning separately for each component, or to construct ensemble models using different rules.

Our quasi experiments evaluated different facets of hyper-parameter tuning in the context of SEE. However, these quasi experiments are not exhaustive, and do not cover the entire body of techniques and dataset used in SEE. First, the quasi experiments focused only on SVR, CART, and ridge regression models. Future studies or extensions to these quasi experiments can evaluate the effect of tuners in other models, such as neural networks, k-nearest neighbors, case-based reasoning, and ensemble and other combination-type models. Studies in tuning for ensembles would have to contemplate tuning in two phases: tuning for the base models, and tuning the hyper-parameters of the ensemble. Second, further studies are necessary for evaluation of multi-objective optimization applied for hyper-parameter tuning. Our quasi experiment showed that multi-objective optimization offers a trade-off between accuracy and stability. In certain cases, such as CART models, multi-objective optimization offered relatively no downsides; while also allowing an user to customize their trade-off between the two metrics. Third, future studies could evaluate the cost of “tuning more”. Based on our results, we determined that evaluating 60 (randomly sampled) hyper-parameters offered the same benefit in accuracy than evaluating over than 4,000. This implies that there is a point in which investing more resources (time, computer power) in tuning stops paying off. Thus, a study on the amount of hyper-parameters searched by an algorithm could determine the stopping point or criteria for tuners, as well as the properties of the dataset and model that require additional

tuning (i.e. more complex models or datasets would require additional tuning).

There are further venues of future work for general software effort estimation, machine learning, and hyper-parameter tuning. As many studies have reported, there is a need for new project data, and low adoption of the existing techniques by the industry. More case studies in collaboration with software development companies potentially attack these two issues, as researchers can work and make publicly available novel software project data, and the companies can experience firsthand the benefits of novel SEE techniques. Another recommendation for future SEE studies is to “look before you leap”, meaning to consider the complexity of your data before selecting a model. For instance, while grid-searched deep learning models may be effective, they may not be efficient in resources when a simple regression model may have had similar accuracy. We identified that very little studies performed online software effort estimation. While the hyper-parameter tuning approaches for such studies are limited, the online scenario better captures the actual effort estimation process of a development company than offline estimation. Lastly, there is potential in researching hyper-parameter tuning for machine learning in other software engineering sub-areas. Tuning has been successfully used in error detection [28] and code smell detection [52], and may be applicable to similar prediction or classification tasks.

# Appendix A

## Hyper-parameter tuning techniques in SEE

Table A.1: List of selected studies.

ID	Year	Title	Authors	Source
S1	2019	Investigating the use of random forest in software effort estimation	Abdelali Z., Mustapha H., Abdelwahed N.	Procedia Computer Science
S2	2019	Whale-crow optimization (WCO)-based Optimal Regression model for Software Cost Estimation	Ahmad, Sumera W.; Bamnote, G. R.	SADHANA-ACADEMY PROCEEDINGS IN ENGINEERING SCIENCES
S3	2019	Performance tuning for machine learning-based software development effort prediction models	Ertuğrul E., Baytar Z., Çatal Ç., Muratlı C.	Turkish Journal of Electrical Engineering and Computer Sciences
S4	2019	Evaluating filter fuzzy analogy homogenous ensembles for software development effort estimation	Hosni M., Idri A., Abran A.	Journal of Software: Evolution and Process
S5	2019	Improved effort estimation of heterogeneous ensembles using filter feature selection	Hosni M., Idri A., Abran A.	ICSOFTE 2018 - Proceedings of the 13th International Conference on Software Technologies
S6	2019	An ensemble-based model for predicting agile software development effort	Malgonde O., Chari K.	Empirical Software Engineering

Continued on next page

Table A.1: List of selected studies. (Continued)

S7	2019	A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation	Minku L.L.	Empirical Software Engineering
S8	2019	Model-based software effort estimation - A robust comparison of 14 algorithms widely used in the data science community	Phannachitta P., Matsumoto K.	International Journal of Innovative Computing, Information and Control
S9	2019	Applicability of Neural Network Based Models for Software Effort Estimation	S. Shukla; S. Kumar	2019 IEEE World Congress on Services (SERVICES)
S10	2019	Software effort interval prediction via Bayesian inference and synthetic bootstrap resampling	Song L., Minku L.L., Xin Y.A.O.	ACM Transactions on Software Engineering and Methodology
S11	2018	Support Vector Regression Based on Grid-Search Method for Agile Software Effort Prediction	A. Zakrani; A. Najm; A. Marzak	2018 IEEE 5th International Congress on Information Science and Technology (CiSt)
S12	2018	On the value of parameter tuning in heterogeneous ensembles effort estimation	Hosni M., Idri A., Abran A., Nassif A.B.	Soft Computing
S13	2018	Support vector regression-based imputation in analogy-based software development effort estimation	Idri, Ali; Abnane, Ibtissam; Abran, Alain	JOURNAL OF SOFTWARE-EVOLUTION AND PROCESS
S14	2018	Ensemble effort estimation using selection and genetic algorithms	Jodpimai P., Sophatsathit P., Lursinsap C.	International Journal of Computer Applications in Technology
S15	2018	Re-estimating software effort using prior phase efforts and data mining techniques	Jodpimai P., Sophatsathit P., Lursinsap C.	Innovations in Systems and Software Engineering
S16	2018	Software effort estimation using FAHP and weighted kernel LSSVM machine	Sehra S.K., Brar Y.S., Kaur N., Sehra S.S.	Soft Computing
S17	2018	A Comparison Study Between Soft Computing and Statistical Regression Techniques for Software Effort Estimation	T. Mohamed Abdellatif	2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)
S18	2018	Software development effort estimation using random forests: An empirical study and evaluation	Zakrani A., Hain M., Namir A.	International Journal of Intelligent Engineering and Systems
S19	2017	A class of hybrid multilayer perceptrons for software development effort estimation problems	de A. Araújo R., Oliveira A.L.I., Meira S.	Expert Systems with Applications

Continued on next page

Table A.1: List of selected studies. (Continued)

S20	2017	Investigating heterogeneous ensembles with filter feature selection for software effort estimation	Hosni M., Idri A., Abran A.	ACM International Conference Proceeding Series
S21	2017	Utilizing cluster quality in hierarchical clustering for analogy-based software effort estimation	J. H. C. Wu; J. W. Keung	2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)
S22	2017	Heterogeneous Ensemble Dynamic Selection for Software Development Effort Estimation	J. T. H. de A. Cabral; R. de A. Araujo; J. P. Nobrega; A. L. I. Oliveira	2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)
S23	2017	A new architecture based on artificial neural network and PSO algorithm for estimating software development effort	Moradbeiky A., Bardsiri A.K.	Journal of Telecommunication, Electronic and Computer Engineering
S24	2017	A stability assessment of solution adaptation techniques for analogy-based software effort estimation	Phannachitta P., Keung J., Monden A., Matsumoto K.	Empirical Software Engineering
S25	2017	Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation	Samareh Moosavi S.H., Khatibi Bardsiri V.	Engineering Applications of Artificial Intelligence
S26	2016	Pareto efficient multi-objective optimization for local tuning of analogy-based estimation	Azzeh M., Nassif A.B., Banitaan S., Almasalha F.	Neural Computing and Applications
S27	2016	Software development efforts prediction using artificial neural network	Bisi M., Goyal N.K.	IET Software
S28	2016	A hybrid model for task completion effort estimation	Dehghan A., Blincoe K., Damian D.	SWAN 2016 - Proceedings of the 2nd International Workshop on Software Analytics, co-located with FSE 2016
S29	2016	A differential evolution-based model to estimate the software services development effort	Khatibi Bardsiri A., Hashemi S.M.	Journal of Software: Evolution and Process
S30	2016	Heterogeneous Ensembles for Software Development Effort Estimation	M. Hosni; A. Idri; A. B. Nassif; A. Abran	2016 3rd International Conference on Soft Computing & Machine Intelligence (ISCMCI)
S31	2016	Hybridizing PSO with SA for optimizing SVR applied to software effort estimation	Novitasari D., Cholissodin I., Mahmudy W.F.	Telkonnika (Telecommunication Computing Electronics and Control)
S32	2015	Model to estimate the software development effort based on in-depth analysis of project attributes	E. Khatibi; V. Khatibi Bardsiri	IET Software

Continued on next page

Table A.1: List of selected studies. (Continued)

S33	2015	Predicting software development effort using tuned Artificial Neural Network	Hota H.S., Shukla R., Singhai S.	Smart Innovation, Systems and Technologies
S34	2015	An empirical analysis of data preprocessing for machine learning-based software cost estimation	Huang J., Li Y.-F., Xie M.	Information and Software Technology
S35	2015	Analogy-based effort estimation: a new method to discover set of analogies from dataset characteristics	M. Azzeh; A. B. Nassif	IET Software
S36	2014	Predicting project effort intelligently in early stages by applying genetic algorithms with neural networks	Li Z.Y.	Applied Mechanics and Materials
S37	2014	A Better Case Adaptation Method for Case-Based Effort Estimation Using Multi-objective Optimization	M. Azzeh; A. B. Nassif; S. Banitaan	2014 13th International Conference on Machine Learning and Applications
S38	2014	Story point approach based agile software effort estimation using various SVR kernel methods	Satapathy S.M., Panda A., Rath S.K.	Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE
S39	2014	Class point approach for software effort estimation using various support vector regression kernel methods	Satapathy S.M., Rath S.K.	ACM International Conference Proceeding Series
S40	2013	Using tabu search to configure support vector regression for effort estimation	Corazza A., Di Martino S., Ferrucci F., Gravino C., Sarro F., Mendes E.	Empirical Software Engineering
S41	2013	Using Ensembles for Web Effort Estimation	D. Azhar; P. Riddle; E. Mendes; N. Mittas; L. Angelis	2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement
S42	2013	Empirical study of homogeneous and heterogeneous ensemble models for software development effort estimation	Elish M.O., Helmy T., Hussain M.I.	Mathematical Problems in Engineering
S43	2013	A PSO-based model to increase the accuracy of software development effort estimation	Khatibi Bardsiri V., Jawawi D.N.A., Hashim S.Z.M., Khatibi E.	Software Quality Journal
S44	2013	Kernel methods for software effort estimation: Effects of different kernel functions and bandwidths on estimation accuracy	Kocaguneli E., Menzies T., Keung J.W.	Empirical Software Engineering

Continued on next page



Table A.1: List of selected studies. (Continued)

S45	2013	A genetic algorithm approach to global optimization of software cost estimation by analogy	Milios D., Stamelos I., Chatzibagias C.	Intelligent Decision Technologies
S46	2013	Ensembles and locality: Insight on improving software effort estimation	Minku L.L., Yao X.	Information and Software Technology
S47	2013	Software effort estimation as a multiobjective learning problem	Minku L.L., Yao X.	ACM Transactions on Software Engineering and Methodology
S48	2013	An Empirical Experiment on Analogy-Based Software Cost Estimation with CUDA Framework	P. Phannachitta; J. Keung; K. Matsumoto	2013 22nd Australian Software Engineering Conference
S49	2013	Software effort estimation using a neural network ensemble	Pai D.R., McFall K.S., Subramanian G.H.	Journal of Computer Information Systems
S50	2013	On the value of outlier elimination on software effort estimation research	Seo Y.-S., Bae D.-H.	Empirical Software Engineering
S51	2013	The impact of parameter tuning on software effort estimation using learning machines	Song L., Minku L.L., Yao X.	ACM International Conference Proceeding Series
S52	2013	Linear combination of multiple case-based reasoning with optimized weight for software effort estimation	Wu D., Li J., Liang Y.	Journal of Supercomputing
S53	2012	Hybrid morphological methodology for software development cost estimation	Araújo R.D.A., Soares S., Oliveira A.L.I.	Expert Systems with Applications
S54	2012	SEffEst: Effort estimation in software projects using fuzzy logic and neural networks	Gonzalez-Carrasco, Israel; Colomo-Palacios, Ricardo; Luis Lopez-Cuadrado, Jose; Garcia Penalvo, Francisco Jose	INTERNATIONAL JOURNAL OF COMPUTATIONAL INTELLIGENCE SYSTEMS
S55	2012	Data Mining Techniques for Software Effort Estimation: A Comparative Study	K. Dejaeger; W. Verbeke; D. Martens; B. Baesens	IEEE Transactions on Software Engineering
S56	2012	Alternative methods using similarities in software effort Estimation	Kosti M.V., Mittas N., Angelis L.	ACM International Conference Proceeding Series
S57	2012	Can cross-company data improve performance in software effort estimation?	Minku L.L., Yao X.	ACM International Conference Proceeding Series
S58	2012	Software cost modelling and estimation using artificial neural networks enhanced by input sensitivity analysis	Papatheocharous E., Andreou A.S.	Journal of Universal Computer Science

Continued on next page

Table A.1: List of selected studies. (Continued)

S59	2012	Increasing the accuracy of software development effort estimation using projects clustering	V. K. Bardsiri; D. N. A. Jawawi; S. Z. M. Hashim; E. Khatibi	IET Software
S60	2011	A shift-invariant morphological system for software development cost estimation	Araújo R.D.A., Oliveira A.L.I., Soares S.	Expert Systems with Applications
S61	2011	Software effort estimation based on optimized model tree	Azzeh M.	ACM International Conference Proceeding Series
S62	2011	Model Tree Based Adaption Strategy for Software Effort Estimation by Analogy	M. Azzeh	2011 IEEE 11th International Conference on Computer and Information Technology
S63	2011	A principled evaluation of ensembles of learning machines for software effort estimation	Minku L.L., Yao X.	ACM International Conference Proceeding Series
S64	2010	How effective is Tabu Search to configure Support Vector Regression for effort estimation?	Corazza A., Di Martino Martino S., Ferrucci F., Gravino C., Sarro F., Mendes E.	ACM International Conference Proceeding Series
S65	2010	Estimating software development effort using Tabu search	Ferrucci F., Gravino C., Oliveto R., Sarro F.	ICEIS 2010 - Proceedings of the 12th International Conference on Enterprise Information Systems
S66	2010	GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation	Oliveira A.L.I., Braga P.L., Lima R.M.F., Cornélio M.L.	Information and Software Technology
S67	2010	Hybrid Intelligent Design of Morphological-Rank-Linear Perceptrons for Software Development Cost Estimation	R. d. A. Araújo; A. L. I. de Oliveira; S. Soares	2010 22nd IEEE International Conference on Tools with Artificial Intelligence
S68	2009	Software Cost Estimation Using SVR Based on Immune Algorithm	J. Lee; K. Kwon	2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing
S69	2009	A study of project selection and feature weighting for analogy based software cost estimation	Li Y.F., Xie M., Goh T.N.	Journal of Systems and Software
S70	2009	Hybrid computational models for software cost prediction: An approach using artificial neural networks and genetic algorithms	Papatheocharous E., Andreou A.S.	Lecture Notes in Business Information Processing
S71	2009	A Morphological-Rank-Linear Approach for Software Development Cost Estimation	R. d. A. Araújo; A. L. I. de Oliveira; S. C. B. Soares	2009 21st IEEE International Conference on Tools with Artificial Intelligence

Continued on next page

Table A.1: List of selected studies. (Continued)

S72	2008	A GA-based feature selection and parameters optimization for support vector regression applied to software effort estimation	Braga P.L., Oliveira A.L.I., Meira S.R.L.	Proceedings of the ACM Symposium on Applied Computing
S73	2008	Theoretical Maximum Prediction Accuracy for Analogy-Based Software Cost Estimation	J. W. Keung	2008 15th Asia-Pacific Software Engineering Conference
S74	2008	Optimization of feature weights and number of neighbors for Analogy based cost Estimation in software project management	Y. F. Li; M. Xie; T. N. Goh	2008 IEEE International Conference on Industrial Engineering and Engineering Management
S75	2006	Improving Accuracy of Multiple Regression Analysis for Effort Prediction Model	K. Iwata; T. Nakashima; Y. Anan; N. Ishii	5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COM SAR'06)
S76	2004	Relation-based neurofuzzy networks with evolutionary data granulation	Oh S.-K., Pedrycz W., Park B.-J.	Mathematical and Computer Modelling
S77	2003	Self-organizing neurofuzzy networks based on evolutionary fuzzy granulation	Sung-Kwun Oh; W. Pedrycz; Byoung-Jun Park	IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans
S78	2002	Self-organising networks in modelling experimental data in software engineering	S. -. Oh; W. Pedrycz; H. -. Park	IEE Proceedings - Computers and Digital Techniques
S79	2001	Quasi-optimal case-selective neural network model for software effort estimation	Jun E.S., Lee J.K.	Expert Systems with Applications

Table A.2: Quality assessment per paper.

ID	QA1: Does the study report its goal or main objective?	QA2: Does the study report research questions?	QA3: Does the study report the datasets that are used?	QA4: Does the study measure accuracy with an unbiased metric?	QA5: Does the study analyze the obtained results?	Total score

Continued on next page

Table A.2: Quality assessment per paper. (Continued)

S1	1.0	0.0	1.0	0.0	1.0	3.0
S2	1.0	0.0	1.0	0.0	0.5	2.5
S3	1.0	1.0	1.0	0.5	0.5	4.0
S4	1.0	1.0	1.0	1.0	1.0	5.0
S5	1.0	0.0	1.0	1.0	1.0	4.0
S6	1.0	0.0	0.5	0.5	1.0	3.0
S7	1.0	1.0	1.0	1.0	1.0	5.0
S8	1.0	0.0	1.0	1.0	1.0	4.0
S9	1.0	0.0	1.0	0.0	0.0	2.0
S10	1.0	1.0	1.0	1.0	1.0	5.0
S11	1.0	1.0	0.5	0.0	0.5	3.0
S12	1.0	1.0	1.0	1.0	1.0	5.0
S13	1.0	1.0	1.0	1.0	1.0	5.0
S14	1.0	0.0	1.0	0.0	1.0	3.0
S15	1.0	0.0	1.0	0.0	1.0	3.0
S16	1.0	0.0	1.0	0.0	0.5	2.5
S17	1.0	0.0	1.0	0.0	1.0	3.0
S18	1.0	0.0	1.0	0.0	1.0	3.0
S19	1.0	0.0	1.0	0.0	1.0	3.0
S20	1.0	1.0	1.0	1.0	1.0	5.0
S21	1.0	0.0	1.0	0.0	1.0	3.0
S22	1.0	0.0	1.0	0.0	1.0	3.0
S23	1.0	0.0	1.0	0.0	0.0	2.0
S24	1.0	1.0	1.0	0.5	1.0	4.5
S25	1.0	0.0	1.0	0.0	1.0	3.0
S26	1.0	1.0	1.0	1.0	1.0	5.0
S27	1.0	0.0	1.0	0.0	1.0	3.0

Continued on next page

Table A.2: Quality assessment per paper. (Continued)

S28	1.0	0.0	0.5	0.0	0.5	2.0
S29	1.0	0.0	1.0	0.0	0.5	2.5
S30	1.0	1.0	1.0	1.0	1.0	5.0
S31	1.0	0.0	1.0	0.5	0.5	3.0
S32	1.0	0.0	1.0	0.5	1.0	3.5
S33	1.0	0.0	1.0	0.5	0.5	3.0
S34	1.0	1.0	1.0	0.5	1.0	4.5
S35	1.0	0.0	1.0	0.5	1.0	3.5
S36	1.0	0.0	1.0	0.0	0.0	2.0
S37	1.0	0.0	1.0	1.0	1.0	4.0
S38	1.0	0.0	1.0	0.5	0.5	3.0
S39	1.0	0.0	1.0	0.5	0.5	3.0
S40	1.0	1.0	1.0	0.5	1.0	4.5
S41	1.0	0.0	0.5	0.5	1.0	3.0
S42	1.0	0.0	1.0	0.0	0.5	2.5
S43	1.0	0.0	1.0	0.0	0.5	2.5
S44	1.0	1.0	1.0	0.5	1.0	4.5
S45	1.0	0.0	1.0	0.0	0.5	2.5
S46	1.0	1.0	1.0	1.0	1.0	5.0
S47	1.0	1.0	1.0	0.5	1.0	4.5
S48	1.0	0.0	1.0	0.5	0.5	3.0
S49	1.0	0.0	0.5	0.0	0.5	2.0
S50	1.0	1.0	1.0	0.0	1.0	4.0
S51	1.0	1.0	1.0	0.5	1.0	4.5
S52	1.0	0.0	1.0	0.0	1.0	3.0
S53	1.0	0.0	1.0	0.0	0.5	2.5
S54	1.0	0.0	1.0	0.0	0.5	2.5

Continued on next page

Table A.2: Quality assessment per paper. (Continued)

S55	1.0	0.0	1.0	0.0	1.0	3.0
S56	1.0	0.0	1.0	0.5	1.0	3.5
S57	1.0	1.0	1.0	1.0	1.0	5.0
S58	1.0	1.0	1.0	0.0	1.0	4.0
S59	1.0	0.0	1.0	0.0	0.5	2.5
S60	1.0	0.0	1.0	0.0	0.5	2.5
S61	1.0	0.0	1.0	0.0	1.0	3.0
S62	1.0	0.0	1.0	0.5	1.0	3.5
S63	1.0	1.0	1.0	0.0	1.0	4.0
S64	1.0	1.0	0.5	0.0	1.0	3.5
S65	1.0	0.0	1.0	0.5	1.0	3.5
S66	1.0	0.0	1.0	0.5	1.0	3.5
S67	1.0	0.0	1.0	0.0	0.5	2.5
S68	1.0	0.0	1.0	0.0	0.5	2.5
S69	1.0	0.0	1.0	0.0	0.5	2.5
S70	1.0	0.0	1.0	0.0	0.5	2.5
S71	1.0	0.0	1.0	0.0	0.5	2.5
S72	1.0	0.0	1.0	0.0	0.5	2.5
S73	1.0	0.0	1.0	0.0	0.5	2.5
S74	1.0	0.0	1.0	0.0	0.5	2.5
S75	1.0	0.0	0.0	0.5	0.5	2.0
S76	1.0	0.0	1.0	0.0	0.5	2.5
S77	1.0	0.0	1.0	0.0	0.5	2.5
S78	1.0	0.0	1.0	0.0	0.5	2.5
S79	1.0	0.0	0.5	0.0	0.5	2.0

Table A.3: Machine learning techniques by sub-type and related SEE studies.

Technique	Sub-type	N	Studies
Bagging		12	S7, S8, S10, S22, S46, S47, S51, S53, S63, S67, S71, S72
Bagging	Bootstrap	1	S10
Bagging	Synthetic Bootstrapping with Relevance Vector Machines	1	S10
Bayesian Model	Naive Bayes	2	S22, S28
Bayesian Model	Bayesian Network	1	S6
Boosting	Adaptative Boost	3	S6, S8, S22
Boosting	Gradient Boosting	3	S3, S6, S8
Boosting		1	S22
Boosting	Gradient Boosting: Multiple additive regression trees	1	S60
Boosting	Stochastic Gradient Boosting	1	S11
Case Based Reasoning	Analogy Based Estimation	16	S8, S13, S21, S29, S32, S35, S43, S45, S47, S48, S50, S56, S59, S62, S69, S74
Case Based Reasoning		8	S34, S37, S40, S55, S61, S64, S65, S79
Case Based Reasoning	Analogy Based Estimation + Genetic Algorithm	3	S29, S69, S74
Case Based Reasoning	Fuzzy Analogy	3	S4, S13, S18
Case Based Reasoning	Analogy Based Estimation + Particle Swarm Optimization	2	S29, S43
Case Based Reasoning	Analogy Based Estimation 0	2	S24, S26
Case Based Reasoning	OneR	2	S22, S28
Case Based Reasoning	Analogy Based Estimation + ANOVA/ANCOVA weighting	1	S32
Case Based Reasoning	Analogy Based Estimation + Differential Evolution	1	S29
Case Based Reasoning	Analogy Based Estimation with CUDA	1	S48
Case Based Reasoning	AQUA+ Analogy Based Estimation	1	S45
Case Based Reasoning	KStar	1	S22
Case Based Reasoning	Non-uniform Analogy Based Estimation	1	S44
Case Based Reasoning	Particle Swarm Optimization Case Based Reasoning	1	S18

Continued on next page

Table A.3: Machine learning techniques by sub-type and related SEE studies. (Continued)

Case Based Reasoning	Particle Swarm Optimized Case Based Reasoning	1		S52
Case Based Reasoning	Stochastic Matrix Analogy Based Estimation	1		S56
Case Based Reasoning	Uniform Analogy Based Estimation	1		S44
Ensemble	Heterogeneous Ensemble	10	S5, S6, S12, S14, S20, S22, S30, S41, S42, S46	
Ensemble	Homogeneous Ensemble	3		S4, S42, S49
Ensemble	Negative Correlation Learning	3		S46, S47, S63
Ensemble	Random Ensemble	3		S46, S47, S63
Ensemble		1		S52
Ensemble	Bucket	1		S59
Ensemble	Dynamic Cross-company Learning	1		S57
Ensemble	Dynamic Weight Majority	1		S57
Ensemble	Dynamical Ensemble Selector	1		S22
Ensemble	Genetic Algorithm Ensemble	1		S14
Ensemble	Harmonic Distance Multi Objective Evolutionary Algorithm Ensemble	1		S47
Ensemble	Heterogeneous OptimalBeta Ensemble	1		S6
Gaussian Process		1		S22
Genetic Algorithm		1		S2
K-Means Clustering	Spherical K-Means Clustering	1		S28
K Nearest Neighbors		17	S3, S5, S6, S9, S10, S12, S14, S15, S20, S22, S23, S28, S30, S41, S46, S51, S73	
Learning Automata		1		S2
Mean		3		S22, S40, S64
Median		3		S7, S40, S64
Neural Network	Multi Layer Perceptron	21	S3, S5, S8, S9, S10, S12, S18, S19, S20, S22, S29, S30, S42, S46, S47, S51, S54, S55, S61, S63, S66	
Neural Network		15	S6, S8, S17, S25, S32, S34, S41, S49, S54, S58, S59, S69, S70, S74, S79	
Neural Network	Radial Basis Function Network	13	S15, S19, S22, S29, S46, S47, S54, S55, S60, S63, S69, S71, S72	
Neural Network	GMDH Polynomial Neural Network	4		S11, S76, S77, S78

Continued on next page



Table A.3: Machine learning techniques by sub-type and related SEE studies. (Continued)

Neural Network	Morphological-Rank-Linear Perceptron	3	S19, S53, S67
Neural Network	Self-Organized Neuro-Fuzzy Network	3	S76, S77, S78
Neural Network	Adaptative Neuro-Fuzzy Interference System	2	S25, S42
Neural Network	Neuro-Fuzzy Network	2	S76, S77
Neural Network	Particle Swarm Optimization Neural Network	2	S23, S27
Neural Network	Cascade-Correlation Neural Network	1	S11
Neural Network	Error Back Propagation Network	1	S33
Neural Network	General Regression Neural Network	1	S11
Neural Network	Genetic Algorithm Particle Swarm Optimization Neural Network	1	S27
Neural Network	Genetic Algoritm Neural Network	1	S36
Neural Network	Multilayer Dilation-Erosion-Linear Perceptron	1	S19
Neural Network	Probabilistic Neural Network	1	S11
Neural Network	Recurrent Neural Network	1	S54
Particle Swarm Optimization	Particle Swarm Optimization-Composite Particle	1	S2
Random Forest		8	S1, S3, S6, S8, S11, S18, S22, S28
Random Forest	Extra Trees	1	S6
Regression	Ordinary Least Squares Regression	15	S2, S3, S6, S8, S9, S14, S15, S19, S22, S35, S41, S50, S55, S60, S71
Regression	Stepwise Regression	12	S23, S25, S29, S32, S35, S41, S43, S59, S61, S64, S65, S74
Regression	Multiple Regression	6	S23, S25, S29, S32, S43, S49
Regression	Multiple Linear Regression	4	S17, S58, S59, S75
Regression	Ridge Regression	4	S3, S6, S8, S55
Regression	Logistic Regression	3	S19, S22, S28
Regression	Least Median Squares Regression	2	S22, S55
Regression	Morphological-Rank-Linear Filter Regression	2	S60, S71

Continued on next page

Table A.3: Machine learning techniques by sub-type and related SEE studies. (Continued)

Regression		1	S79
Regression	ABC-PSO Regression	1	S11
Regression	Automatically Transformed Linear Model	1	S10
Regression	Crow Search Algorithm + Kernel Regression	1	S2
Regression	Crow Search Algorithm + Linear Regression	1	S2
Regression	Elastic-Net Regression	1	S8
Regression	Empirical Automatically Transformed Linear Model	1	S10
Regression	Kernel Regression	1	S2
Regression	Lasso Regression	1	S3
Regression	Least Absolute Shrinkage and Selection Operator Regression	1	S8
Regression	Least Angle Regression	1	S8
Regression	Log-Linear Regression	1	S17
Regression	Logarithmic Linear Regression	1	S7
Regression	Manual Regression	1	S11
Regression	Manual Stepwise Regression	1	S40
Regression	Multivariate Adaptive Regression Splines	1	S55
Regression	Partial Least Squares Regression	1	S41
Regression	Polynomial Regression	1	S78
Regression	Principal Component Analysis Regression	1	S41
Regression	Robust Regression	1	S55
Regression	Tabu Search Regression	1	S65
Regression	Whale-Crow Optimization + Kernel Regression	1	S2
Regression	Whale-Crow Optimization + Linear Regression	1	S2
Regression Tree	Classification and Regression Trees	16	S3, S8, S15, S23, S25, S29, S32, S34, S35, S41, S43, S55, S59, S60, S69, S74
Regression Tree		13	S1, S5, S6, S7, S10, S11, S12, S14, S18, S19, S20, S47, S51
Regression Tree	M5P Model Tree	4	S18, S61, S62, S66
Regression Tree	M5 Tree	2	S22, S55
Regression Tree	REPTree Regression Trees	2	S46, S63

Continued on next page

Table A.3: Machine learning techniques by sub-type and related SEE studies. (Continued)

Regression Tree	C4.5 Decision Tree	1	S28
Regression Tree	Chaid Decision Tree	1	S28
Regression Tree	J48 Tree	1	S22
Regression Tree	M5Prime	1	S30
Regression Tree	RepTree	1	S22
Regression Tree	REPTree	1	S57
Relevance Vector Machines		1	S10
Relevance Vector Machines	Empirical Relevance Vector Machines	1	S10
Rule Based Estimation	Conjunctive Rules	2	S22, S46
Rule Based Estimation	Decision Table	2	S22, S46
Rule Based Estimation	M5 Rules	2	S22, S46
Stacking		4	S6, S8, S28, S62
Stacking	Voting	1	S28
Support Vector Regression		31	S3, S5, S6, S8, S9, S10, S11, S12, S15, S17, S18, S19, S20, S22, S28, S30, S31, S38, S39, S40, S42, S53, S54, S60, S64, S66, S67, S68, S69, S71, S72
Support Vector Regression	Fuzzy Analytic Hierarchy Process Least Squares Support Vector Regression	1	S16
Support Vector Regression	Least Squares Support Vector Machine	1	S55
Support Vector Regression	Least Squares Support Vector Regression	1	S16

## Appendix B

# ChimeraHPT list of technique and parameters

1

Table B.1: Categories, techniques, parameters and possible values for the AS file.

Technique	Parameter	Type	Description	Ex.
none	-	-	-	-
variancethreshold	threshold	float	Features with a training-set variance lower than this threshold will be removed	0.1
correlationpercentile	percentile	int	Percent of features to keep	10
kbest	score_func	function	Function that takes a set of features and returns a score	-
	k	int	Number of features to keep	3
fpr	score_func	function	Function that takes a set of features and returns a score	-
	alpha	float	The highest p-value for features to be kept	0.05
fdr	score_func	function	Function that takes a set of features and returns a score	-
	alpha	float	The highest p-value for features to be kept	0.05
fwe	score_func	function	Function that takes a set of features and returns a score	-
	alpha	float	The highest p-value for features to be kept	0.05
rfe	n_features_to_select	int	Number of features to select	6
	step	int or float	Amount (int) or percentage (float) of features to remove at each iteration	0.15
Random Forest	n_estimators	int	Amount of regression tree models that comprise the forest	100

Table B.2: Categories, techniques, parameters and possible values for the LA file.

Technique	Parameter	Type	Description	Ex.
knn	n_neighbors	int	Number of neighbors to use	5
	weights	str or function	Weight function used in prediction	'uniform'
	algorithm	str	Algorithm used to compute the nearest neighbors	'auto'
	leaf_size	int	Leaf size passed to BallTree or KDTree	30
	p	int	Power parameter for the Minkowski metric	2
	metric	str or function	The distance metric to use for the tree	'minkowski'
mlp	n_layers	int	Amount of hidden layers	1
	n_hidden	int	Amount of neurons in each hidden layer	100
	activation	str	Activation function for the hidden layer	'relu'
	solver	str	The solver for weight optimization	'adam'
	alpha	float	L2 penalty (regularization term) parameter	0.0001
	batch_size	int	Size of minibatches for stochastic optimizers	200
	learning_rate	str	Learning rate schedule for weight updates, only used when solver='sgd'	'constant'
	learning_rate_init	float	The initial learning rate used, only with solver='sgd' or 'adam'	0.001
	power_t	float	The exponent for inverse scaling learning rate, only used when solver='sgd'	0.5
	max_iter	int	Maximum number of iterations for the solver	200
	shuffle	bool	Whether to shuffle samples in each iteration	True
	tol	float	Tolerance for the optimization	0.0001
	momentum	float	Momentum for gradient descent update, between 0 and 1, only used when solver='sgd'	0.9
	nesterovs_momentum	bool	Whether to use Nesterov's momentum, only used when solver='sgd'	True
	early_stopping	bool	Whether to use early stopping to terminate training when validation score is not improving	False
	validation_fraction	float	The proportion of training data to set aside as validation set for early stopping	0.1
	beta_1	float	Exponential decay rate for estimates of first moment vector in adam	0.9
beta_2	float	Exponential decay rate for estimates of second moment vector in adam	0.999	
epsilon	float	Value for numerical stability in adam	0.01	
lr	fit_intercept	bool	Whether to calculate the intercept for this model	True

Continued on next page

Table B.2: Categories, techniques, parameters and possible values for the LA file. (Continued)

	normalize	bool	When fit_intercept is True, the input data will be normalized	False
	copy_X	bool	If True, X will be copied	True
ridge	alpha	float	Regularization strength; must be a positive float	1.0
	fit_intercept	bool	Whether to calculate the intercept for this model	True
	normalize	bool	When fit_intercept is True, the input data will be normalized	False
	copy_X	bool	If True, X will be copied	True
	max_iter	int	Maximum number of iterations for conjugate gradient solver	100
	tol	float	Precision of the solution	0.001
	solver	str	Solver to use in the computational routines	'auto'
svr	kernel	str	Kernel type	'rbf'
	degree	int	Degree of the polynomial kernel function	3
	gamma	float	Kernel coefficient for 'rbf', 'poly' and 'sigmoid'	3
	coef0	float	Independent term in kernel function, for 'poly' and 'sigmoid'	0
	tol	float	Tolerance for stopping criterion	0.001
	C	float	Regularization parameter	1.0
	epsilon	float	Epsilon in the epsilon-SVR model	0.1
	shrinking	bool	Whether to use the shrinking heuristic	True
regressortree	criterion	str	The function to measure the quality of a split	'mse'
	splitter	str	The strategy used to choose the split at each node	'best'
	max_depth	int	The maximum depth of the tree	None
	min_samples_split	int or float	The minimum number of samples required to split an internal node	2
	min_samples_leaf	int or float	The minimum number of samples required to be at a leaf node	1
	min_weight_fraction_leaf	float	The minimum weighted fraction of the sum total of weights required to be at a leaf node	0.0
	max_features	int or float	The number of features to consider when looking for the best split	None
	max_leaf_nodes	int	Grow a tree with max_leaf_nodes in best-first fashion	None
	min_impurity_decrease	float	A node will be split if this split induces a decrease of the impurity greater than or equal to this value	0.0

Continued on next page

Table B.2: Categories, techniques, parameters and possible values for the LA file. (Continued)

bagging	base_estimator	object	The base estimator to fit on random subsets of the dataset, currently not supported	None
	n_estimators	int	The number of base estimators in the ensemble	10
	max_samples	int or float	The number of samples to draw from X to train each base estimator, using bootstrap	1.0
	max_features	int or float	The number of features to draw from X to train each base estimator	1.0
	bootstrap	bool	Whether samples are drawn with replacement	True
	bootstrap_features	bool	Whether features are drawn with replacement	False
	oob_score	bool	Whether to use out-of-bag samples to estimate the generalization error	False
zeror	strategy	str	Strategy to use to generate predictions	'mean'
	constant	int or float	The explicit constant as predicted by the 'constant' strategy	6
	quantile	float	The quantile to predict using the 'quantile' strategy	0.5
*	verbose	int	Controls the verbosity when fitting and predicting	0
	n_jobs	int	The number of jobs to run in parallel for both fit and predict	-1
	random_state	int	Random seed	None

Table B.3: Categories, techniques, parameters and possible values for the PT file.

Technique	Parameter	Type	Description	Ex.
default	-	-	-	-
grid search	-	-	-	-
random search	n_iter	int	Number of parameter settings that are sampled	10
random range search	n_iter	int	Number of parameter settings that are sampled	60
de	mutation_rate	float	Probability that a gene is mutated	0.7
	crossover_rate	float	Percentage of genes that are carried over from parents	0.8
	population_size	int	Individuals generated per iteration	10
	iterations	int	Amount of iterations	60
flash	budget	int	Amount of solutions that are explored	60
	population_size	int	Amount of randomly generated solutions	2880

Continued on next page

Table B.3: Categories, techniques, parameters and possible values for the PT file. (Continued)

	initial_size	int	Amount of solutions that are randomly explored before the prediction model is used	15
dodge	initial_size	int	Amount of solutions that are randomly explored before the prediction model is used	15
	population_size	int	Amount of solutions that are explored, includes initial_size	60
ga	max_num_iteratio	int	The termination criterion of geneticalgorithm, max amount of iterations	15
	population_size	int	Determines the number of trial solutions in each iteration	100
	mutation_probability	float	Determines the chance of each gene in each individual solution to be replaced by a random value	0.1
	elit_ratio	int	Determines the number of elites in the population	0
	crossover_probability	float	Determines the chance of an existed solution to pass its genome to new trial solutions	0.5
cga	budget	int	Amount of explored solutions	60
oneplusone	budget	int	Amount of explored solutions	60
pso	budget	int	Amount of explored solutions	60
	popsize	int	Amount of explored solutions per iteration	10
bo	budget	int	Amount of explored solutions	60
tabu	tabu_size	int	Size of the tabu list, adds that many additional explored solutions	15
	max_steps	int	Amount of iterations	10
	neighborhood_size	int	Amount of solutions explored per iteration	12
harmony	memory_size	int	Size of the harmony set, adds that many additional explored solutions	10
	memory_considering_rate	float	Probability that a generated solution considers the harmony set	0.4
	pitch_adjustment_rate	float	Probability that a parameter taken form the harmony set is mutated	0.1
	fret_width	float	Maximum value for the mutation for pitch_adjustment_rate	3
	max_steps	int	Amount of iterations	6
*	scoring	str or list	Metrics to evaluate the performance of the cross-validated model on the test set	'MMRE'
	n_jobs	int	The number of jobs to run in parallel for both fit and predict	-1
	refit	bool or str	Refit an estimator using the best found parameters on the whole dataset, for multi-objective optimization it needs to be hypervolume	True

Continued on next page



Table B.3: Categories, techniques, parameters and possible values for the PT file. (Continued)

cv	int	Number of folds to use in k-fold cross-validation	5
random_state	int	Seed for pseudo random number generator	None

## Appendix C

# Paper 1: Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: un mapeo de literatura

**Reference** Villalobos-Arias, L., Quesada-López, C., Martínez, A., & Jenkins, M. (2021). Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: un mapeo de literatura. *Revista Ibérica de Sistemas e Tecnologías de Informação*, (E42), 305-318.

# Status Indexed in Scopus



Search Sources Lists SciVal

[Create account](#)
[Sign in](#)

< Back to results | 1 of 1

Export
 Download
 Print
 E-mail
 Save to PDF
 Add to List
 [More...](#)

## Document type

Article

## Source type

Journal

## ISSN

16469895

[View more](#)

RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao • Volume 2021, Issue E42, Pages 295 - 308  
• February 2021

[Machine learning hyper-parameter tuning techniques for software effort estimation: A systematic mapping study] [Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: Un mapeo de literatura]

Villalobos-Arias L. , Quesada-Lopez C. , Martínez A. , Jenkins M.

Save all to author list

Universidad de Costa Rica, San Pedro, Costa Rica

## Abstract

Author keywords

SciVal Topics

Funding details

## Abstract

Different machine learning (ML) algorithms have been used to support software effort estimation (SEE) processes. However, the performance of these algorithms are sensible to multiple factors, including the choice of hyper-parameters. More recently, hyper-parameter tuning has risen as a SEE research area to improve the performance of ML models. In this paper, we perform a systematic mapping study to characterize hyper-parameter tuning techniques for ML algorithms in the context of SEE. We present the results of 67 studies identified between 2010 and 2019, and classified the hyper-parameter tuning techniques, ML algorithms, and datasets. Likewise, we reported the challenges as a roadmap for future research in the area. © 2021, Associacao Iberica de Sistemas e Tecnologias de Informacao. All rights reserved.

## Author keywords

Hyper-parameters; Machine learning; Software effort estimation; Systematic literature mapping study

**Topic name** Effort Estimation; Functional Size Measurement; COSMIC

**Prominence percentile** 92.766

Funding sponsor	Funding number	Acronym
Universidad de Costa Rica	834-B8-A27	UCR

[See opportunities by UCR](#)

## Funding text

Este estudio fue apoyado por la Universidad de Costa Rica No. 834-B8-A27. Nuestro agradecimiento al Empirical Software Engineering Group (ESEG) de la Universidad de Costa Rica retroalimentación sobre este trabajo.

References (26)

[View in search results format](#)

All
 Export
 Print
 E-mail
 Save to PDF
 [Create bibliography](#)

## Metrics

[View all metrics](#)

2 Views Count 2021

Last updated on:

18 June 2021

2 2012-2021



PlumX Metrics

Usage, Captures, Mentions, Social Media and Citations beyond Scopus.

## Cited by 0 documents

Inform me when this document is cited in Scopus:

[Set citation alert](#)

## Related documents

Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation

Villalobos-Arias, L., Quesada-López, C., Guevara-Coto, J., (2020) PROMISE 2020 - Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Co-located with ESEC/FSE 2020

Software development effort estimation using feature selection techniques

Hosni, M., Idris, A., (2018) Frontiers in Artificial Intelligence and Applications

Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation

Villalobos-Arias, L., Quesada-López, C., Martínez, A., (2021) Advances in Intelligent Systems and Computing

# Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: un mapeo de literatura

Leonardo Villalobos-Arias, Christian Quesada-López, Alexandra Martínez, Marcelo Jenkins

{leonardo.villalobosarias, cristian.quesadalopez, alexandra.martinez, marcelo.jenkins}@ucr.ac.cr

Universidad de Costa Rica, San Pedro, Costa Rica.

DOI: 10.17013/risti.n.pi-pf

**Resumen:** Distintos algoritmos de aprendizaje automático (ML) han sido utilizados para apoyar los procesos de estimación de esfuerzo de desarrollo del software (EES). Sin embargo, el desempeño de estos algoritmos puede verse impactado por varios factores, uno de los cuales es la escogencia de los hiperparámetros. En los últimos años, el ajuste de hiperparámetros ha surgido como un área de investigación de interés para la EES que busca optimizar el desempeño de los modelos de ML. En este trabajo, realizamos un mapeo sistemático de literatura para caracterizar las técnicas de ajuste automático de hiperparámetros de algoritmos de ML utilizados en el contexto de la EES. Presentamos los resultados de 67 estudios identificados entre el 2010 y el 2019 y clasificamos las técnicas de ajuste de hiperparámetros, los algoritmos de ML y los conjuntos de datos dónde se han aplicado. Asimismo, reportamos los retos reportados como mapa de ruta para futuras investigaciones en el área.

**Palabras-clave:** hiperparámetros; aprendizaje automático; estimación de esfuerzo del desarrollo de software; mapeo sistemático de literatura.

## ***Machine learning hyper-parameter tuning techniques for software effort estimation: a systematic mapping study***

**Abstract:** Different machine learning (ML) algorithms have been used to support software effort estimation (SEE) processes. However, the performance of these algorithms are sensible to multiple factors, including the choice of hyper-parameters. More recently, hyper-parameter tuning has risen as a SEE research area to improve the performance of ML models. In this paper, we perform a systematic mapping study to characterize hyper-parameter tuning techniques for ML algorithms in the context of SEE. We present the results of 67 studies

identified between 2010 and 2019, and classified the hyper-parameter tuning techniques, ML algorithms, and datasets. Likewise, we reported the challenges as a roadmap for future research in the area.

**Keywords:** hyper-parameters; machine learning; software effort estimation; systematic literature mapping study.

## 1. Introducción

La estimación de esfuerzo de desarrollo del software (EES) es una tarea esencial en la administración de proyectos (Boehm, 1984). La exactitud de los modelos de EES es esencial, dado que la sobre estimación puede causar la pérdida de oportunidades de negocio y la subestimación puede afectar el éxito de un proyecto (Lederer & Prasad, 1995).

Los algoritmos de aprendizaje automático (ML) han sido investigados en el área de la EES por décadas (Wen, et al., 2012). Estos estudios evalúan distintas técnicas de validación, pre-procesamiento, selección de atributos y aprendizaje automático que buscan minimizar el error de las estimaciones. El principal beneficio de estos modelos es que pueden apoyar a los profesionales en sus procesos de toma de decisiones durante el desarrollo de los proyectos (Minku, 2019).

En los últimos años, el ajuste automático de los hiperparámetros ha emergido como área de investigación que potencialmente pueden impactar el desempeño de los modelos de SEE (Song et al., 2013; Idri et al., 2016; Hosni, et al., 2018; Minku, 2019). El ajuste de hiperparámetros ha demostrado su impacto en distintas áreas tales como la predicción de defectos (Agrawal et al., 2019).

Los hiperparámetros son configuraciones ajustables que se eligen para entrenar el modelo y que rigen proceso de entrenamiento (Luo, 2016). Las técnicas de ajuste de hiperparámetros permiten obtener de manera automática las configuraciones que optimizan la exactitud de las estimaciones, estas incluyen técnicas de búsqueda exhaustivas, aleatorias, genéticas y otras (Luo, 2016). Muchos de los estudios en EES no reportan las configuraciones de los hiperparámetros, utilizan las configuraciones por defecto, o no utilizan técnicas de ajuste (Minku & Yao, 2013). La evaluación del ajuste automático de hiperparámetros en modelos de SEE es necesaria para determinar el impacto en la exactitud de las estimaciones de esfuerzo (Minku, 2019).

En este trabajo, realizamos un mapeo sistemático de literatura para caracterizar las técnicas de ajuste automático de hiperparámetros de algoritmos de ML utilizados en el contexto de la EES. Presentamos los resultados preliminares basado en el análisis de 67 estudios publicados entre el 2010 y el 2019. Para esto, clasificamos las técnicas de ajuste de hiperparámetros, los algoritmos de ML, los conjuntos de datos dónde se

han aplicado y las métricas de desempeño con las cuáles se han evaluado los modelos de EES.

El contenido del resto del artículo se estructura de la siguiente manera. La Sección 2 presenta estudios de literatura realizados en el área de la EES. La Sección 3 describe el diseño del mapeo. La Sección 4 presenta y discute los resultados. Finalmente, la Sección 5 establece las conclusiones.

## **2. Trabajo relacionado**

Jorgensen & Shepperd (2006) realizaron una revisión sistemática de literatura sobre la estimación de costos analizando sus oportunidades de mejora. Identificaron 304 estudios que clasificaron de acuerdo a su objetivo de investigación, enfoque de estimación y de investigación, conjuntos de datos y contexto. Wen et al. (2012) realizaron una revisión sistemática de literatura sobre estudios empíricos de modelos de aprendizaje automático (ML) para la EES. Clasificaron 84 estudios de acuerdo a los tipos de técnicas, los resultados de exactitud, los métodos de comparación y el contexto. Idri et al. (2016) reportan una revisión sistemática de literatura de 24 estudios en la que identifican modelos ensamblados de estimación de esfuerzo, su exactitud, las reglas para la combinación de técnicas y las metodologías utilizadas.

Malgorta et al. (2017) realizaron una revisión sistemática que identificó 78 estudios sobre la capacidad de los algoritmos basados en búsqueda para modelos de predicción. Los autores clasificaron las investigaciones de acuerdo a las técnicas de búsqueda, las configuraciones para los experimentos, las funciones de aptitud, el desempeño, las técnicas estadísticas, y las ventajas y desventajas de su utilización. Sehra et al. (2017) estudiaron las tendencias y patrones de la investigación sobre modelos de estimación de esfuerzo. El estudio identificó 12 áreas de investigación y su mapeo semántico con 60 tendencias de investigación.

Este mapeo sistemático de literatura agrega evidencia sobre la caracterización de las técnicas de ajuste automático de hiperparámetros de algoritmos de ML en el contexto de la EES. En nuestro entendimiento, no existe un mapeo de literatura que presente esta caracterización en el área de la EES.

## **3. Metodología**

El diseño del estudio se realizó basado en los lineamientos descritos por Petersen et al. (2015). El objetivo del estudio fue caracterizar las técnicas de ajuste automático de hiperparámetros utilizadas en el contexto de la estimación de esfuerzo (EES). Para esto, identificamos los algoritmos de aprendizaje automático (ML) y los conjuntos de datos reportados en los estudios que aplican ajuste automático de hiperparámetros. El detalle de cada uno de los formularios producto de la ejecución del protocolo se encuentra disponible en <https://tinyurl.com/y5z7kcuc>.

### 3.1. Preguntas de investigación

Las siguientes preguntas de investigación se analizan en el contexto de la EES para alcanzar el objetivo de investigación:

- **RQ1:** ¿Cuáles técnicas de ajuste automático de hiperparámetros han sido utilizadas en modelos de estimación que utilizan algoritmos de ML?
- **RQ2:** ¿Cuáles conjuntos de datos de proyectos han sido utilizados en los estudios de ajuste automático de hiperparámetros?

La RQ1 identifica y clasifica las técnicas utilizadas en los estudios de ajuste de hiperparámetros que incluyen las técnicas de validación, pre-procesamiento, selección de atributos, aprendizaje automático y ajuste automático. La RQ2 identifica los conjuntos de datos utilizados y sus características.

Para la construcción, refinamiento y validación del protocolo de diseño del mapeo, utilizamos 15 artículos de control (Corazza et al., 2010; Oliveira et al., 2010; Azzeh, 2011; Dejaeger et al., 2011; Araújo et al., 2012; Azhar et al., 2013; Corazza et al., 2013; Kocaguneli et al., 2013; Song et al., 2013; Azzeh et al., 2014; Hota et al., 2015; Hosni et al., 2016; 2017; 2018; Minku et al., 2019). Los estudios de control fueron identificados mediante una búsqueda exploratoria para identificar artículos que evalúan técnicas de ajuste automático de hiperparámetros en el contexto de la EES. Estos fueron utilizados para refinar las cadenas de búsqueda y los procesos de selección de artículos relevantes.

### 3.2. Estrategia de búsqueda y selección

La cadena de búsqueda se construyó a partir de las preguntas de investigación y las palabras clave del título y resumen de los artículos de control. En el mapeo realizamos un conjunto de búsquedas automatizadas y la aplicación de criterios de inclusión y exclusión para identificar los estudios primarios relevantes. Para guiar el proceso de construcción de la cadena de búsqueda utilizamos el modelo PICO que permite organizar las palabras clave en grupos de búsqueda a partir de la población (P), intervención (I), comparación (C) y salidas esperadas (O). En el caso de nuestro estudio la población buscada son los artículos de modelos de EES, la intervención son las técnicas de ajuste automático de hiperparámetros y los algoritmos de ML, no consideramos aspectos de comparación, y las salidas son las técnicas utilizadas para la construcción de los modelos de ML en EES, los conjuntos de datos y las métricas de evaluación. Basado en el PICO y a partir de múltiples corridas de refinamiento establecimos la siguiente la cadena de búsqueda:

*(“software” AND (“effort estimation” OR “effort prediction” OR “cost estimation” OR “cost prediction”)) AND (“tuning” OR “optim\*” OR “setting\*” OR “combinat\*” OR “ensemble\*” OR “scheme\*”)*

Las búsquedas automatizadas se realizaron en las bases de datos *Scopus*, *IEEE Xplore*, *Web of Science* y *ScienceDirect* en febrero del 2020. Estas fueron adaptadas a cada una de las bases de datos. Los buscadores se configuraron para identificar artículos basado en el título, resumen y palabras clave. Una vez que se obtuvieron los artículos de las búsquedas en las bases de datos se procedió con la eliminación de los duplicados. En total se identificaron un total de 644 artículos sin duplicar desde el 2010.

### ***Inclusión y exclusión***

La identificación de los artículos relevantes se realizó mediante del proceso de inclusión y exclusión. Para esto se aplicaron un conjunto de criterios a cada uno de los artículos obtenidos en la búsqueda automatizada a partir del título, resumen y palabras clave. En caso de duda, se realizó la lectura a texto completo del artículo para determinar su relevancia para responder las preguntas de investigación. Se incluyeron los artículos: (I1) que son estudios primarios, (I2) en el campo de la EES, (I3) que utiliza algoritmos de aprendizaje automático y (I4) que utiliza técnicas de ajuste automático de hiperparámetros. Se excluyeron artículos (E1) que no estuvieran escritos en inglés y (E2) que no estuvieran disponibles en texto completo. Después del proceso de inclusión y exclusión se determinaron 67 artículos relevantes. El principal criterio para descartar artículos fue el I4, dado que muchos de los artículos obtenidos de las búsquedas no aplicaban algoritmos de ajuste automático de hiperparámetros, dado que seleccionaban los hiperparámetros manualmente o utilizaban los valores por defecto de los algoritmos de ML.

### ***Evaluación de calidad***

Para cada uno de los artículos relevantes, realizamos una evaluación de calidad para determinar el nivel de detalle para responder las preguntas de investigación. Se evaluaron y reportaron los criterios relacionados con: (Q1) el reporte del objetivo y (Q2) las preguntas de investigación, (Q3) el reporte de los conjuntos de datos utilizados, (Q4) la calidad de las métricas de evaluación y (Q5) el análisis de los resultados obtenidos. Cada criterio se puntúa en una escala de 0 a 1, donde 0 significa que no se reporta, 0.5 parcialmente y 1 completamente. Para calcular la puntuación de calidad de cada artículo se suman los resultados de los criterios, para un máximo posible de 5 puntos. En nuestro estudio, la evaluación de calidad no fue utilizada para excluir artículos durante el análisis.

### **3.3. Extracción, clasificación y análisis**

Para cada una de las preguntas de investigación se definieron los componentes a extraer a partir del contenido de cada artículo. Primero se extrae información general del estudio, y luego la correspondiente a cada pregunta de investigación. Durante la extracción se realizó la clasificación de cada artículo utilizando las recomendaciones para la construcción de esquemas de clasificación de estudios de mapeo aplicado a



los estudios de control, que se complementó durante la extracción de todos los artículos relevantes (Petersen et al., 2015). La Tabla 1 detalla cada uno de los componentes extraídos y ejemplos de categorías para su clasificación.

Tabla 1 – Formulario de extracción

RQ	Componentes
<i>General</i>	Año, autores, foro, tipo de foro, tipo de estudio, objetivo, preguntas de investigación
<i>RQ1</i>	Técnicas de ajuste automático de hiperparámetros ( <i>grid search</i> , <i>random search</i> , <i>genetic algorithm</i> , otros), validación ( <i>k-fold</i> , <i>leave-one-out</i> , otros), transformación ( <i>standardization</i> , <i>logarithm</i> , otras), selección de atributos ( <i>correlation-based</i> , <i>sequential search</i> , <i>genetic algorithm</i> , otros), algoritmos de aprendizaje automático ( <i>regression</i> , <i>neural networks</i> , <i>regression trees</i> , <i>support vector machines</i> , otros), hiperparámetros y rangos, proceso de estimación ( <i>on-line</i> , <i>off-line</i> )
<i>RQ2</i>	Conjuntos de datos ( <i>ISBSG</i> , <i>Albrecht</i> , <i>COCOMO</i> , <i>NASA</i> , <i>Desharnais</i> , otros), tipo de conjunto de datos ( <i>Cross Company</i> , <i>Single Company</i> )

Toda la información de los artículos fue tabulada en una hoja de extracción de datos. Para responder cada pregunta de investigación, se obtienen los grupos a partir de las clasificaciones y se presentan las tendencias y patrones identificados.

### 3.4. Amenazas a la validez

Las siguientes amenazas a la validez del estudio de mapeo deben considerarse. El análisis se realiza considerando estudios primarios del 2010 al 2019 lo que puede dejar por fuera artículos relevantes de años anteriores. Asimismo, la cadena de búsqueda fue definida y calibrada mediante múltiples corridas de prueba; sin embargo, estudios relevantes pueden no ser identificados en las búsquedas automatizadas porque no utilizan las palabras clave seleccionadas. Para las búsquedas se utilizaron bases de datos reconocidas en el área de la ingeniería de software y la computación. El proceso de inclusión y exclusión, así como la extracción de los artículos fue realizada por el autor principal de este mapeo, lo que podría introducir un sesgo en la selección. Durante la inclusión y exclusión, un segundo autor realizó un muestro aleatorio para validar este proceso. Finalmente, los resultados del mapeo solo pueden ser generalizados al conjunto de literatura seleccionada para el análisis.

## 4. Resultados

En esta sección presentamos los resultados del mapeo de literatura. El listado de los artículos relevantes y el formulario de extracción se encuentra disponible en <https://tinyurl.com/y5z7kcuc>, los estudios son identificados con la nomenclatura

S1-S67 que es utilizada para el reporte de resultados. La Figura 1 muestra los artículos seleccionados del mapeo. La búsqueda automatizada en las bases de datos recuperó un total de 1,310 artículos. Después del proceso de eliminación de duplicados se obtuvieron un total de 644 artículos entre el 2010-2019. El proceso de inclusión y exclusión seleccionó un total de 67 estudios primarios relevantes que fueron extraídos, evaluados y analizados. Los resultados de calidad obtenidos para el conjunto de artículos relevantes variaron entre 2 y 5, con una media de 3.5, una mediana de 3 y una desviación de 1. Los resultados indican que los estudios brindan un nivel de detalle aceptable para responder a las preguntas de investigación. Asimismo, se denota la necesidad de mejorar los reportes utilizando métricas de exactitud estandarizadas.

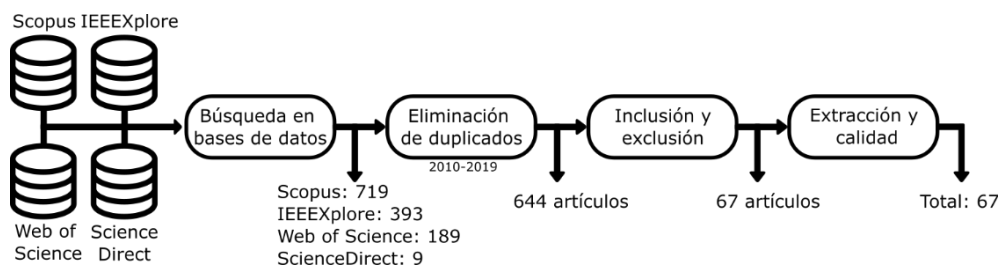


Figura 1 – Artículos seleccionados a partir de la estrategia de búsqueda

La distribución de los artículos fue la siguiente: 4 artículos fueron publicados en el 2010 y 4 en el 2011, 7 en el 2012, 13 en el 2013, 4 en el 2014 y 4 en el 2015, 6 en el año 2016, 7 en el 2017, 8 en el 2018, y finalmente 10 artículos en el 2019. La distribución de los artículos muestra que la investigación sobre el ajuste automático de hiperparámetros en la EES es un área vigente.

#### 4.1. Técnicas de ajuste de hiperparámetros

La Tabla 2 lista las técnicas de ajuste automático de hiperparámetros reportadas en los estudios de EES que utilizan algoritmos de ML. En total se identificaron 12 técnicas de ajuste. Las técnicas más utilizadas fueron *Grid Search* con 53 estudios, *Genetic Algorithms* con 8 y *Particle Swarm Optimization* con 6 estudios.

*Grid Search* consiste en evaluar todas las configuraciones de hiperparámetros de un espacio de búsqueda. A partir de los valores predefinidos para cada hiperparámetro se exploran todas las combinaciones posibles, evaluando los resultados del modelo para cada una. Por su parte, *Genetic Algorithms* es un método de optimización inspirado en la naturaleza que genera una población inicial de individuos, llamados cromosomas, que representan las configuraciones de hiperparámetros. La función de aptitud de cada cromosoma es evaluada mediante una métrica de desempeño y se conservan solamente los individuos más aptos y se generan nuevos individuos a par-

tir de la combinación de los existentes. De manera similar, el *Particle Swarm Optimization* simula el comportamiento de aves buscando comida en un área. Diferentes soluciones potenciales de hiperparámetros son representadas como una partícula en el espacio. Para determinar cuál de las partículas está más cercana a la solución óptima, se calcula su aptitud. Las demás partículas se mueven para estar más cercanas a la solución óptima. Este proceso se repite por una cantidad de iteraciones o hasta que se encuentre una solución satisfactoria.

Tabla 2 – Técnicas de ajuste de hiperparámetros

Técnica	Cant.	Estudios
Grid Search	53	S1, S3, S4, S5, S6, S8, S10, S11, S12, S14, S15, S17, S18, S19, S20, S22, S23, S24, S26, S27, S28, S29, S30, S31, S32, S33, S34, S35, S36, S38, S39, S40, S41, S42, S43, S44, S46, S47, S48, S49, S50, S51, S52, S55, S56, S57, S58, S59, S60, S62, S63, S65, S66
Genetic Algorithm	8	S18, S37, S42, S45, S53, S54, S66, S67
Particle Swarm Optimization	6	S2, S12, S13, S25, S31, S37
Tabu Search	3	S2, S40, S64
Random Search	2	S21, S40

Las técnicas siguientes fueron reportadas en un estudio: Bee's Algorithm [S61], Hill Climbing [S7], K-Fold Cross-Validation [S16], Linear Size Adjustment [S37], Online Supervised Tuning Procedure [S7], Regression Towards the Mean [S37] y Satin Bowerbird Optimization [S25].

La Tabla 3 lista los algoritmos de ML reportados en los estudios de EES. En total se identificaron 18 técnicas de ML utilizadas para la construcción de modelos de EES. Las técnicas más utilizadas reportadas fueron: *Neural Networks* con 39 estudios, *Regression Trees* con 35, *Regressions* con 31 estudios, *Case Based Reasoning* con 29 y *Support Vector Regressions* con 29 estudios.

Los hiperparámetros estudiados para las *Neural Networks* corresponden principalmente a la arquitectura de la red (cantidad de capas ocultas o intermedias y la cantidad de neuronas en cada una de estas capas), valores de ajuste del entrenamiento (tasa de aprendizaje, *momentum*, épocas de entrenamiento) y la función de activación. En la EES se ha investigado principalmente arquitecturas pequeñas, usualmente una capa oculta con menos de 20 neuronas. Entre los hiperparámetros estudiados para los *Regression Trees* se encuentran la profundidad máxima del árbol y los criterios para determinar si se divide un nodo (función de partición, cantidad mínima de instancias por hoja, incremento mínimo de rendimiento o varianza). En el caso de *Regression*, las regresiones lineares, polinomiales, logarítmicas, o modelos múltiples no cuentan con hiperparámetros, pero han sido estudiadas en conjunto

con otros algoritmos para generar modelos complejos, o como líneas base. *Case Based Reasoning* ha sido estudiado principalmente mediante *analogy based estimation* y analizando 3 hiperparámetros: la cantidad de casos similares a considerar, la función de distancia para determinar los casos más cercanos, y la función de ajuste. Para la *Support Vector Regression* se han estudiado hiperparámetros para aumentar la cantidad de dimensiones y facilitar la búsqueda de un margen de separación. En este caso se han ajustado la penalización por errores (complejidad), el tamaño del margen de tolerancia de errores, la función de *kernel*, y el parámetro para la función de *kernel*.

Tabla 3 – Algoritmos de ML

Algoritmo	Cant.	Estudios
Neural Network	39	S3, S5, S6, S8, S9, S10, S11, S12, S15, S17, S18, S19, S20, S22, S23, S25, S27, S29, S30, S32, S33, S34, S36, S41, S42, S46, S47, S49, S51, S53, S54, S55, S58, S59, S60, S61, S63, S66, S67
Regression Tree	35	S1, S5, S6, S7, S8, S10, S11, S12, S14, S15, S18, S19, S20, S22, S23, S25, S28, S29, S30, S32, S34, S35, S41, S43, S46, S47, S51, S55, S57, S59, S60, S61, S62, S63, S66
Regression	31	S2, S3, S6, S7, S8, S9, S10, S11, S14, S15, S17, S19, S22, S23, S25, S28, S29, S32, S35, S40, S41, S43, S49, S50, S55, S58, S59, S60, S61, S64, S65
Case Based Reasoning	29	S4, S8, S13, S18, S21, S22, S24, S26, S28, S29, S32, S34, S35, S37, S40, S43, S44, S45, S47, S48, S50, S52, S55, S56, S59, S61, S62, S64, S65
Support Vector Regression	29	S3, S5, S6, S8, S9, S10, S11, S12, S15, S16, S17, S18, S19, S20, S22, S28, S30, S31, S38, S39, S40, S42, S53, S54, S55, S60, S64, S66, S67
Ensemble	17	S4, S5, S6, S12, S14, S20, S22, S30, S41, S42, S46, S47, S49, S52, S57, S59, S63
K Nearest Neighbors	16	S3, S5, S6, S9, S10, S12, S14, S15, S20, S22, S23, S28, S30, S41, S46, S51

Los siguientes algoritmos de ML fueron reportados en 10 estudios o menos: Bagging [S7, S8, S10, S22, S46, S47, S51, S53, S63, S67], Random Forest [S1, S3, S6, S8, S11, S18, S22, S28], Boosting [S3, S6, S8, S11, S22, S60], Stacking [S6, S8, S28, S62], Bayesian Model [S6, S22, S28], Rule Based Estimation [S22, S46], Gaussian Process [S22], Genetic Algorithm [S2], K-Means Clustering [S28], Particle Swarm Optimization [S2] y Relevance Vector Machines [S10].

Se identificaron 5 técnicas de validación cruzada (*Cross-Validation*): *Leave-One-Out Cross-Validation* fue reportada en 32 estudios [S4, S5, S8, S11, S12, S13, S18, S19, S20, S21, S22, S23, S24, S26, S27, S29, S30, S35, S36, S37, S40, S41, S44, S49, S50, S53, S55, S56, S60, S62, S64, S66], *K-Fold Cross-Validation* en 27 [S1, S3, S10, S11, S13, S14, S15, S16, S17, S18, S19, S25, S27, S28, S31, S32, S38, S39, S40, S43, S50, S54, S59, S61, S62, S65, S66], *Hold-out Split* en 24 [S1, S2, S3, S9, S17, S18, S22, S33, S34, S38, S42, S45, S46, S47, S52, S53, S54, S55, S58, S60, S63, S64, S66,

S67], *Online* en 4 estudios [S7, S45, S51, S57] y finalmente, *Blocked Cross-Validation* en un estudio [S6]. Para la EES se recomienda *Leave-One-Out Cross-Validation* ya que las particiones generadas por este algoritmo son determinísticas, incrementando la replicabilidad de los resultados de las evaluaciones.

Con respecto a los algoritmos de pre procesamiento se reportaron 22 distintas: *Unit Range [0,1]* en 21 estudios [S8, S15, S17, S19, S25, S26, S31, S32, S33, S34, S37, S38, S39, S41, S43, S45, S46, S53, S60, S62, S67], *Logarithm* en 13 [S8, S14, S15, S17, S35, S40, S41, S47, S50, S55, S58, S61, S64], *K Nearest Neighbors* en 7 [S10, S13, S46, S47, S50, S51, S63], *K-Means Clustering* en 5 estudios [S46, S47, S50, S55, S63] al igual que *Principal Component Analysis* [S3, S8, S27, S41, S42] y *Standardization* [S3, S10, S14, S15, S49]. *One-hot Encoding* fue reportado en 4 estudios [S3, S10, S35, S55] y *Binning* [S3, S66], *BoxCox* [S15, S55], *Cook's Distance* [S15, S50], *Interquartile Range* [S15, S50], *Mean* [S34, S54], *Median* [S54, S55], *Unit Range [-1,1]* [S34, S58] en 2 estudios. Finalmente, los siguientes fueron reportados en 1 estudio: *Binary Encoding* [S49], *Jarque-Bera Test* [S15], *Least Trimmed Squares* [S50], *Listwise Deletion* [S34], *Mantel Leverage Metric* [S50], *Random*[S54], *Support Vector Regression* [S13], y *Z-Score* [S15]. El *Unit Range [0, 1]* se utiliza para eliminar el sesgo de la escala de unidad de las variables, y la transformación *Logarithmic* para que los datos cumplan los supuestos de normalidad. El método de imputación de datos faltantes más reportado es *K Nearest Neighbors* dado que considera las propiedades de los datos.

Las técnicas de selección de atributos reportadas fueron 9: *Correlation Based Feature Selection* en 10 estudios [S4, S5, S6, S15, S20, S24, S34, S41, S58, S63], *Genetic Algorithm* en 5 [S18, S42, S53, S66, S67], *Pearson correlation* en 4 [S4, S5, S9, S40], *RRelief Based Feature Selection* en 3 [S4, S5, S20], *Backward Feature Elimination* en 2 [S8, S55] y *Exhaustive Search* [S35], *Particle Swarm Optimization* [S31], *Principal Component Analysis* [S24] y *Regression* [S41] en un artículo. Cabe resaltar que *Genetic Algorithms* y *Particle Swarm Optimization* han sido utilizadas para la selección de atributos y el ajuste de hiperparámetros.

#### 4.2. Conjuntos de datos

La Tabla 4 muestra los estudios por los conjuntos de datos reportados para las evaluaciones de los modelos de EES clasificados con respecto a su repositorio de origen. El repositorio de datos más utilizado es el repositorio abierto PROMISE. Este contiene conjuntos de datos *sigle-company* (*Deshanrais*, *Albrecht*, *Kemerer*) como *cross-company* (*Miyazaki94*, *Coco-moNasa2* o *Nasa93*). La mayoría de estos conjuntos contienen menos de 100 proyectos, en la mayoría de casos. El repositorio del *International Software Benchmarking Standards Group* (ISBSG, R8, R10, R11) es *cross-company* y contiene proyectos de múltiples compañías alrededor del mundo. Por ejemplo, el *release 11* está conformado por 5052 proyectos de 24 países distintos. Dada la gran cantidad de organizaciones y sus diferentes prácticas, los datos de este

conjunto deben ser pre procesados previo a su uso para estimación. El conjunto *Open* se refiere a los repositorios de datos que se encuentran disponibles de manera abierta (*NasaBailey* y *KotenGray*). Estos se componen de datos *single-company*, reportando menos de 50 proyectos. El origen de datos *Tukutuku* corresponde con proyectos de desarrollo web. Se identificaron 3 conjuntos de datos privados que corresponden a la compañía IBM, y finalmente, un estudio utilizó 2 conjuntos de datos generados de manera artificial, marcados como *Artificial*.

Tabla 4 – Conjuntos de datos de estimación de esfuerzo

Tipo	Cant.	Conjuntos de datos y estudios
PROMISE	52	Desharnais [S2, S4, S5, S8, S9, S12, S13, S14, S18, S19, S20, S21, S22, S23, S24, S26, S27, S29, S31, S34, S35, S40, S42, S44, S45, S46, S47, S48, S50, S52, S53, S55, S56, S58, S59, S61, S62, S63, S65, S66, S67]. Cocomo81 [S1, S2, S4, S5, S8, S10, S12, S13, S14, S16, S18, S19, S20, S21, S23, S24, S26, S27, S35, S36, S37, S42, S44, S46, S47, S48, S53, S55, S57, S61, S62, S63, S66, S67]. Albrecht [S4, S5, S8, S12, S13, S18, S19, S20, S22, S24, S25, S26, S27, S30, S35, S37, S40, S42, S44, S48, S53, S61, S62, S66]. Kemerer [S8, S12, S13, S14, S16, S18, S19, S20, S21, S24, S25, S26, S35, S37, S40, S44, S48, S53, S61, S62, S66]. Maxwell [S8, S10, S14, S21, S22, S23, S24, S26, S33, S35, S37, S40, S42, S44, S48, S51, S55, S59, S61, S62]. CocomoNasa2 [S2, S8, S10, S17, S21, S22, S24, S37, S44, S46, S47, S48, S57, S61, S63]. China [S4, S5, S12, S13, S20, S21, S24, S26, S33, S35, S37, S40, S48, S62]. Miyazaki94 [S4, S5, S8, S12, S13, S14, S20, S24, S30, S40, S42, S44, S48, S52]. Telecom [S14, S26, S35, S37, S40, S44, S48, S61]. CocomoSDR [S8, S22, S24, S44, S46, S47, S63]. CocomoNasa [S2, S22, S55, S57]. Kitchenham [S10, S34, S48, S51]. USPO5 [S34, S48, S55].
ISBSG	29	ISBSG R10 [S7, S10, S26, S34, S35, S45, S46, S47, S51, S57, S61, S62, S63]. ISBSG R8 [S1, S4, S5, S12, S13, S18, S50]. ISBSG R11 [S3, S25, S29, S32, S43, S55]. ISBSG R? [S22, S54]. ISBSG R9 [S58].
Open	18	NasaBailey [S19, S26, S27, S35, S53, S60, S63, S65, S66]. Finnish [S8, S24, S40, S44]. KotenGray [S8, S19, S53, S66]. CF [S43, S56]. Costagliola05 [S39]. Jodpimai18 [S15]. Ziauddin12 [S38].
Tukutuku	5	Tukutuku [S1, S18, S40, S41, S64].
Private	4	ESA [S55]. Euroclear [S55]. Experience [S55]. IT University [S6]. IVR [S16]. Pai13 [S49].
Unknown	3	Bank [S50]. Stock [S50]. Unidentified [S16]. Zakrani18 [S11].
IBM	2	DPS [S43]. RQM [S28]. RTC [S28].
Artificial	1	Moderate [S29]. Severe [S29].

#### 4.5. Discusión

Los estudios identificados establecen la complejidad de la selección de los valores de los hiperparámetros manualmente para los distintos algoritmos de ML. Para abordar este problema los autores plantean el uso de las técnicas de ajuste automático [S4, S7, S11, S12, S13, S16, S20, S21, S26, S33, S35, S37, S40, S45, S51, S61, S62, S63, S64,

S66]. Sin embargo, también se ha reportado la falta de técnicas de ajuste [S7, S12, S41] y distintas limitaciones en estas técnicas de ajuste y en los algoritmos de ML [S1, S2, S4, S6, S10, S12, S13, S16, S17, S18, S19, S20, S24, S31, S32, S35, S40, S45, S55]. Los autores plantean la necesidad de técnicas que guíen las búsquedas de hiperparámetros a partir de las características de los conjuntos de datos.

El teorema de “*No Free Lunch*”, que establece que no existe un modelo de aprendizaje universal para resolver todos los problemas, ha sido reportado por los autores en el campo de la EES [S1, S2, S5, S9, S12, S14, S15, S16, S18, S20, S22, S24, S26, S30, S41, S42, S52, S61, S63]. Para distintos conjuntos de datos, un algoritmo de ML puede obtener resultados contradictorios, por lo que se ha establecido la necesidad de estudios comparativos [S18, S41, S55] y la implementación de métricas estandarizadas para la comparación sin sesgo entre estudios [S24, S35, S55, S66]. Dado este problema, los autores reportan las limitaciones para la generalización de los resultados. Estas limitaciones se incrementan dada la cantidad y tamaño de los conjuntos de datos disponibles para este tipo de estudios [S9, S55].

La necesidad de evidencia empírica sobre la efectividad de las técnicas de ajuste de hiperparámetros para los algoritmos de ML ha sido reportado [S8, S10, S11, S13, S15, S16, S17, S18, S30, S34, S44, S50, S51, S58, S63]. Esto incluye investigaciones relacionadas con la construcción de algoritmos de ensamblaje de ML [S8, S18, S30, S60], la evaluación de algoritmos de ML [S16, S17], la evaluación del impacto del ajuste de hiperparámetros [S51], el uso de los métodos de *kernel* [S44], técnicas de imputación de valores faltantes [S13], técnicas de detección y eliminación de valores atípicos [S50], análisis de importancia de variables de entrada [S58], estimación en intervalos [S10], estimación ágil [S11] y a través del ciclo de vida de desarrollo [S15]. Los estudios indican que la estabilidad de las estimaciones depende de distintos factores durante la construcción de los modelos de EES (conjuntos de datos, transformaciones, selección de atributos e hiperparámetros) [S8, S12, S17, S18, S20, S24, S26, S37, S41, S46, S51, S61, S66]. Todos estos factores deben ser estudiados y reportados para habilitar la replicación y la agregación de evidencia [S2, S24, S41, S46, S51, S55, S57, S58, S63], lo que implica la mejora en la completitud de los reportes de los estudios.

Los riesgos del sobre y sub ajuste en las estimaciones han sido advertidos en el área [S1, S3, S9, S10, S11, S13, S15, S17, S19, S22, S25, S26, S27, S29, S42, S46, S47, S49, S50, S54, S60, S63, S67]. Asimismo, la baja adopción de estos modelos de estimación en la industria [S2, S3, S16, S19, S53, S55, S66] en comparación con la estimación de juicio experto. Se plantea la necesidad de mejorar los resultados de las estimaciones [S14, S16, S22, S32, S57] y su adaptación para escenarios reales en-línea [S7, S51, S57] para mejorar la adopción.

La dificultad y el costo de recolección de datos [S7, S44, S47, S53, S58], la calidad de los datos recolectados [S4, S6, S7, S13, S16, S18, S24, S50, S59, S62], los valores faltantes [S4, S13], la heterogeneidad [S7, S16] y los valores atípicos [S4, S16, S18, S50, S62] de los conjuntos de datos fueron reportados como restos del área. Finalmente, el costo computacional de las técnicas de ajuste ha sido indicada estableciendo la

necesidad de técnicas más eficientes [S14, S26, S32, S34, S40, S44, S45, S48, S53, S62, S67].

## 5. Conclusiones

En este trabajo se identificaron estudios de ajuste de hiperparámetros en el contexto de la EES. Hemos reportado las técnicas de ajuste automático, los algoritmos de ML, los conjuntos de datos y las métricas de desempeño utilizadas para evaluar los modelos de EES. Los resultados ofrecen un mapeo que puede servir como guía para la selección de técnicas y algoritmos para estudios en EES. La clasificación de técnicas de ajuste indica que *Grid Search*, *Genetic Algorithms* y *Particle Swarm Optimization* son las más utilizadas. En cuanto a los algoritmos de ML, las *Neural Networks*, *Regression Trees*, *Regressions*, *Case Based Reasoning* y *Support Vector Regressions* son las más evaluadas. Asimismo, los estudios reportan múltiples algoritmos de validación cruzada, pre procesamiento y selección de atributos que se han combinado para evaluar el impacto en los resultados de las estimaciones.

Basado en los resultados del mapeo se identifican posibles problemas abiertos en el área de ajuste automático de hiperparámetros. No solo se deben realizar estudios comparativos para evaluar el impacto de las técnicas de ajuste automático en la exactitud de los modelos de EES, sino también estudiar nuevas técnicas que reduzcan la cantidad de hiperparámetros seleccionados, manteniendo la exactitud de las estimaciones. Se requiere la evaluación de la efectividad de los ensambles de algoritmos de ML (y sus hiperparámetros) en la EES. Las técnicas de ajuste no solo deben ser utilizadas en los algoritmos de ML que determinan el modelo, sino también cuando en las etapas de pre procesamiento y selección de atributos, si los algoritmos utilizados lo permiten. Se requiere la evaluación de nuevos conjuntos de datos y de la industria. El uso de estrategias que se acerquen a la práctica de la industria, tal como la estimación en-línea, debe ser evaluada. Los investigadores deben utilizar métricas de exactitud estandarizadas para habilitar la comparación de resultados independientemente de los conjuntos de datos.

## Agradecimientos

Este estudio fue apoyado por la Universidad de Costa Rica No. 834-B8-A27. Nuestro agradecimiento al Empirical Software Engineering Group (ESEG) de la Universidad de Costa Rica retroalimentación sobre este trabajo.

## Referencias

Agrawal, A., Fu, W., Chen, D., Shen, X., & Menzies, T. (2019). How to "DODGE" Complex Software Analytics. *IEEE Transactions on Software Engineering*.



- Araújo, R. D. A., Soares, S., & Oliveira, A. L. (2012). Hybrid morphological methodology for software development cost estimation. *Expert Systems with Applications*, 39(6), 6129-6139.
- Azhar, D., Riddle, P., Mendes, E., Mittas, N., & Angelis, L. (2013). Using ensembles for web effort estimation. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 173-182). IEEE.
- Azzeh, M. (2011). Software effort estimation based on optimized model tree. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering* (pp. 1-8).
- Azzeh, M., Nassif, A. B., & Banitaan, S. (2014). A better case adaptation method for case-based effort estimation using multi-objective optimization. In *2014 13th International Conference on Machine Learning and Applications* (pp. 409-414). IEEE.
- Boehm, B. W. (1984). Software engineering economics. *IEEE transactions on Software Engineering*, (1), 4-21.
- Corazza, A., Di Martino, S., Ferrucci, F., Gravino, C., Sarro, F., & Mendes, E. (2010). How effective is tabu search to configure support vector regression for effort estimation?. In *Proceedings of the 6th international conference on predictive models in software engineering* (pp. 1-10).
- Corazza, A., Di Martino, S., Ferrucci, F., Gravino, C., Sarro, F., & Mendes, E. (2013). Using tabu search to configure support vector regression for effort estimation. *Empirical Software Engineering*, 18(3), 506-546.
- Dejaeger, K., Verbeke, W., Martens, D., & Baesens, B. (2011). Data mining techniques for software effort estimation: a comparative study. *IEEE transactions on software engineering*, 38(2), 375-397.
- Idri, A., Hosni, M., & Abran, A. (2016). Systematic literature review of ensemble effort estimation. *Journal of Systems and Software*, 118, 151-175.
- Jorgensen, M., & Shepperd, M. (2006). A systematic review of software development cost estimation studies. *IEEE Transactions on software engineering*, 33(1), 33-53.
- Hosni, M., Idri, A., Nassif, A. B., & Abran, A. (2016). Heterogeneous ensembles for software development effort estimation. In *2016 3rd International Conference on Soft Computing & Machine Intelligence (ISCMi)* (pp. 174-178). IEEE.
- Hosni, M., Idri, A., & Abran, A. (2017). Investigating heterogeneous ensembles with filter feature selection for software effort estimation. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement* (pp. 207-220).

- Hosni, M., Idri, A., Abran, A., & Nassif, A. B. (2018). On the value of parameter tuning in heterogeneous ensembles effort estimation. *Soft Computing*, 22(18), 5977-6010.
- Hota, H. S., Shukla, R., & Singhai, S. (2015). Predicting Software Development Effort Using Tuned Artificial Neural Network. In *Computational Intelligence in Data Mining-Volume 3* (pp. 195-203). Springer, New Delhi.
- Kocaguneli, E., Menzies, T., & Keung, J. W. (2013). Kernel methods for software effort estimation. *Empirical Software Engineering*, 18(1), 1-24.
- Lederer, A. L., & Prasad, J. (1995). Causes of inaccurate software development cost estimates. *Journal of systems and software*, 31(2), 125-134.
- Luo, G. (2016). A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1), 18.
- Malhotra, R., Khanna, M., & Raje, R. R. (2017). On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions. *Swarm and Evolutionary Computation*, 32, 85-109.
- Minku, L. L., & Yao, X. (2013). Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, 55(8), 1512-1528.
- Minku, L. L. (2019). A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. *Empirical Software Engineering*, 24(5), 3153-3204.
- Oliveira, A. L., Braga, P. L., Lima, R. M., & Cornélio, M. L. (2010). GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *information and Software Technology*, 52(11), 1155-1166.
- Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64, 1-18.
- Sehra, S. K., Brar, Y. S., Kaur, N., & Sehra, S. S. (2017). Research patterns and trends in software effort estimation. *Information and Software Technology*, 91, 1-21.
- Song, L., Minku, L. L., & Yao, X. (2013). The impact of parameter tuning on software effort estimation using learning machines. In *Proceedings of the 9th international conference on predictive models in software engineering* (pp. 1-10).
- Wen, J., Li, S., Lin, Z., Hu, Y., & Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1), 41-59.

## Appendix D

# Paper 2: Evaluation of Grid and Random Search for Support Vector Regression

**Reference** Villalobos-Arias, L., Quesada-López, C., Guevara-Coto, J., Martínez, A., & Jenkins, M. (2020, November). Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation. In Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering (pp. 31-40).

## Status Indexed in ACM

ACM DIGITAL LIBRARY
Universidad de Costa Rica [Browse](#) [About](#) [Sign in](#) [Register](#)

Journals Magazines Proceedings Books SIGs Conferences People
Search ACM Digital Library  [Advanced Search](#)

Conference Proceedings Upcoming Events Authors Affiliations Award Winners

Home > Conferences > ICSE > Proceedings > PROMISE 2020 > Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation

RESEARCH-ARTICLE

### Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation

**Authors:** Leonardo Villalobos-Arias, Christian Ouesada-López, Jose Guevara-Coto, Alexandra Martínez, Marcelo Jenkins [Authors Info & Affiliations](#)

PROMISE 2020: Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering • November 2020 • Pages 31–40 • <https://doi.org/10.1145/3416508-3417121>

**Published:** 08 November 2020

38

PROMISE 2020: Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering

Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation

Pages 31–40

[← Previous](#)
[Next →](#)

[ABSTRACT](#)  
[References](#)  
[Index Terms](#)  
[Comments](#)

ACM DIGITAL LIBRARY

**ABSTRACT**

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

**References**

1. Amritanshu Agrawal, Wei Fu, Di Chen, Xipeng Shen, and Tim Menzies. 2019. How to "DODGE" Complex Software Analytics. IEEE Transactions on Software Engineering ( 2019 ). [6](#)

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

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

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

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

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

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

## ABSTRACT

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

## CCS CONCEPTS

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

## KEYWORDS

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

## ACM Reference Format:

Leonardo Villalobos-Arias, Christian Quesada-López, Jose Guevara-Coto, Alexandra Martínez, and Marcelo Jenkins. 2020. Evaluating Hyper-parameter Tuning using Random Search in Support Vector Machines for Software Effort Estimation. In *Proceedings of the 16th ACM International Conference*

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

PROMISE '20, November 8–9, 2020, Virtual, USA

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-8127-7/20/11...\$15.00

<https://doi.org/10.1145/3416508.3417121>

*on Predictive Models and Data Analytics in Software Engineering (PROMISE '20), November 8–9, 2020, Virtual, USA. ACM, New York, NY, USA, 10 pages.*  
<https://doi.org/10.1145/3416508.3417121>

## 1 INTRODUCTION

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

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

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

## 2 RELATED WORK

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

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

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

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

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

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

## 3 STUDY DESIGN

The main objective of this study is to investigate to what extent the random search hyper-parameter tuning approach affects the accuracy and stability of support vector regression for software effort estimation. We compared the results against those obtained

from ridge regression models and grid search-tuned models. The following research questions were posed:

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

### 3.1 Data Set

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

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

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

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

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

**Table 1: ISBSG 2018 Release 1 data sets**

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

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

### 3.2 Machine Learning Model Evaluation

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

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

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

### 3.3 Machine Learning Algorithms and Hyper-parameter Settings

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

**3.3.1 Hyper-parameter Tuning. Grid search.** Grid search is a hyper-parameter tuning approach that evaluates each possible parameter combination in the search space [4]. The search space is formed by a hyper-parameter grid: a multi-dimensional space with one dimension per parameter. A point in the search space is defined by a value along each dimension. For example, a valid point in the search space comprised by hyper-parameters  $C = \{100, 150, 200\}$  and  $\gamma =$

$\{0.001, 0.01, 0.1\}$  would be ( $C = 150, \gamma = 0.01$ ). Grid search explores all combinations of values for each parameter (i.e., the entire search space). For each such combination (point in the search space), a model is built and evaluated using those hyper-parameter values. The hyper-parameter combination with the highest accuracy is reported. The scikit-learn implementation of grid search search uses cross-validation for the search process. We selected a 10-fold cross-validation approach.

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

**Default hyper-parameters.** To represent the scenario without hyper-parameter tuning, we used the default hyper-parameter values for each studied method, as defined by the scikit-learn library. This is our baseline for determining the accuracy improvement of hyper-parameter tuning. In scikit-learn, the default hyper-parameters for SVR are: a) kernel = rbf, b)  $C = 1$ , and  $\epsilon = 0.1$  c)  $\gamma = 1/(n * var)$  where  $n$  = number of features,  $var$  = variance of the data set. The default hyper-parameter for ridge regression is  $\alpha = 1.0$ . The default hyper-parameter for VT is  $threshold = 0$ . The default hyper-parameter for CP is  $percentile = 10$ . Regarding the kernel hyper-parameter for SVR, its default value is radial basis function (rbf). This kernel has been successfully used in multiple SEE studies [8, 17], and has shown to work well in both high and low dimensional feature spaces when adequately adjusted [18].

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

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



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

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

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

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

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

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

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

### 3.4 Performance Metrics

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

**Table 2: Hyper-parameter values**

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

the mean absolute residual (*MAR*):

$$MAR = \frac{1}{n} \sum_{i=1}^n AR_i. \quad (1)$$

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

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

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

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

$$SA = 1 - \frac{MdAR}{MdAR_{p_0}}, \quad (3)$$

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

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

$$\Delta = \frac{MdAR_1 - MdAR_2}{SdAR_2}, \quad (4)$$

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

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



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

$$\text{stability ratio} = \frac{SdAR_1}{SdAR_2} \quad (6)$$

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

### 3.5 Threats to Validity

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

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

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

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

## 4 RESULTS

### 4.1 Prediction Accuracy Improvement when Using Random Search

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

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

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

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

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

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

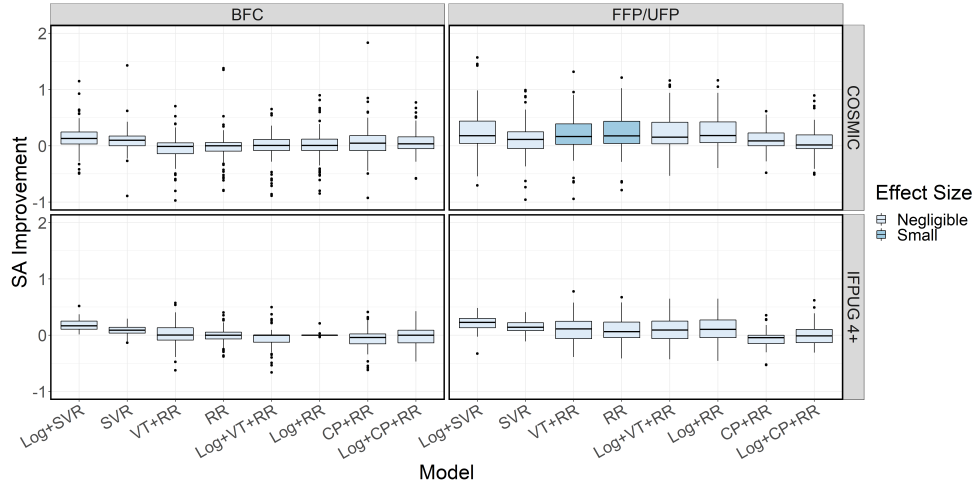


Figure 1: Accuracy improvement and effect size of random search.

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

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

## 4.2 Prediction Accuracy Stability when Using Random Search

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

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

The accuracy of the Log+SVR and SVR models became more stable due hyper-parameter tuning. These techniques boast a median

stability ratio of 0.842 to 0.986 for Log+SVR, and 0.840 to 1.004 for SVR, as shown on table 4. For ridge regression models, the stability ratio varies from 0.880 to 1.040. Most cases in which regression models had a median increase in variability were on the IFPUG 4+ data sets.

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

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

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

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

### 4.3 Ranking of SEE Models Based on Standardized Accuracy

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

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

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

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

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

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

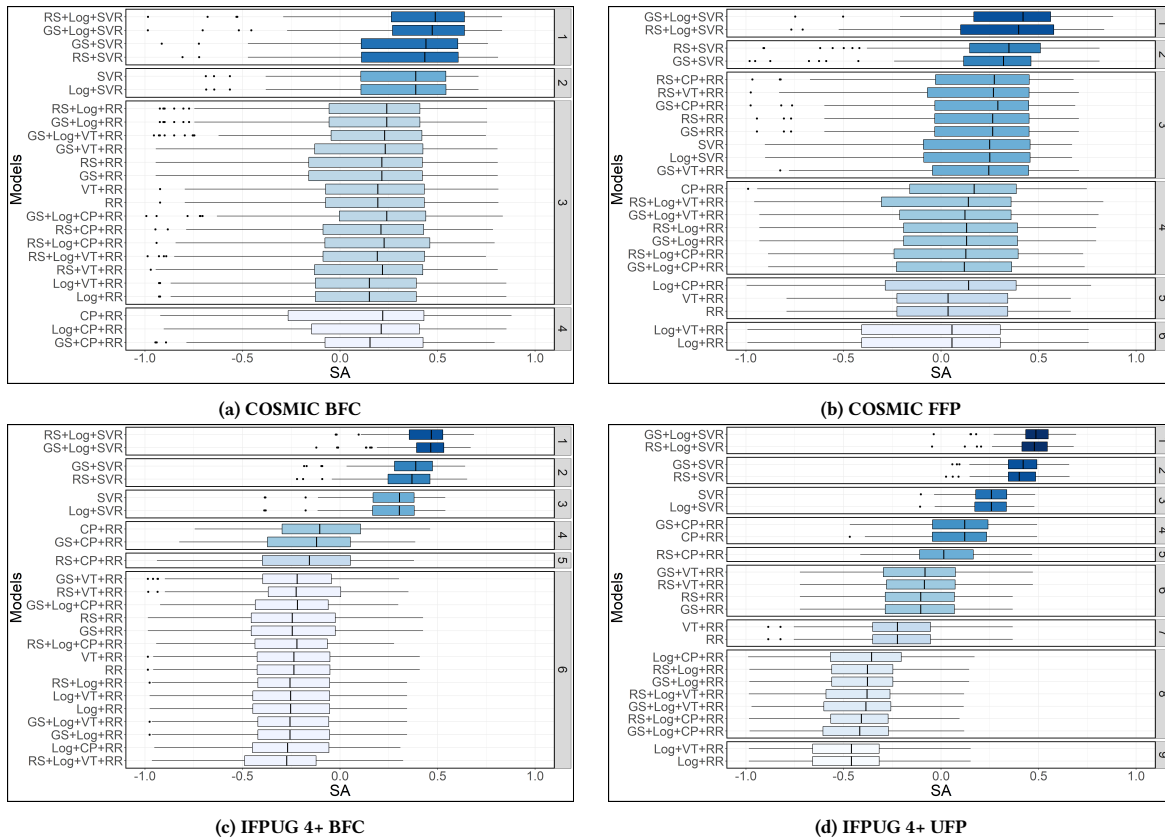


Figure 2: Scott-Knott clusters and SA of the studied SEE models.

## 5 DISCUSSION

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

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

For COSMIC function points, the prediction accuracy for SVR models was higher when models were trained and tested with data sets that used basic functional components over function point total. This goes against previous studies that use the ISBSG repository [9, 11, 17, 23, 32], as they choose to select the function point total over the individual components. Our results show that SEE models

could benefit from the use of BFC as features. For instance, SEE models could determine if BFC affected effort differently, depending on other features. For example, input points could involve more effort in projects developed with programming language A over programming language B. Future work could research use BFC alongside total functional size for SEE models.

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

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

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

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

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

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

Hosni *et al.* [17] study the effect of hyper-parameter values on heterogeneous ensemble effort estimation. We focus on the results obtained in the first experimental study presented by the paper, in which four base techniques, including SVR, are trained on the

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

## 6 CONCLUSION

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

Furthermore, this study demonstrated that use of RS for model tuning could provide an improvement in model stability. This observation was considerable for SVR models, which had the best median metrics for accuracy and stability. Because RS searches a limited parameter space or number of iterations, compared to



grid search, this could have accounted for some instances where our results were lower than expected. Finally, we identified that model tuning and data processing, in this case using logarithmic transformation, improved model performance.

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

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

## ACKNOWLEDGMENTS

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

## REFERENCES

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

## Appendix E

# Paper 3: Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation

**Reference** Villalobos-Arias, L., Quesada-López, C., Martínez, A., & Jenkins, M. (2021). Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation. In *Trends and Applications in Information Systems and Technologies: Volume 3 9* (pp. 589-598). Springer International Publishing.

# Status Indexed in Scopus



Search Sources Lists SciVal

[Create account](#)
[Sign in](#)

< Back to results | 1 of 1

Export
 Download
 Print
 E-mail
 Save to PDF
 Add to List
 More... >

[View at Publisher](#)

**Document type**  
Conference Paper  
**Source type**  
Book Series  
**ISSN**  
21945357  
**ISBN**  
978-303072659-1  
**DOI**  
10.1007/978-3-030-72660-7\_56  
 Advances in Intelligent Systems and Computing • Volume 1367 AISC, Pages 589 - 598 • 2021 • World Conference on Information Systems and Technologies, WorldCIST 2021, 30 March 2021 - 2 April 2021  
 Villalobos-Arias L , Quesada-Lopez C , Martinez A   
 Jenkins M   
[Save all to author list](#)

[View more](#) ▾

**Abstract**

**Author keywords**

**Indexed keywords**

**SciVal Topics**

**Funding details**

**Abstract**  
 Classification and regression trees (CART) have been reported to be competitive machine learning algorithms for software effort estimation. In this work, we analyze the impact of hyper-parameter tuning on the accuracy and stability of CART using the grid search, random search, and DODGE approaches. We compared the results of CART with support vector regression (SVR) and ridge regression (RR) models. Results show that tuning improves the performance of CART models up to a maximum of 0.153 standardized accuracy and reduce its stability ratio to a minimum of 0.819. Also, CART proved to be as competitive as SVR and outperformed RR. © 2021, The Author(s), under exclusive license to Springer Nature Switzerland AG.

**Author keywords**  
 Hyper-parameter tuning; ISBSG; Software effort estimation

**Engineering controlled terms**  
 Forestry; Information systems; Information use; Learning algorithms; Machine learning; Object oriented programming; Tuning

**Engineering uncontrolled terms**  
 CART models; Classification and regression tree; Grid search; Hyper-parameter; Random searches; Ridge regression; Software effort estimation; Support vector regression (SVR)

**Engineering main heading**  
 Support vector regression

**Topic name** Effort Estimation; Functional Size Measurement; COSMIC

**Prominence percentile** 92.766

Funding sponsor	Funding number	Acronym
Universidad de Costa Rica		UCR

[See opportunities by UCR](#)

**Metrics** [View all metrics](#) >

**Cited by 0 documents**  
 Inform me when this document is cited in Scopus.  
[Set citation alert](#) >

**Related documents**  
 Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation  
 Villalobos-Arias, L., Quesada-López, C., Guevara-Coto, J. (2020) PROMISE 2020 - Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Co-located with ESEC/FSE 2020

Machine learning hyper-parameter tuning techniques for software effort estimation. A systematic mapping study | Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo. Un mapeo de literatura  
 Villalobos-Arias, L., Quesada-López, C., Martínez, A. (2021) RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao

A comparative study of data mining and machine learning techniques for software effort estimation using function points | Un estudio comparativo de técnicas de minería de datos y aprendizaje máquina para la estimación del esfuerzo utilizando puntos de función  
 López, C.Q., Murillo-Morera, J., Jenkins, M. (2019) RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao

[View all related documents based on](#)





# Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation

Leonardo Villalobos-Arias<sup>(✉)</sup>, Christian Quesada-López, Alexandra Martínez,  
and Marcelo Jenkins

Universidad de Costa Rica, San Pedro, Costa Rica  
{leonardo.villalobosarias,cristian.quesadalopez,  
alexandra.martinez,marcelo.jenkins}@ucr.ac.cr

**Abstract.** Classification and regression trees (CART) have been reported to be competitive machine learning algorithms for software effort estimation. In this work, we analyze the impact of hyper-parameter tuning on the accuracy and stability of CART using the grid search, random search, and DODGE approaches. We compared the results of CART with support vector regression (SVR) and ridge regression (RR) models. Results show that tuning improves the performance of CART models up to a maximum of 0.153 standardized accuracy and reduce its stability radio to a minimum of 0.819. Also, CART proved to be as competitive as SVR and outperformed RR.

**Keywords:** Software effort estimation · Hyper-parameter tuning · ISBSG

## 1 Introduction

Machine learning (ML) has been studied for many years to increase the accuracy of software effort estimations (SEE) [8]. Hyper-parameter tuning is the process of exploring and selecting the optimal ML hyper-parameters, and it is considered a crucial step for building accurate SEE models [28]. Automated hyper-parameter tuning approaches have been evaluated in SEE to improve model performance, but they come at a computational cost. For example, grid search exhaustively executes all possibilities in a search space, hence poorly scales to larger search spaces [4] and requires a large amount of resources [28].

Recent research has endorsed faster tuners that can find good (or even optimal) hyper-parameter values [1, 27], such as random search and DODGE. Random search selects and evaluates random samples from a large search space [4]. DODGE stores previously visited hyper-parameter values on a tree, selecting the most promising branches for further exploration [1].

Classification and regression trees (CART) has been shown to be competitive ML algorithms for SEE, which can handle heterogeneous project data [15].

CART can also outperform other machine learning techniques such as regression, multilayer perceptron and case based reasoning models [16]. Previous studies have analyzed the impact of hyper-parameter tuning on the accuracy of CART and have reported promising results [27].

In this paper, we analyze the impact of hyper-parameter tuning on the accuracy and stability of CART. In a previous work, we evaluated the impact of grid search and random search hyper-parameter tuners in support vector regression (SVR) and ridge regression (RR) models using the ISBSG 2018 R1 dataset [25]. This paper extends our previous work by including the DODGE tuner and CART model. We compare the impact of these hyper-parameter tuning algorithms and learning models on four subsets of the ISBSG 2018 R1 dataset.

This paper is organized as follows. Section 2 presents related work and Sect. 3 presents the study design. Section 4 details the results and Sect. 5 presents the conclusions.

## 2 Related Work

Minku [15] proposed and evaluated an online SEE hyper-parameter tuning and clustering approach, including both regression trees and random forest. The OIL tuning architecture was presented by Xia *et al.* [27] to evaluate 3 tuners applied to CART and k-nearest neighbors (KNN). Azzeh [3] investigates the Bee's algorithm to find high-performing parameter values for regression trees. Song *et al.* [21] compare the performance of default, best and worst hyper-parameters for CART and other learners. Ertuğrul *et al.* [9] compared 9 different grid search tuned SEE models and showed that CART achieved the lowest mean average error. Huang *et al.* [12] show that the use of data pre-processing techniques combined with CART should go through a careful selection process.

Hyper-parameter tuned CART models have been researched in other contexts. The defect prediction studies by Tantithamthavorn *et al.* [24] evaluate grid search tuned models for improving prediction accuracy and stability, as well as the computational cost of hyper-parameter tuning. Fu *et al.* [10] employ the differential evolution tuner with CART for defect prediction, showing that tuning can change previously reported rankings of ML. The DODGE tuning tool by Agrawal *et al.* [1] improves on exhaustive approaches by avoiding redundant hyper-parameter settings, running orders of magnitude faster, and generating more accurate models.

## 3 Experimental Design

The main objective of this study was to evaluate to what extent does hyper-parameter tuning impact the accuracy and stability of classification and regression trees (CART) in the context of SEE. This study extends our previous work [25] using the DODGE algorithm and the CART learner. To meet this objective, the following research questions were posed:

RQ1. What is the impact of hyper-parameter tuning on the accuracy and stability of CART models?

RQ2. How competitive are CART models compared to the SVR and RR models?

**Dataset.** We used the International Software Benchmarking Standards Group (ISBSG) Development & Enhancement 2018 Repository Release 1. The dataset was pre-processed following the guidelines stated in [25]. We selected projects that (a) have A or B of quality, and have function point quality A or B for IFPUG [8, 11], (b) recorded effort that accounts only for the development team, (c) are new development projects, and (d) were measured with COSMIC or IFPUG 4+ approaches [17]. We split the data into subsets by their function point measure unit (IFPUG 4+ and COSMIC) and generate two datasets: one using base functional components (BFC) and one using the total function points (FP). Thus, the study used four datasets: IFPUG 4+ UFP, IFPUG 4+ BFC, COSMIC FFP, and COSMIC BFC.

Feature selection criteria were applied independently for each subset, based on the protocol of Dejaeger *et al.* [8] and recommendations detailed in González-Ladrón-de-Guevara *et al.* [11]. We kept only those features that (a) are relevant for effort estimation, (b) are not redundant with other features, (c) are available at the time of estimation, and (d) have less than 25% missing values. The final dataset description and selected features are available in [25].

**Machine Learning Model Evaluation.** In order to evaluate the machine learning models, two validation approaches were used. The first validation approach was a hold-out set, where 90% of the data was used for model construction and evaluation, and the remaining 10% was used as a test set. The accuracy metrics presented in this study were obtained from training and evaluating the models in the 90% partition. A second validation approach was used on the 90% selected data to construct the models. We validated the performance of the investigated SEE approaches through 10 times 10-fold Cross-Validation (CV), based on recent work by Song *et al.* [22, 23] to avoid high variance problems that may rise from using leave-one-out in the smaller (i.e. COSMIC) datasets. The accuracy metrics were calculated for each validation fold. The test set (10%) is used to corroborate the results reported in this study.

**Experimental Framework.** Our framework was built using Python 3.7 and the scikit-learn library [25]. Our DODGE algorithm implementation is based on the work of Agrawal *et al.* [1] adapted to the scikit-learn tuning interface.

**Machine Learning Techniques.** We constructed the SEE models by combining the following techniques: the logarithmic (Log) [7] data transformation (DT), and 3 ML algorithms, classification and regression trees (CART) [6], support vector regression (SVR) [26] and ridge regression (RR) [8]. Models were also

**Table 1.** Hyper-parameter values.

Model	Hyper-parameters and their values
CART	$\text{min\_samples\_leaf} = \{1, 2, 3, \dots, 20\}$ ; $\text{min\_impurity\_decrease} = (10^x, x = \{-5, -4.5, -4, \dots, 0\})$ ; $\text{max\_depth} = \{1, 2, 3, \dots, 20\}$
SVR	$\text{kernel} = \{\text{rbf}, \text{sigmoid}\}$ ; $\gamma = (10^x, x = \{-3, -2.5, -2, \dots, -0.5\})$ ; auto, scale; $C = 1, 5, 15, 30, \{50, 100, \dots, 450\}, \{500, 1000, \dots, 15000\}$ ; $\epsilon = 10^x, x = \{-3, -2.5, -2, \dots, -0.5\}$
RR	$\alpha = 1, \{5, 10, \dots, 45\}, \{50, 75, \dots, 500\}$
VT+RR	$\text{threshold} = 0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5$ ; parameters from RR
CP+RR	$\text{percentile} = \{10, 20, 30, \dots, 90\}$ ; parameters from RR

constructed using no data transformation. For model tuning, we used 3 hyper-parameter tuning (PT) approaches: random search (RS) [4], grid search (GS) [5], and DODGE (DG) [1]. Models were also build using default parameters. For ridge regression, 2 feature selection (FS) methods were used: variance threshold (VT) [2] and correlation percentile (CP) [19]. SVR and CART were evaluated using these feature selectors, but they did not provide significant differences in accuracy and were not reported. A total of forty models were compared, product of the 4 hyper-parameter tunings (3 techniques + default), 2 data transformations (Log + None), and 5 ML models (CART, SVR, RR, VT+RR, CP+RR).

**Hyper-Parameter Values.** The hyper-parameter values researched in this study are shown in Table 1. These values were selected from existing recommendations in the literature [14, 22, 25]. For the DODGE hyper-parameter tuning, the range of values used for numerical features corresponds with the minimum and maximum values of the search space. The default hyper-parameters are those specified by the scikit-learn library, and are included in the hyper-parameter search space. In total, an exhaustive approach like grid search would explore 4,851 hyper-parameter combinations for CART, 4,128 for SVR, 29 for RR, 232 for VT+RR, and 261 for CP+RR. In contrast, the Random and Dodge tuners were configured to explore a maximum of 60 hyper-parameter settings.

**Evaluation Metrics.** We measured the accuracy of SEE models using absolute residual (i.e., the absolute difference between the predicted and actual effort values) metrics: the median absolute residual (*MdAR*) and the standardized accuracy (*SA*) [20]. We measured the improvement on accuracy (with respect to default parameters) using the improvement ratio (*imp*) [24]. To measure the stability of the tuned approaches, we employed the stability ratio [24] based on the standard deviation of the absolute residual (*SdAR*). *MdAR* and *SdAR* are respectively obtained by calculating the median and standard deviation of the absolute residuals. We use the version of *SA* by Minku [15], using the random estimation baseline model  $MdAR_{p0}$  recommended by Langdon *et al.* [13].

### 3.1 Threats to Validity

*Internal validity:* The accuracy of the constructed models is measured in standardized accuracy using a baseline model [13]. We employed 10 times 10-fold cross-validation to calculate these metrics. *Construct validity:* To mitigate the ISBSG dataset variability, we selected only those projects with high data quality rating. The search space for the hyper-parameter tuning techniques was selected based on previous literature [14,22,25]. Missing value imputation introduces some threats to construct validity. *External validity:* This study covers only the ISBSG 2018 Release 1 dataset. The results of this study cannot be generalized to other datasets, projects, or the software development industry. *Conclusion validity:* We use one hundred cross-validation iterations to provide enough experimental runs and measurements. We employed the Scott-Knott algorithm to quantify whether tuning had an effect on the accuracy and stability.

## 4 Results

Table 2 shows the improvement and stability ratio of tuned models with respect to default parameters. The background color indicates the ranking determined by the Scott-Knott analysis [18]. Results are presented for each model, data transformation, hyper-parameter tuning approach, and dataset.

To determine if tuning impacted the accuracy and stability, we ranked all constructed models using the Scott-Knott algorithm. This method uses a hierarchical clustering algorithm to partition the treatments into equivalent groups. The Scott-Knott analysis was performed for once for each dataset, assigning each model and tuner a rank. The analysis was performed separately on the accuracy (SA) and stability (SdAR) achieved by the models. The Box-Cox transformation was used on these metrics before the analysis, as Scott-Knott requires its input to follow a normal distribution.

### 4.1 Impact of Tuning on Accuracy and Stability

**Accuracy.** For CART models, grid search increased the median accuracy by up to 0.143 SA, random search by up to 0.153 SA, and DODGE by up to 0.141 SA. The three tuning approaches provided a similar increase in SA, the largest difference in performance being around 0.038 SA for CART and 0.012 for Log+CART. The statistical analysis shows that the three tuning approaches have performance above default parameters, and have equivalent improvement ratios when applied on CART. Moreover, tuned CART models were classified as the top-performing models on the COSMIC FFP dataset. We can thus determine that studied hyper-parameter tuners similarly improve the accuracy of CART.

For SVR models, grid search increased the median accuracy by up to 0.223 SA, random search by up to 0.227 SA, and DODGE by up to 0.228 SA. The Scott-Knott analysis determined that DG+SVR performs equally to default

**Table 2.** Median improvement ratio and stability ratio of tuned models.

		Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8	Rank 9
Model	Tuning	improvement ratio (RQ1)				stability ratio (RQ2)				
		COSMIC		IFPUG 4+		COSMIC		IFPUG 4+		
		BFC	FFP	BFC	UFP	BFC	FFP	BFC	UFP	
CART	DG	0.141	0.115	0.088	0.107	0.883	0.954	0.916	0.819	
	RS	0.136	0.153	0.071	0.100	0.893	0.899	0.935	0.838	
	GS	0.121	0.138	0.103	0.105	0.917	0.901	0.934	0.882	
Log+CART	DG	0.136	0.104	0.085	0.089	0.867	0.964	0.862	0.827	
	RS	0.145	0.111	0.097	0.097	0.890	0.919	0.862	0.862	
	GS	0.143	0.114	0.092	0.101	0.898	0.908	0.824	0.875	
SVR	DG	0.045	0.000	0.000	0.000	0.998	0.997	1.000	1.000	
	RS	0.099	0.113	0.088	0.140	1.004	0.840	0.958	0.877	
	GS	0.102	0.092	0.105	0.156	1.002	0.885	0.958	0.864	
Log+SVR	DG	0.084	0.163	0.159	0.228	0.978	0.829	0.911	0.850	
	RS	0.129	0.180	0.166	0.227	0.986	0.842	0.911	0.915	
	GS	0.128	0.171	0.177	0.223	0.984	0.845	0.922	0.909	
RR	DG	0.000	0.191	-0.012	0.097	1.000	0.990	1.033	1.007	
	RS	0.000	0.177	0.000	0.064	1.000	0.971	1.000	1.006	
	GS	0.000	0.177	0.000	0.064	1.000	0.971	1.000	1.006	
Log+RR	DG	0.000	0.184	0.000	0.091	1.000	0.904	1.000	1.010	
	RS	0.005	0.183	0.000	0.104	0.997	0.880	1.000	1.018	
	GS	0.005	0.183	0.000	0.104	0.997	0.880	1.000	1.018	
CP+RR	DG	0.003	0.120	0.000	0.000	0.998	1.006	1.000	1.000	
	RS	0.047	0.081	-0.041	-0.045	0.981	0.996	1.000	1.000	
	GS	0.018	0.083	0.000	0.000	0.980	0.996	1.000	1.000	
Log+CP+RR	DG	0.000	0.000	0.000	0.000	0.993	1.000	1.000	1.000	
	RS	0.036	0.014	0.000	-0.012	0.980	0.998	0.999	1.001	
	GS	0.040	0.001	-0.014	-0.002	0.988	1.000	0.999	1.000	
VT+RR	DG	0.000	0.169	0.000	0.114	1.000	0.980	1.053	1.026	
	RS	-0.012	0.165	0.002	0.113	0.999	0.971	1.040	1.010	
	GS	0.000	0.160	0.000	0.108	1.000	0.968	1.005	1.009	
Log+VT+RR	DG	0.000	0.089	0.000	0.158	1.000	0.905	1.000	1.039	
	RS	0.005	0.152	0.000	0.091	0.988	0.892	1.000	1.024	
	GS	0.007	0.158	0.000	0.093	0.985	0.899	1.000	1.023	

SVR. The remaining tuners had significantly different SA from default parameters, consistently obtaining the second-highest rank among all methods. In the Log+SVR model, the three tuners were both above default parameters and had similar performance (excepting DG+Log+SVR on COSMIC BFC). Moreover, the three models obtained the top rank across all datasets.

For RR models, grid search increased the median accuracy by up to 0.183 SA, random search by up to 0.183 SA, and DODGE by up to 0.191 SA. The impact of tuning depended on the type of dataset and model. Tuning never improved the performance of CP+RR (except in IFPUG 4+ UFP) and Log+CP+RR. Tuning improved accuracy of all other RR models in full function point datasets, but it was not effective in BFC datasets.



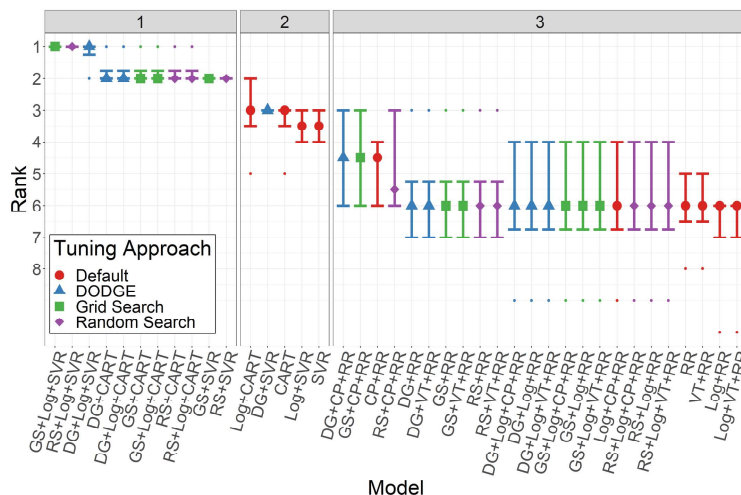
**Stability.** For CART models, grid search achieved a median stability ratio of 0.825, random search of 0.838, and DODGE of 0.819. The statistical analysis shows that the three tuning approaches have less deviation than default parameters, and have equivalent stability ratios when applied on CART. The DODGE approach provided the lowest stability ratio on several datasets (5 out of 8). We can thus determine that the studied hyper-parameter tuners improve the stability of CART.

For SVR models, grid search achieved a median stability ratio of 0.845, random search of 0.840, and DODGE of 0.829. The Scott-Knott analysis determined that GS+SVR, RS+SVR, and DG+SVR are not different from default SVR in the COSMIC BFC dataset. In addition, DODGE had stability equal to default parameters in IFPUG 4+ datasets. In the Log+SVR model, the three tuners were both more stable than default parameters and achieved similar ratios.

For RR models, grid search achieved a median stability ratio of 0.880, random search of 0.840, and DODGE of 0.904. The three tuning approaches were similar with respect to the level of stability they provide. Tuning did not improve stability on the IFPUG 4+ datasets, nor for the CP+RR and Log+CP+RR models.

### 4.2 Ranking of SEE Models

Figure 1 shows the distribution of ranks of each SEE model researched in this study. The models are grouped based on the results of a second Scott-Knott analysis. The first group consists of all tuned (Log+)CART models, and all tuned (Log+)SVR models except DG+SVR. The second group consists of DG+SVR, untuned (Log+)SVR, and untuned (Log+)CART. The third group is comprised of all RR models. Two models rank first across all datasets: RS+Log+SVR, and GS+Log+SVR. The DG+Log+SVR model ranked first in all but one dataset.



In general terms, tuned SVR and CART outperformed untuned SVR and CART models, which in turn ranked above ridge regression. Tuned CART and SVR models provide similar results, but vary depending on the dataset. In the case of RR models, parameter tuning does not greatly affect their overall performance. However, the log transformation does appear to reduce the performance of RR models. The best models were a combination of tuning, log transformation and the SVR model.

### 4.3 Discussion

The results confirm previous reports in SEE literature that state that hyper-parameter tuning improves the accuracy of machine learning models. These techniques have shown to either maintain or improve model stability, as in [25].

The three tuning approaches provided equivalent increases in accuracy (33 out of 40) and stability (31 out of 40), but their use of resources was different. Random search and DODGE explored a total of 60 different hyper-parameter values. This exploration requires training and testing a model to measure the performance of the hyper-parameters. On the other hand, grid search explored a much higher amount of parameter values: 4851 for CART, 4128 for GS, 242 for VT+RR, and 261 for CP+RR. This amounted to a higher requirement of computational resources and time for grid search to obtain results that are similar to the simple, faster approaches. Our recommendation for researchers is to favor simple but effective tuners as DODGE and random search when constructing SEE models for CART and SVR on datasets similar to ISBSG. Further comparisons using other ML algorithms and datasets are necessary to make this statement more general.

The effect of tuning depended on whether the estimation used total function points or the basic functional components. CART and SVR models had better prediction when using BFC for COSMIC, and when using UFP for IFPUG 4+. This shows that there is potential in performing SEE using basic functional components instead of function point totals.

There are some combinations of techniques that increase effectiveness. For example, using the logarithm transformation increased the accuracy and stability of SVR, but had mostly no effect on CART. Moreover, applying the Log transformation decreased the accuracy of ridge regression models, regardless of tuning or feature selection. In the case of ridge regression, feature selection provided improvements when used in combination with tuning. Although RR models are less accurate than CART or SVR, they have the advantage of being more stable.

## 5 Conclusion

In this study, we evaluated the impact of hyper-parameter tuning when applied to classification and regression trees. We evaluated three different hyper-parameter tuning approaches—grid search, random search, and DODGE. Tuned CART models were also compared against similarly tuned SVR and ridge regression models. The models were evaluated on their ability to estimate development effort on 4 sub sets derived from the ISBSG 2018 Release 1 dataset.



The main finding of this study was that all of the hyper-parameter tuning approaches improved the accuracy of CART and SVR models. Moreover, the three tuning approaches provided similar increases to accuracy, even though they used different amounts of resources (i.e. 4,851 in GS+CART against 60 in RS+CART and DG+CART). The stability of CART and SVR models increased when using tuning approaches. SEE practitioners or researchers could employ a non-exhaustive tuning approach and obtain improvements in accuracy using less computational resources. Of all the studied models, the combination of tuning, Log transformation, and SVR was the most accurate for predicting software effort in the studied dataset.

Future extensions of this study would explore datasets commonly used in SEE studies, as well as incorporate further machine learning algorithms, feature selectors, data transformations, hyper-parameter tuners, and the tuned hyper-parameters. Lastly, there is potential in researching the parameter values of hyper-parameter approaches, such as the parameter  $\epsilon$  of DODGE, and their effect in the accuracy of the models.

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

## References

1. Agrawal, A., Yang, X., Agrawal, R., Shen, X., Menzies, T.: Simpler hyperparameter optimization for software analytics: why, how, when? arXiv preprint [arXiv:2008.07334](https://arxiv.org/abs/2008.07334) (2020)
2. Albon, C.: Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning. O'Reilly Media, Inc., Newton (2018)
3. Azzeh, M.: Software effort estimation based on optimized model tree. In: Proceedings of the 7th International Conference on Predictive Models in Software Engineering, pp. 1–8 (2011)
4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **13**(Feb), 281–305 (2012)
5. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, pp. 2546–2554 (2011)
6. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. CRC Press, Boca Raton (1984)
7. Corazza, A., Di Martino, S., Ferrucci, F., Gravino, C., Sarro, F., Mendes, E.: How effective is tabu search to configure support vector regression for effort estimation? In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, pp. 1–10 (2010)
8. Dejaeger, K., Verbeke, W., Martens, D., Baesens, B.: Data mining techniques for software effort estimation: a comparative study. *IEEE Trans. Software Eng.* **38**(2), 375–397 (2011)
9. Ertuğrul, E., Baytar, Z., Çatal, Ç., Muratlı, Ö.C.: Performance tuning for machine learning-based software development effort prediction models. *Turkish J. Electr. Eng. Comput. Sci.* **27**(2), 1308–1324 (2019)

10. Fu, W., Menzies, T., Shen, X.: Tuning for software analytics: is it really necessary? *Inf. Softw. Technol.* **76**, 135–146 (2016)
11. González-Ladrón-de Guevara, F., Fernández-Diego, M., Lokan, C.: The usage of ISBSG data fields in software effort estimation: a systematic mapping study. *J. Syst. Softw.* **113**, 188–215 (2016)
12. Huang, J., Li, Y.F., Xie, M.: An empirical analysis of data preprocessing for machine learning-based software cost estimation. *Inf. Softw. Technol.* **67**, 108–127 (2015)
13. Langdon, W.B., Dolado, J., Sarro, F., Harman, M.: Exact mean absolute error of baseline predictor, marp0. *Inf. Softw. Technol.* **73**, 16–18 (2016)
14. Malgonde, O., Chari, K.: An ensemble-based model for predicting agile software development effort. *Empir. Softw. Eng.* **24**(2), 1017–1055 (2019)
15. Minku, L.L.: A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. *Empirical Softw. Eng.* **24**, 1–52 (2019)
16. Najm, A., Marzak, A., Zakrani, A.: Systematic review study of decision trees based software development effort estimation. *Organization* **11**(7) (2020)
17. Quesada-López, C., Murillo-Morera, J., Jenkins, M.: Un estudio comparativo de técnicas de minería de datos y aprendizaje máquina para la estimación del esfuerzo utilizando puntos de función. *Revista Ibérica de Sistemas e Tecnologías de Informação (E17)*, 595–609 (2019)
18. Scott, A.J., Knott, M.: A cluster analysis method for grouping means in the analysis of variance. *Biometrics* 507–512 (1974)
19. Shalev-Shwartz, S., Ben-David, S.: *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, Cambridge (2014)
20. Shepperd, M., MacDonell, S.: Evaluating prediction systems in software project estimation. *Inf. Softw. Technol.* **54**(8), 820–827 (2012)
21. Song, L., Minku, L.L., Yao, X.: The impact of parameter tuning on software effort estimation using learning machines. In: *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, pp. 1–10 (2013)
22. Song, L., Minku, L.L., Yao, X.: The potential benefit of relevance vector machine to software effort estimation. In: *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, pp. 52–61 (2014)
23. Song, L., Minku, L.L., Yao, X.: Software effort interval prediction via Bayesian inference and synthetic bootstrap resampling. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **28**(1), 1–46 (2019)
24. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: The impact of automated parameter optimization on defect prediction models. *IEEE Trans. Software Eng.* **45**(7), 683–711 (2018)
25. Villalobos-Arias, L., Quesada-López, C., Guevara-Coto, J., Martínez, A., Jenkins, M.: Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation. In: *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2020)*. ACM (2020)
26. Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C.: Systematic literature review of machine learning based software development effort estimation models. *Inf. Softw. Technol.* **54**(1), 41–59 (2012)
27. Xia, T., Krishna, R., Chen, J., Mathew, G., Shen, X., Menzies, T.: Hyperparameter optimization for effort estimation. *arXiv preprint [arXiv:1805.00336](https://arxiv.org/abs/1805.00336)* (2018)
28. Yang, L., Shami, A.: On hyperparameter optimization of machine learning algorithms: theory and practice. *Neurocomputing* **415**, 295–316 (2020)

## Appendix F

# Paper 4: Hyper-parameter Tuning using Genetic Algorithms for Software Effort Estimation

**Reference** Villalobos-Arias, L., Quesada-López, C., Martínez, A., & Jenkins, M. (2021). Hyper-parameter Tuning using Genetic Algorithms for Software Effort Estimation. In Proceedings of the 16th Iberic Conference on Information and Technology Systems.

## Status Indexed in IEEE Xplore

IEEE.org | IEEE Xplore | IEEE-SA | IEEE Spectrum | More Sites
SUBSCRIBE | Cart | Create Account | Personal Sign In

[Browse](#) | [My Settings](#) | [Help](#) | [Institutional Sign In](#)

All

Q

ADVANCED SEARCH

Conferences > 2021 16th Iberian Conference ...

### Hyper-parameter Tuning using Genetic Algorithms for Software Effort Estimation

**Publisher:** IEEE [Cite This](#) [PDF](#)

Leonardo Villalobos-Arias ; Christian Quesada-López ; Marcelo Jenkins ; Juan Murillo-Morera [All Authors](#)

[R](#) [Share](#) [©](#) [Print](#) [Alert](#)

---

**Abstract**

[Authors](#)

[Keywords](#)

**Abstract:**  
 Research on software effort estimation has investigated the impact of hyper-parameter tuning approaches for machine learning. Many automated tuning approaches have been proposed in the literature to improve the performance of their base models. In this study we compare the Dodge, Grid, Harmony, Tabu, and Random Search algorithms against Standard Genetic, 1+1 Genetic, and Compact Genetic Algorithms in terms of their impact to model accuracy and stability. This evaluation was performed using the ISBSG R18 dataset and on the Support Vector Regression, Classification and Regression Trees, and Ridge Regression techniques. Results of the Scott-Knott analysis show that the genetic algorithms perform similar or better to the other tuners, achieving an improvement of standardized accuracy of up to 0.21 and final values of up to 0.53. Genetic algorithm-tuned Support Vector Regression achieves the highest accuracy while improving estimation stability relative to other tuners.

**Published in:** 2021 16th Iberian Conference on Information Systems and Technologies (CISTI)

**Date of Conference:** 23-26 June 2021      **DOI:** 10.23919/CISTI52073.2021.9476459

**Date Added to IEEE Xplore:** 12 July 2021      **Publisher:** IEEE

► **ISBN Information:**      **Conference Location:** Chaves, Portugal

**Print on Demand(PoD) ISSN:** 2168-0727

[Sign in to Continue Reading](#)

---

[Authors](#)
▼

[Keywords](#)
▼

**More Like This**

Classification of power system stability using support vector machines  
 IEEE Power Engineering Society General Meeting, 2005  
 Published: 2005

Prediction of loadability margin of power system using Support Vector Machine  
 2013 International Conference on Energy Efficient Technologies for Sustainability  
 Published: 2013

[Show More](#)

# Ajuste Automático de Hiperparámetros mediante Algoritmos Genéticos para la Estimación de Esfuerzo

## *Hyper-parameter Tuning using Genetic Algorithms for Software Effort Estimation*

Leonardo Villalobos-Arias, Christian Quesada-López,  
Marcelo Jenkins  
Escuela de Ciencias de la Computación e Informática  
Universidad de Costa Rica, San Pedro, Costa Rica  
{leonardo.villalobosarias, cristian.quesadalopez, marce-  
lo.jenkins}@ucr.ac.cr

Juan Murillo-Morera  
Escuela de Informática  
Universidad Nacional de Costa Rica, Heredia, Costa Rica  
juan.murillo.morera@una.cr

*Resumen*—Estudios en el área de la estimación del esfuerzo del desarrollo de software han reportado el impacto del ajuste de hiperparámetros en los modelos basados en técnicas de aprendizaje automático. Múltiples algoritmos de ajuste han sido propuestos en la literatura con el objetivo de mejorar el desempeño de las estimaciones. En este estudio comparamos los algoritmos de ajuste de hiperparámetros Dodge, Grid, Harmony, Tabu y Random Search con el Genetic Standard, 1+1 y Compact para evaluar el impacto en la exactitud y la estabilidad de los modelos de estimación de esfuerzo. Realizamos la evaluación utilizando el conjunto de datos ISBSG R18 y ajustamos automáticamente los hiperparámetros de las técnicas Support Vector Regression, Classification and Regression Trees y Ridge Regression. Los resultados del análisis de Scott-Knott muestran que los algoritmos genéticos obtienen un desempeño superior o similar a los demás algoritmos de ajuste, incluyendo Grid Search. El ajuste de hiperparámetros alcanza un impacto de hasta un máximo de 0,21 en la exactitud estandarizada con valores finales de hasta 0,53. Los modelos de Support Vector Regression optimizados con algoritmos genéticos muestran la mayor exactitud y mejoran la estabilidad de las estimaciones con respecto a los demás algoritmos de ajuste.

*Palabras Clave* - optimización; hiperparámetros; algoritmos evolutivos; aprendizaje automático; estudio empírico.

*Abstract* — Research on software effort estimation has investigated the impact of hyper-parameter tuning approaches for machine learning. Many automated tuning approaches have been proposed in the literature to improve the performance of their base models. In this study we compare the Dodge, Grid, Harmony, Tabu, and Random Search algorithms against Standard Genetic, 1+1 Genetic, and Compact Genetic Algorithms in terms of their impact to model accuracy and stability. This evaluation was performed using the ISBSG R18 dataset and on the Support Vector Regression, Classification and Regression Trees, and Ridge Regression techniques. Results of the Scott-Knott analysis show that the genetic algorithms perform similar or better to the other tuners, achieving an improvement of standardized accuracy of up to 0,21 and final values of up to 0,53. Genetic algorithm-tuned Support Vector Regression achieves the highest accuracy while improving estimation stability relative to other tuners.

*Keywords* - optimization; hyper-parameters; evolutive algorithm; machine learning; empirical study.

### I. INTRODUCCIÓN

La estimación del esfuerzo del desarrollo de software es una tarea esencial para la administración de proyectos [11]. Distintas técnicas de aprendizaje automático han sido estudiadas para mejorar los procesos de estimación de esfuerzo [12]. El principal beneficio de estas técnicas es que pueden apoyar a los profesionales en sus procesos de toma de decisiones durante el desarrollo de los proyectos [13].

El ajuste automático de los hiperparámetros ha sido estudiado para evaluar el impacto en el desempeño de los modelos de estimación de esfuerzo, con el objetivo de mejorar su capacidad de predicción [13, 30]. Las técnicas de ajuste automático de hiperparámetros obtienen las configuraciones del modelo que mejoran la exactitud de las estimaciones, al hacer que el modelo se adapte mejor a los datos de entrada [13]. Las técnicas de ajuste incluyen búsquedas exhaustivas, aleatorias, genéticas y otras [15]. Particularmente, los algoritmos bioinspirados, principalmente los genéticos, se han utilizado en el área de estimación de esfuerzo [16, 32]. En trabajos previos hemos evaluado el impacto de hiperparámetros en modelos de aprendizaje automático [30, 31], y en este estudio extendemos este trabajo para incluir enfoques de ajuste evolutivos.

En este artículo, analizamos el impacto del ajuste automático de hiperparámetros mediante algoritmos genéticos en la exactitud y estabilidad de modelos de estimación de esfuerzo. Para esto comparamos los algoritmos de ajuste Genetic Standard, 1+1 y Compact con Dodge, Grid, Harmony, Tabu y Random Search. Realizamos la evaluación utilizando el conjunto de datos ISBSG R18 y ajustamos automáticamente los hiperparámetros de las técnicas Support Vector Regression, Classification and Regression Trees y Ridge Regression.

El reporte se estructura de la siguiente manera: la sección II presenta el trabajo relacionado, la sección III describe la metodología, el análisis de resultados se presenta en la sección IV y finalmente, la sección V presenta las conclusiones.

## II. TRABAJO RELACIONADO

Listamos estudios que utilizan técnicas bioinspiradas, principalmente algoritmos genéticos para mejorar los resultados de los modelos de estimación mediante el ajuste de hiperparámetros y la selección de atributos.

Ali et al. [16] presentan una revisión sistemática de literatura para identificar los diferentes algoritmos bioinspirados y su impacto en la parametrización en la estimación de esfuerzo. Los autores identificaron que los algoritmos genéticos y Particle Swarm Optimization son ampliamente utilizados para la parametrización de los modelos obteniéndose mejores resultados que las técnicas tradicionales. Del mismo modo, Villalobos-Arias et al. [32] identificaron que los enfoques genéticos son de los más utilizados para el ajuste en el área de estimación de esfuerzo.

Palaniswamy et al. [17] presentan un método para mejorar la estimación de esfuerzo utilizando el ajuste de hiperparámetros mediante el uso de métodos evolutivos, Particle Swarm Optimization y algoritmos genéticos.

Ali et al. [18], analiza la selección de atributos con el fin de determinar el impacto de cada uno de ellos en la exactitud de las estimaciones. Para esto utilizan algoritmos bioinspirados Genetic Algorithm, Particle Swarm Optimization, Ant Colony Optimization, Tabu Search, Harmony Search y Firefly. Los autores determinan que estos algoritmos obtienen mejores resultados que los tradicionales.

Oliveira et al. [4], investigan el uso de los algoritmos genéticos para seleccionar de forma óptima un subconjunto de características y optimizar los parámetros de los métodos de aprendizaje automático. Los resultados fueron comparados con las técnicas clásicas. Las simulaciones mostraron que el método basado en algoritmos genéticos incrementó el desempeño de los modelos de estimación. Además, identificaron que el uso combinado de características y parámetros mejora la exactitud de las estimaciones reduciendo la complejidad de los modelos.

A pesar que los estudios sobre algoritmos genéticos han mostrado mejoras considerables en los modelos, estos algoritmos no son normalmente comparados contra algoritmos de ajuste tradicionales. Nuestro estudio evalúa y compara algoritmos de ajuste de hiperparámetros con distintos algoritmos del estado del arte confirmando su potencial para la optimización de modelos de estimación para el conjunto de datos ISBSG R18.

## III. DISEÑO DEL ESTUDIO

El objetivo general del estudio fue evaluar el impacto del ajuste de hiperparámetros mediante algoritmos genéticos en la exactitud y estabilidad de modelos de estimación de esfuerzo. Para ellos comparamos los algoritmos de ajuste, aplicados a las técnicas de aprendizaje automático. Este estudio extiende nuestros trabajos previos [30, 31] evaluando los enfoques de ajuste evolutivos mencionados. Las preguntas de investigación son las siguientes:

- RQ1: ¿Cuál es el impacto del ajuste automático de hiperparámetros mediante algoritmos genéticos en la exactitud y estabilidad de los modelos de estimación?

- RQ2: ¿Cuál es la clasificación de modelos de estimación ajustados automáticamente mediante algoritmos genéticos basado en su desempeño?

### A. Conjunto de datos

Realizamos la evaluación utilizando el conjunto de datos del International Software Benchmarking Standards Group Development & Enhancement 2018 Repository Release 1 (ISBSG R18). El conjunto de datos fue pre procesado basado en los lineamientos descritos en [30, 31]. Seleccionamos proyectos (a) con calidad A o B y calidad de puntos de función A o B para IFPUG, (b) el esfuerzo registrado solo para el equipo de desarrollo, (c) nuevos desarrollos, y (d) medido con el método de medición IFPUG 4+ o COSMIC.

El conjunto de datos resultante fue dividido en dos subconjuntos, uno para cada método de medición. Además, para método generamos dos subconjuntos: uno considerando los puntos de función agregados y otro con los componentes funcionales básicos. Se utilizaron los cuatro subconjuntos: COSMIC FFP, COSMIC BFC e IFPUG UFP e IFPUG BFC.

La selección de atributos fue aplicada independientemente para cada subconjunto basado en los lineamientos descritos en [1] y las recomendaciones en [2]. Solo se utilizaron atributos (a) relevantes para la estimación de esfuerzo, (b) no redundantes con otros atributos, (c) están disponibles en los procesos de estimación, y (d) tienen menos de 25% de valores faltantes.

### B. Técnicas de ajuste y aprendizaje automático evaluadas

Los modelos de estimación de esfuerzo se construyeron combinando técnicas de pre procesamiento, selección de atributos y aprendizaje automático. Se combinaron la transformación logarítmica (DT) [20], las técnicas Support Vector Regression (SVR) [12], Classification and Regression Trees (CART) [21] y Ridge Regression (RR) [1]. Para el ajuste de hiperparámetros se utilizaron los algoritmos Genetic Standard (GA) [28], 1+1 (1+1GA) [26], Compact (CGA) [27], Dodge (DG) [14], Grid (GS) [24], Harmony (HS) [29], Tabu (TS) [20] y Random Search (RS) [25]. Los modelos se construyeron con los hiperparámetros por defecto y con cada uno de los algoritmos de ajuste.

Para RR se utilizaron los métodos de selección de atributos variance threshold (VT) [22] y correlation percentile (CP) [23]. Las técnicas SVR y CART fueron evaluadas con estos selectores, pero no fueron reportados al no presentar mejoras significativas. Un total de 90 modelos fueron comparados producto de 9 algoritmos de ajuste (GA, 1+1, CGA, DG, GS, HS, TS, RS, Default), 2 transformaciones, (Log, None), y 5 técnicas (SVR, CART, RR, VT+RR, CP+RR).

TABLE I. VALORES DE LOS HIPERPARÁMETROS

Técnica	Valores
SVR	kernel = {rbf, sigmoid}; $\gamma = (10^x, x = \{-3, -2.5, -2, \dots, -0.5\})$ ; auto, scale; C = 1, 5, 15, 30, {50, 100, ..., 450}, {500, 1000, ..., 15000}; $\epsilon = 10^x, x = \{-3, -2.5, -2, \dots, -0.5\}$ min_samples_leaf = {1, 2, 3, ..., 20};
CART	min_impurity_decrease = $(10^x, x = \{-5, -4.5, -4, \dots, 0\})$ ; max depth = {1, 2, 3, ..., 20}
RR, VT+RR, CP+RR	$\alpha = 1; \{5, 10, \dots, 45\}, \{50, 75, \dots, 500\}$ ; threshold = 0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5; percentile = {10, 20, 30, ..., 90}

### C. Valores de los hiperparámetros

Los valores de los hiperparámetros utilizados para las técnicas de aprendizaje automático se listan en la Tabla I. Los valores se basan en los reportados en estudios previos [4, 5, 30]. Los hiperparámetros default son los especificados por la librería scikit-learn, y son incluidos en el espacio de búsqueda de los algoritmos de ajuste. Los modelos default son utilizados como línea base de comparación para los modelos ajustados. Para los algoritmos de ajuste no aleatorios y no exhaustivos, el rango de valores usados para los valores numéricos corresponde al mínimo y máximo valor del espacio de búsqueda. En total, los enfoques exhaustivos como GS exploran 4.851 combinaciones de hiperparámetros para CART, 4.128 para SVR, 29 para RR, 232 para VT+RR, y 261 para CP+RR. En contraste, los otros algoritmos de ajuste fueron configurados para explorar un máximo de 60 valores. Los tiempos no se reportan, dado que los algoritmos corrieron en hardware diferente.

### D. Evaluación de los modelos de aprendizaje automático

Para evaluar los modelos utilizamos dos enfoques de validación. En el primero el hold-out set fue utilizado con 90% de los datos para entrenamiento y 10% para evaluación. En el segundo, con el 90% de los datos se construyen los modelos y se evalúa el rendimiento con un 10 times 10-fold Cross-Validation (CV) basado en las recomendaciones de [3] para reducir el problema de leave-one-out en conjuntos de datos pequeños. La métrica de exactitud fue calculada para cada fold. El conjunto de pruebas (10%) fue utilizado para corroborar los resultados reportados.

### E. Métricas de evaluación

La exactitud de los modelos de estimación se calcula utilizando métricas basadas en los residuos absolutos. Se calculan la exactitud estandarizada (SA) utilizando un modelo de línea base de estimación aleatoria (MARp0) y el promedio absoluto de residuos (MAR) [6, 9]. El impacto en la exactitud (con respecto a los parámetros por defecto) se calcula usando la ratio de mejora (IR) [7]. Para el análisis de estabilidad se utiliza la ratio de estabilidad (SR) [7] basado una métrica que denominamos estabilidad estandarizada (SS). Similar a SA, esta se calcula utilizando un modelo de línea base de estimación aleatoria (MARp0), pero con desviación estándar de los residuos absolutos (SdAR) de los modelos en lugar del promedio. MAR y SdAR se obtienen calculando el promedio y la desviación de los residuos absolutos.

## IV. RESULTADOS

La Tabla II muestra la mejora promedio en la exactitud y la estabilidad para los modelos con hiperparámetros ajustados con respecto a los modelos con valores por defecto. La escala de grises resalta las cuatro clasificaciones con mayor impacto de acuerdo al análisis de Scott-Knott [10]. El análisis utiliza un algoritmo de agrupamiento para particionar los tratamientos en grupos equivalentes. El análisis fue realizado por separado para las métricas de exactitud (SA) y estabilidad (SS) de los modelos. Los resultados de exactitud y estabilidad son presentados para cada modelo de estimación por subconjunto de datos, indicando técnica de ajuste y de aprendizaje automático. No se muestran los modelos RR porque fueron los que mostraron menor desempeño en la exactitud. Estos modelos presentan alta

estabilidad dado que el impacto del ajuste de hiperparámetros es poco.

### A. Impacto del ajuste de hiperparámetros en la exactitud y la estabilidad

*Exactitud.* Para los modelos ajustados con algoritmos genéticos se presentaron mejoras de hasta 0,21 en la SA con una mediana de 0,11. En particular, Genetic Standard obtuvo mejoras de hasta 0,21 con mediana de 0,14, 1+1 Genetic de hasta 0,21 con mediana de 0,10 y el Genetic Compact de hasta 0,21 con mediana de 0,11 en el SA. Los mayores impactos se obtuvieron en el conjunto de datos COSMIC BFC para el modelo Log+CART. En el conjunto de datos COSMIC FFP el impacto máximo fue de 0,19 con mediana de 0,07. En este caso el mejor modelo fue Log+SVR. En el conjunto de datos IFPUG BFC el impacto máximo fue de 0,14 con mediana de 0,11 en el modelo Log+CART. En el conjunto de datos IFPUG UFP el impacto máximo fue de 0,17 con mediana de 0,13 para el modelo Log+SVR. Los algoritmos genéticos alcanzan impactos similares a los obtenidos por Grid Search que realiza la exploración de todo el espacio de búsqueda. Asimismo, los algoritmos Dodge, Harmony, Tabu y Random Search alcanzan impactos similares. Esto comprueba la competitividad de los algoritmos genéticos con algoritmos exhaustivos y demás reportados en el estado del arte.

*Estabilidad.* Para los modelos ajustados con algoritmos genéticos se presentaron mejoras de hasta 0,23 en la estabilidad con una mediana de 0,10. En particular, Genetic Standard obtuvo mejoras de hasta 0,23 con mediana de 0,10, 1+1 Genetic de hasta 0,14 con mediana de 0,07 y el Genetic Compact de hasta 0,22 con mediana de 0,10 en la estabilidad. Los mayores impactos se obtuvieron en el conjunto de datos IFPUG UFP para el modelo Log+SVR. En el conjunto de datos COSMIC BFC el impacto máximo fue de 0,11, en los modelos CART y Log+CART, con mediana de 0,09. En el conjunto de datos COSMIC FFP el impacto máximo fue de 0,20 con mediana de 0,08. En este caso el modelo con mayor mejora en la estabilidad fue Log+SVR. En el conjunto de datos IFPUG BFC el impacto máximo fue de 0,13, en el modelo Log+CART, con mediana de 0,09. Para el análisis de estabilidad basado en Scott-Knott, los algoritmos Genetic Standard y Genetic Compact se clasifican en el grupo más estable (primer grupo) con respecto a la variación del error absoluto de las estimaciones para los modelos CART, Log+SVR, Log+CART y RR. El algoritmo 1+1 Genetic se clasificó en el segundo grupo para los mismos modelos. Todos los algoritmos de ajuste para RR clasificaron en el primer grupo, dado que RR presenta poco impacto por el ajuste de hiperparámetros.

En resumen, los algoritmos de ajuste automático mejoraron el desempeño de los modelos por defecto similarmente. Para el conjunto COSMIC BFC, todas las combinaciones de SVR y CART con Log y ajuste obtuvieron la precisión más alta (grupo 1). Para los demás conjuntos, el nivel de exactitud logrado por las técnicas fue aceptable (grupos 2-4). Las combinaciones de GA y CGA con Log+SVR obtuvieron el desempeño más alto (grupo 1), independientemente del conjunto. En todos los casos, las técnicas de ajuste mejoraron la estabilidad de CART y SVR, obteniendo rango 4 o superior en la mayoría de casos (103 de 128 modelos).

TABLE II. MEJORA PROMEDIO EN LA EXACTITUD Y LA ESTABILIDAD PARA LOS MODELOS DE LOS CONJUNTOS DE DATOS ESTUDIADOS

Ajuste	Modelo	Mejora Exactitud				Mejora Estabilidad			
		COSMIC		IFPUG		COSMIC		IFPUG	
		BFC	FFP	BFC	UFP	BFC	FFP	BFC	UFP
1+1	CART	0,19	0,07	0,10	0,13	0,10	0,01	0,06	0,12
1+1	Log+CART	0,21	0,09	0,12	0,13	0,10	0,07	0,08	0,14
1+1	Log+SVR	0,06	0,15	0,09	0,12	0,01	0,14	0,06	0,10
1+1	SVR	0,02	0,05	0,01	0,02	0,00	0,11	0,00	0,02
CGA	CART	0,18	0,07	0,11	0,15	0,11	0,08	0,10	0,16
CGA	Log+CART	0,21	0,06	0,14	0,14	0,09	0,04	0,13	0,17
CGA	Log+SVR	0,06	0,19	0,11	0,17	0,04	0,20	0,10	0,22
CGA	SVR	0,03	0,02	0,02	0,01	0,02	0,07	0,02	0,04
DG	CART	0,17	0,07	0,10	0,14	0,10	0,03	0,06	0,13
DG	Log+CART	0,19	0,07	0,14	0,12	0,12	0,02	0,11	0,13
DG	Log+SVR	0,05	0,15	0,09	0,14	0,02	0,13	0,07	0,12
DG	SVR	0,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00
GA	CART	0,18	0,10	0,11	0,14	0,11	0,04	0,09	0,17
GA	Log+CART	0,21	0,07	0,14	0,13	0,11	0,02	0,10	0,18
GA	Log+SVR	0,07	0,18	0,11	0,17	0,03	0,19	0,11	0,23
GA	SVR	0,04	0,06	0,03	0,03	0,01	0,10	0,03	0,04
GS	CART	0,15	0,12	0,10	0,11	0,07	0,07	0,05	0,10
GS	Log+CART	0,20	0,11	0,15	0,11	0,08	0,05	0,13	0,09
GS	Log+SVR	0,07	0,15	0,09	0,11	0,01	0,12	0,06	0,07
GS	SVR	0,05	0,07	0,04	0,08	0,00	0,09	0,03	0,11
HS	CART	0,18	0,10	0,11	0,13	0,08	0,05	0,08	0,11
HS	Log+CART	0,20	0,07	0,13	0,13	0,10	0,04	0,10	0,11
HS	Log+SVR	0,06	0,16	0,09	0,13	0,02	0,14	0,06	0,10
HS	SVR	0,04	0,03	0,02	0,03	0,01	0,02	0,01	0,02
RS	CART	0,17	0,09	0,09	0,13	0,10	0,07	0,05	0,10
RS	Log+CART	0,20	0,07	0,13	0,12	0,09	0,05	0,10	0,12
RS	Log+SVR	0,06	0,14	0,09	0,11	0,01	0,13	0,07	0,07
RS	SVR	0,04	0,07	0,04	0,08	0,00	0,13	0,03	0,10
TS	CART	0,18	0,08	0,12	0,13	0,10	0,02	0,06	0,13
TS	Log+CART	0,21	0,08	0,13	0,13	0,08	0,01	0,08	0,10
TS	Log+SVR	0,07	0,16	0,09	0,14	0,02	0,12	0,07	0,12
TS	SVR	0,00	0,11	0,04	0,09	0,00	0,16	0,03	0,13

### B. Clasificación de los modelos de estimación de esfuerzo

La Figura 1 muestra la distribución de la clasificación de cada modelo de estimación estudiado de acuerdo a la exactitud. Los modelos fueron agrupados aplicando el análisis Scott-Knott sobre los rangos de grupos de la Tabla II. Los rangos se normalizaron en una escala de [0, 1]. Para cada conjunto de datos, se transformaron los rangos utilizando la fórmula  $1 - (r_i - 1)/(R - 1)$  donde  $r_i$  es el  $i$ -enésimo rango y  $R$  es la cantidad de rangos del conjunto. Por ejemplo, para el subconjunto de datos COSMIC FFP el análisis identificó 7 grupos, el modelo CGA+Log+SVR con rango 1 tiene un puntaje normalizado de 1,0, el modelo TS+SVR de rango 4 un puntaje de 0,5, y la técnica SVR de rango 7 tiene un puntaje de 0,0. Las distribuciones mostradas corresponden al rango normalizado de cada modelo de estimación.

El análisis Scott-Knott identificó dos grupos. El primer grupo lista los modelos con mayor desempeño. Los tres modelos con mejor desempeño fueron los que utilizaron la combinación de las técnicas Log+SVR optimizados con los algoritmos de ajuste Genetic Standard, 1+1 y Compact Genetic. En total, nueve de los 25 modelos del primer grupo fueron los optimizados con algoritmos genéticos para las técnicas de aprendizaje automático SVR y CART. En este grupo, los algoritmos de ajuste genéticos, el Grid, Random, Dodge, Harmony y Tabu lograron un buen desempeño para la optimización de los modelos basados en Log+SVR, SVR, CART y Log+CART. Los modelos de estimación basados en Log+SVR presentaron la mejor exactitud y clasificación para los cuatro subconjuntos de datos COSMIC FFP y BFC e IFPUG UFP y BFC. En el caso de los modelos Log+SVR, Log+CART y CART presentaron la



mejor clasificación para el subconjunto COSMIC BFC. En el segundo grupo se incluyen todas las técnicas de RR, las SVR con ajuste automático (excepto TS), y los modelos Log+CART, Log+SVR con hiperparámetros por defecto.

Dentro del primer grupo, 2 modelos obtuvieron el máximo puntaje en los 4 conjuntos de datos: GA+Log+SVR y CGA+Log+SVR. La exactitud de las técnicas varió dependiendo del conjunto de datos desde 0,34 hasta 0,53 para el SA en GA+Log+SVR y desde 0,34 hasta 0,53 en CGA+Log+SVR. El siguiente conjunto de técnicas con la clasificación más alta correspondió al Log+SVR optimizado con los algoritmos 1+1, Dodge, Harmony y Tabu Search, obteniendo valores de SA por conjunto desde 0,25 hasta 0,50. La combinación de técnicas que incluye CART con mayor puntaje corresponde con CGA+Log+CART, con puntajes de SA desde 0,36 hasta 0,49. Las técnicas restantes de CART dentro del primer grupo incluyen todos modelos optimizados, con y sin Log, con valores de SA desde 0,34 hasta 0,49. Con respecto a SVR, las técnicas restantes del primero grupo corresponden con RS+Log+SVR, GS+Log+SVR, y TS+SVR, con valores de SA desde 0,30 hasta 0,47. En todos los casos, GA y CGA obtuvieron resultados similares, con menos de una diferencia de 0,05 SA. Para los modelos y datos estudiados, las técnicas podrían utilizarse de manera intercambiable.

Dentro del segundo grupo, los dos modelos con la media más alta son GS+SVR y RS+SVR. Estos modelos obtuvieron valores de SA desde 0,31 hasta 0,45. Con respecto a RR, el modelo con los puntajes más altos corresponde a TS+CP+RR, con valores de SA desde 0,26 hasta 0,43. El segundo grupo además contiene los modelos que utilizan parámetros por defecto. Los modelos SVR sin optimizar obtuvieron puntajes desde 0,27 hasta 0,37, los modelos CART sin optimizar, desde

0,12 hasta 0,35, y los modelos RR sin optimizar, desde 0,25 hasta 0,43.

Las mejores técnicas estudiadas incluyen los modelos Log+SVR, CART, y Log+CART optimizados. Para los modelos basados en regresión y SVR sin Log, el ajuste automático no logra una mejora en exactitud considerable. El impacto del ajuste de hiperparámetros depende la complejidad del modelo y sus parámetros: modelos más complejos como CART y SVR se ven más beneficiados que modelos como RR. La transformación Log además mejora el impacto del ajuste en CART y SVR.

A pesar que el análisis Scott-Knott determinó que las técnicas de ajuste tienen similar desempeño, hay una diferencia de consistencia en las técnicas. Particularmente, los algoritmos genéticos encontraron los valores de hiperparámetros que generan el modelo con más exactitud, independientemente del conjunto de datos. Las demás técnicas de optimización presentaban variaciones en su desempeño dependiendo de las cualidades de los datos. Una posible explicación en la diferencia de desempeño de los algoritmos de ajuste es la forma en la que estos exploran el espacio. Grid y Random están limitados en su espacio de búsqueda, al solo usar puntos dentro de los valores predeterminados. Los demás algoritmos, a pesar de tener mayor libertad en la búsqueda, podrían experimentar problemas de localidad. Dodge, Tabu, y Harmony toman un punto de exploración como base, y tratan de buscar valores cercanos que mejoren la exactitud del modelo. Por este motivo, es posible que estos algoritmos no encuentren los valores que causan mejor desempeño, al quedarse atascados en estos máximos locales. Los algoritmos genéticos evitan este problema al introducir la mutación aleatoria, que permite explorar puntos en el espacio que normalmente no serían considerados.

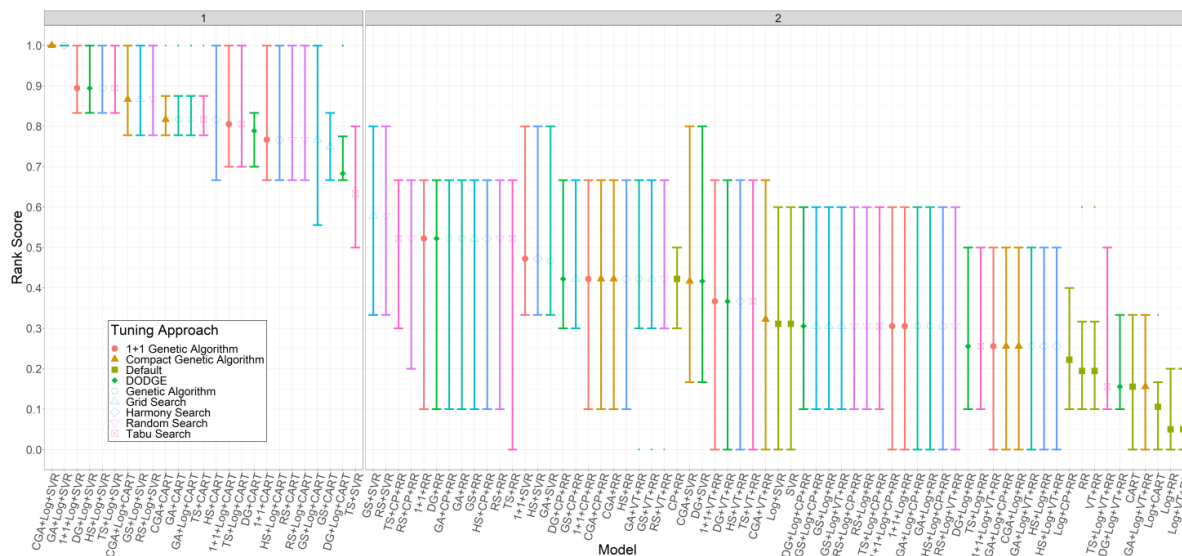


Figure 1. Clasificación basada en Scott-Knott para los modelos de los conjuntos de datos estudiados.

## V. CONCLUSIONES

En este artículo evaluamos el impacto del ajuste automático de hiperparámetros mediante los algoritmos Genetic Standard, 1+1 y Compact, que comparamos contra Dodge, Grid, Harmony, Tabu y Random Search con respecto a su impacto en la exactitud y estabilidad de modelos de estimación de esfuerzo.

Los resultados muestran que los algoritmos genéticos obtienen un desempeño superior o similar a los demás algoritmos de ajuste, incluyendo Grid Search, que realiza la exploración de todos los puntos del espacio de búsqueda predeterminado y que se utiliza como línea base. El ajuste de hiperparámetros alcanza un impacto de hasta un máximo de 0,21 en la exactitud estandarizada con valores finales de hasta 0,53. Los modelos de Support Vector Regression con logaritmo y optimizados con algoritmos genéticos muestran la mayor exactitud y estabilidad.

El trabajo futuro de este estudio incluye el análisis de los algoritmos de ajuste en otros conjuntos de datos comúnmente usados en el área de la estimación de esfuerzo y con otras técnicas de aprendizaje automático. Finalmente, es de interés realizar la evaluación del ajuste de los valores de los parámetros de los algoritmos de ajuste.

## AGRADECIMIENTOS

Este estudio fue apoyado por la Universidad de Costa Rica No. 834-B8-A27. Nuestro agradecimiento al Empirical Software Engineering Group (ESEG) de la Universidad de Costa Rica.

## REFERENCIAS

- [1] Dejaeger, K., Verbeke, W., Martens, D., Baesens, B.: Data mining techniques for software effort estimation: a comparative study. *IEEE transactions on software engineering* 38(2), 375-397 (2011).
- [2] Gonzalez-Ladron-de Guevara, F., Fernandez-Diego, M., Lokan, C.: The usage of isbsg data fields in software effort estimation: A systematic mapping study. *Journal of Systems and Software* 113, 188-215 (2016).
- [3] Song, L., Minku, L.L., Yao, X.: Software effort interval prediction via bayesian inference and synthetic bootstrap resampling. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28(1), 1-46 (2019).
- [4] Song, L., Minku, L.L., Yao, X.: The potential benefit of relevance vector machine to software effort estimation. In: *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, pp. 52-61 (2014).
- [5] Malgonde, O., Chari, K.: An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering* 24(2), 1017-1055 (2019).
- [6] Shepperd, M., MacDonell, S.: Evaluating prediction systems in software Project estimation. *Information and Software Technology* 54(8), 820-827 (2012).
- [7] Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering* 45(7), 683-711 (2018).
- [8] Minku, L. L. (2019). A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. *Empirical Software Engineering*, 24(5), 3153-3204.
- [9] Langdon, W.B., Dolado, J., Sarro, F., Harman, M.: Exact mean absolute error of baseline predictor, marp0. *Information and Software Technology* 73, 16-18 (2016).
- [10] Scott, A.J., Knott, M.: A cluster analysis method for grouping means in the analysis of variance. *Biometrics* pp. 507-512 (1974).
- [11] Boehm, B. W. Software engineering economics. *IEEE transactions on Software Engineering*, (1), 4-2 (1984).
- [12] Wen, J., Li, S., Lin, Z., Hu, Y., & Huang, C. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1), 41-59 (2012).
- [13] Minku, L. A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. *Empirical Software Engineering*, 24(5), 3153-3204 (2019).
- [14] Agrawal, A., Fu, W., Chen, D., Shen, X., & Menzies, T. How to "DODGE" Complex Software Analytics. *IEEE Transactions on Software Engineering* (2019).
- [15] Luo, G. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1), 18 (2016).
- [16] Ali, A., & Gravino, C. Using Bio-Inspired Features Selection Algorithms in Software Effort Estimation: A Systematic Literature Review. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 220-227). IEEE (2019).
- [17] Palaniswamy, S. K., & Venkatesan, R. Hyperparameters tuning of ensemble model for software effort estimation. *Journal of Ambient Intelligence and Humanized Computing*, 1-11 (2020).
- [18] Ali, A., & Gravino, C. Improving Software Effort Estimation using Bio-Inspired Algorithms to select relevant features: An Empirical Study. *Science of Computer Programming* (2021).
- [19] Oliveira, A. L., Braga, P. L., Lima, R. M., & Cornélio, M. L. GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *information and Software Technology*, 52(11), 1155-1166 (2010).
- [20] Corazza, A., Di Martino, S., Ferrucci, F., Gravino, C., Sarro, F., Mendes, E.: How effective is tabu search to configure support vector regression for effort estimation. In *Proceedings of the 6th international conference on predictive models in software engineering*, pp. 1-10 (2010).
- [21] Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: *Classification and regression trees*. CRC press (1984).
- [22] Albon, C.: *Machine learning with python cookbook: Practical solutions from preprocessing to deep learning*. " O'Reilly Media, Inc." (2018).
- [23] Shalev-Shwartz, S., Ben-David, S.: *Understanding machine learning: From theory to algorithms*. Cambridge university press (2014).
- [24] Bergstra, J.S., Bardenet, R., Bengio, Y., Kegl, B.: Algorithms for hyper-parameter optimization. In: *Advances in neural information processing systems*, pp. 2546-2554 (2011).
- [25] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. *Journal of machine learning research* 13(Feb), 281-305 (2012).
- [26] Doerr, B., Doerr, C., & Ebel, F. (2015). From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567, 87-104.
- [27] Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1999). The compact genetic algorithm. *IEEE transactions on evolutionary computation*, 3(4), 287-297.
- [28] Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing* (Vol. 53, p. 18). Berlin: Springer.
- [29] Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A new heuristic optimization algorithm: harmony search. *simulation*, 76(2), 60-68.
- [30] Villalobos-Arias, L., Quesada-López, C., Guevara-Coto, J., Martínez, A., & Jenkins, M. (2020, November). Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation. In *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering* (pp. 31-40).
- [31] Villalobos-Arias, L., Quesada-López, C., Martínez, A., & Jenkins, M. (2021, March). Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation. In *Proceedings of the 9th World Conference on Information Systems and Technologies*.
- [32] Villalobos-Arias, L., Quesada-López, C., Martínez, A., & Jenkins, M. (2021). Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: un mapeo de literatura. *Revista Ibérica de Sistemas e Tecnologías de Informação*, (E42), 305-318.

## Appendix G

# Paper 5: Multi-objective Hyper-parameter Tuning for Software Effort Estimation

**Reference** Villalobos-Arias, L., Quesada-López, C., Martínez, A., & Jenkins, M. (2021). Multi-objective Hyper-parameter Tuning for Software Effort Estimation. In Proceedings of the 24th Ibero-American Conference on Software Engineering. To be published.

**Status** Camera-ready version submitted. Awaiting publication.

# Multi-objective Hyper-parameter Tuning for Software Effort Estimation

Leonardo Villalobos-Arias, Christian Quesada-López, Alexandra Martinez, and  
Marcelo Jenkins

University of Costa Rica, San Pedro de Montes de Oca, Costa Rica  
{leonardo.villalobosarias, cristian.quesadalopez, alexandra.martinez,  
marcelo.jenkins}@ucr.ac.cr

**Abstract.** Studies of software effort estimation (SEE) have employed machine learning techniques to improve the accuracy of their predictions but concerns about the stability (and thus, replicability) of these models have been raised. In this study, we evaluate the impact of multi-objective hyper-parameter tuning on the accuracy and stability of five effort estimation algorithms: support vector regression (SVR), classification and regression trees (CART), and three versions of ridge regression (RR). Each model had two variants, for a total of 10: no data transformation and using logarithmic (Log). We compare the single- and multi-objective implementations of twelve hyper-parameter tuners, and evaluate their accuracy and stability on four subsets of the ISBSG R18 dataset. We evaluate the random search, flash, dodge, differential evolution, particle swarm optimization, tabu search, genetic algorithm (GA), one plus one GA, compact GA, bayesian optimization, harmony search, and hyperband tuners. The results indicated that, out of the 480 total (10 models  $\times$  12 tuners  $\times$  4 datasets) studied models, multi-objective tuning improved their stability on 159 cases (33%) and maintained it on 268 cases (56%), while improving accuracy on 60 cases (13%) and maintaining it on 247 (51%) cases. The effect of multi-objective tuning depended on the underlying model. For Log+SVR models, it was particularly effective, improving or maintaining accuracy in 65% of cases, and improving stability in 77% of cases. Multi-objective tuning also had a positive effect on CART and Log+CART models. In terms of accuracy, the most effective tuners were differential evolution, flash, genetic algorithm, and compact genetic algorithm when paired with Log+SVR.

**Keywords:** software effort estimation · hyper-parameter tuning · machine learning · multi-objective optimization · empirical study

## 1 Introduction

Software effort estimation (SEE), is an essential task of project management [6]. A key issue for software organizations is to accurately and consistently predict effort, as either overestimates and underestimates can result in missed business opportunities and threats for project development [14]. For these reasons, machine

learning techniques have been used in SEE to create higher accuracy estimation models [8]. Recent SEE studies apply hyper-parameter tuning (the process of automatically finding optimal model hyper-parameters) [2, 35] for building accurate models [36].

Machine learning research on software engineering had centered around improving the accuracy of the model [8], but it has recently shifted its focus to include additional model qualities such as stability [29]. Stability refers to an effect that holds for different situations [16]. In the context of SEE, a stable model should have similar accuracy when replicated or tested on different datasets. On the contrary, an unstable model may produce inconsistent results under different circumstances [17]. The introduction of hyper-parameter tuning techniques can affect the stability and accuracy of the SEE models [26]. Fortunately, current defect prediction studies [29] demonstrate that tuned models are as stable as untuned models.

Our previous work has focused on the evaluation of hyper-parameter tuning approaches with respect to their ability to improve accuracy and stability [31–33]. Such approaches seek to minimize the average prediction error, but overlook the importance of producing effort estimates that lie in a consistent margin of error [21]. By incorporating multi-objective optimization, a hyper-parameter tuner could accomplish both objectives simultaneously.

In this paper, we investigate multi-objective tuning by optimizing the hyper-parameter search for both model accuracy and stability. Our aim is to determine how multi-objective tuning compares to single-objective tuning. To accomplish this, we evaluated 14 single-objective hyper-parameter tuners (including default parameters) and 12 multi-objective hyper-parameter tuners, applied to five machine learning algorithms, two transformations, and four datasets. We extended the scope of our most recent work [33] by evaluating additional hyper-parameter tuning approaches and incorporating multi-objective tuning. This paper is organized as follows: Section 2 presents related work; Section 3 describes the study design; Section 4 shows the results; and section 5 outlines the conclusions.

## 2 Related Work

Langsari et al. [13] employed a fuzzy multi-objective particle swarm optimization approach to find the optimal parameter values for the COCOMO II model, and compared these results to a COCOMO II model with traditional values. Their adjusted COCOMO model performed more accurate estimations than the default model. Similarly, Azzeh et al. [5] investigated multi-objective particle swarm optimization to adjust the hyper-parameters of analogy-based estimation. Their results highlight the importance of simultaneously tuning all hyper-parameters and its noticeable impact to analogy based estimation. A study by Sarro et al. [21] introduced a dual-objective algorithm for effort estimation, which optimizes both the sum of absolute errors and the prediction confidence interval. They compared it with state of the art techniques. Their results show that the proposed approach outperforms state of the art techniques in terms of

standardized accuracy, and that multi-objective approaches can improve model performance.

Recent literature on software engineering has studied the effectiveness of heuristics-based fast tuners as an alternative to exhaustive search. Agrawal *et al.* [1, 2] proposed the dodge as a novel optimizer that searches for “interesting” hyper-parameter combinations and avoids redundant settings. Fu *et al.* [10, 9] analyzed the effectiveness of a differential evolution algorithm applied to the tuning of defect prediction models. In the area of configurable software systems, Nair *et al.* [18] used flash tuner, a method that determines the most promising exploration points based on a prediction model. Xia *et al.* [35] evaluated both flash tuner and differential evolution algorithm in the context of effort estimation.

Researchers have also studied bio-inspired tuning methods. Ali *et al.* [4] used genetic algorithms, particle swarm optimization, ant colony optimization, and tabu, harmony, and firefly searches for feature selection and tuning. The study by Palaniswamy *et al.* [20] used particle swarm optimization and genetic algorithms to improve the performance of ensemble learners. Oliviera *et al.* [19] employed genetic algorithms for simultaneous tuning of hyper-parameters and selection of features in SEE models.

This study further investigates the use of multi-objective optimization in SEE in conjunction with many of the hyper-parameter tuners used in the literature.

### 3 Study design

The main objective of this study was to evaluate the impact of multi-objective optimization hyper-parameter tuning on accuracy and stability, in the context of SEE models. The following research questions were proposed to guide the study:

- RQ1** What is the prediction accuracy and stability of single-objective hyper-parameter tuning?
- RQ2** Does multi-objective hyper-parameter tuning improve the prediction accuracy and stability?

Figure 1 shows the model training and evaluation process used in this study. Each task on the process corresponds with a section in the paper. The process is supported by our evaluation framework, which implements or adapts the machine learning techniques used in this study. This framework was built upon the scikit-learn library using Python 3.7, and supports three machine learning activities: (1) preprocessing, (2) training and evaluation, and (3) statistical analysis. Section 3.1 shows the dataset used in this study and preprocessing performed. Section 3.2 describes the process used to train and evaluate SEE models. Section 3.3 contains the studied algorithms that are combined to create SEE models, the evaluation metrics, and configurations for multi-objective optimization. The statistical analysis performed to the results of this study is detailed in section 4.

4 L. Villalobos-Arias et al.

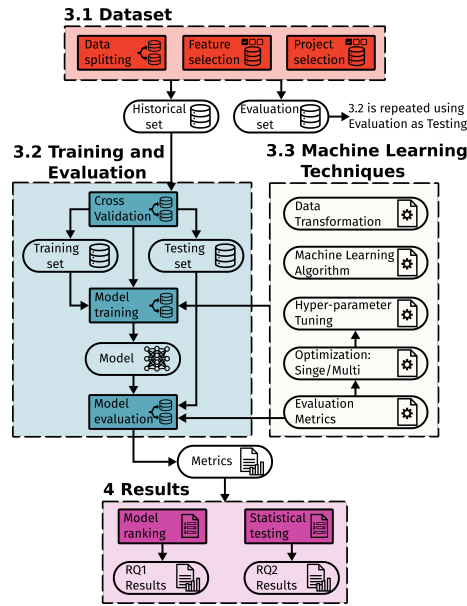


Fig. 1: Machine learning framework and process.

### 3.1 Dataset

We employed the International Software Benchmarking Standards Group Development & Enhancement 2018 Repository Release 1 (ISBSG R18) dataset. Based on previous research [8, 26, 32], we performed a three-step preprocessing that included data splitting, project selection, and feature selection.

**Data splitting** We splitted the data by their function point measurement unit, opting to use IFPUG 4+ and COSMIC. From these, we generated two datasets: one using total function points (FP), and another using the base functional components (BFC). This produced four datasets: IFPUG 4+ UFP (unified function points), IFPUG 4+ BFC, COSMIC FFP (full function points), and COSMIC BFC.

**Project selection** We selected projects that met these criteria: (a) data point quality A or B, (b) function point quality A or B if function point type is IFPUG 4+, (c) development type is “new development”, and (d) projects with recorded effort values for the development team. The reason being that we were interested in investigating effort estimation of new projects using reliable project data, based on literature recommendations [8, 26].

**Feature selection** We selected those features that are both available and pertinent to the effort estimation process [8, 11]. Besides functional size features,

the final feature set included Application Group, Architecture, Case Tool Used, Development Methodologies, Development Platform, Industry Sector, Intended Market, Language Type, Max Team Size, Team Size Group, Used Methodology, and Year of Project. We used Summary Work Effort as our target feature.

We randomly selected and removed 10% of the data in each dataset, to be used later in the evaluation (evaluation set). The remaining 90% of the data (historical set) was used in the training and evaluation activity. The evaluation set is used to review the metrics obtained in this study and discard problems such as overfitting and data mismatch.

### 3.2 Training and evaluation

We performed two evaluation runs using a set of 14 hyper-parameter tuners: one using single-objective optimization on prediction accuracy, and another using multi-objective optimization on prediction accuracy and stability. For the single-objective evaluation run, we used 5 machine learning algorithms, 2 transformation alternatives (log data transformation and no transformation), and 14 hyper-parameter tuners (including default parameters). A total of  $5 \times 2 \times 14 = 140$  machine learning models were constructed by combining these configurations. For the multi-objective evaluation run, we only used 12 hyper-parameter tuners, plus the same 5 algorithms and 2 transformations as before. This yielded  $5 \times 2 \times 12 = 120$  additional models. The two tuners excluded from the multi-objective run were default parameters and grid search.

We trained each model using 10 times 10-fold cross-validation, as suggested by previous research [27,28]. To do this, we partitioned the dataset into 10 random groups of even size. Then we chose one partition as test set and the remaining nine as training set, and repeated this until each partition had been used once as testing set. The entire process was repeated ten times, for a total of 100 runs. All 100 partitions were randomly created beforehand to ensure that the 260 models were run over the same data. Each model instance was trained using its corresponding training set. Each hyper-parameter tuning internally performed a sub-partition using 10-fold cross-validation to adjust find optimal values. The model is re-trained with the selected parameter(s) using the entire training set. The evaluation metrics (section 3.3) are calculated on every testing set for a total sample of 100 metrics per model and dataset.

### 3.3 Machine learning techniques

**Machine learning algorithms:** Five machine learning algorithms were used to predict software effort values: classification and regression trees (CART) [7], support vector regression (SVR) [34], and three variants of ridge regression (RR) as a base for comparison [8]. The RR variants used were: normal ridge regression, ridge regression with percentile correlation as feature selection (CP+RR) [23], and ridge regression with threshold variance as feature selection (VT+RR) [3]. Table 1 shows the hyper-parameter search space, selected based on literature suggestions [27, 15]. Feature selection was not used for CART and SVR models,



Table 1: Hyper-parameter search space.

Model	Hyper-parameters' values
CART	min_samples_leaf = {1, 2, 3, ..., 20}; max_depth = {1, 2, 3, ..., 20}; min_impurity_decrease = ( $10^x$ , $x = \{-5, -4.5, -4, \dots, 0\}$ )
SVR	$C = 1, 5, 15, 30, \{50, 100, \dots, 450\}, \{500, 1000, \dots, 15000\}$ ; kernel = {rbf, sigmoid}; $\gamma = (10^x, x = \{-3, -2.5, -2, \dots, -0.5\})$ ; auto, scale; $\epsilon = 10^x, x = \{-3, -2.5, -2, \dots, -0.5\}$
RR	$\alpha = 1, \{5, 10, \dots, 45\}, \{50, 75, \dots, 500\}$
VT+RR	threshold = 0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5; $\alpha$ from RR
CP+RR	percentile = {10, 20, 30, ... , 90}; $\alpha$ from RR

as we determined in previous iterations of this experiment that the two studied feature selectors did not affect the accuracy and stability of the CART and SVR models. For tuning techniques that explore ranges of values (i.e., GA and DODGE), the minimum and maximum values were used for numerical features. Two variations of each model were used: one with no data transformation, and one with the logarithmic (Log) transformation.

**Hyper-parameter tuners:** Our evaluation comprised 14 different tuners used in SEE literature [33] and other research areas [36]. We employed the scikit-learn implementation of default hyper-parameters, random search (RS), and grid search (GS) as performance baseline. We also evaluated the flash (FS), dodge (DG), and differential evolution (DE) tuners, since recent SE literature has endorsed the use of such fast and effective optimizers [9, 18, 2]. As traditional tuners from the SEE literature, we selected particle swarm optimization (PSO), tabu search (TS), and three variants of genetic algorithms: traditional genetic (GA), one plus one (1+1), and compact genetic algorithm (CGA). We also evaluated algorithms used in other areas: Bayesian optimization (BO), harmony search (HS) and hyperband (HB). For each of the 14 methods, two variants were run: a single-objective variant (optimizing for standardized accuracy) and a multi-objective variant (optimizing for hyperspace). Grid search was not run using multi-objective optimization as it was too computer-intensive.

**Evaluation metrics:** The **accuracy** of the SEE models was measured using absolute residual metrics (based on the absolute difference between predicted and actual values): the median absolute residual ( $MdAR$ ) and the standardized accuracy ( $SA$ ) [25]. To measure the improvement on accuracy (with respect to default parameters), we used the improvement ratio ( $imp$ ) [29]. On the other hand, the **stability** of the tuners was measured by the stability ratio [29] based on the standard deviation of the absolute residual (SdAR), and by the standardized stability ( $SD$ ).

$MdAR$  and  $SdAR$  correspond to the median and standard deviation of the absolute residuals, respectively.  $SA$  is computed as proposed by Shepperd and MacDonell [24], using the random estimation baseline model  $MdAR_{p0}$  recommended by Langdon *et al.* [12]. Additionally, we included our own metric  $SD$ , similar to  $SA$ , for which the  $SdAR$  is normalized using the standard deviation of random estimation baseline model,  $SdAR_{p0}$ .  $SD$  is calculated as  $1 - SdAR/SdAR_{p0}$ .

In order to calculate the metrics for the multi-objective approaches, we computed the average of the points in the Pareto front, as done in previous studies [21, 30] to compare single- and multi-objective methods. In real SEE applications, one of these points would be selected based on certain preference or decision. However, for the purpose of this study we wanted to consider all possible points and, to not favor specific solutions, we used the mean aggregated metrics.

**Multi-objective optimization** Implementations of hyper-parameter tuning algorithms are traditionally designed to (1) optimize only one metric and (2) return the best solution –usually one point in the search space. Thus, our tuners had to be adapted for multi-objective optimization. First, a composite metric was used to weight in both stability and accuracy. Second, the Pareto front of the search space was employed.

*Multi-objective function:* Random and grid search hyper-parameter tuners evaluate a set of points from the search space, either by random sampling or considering all possibilities. However, most of the hyper-parameter tuners in this study take a heuristic approach: they determine what is a ‘good’ (or ‘bad’) point in the search space, and select the next portion of the space to explore based on this information. A function ( $SA$  in our case) is then used to score each explored point in the space. For multi-objective tuning, the tuners maximized the function  $(SA + 1) \times (SD + 1)$ .

*Evaluation of multiple solutions:* The Pareto front of the searched values was identified. This set consists of all non-dominant solutions. A solution  $p_1$  is said to dominate another solution  $p_2$  if  $p_1$  is equally good or better than  $p_2$  for every metric. Conversely, a solution  $p_2$  cannot be dominated by another solution  $p_1$  if either  $p_2$  has the same values as  $p_1$  for each metric, or  $p_2$  has at least one metric better than  $p_1$ . Thus, all points in the Pareto front offer different but equally viable trade-offs between accuracy and stability.

## 4 Results

### 4.1 Accuracy and stability of single-objective tuning (RQ1)

In order to rank the studied SEE models, we used the Scott-Knott algorithm [22], which employs hierarchical clustering to partition treatments into equivalent

groups. This analysis was performed twice for each dataset: one for *SA* and one for *SD*. The Box-Cox transformation was applied to these metrics before the analysis, as Scott-Knott requires its input to follow a normal distribution. Ridge regression techniques were excluded from the analysis due to their low accuracy. Even hyper-parameter tuning did not improve the accuracy of ridge models. We refer to the ranks determined by the Scott-Knott analysis as dataset ranks.

A second Scott-Knott analysis grouped the techniques based on their overall performance. This analysis was applied on the ranks obtained from the first analysis, normalized to the  $[0, 1]$  scale. For this, we used the transformation  $1 - (r_i - 1)/(R - 1)$ , where  $r_i$  is the rank of the  $i$ -th model and  $R$  is the amount of ranks. We refer to the ranks determined by the second Scott-Knott analysis as general ranks. Figure 2 shows the distribution of dataset ranks for each technique, grouped by their general rank.

The analysis on *SA* determined four general groups: the first group comprises tuned Log+SVR models, except traditional RS and GS tuners; the second group contains all tuned CART and Log+CART models as well as GS+Log+SVR and RS+Log+SVR; the third group has all tuned SVR models except those with CGA and DG; the fourth group contains the four untuned models, plus CGA+SVR and DG+SVR. Four models obtained the first dataset rank in all cases: CGA+Log+SVR, DE+Log+SVR, FS+Log+SVR, and GA+Log+SVR.

When performed on *SD*, the Scott-Knott analysis determined three general groups: the first group contains all tuned CART, Log+CART, and (CGA, DE, FS, GA, DG, DE, and GA)+Log+SVR; the second group includes the remaining tuned Log+SVR models, TS+SVR, RS+SVR, PSO+SVR, GS+SVR, and CGA+SVR; the third group comprises the remaining tuned SVR models and the four untuned models. Within the first group, four models obtained the first dataset ranking in all except IFPUG 4+ BFC, where they obtained the second ranking. These models are CGA+Log+SVR, DE+Log+SVR, FS+Log+SVR, and GA+Log+SVR.

From these groups, it can be observed that hyper-parameter tuning improved both the accuracy and the stability of CART, Log+CART, SVR, and Log+SVR models. For SVR, only the TS, PSO, GS and RS tuners improved both qualities. For CART and Log+CART models, all tuners provided similar improvements in both accuracy and stability. For SVR and Log+SVR, the choice of tuner affected the outcome, as demonstrated by the general groups. Overall, SVR benefited the most from DE tuning, and Log+SVR from DE, GA, CGA, and FS.

Table 2 shows the *SA* and *SD* scores, and highlights the most accurate models (with database rank 1 and 2). Log+SVR is shown as a reference point for untuned models. Hyper-parameter tuning provided significant improvements in mean *SA* in the top models with respect to its default counterpart. The gain in accuracy was at its peak in FFP datasets, ranging from 0.139 to 0.2 *SA* in COSMIC FFP, and from 0.123 to 0.168 *SA* in IFPUG 4+ UFP. Tuning in BFC datasets was less impactful, ranging from 0.089 to 0.116 *SA* in IFPUG 4+ BFC, and from 0.055 to 0.075 *SA* in COSMIC BFC. A similar pattern occurs in *SD*, where tuning had a larger improvement in stability on IFPUG 4+ UFP, ranging

Table 2: Mean  $SA$ ,  $SD$ , and dataset rank of the most accurate models.

Model	SA				SD			
	IFPUG 4+		COSMIC		IFPUG 4+		COSMIC	
	FFP	BFC	FFP	BFC	FFP	BFC	FFP	BFC
CGA+Log+SVR	0.534	0.481	0.479	0.334	0.426	0.313	0.408	0.255
DE+Log+SVR	0.534	0.476	0.461	0.346	0.430	0.311	0.394	0.240
FS+Log+SVR	0.537	0.484	0.469	0.347	0.450	0.321	0.416	0.239
GA+Log+SVR	0.532	0.480	0.459	0.338	0.440	0.312	0.410	0.237
BO+Log+SVR	0.502	0.463	0.424	0.346	0.318	0.281	0.353	0.222
HB+Log+SVR	0.498	0.463	0.431	0.347	0.308	0.281	0.354	0.235
HS+Log+SVR	0.492	0.458	0.439	0.349	0.312	0.279	0.351	0.232
DG+Log+SVR	0.503	0.464	0.424	0.346	0.329	0.286	0.330	0.248
1+1+Log+SVR	0.496	0.459	0.418	0.344	0.315	0.282	0.365	0.231
PSO+Log+SVR	0.501	0.458	0.425	0.329	0.318	0.275	0.347	0.227
TS+Log+SVR	0.505	0.457	0.426	0.345	0.334	0.279	0.341	0.235
Log+SVR	0.369	0.368	0.279	0.274	0.216	0.222	0.209	0.212
			Rank 1	Rank 2	Rank 3+			

from 0.092 to 0.234  $SD$ , and COSMIC FFP, ranging from 0.121 to 0.207  $SD$ . In contrast, the effect was lower on IFPUG 4+ BFC, ranging from 0.053 to 0.099  $SD$ , and COSMIC BFC, ranging from 0.01 to 0.043  $SD$ .

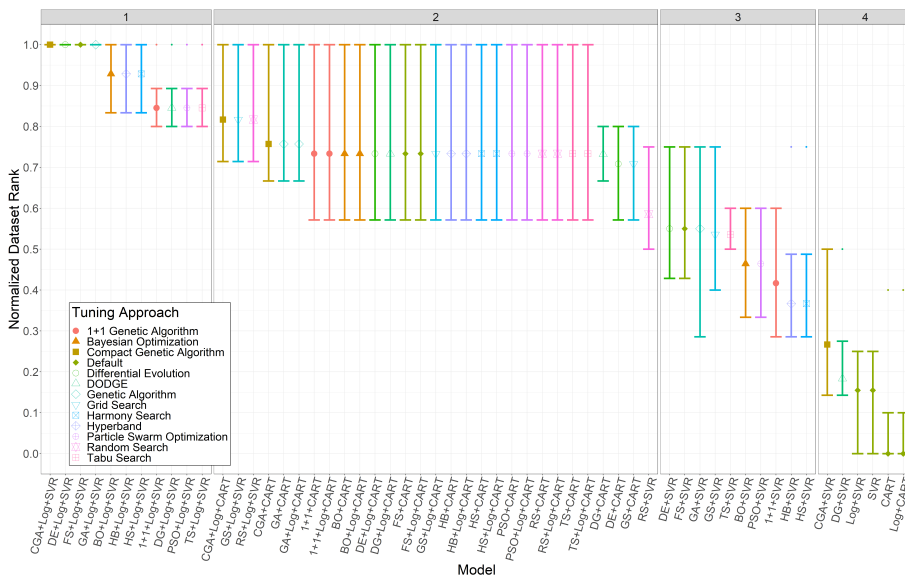
Single-objective hyper-parameter tuning improved the accuracy and stability of CART and SVR based models, but had no effect on RR. In both accuracy and stability, the models with the highest metrics were Log+SVR tuned with differential evolution, flash, genetic algorithm, or compact genetic algorithm.

## 4.2 Accuracy and stability of multi-objective tuning (RQ2)

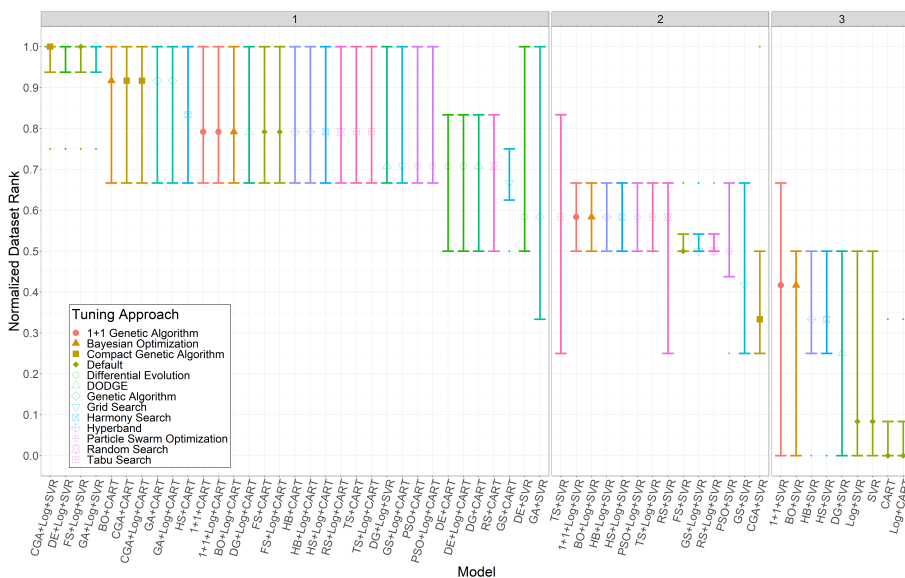
Comparing the performance of single- and multi- objective tuning was done by means of Wilcoxon signed-rank tests (at 0.05 significance level). These tests were performed for the two metrics  $SA$  and  $SD$ . Separate tests were done for each of the four datasets and 120 models, yielding a total of 480 tests for each metric.

Table 3 shows the win-tie-loss statistics on the results of the Wilcoxon signed-rank test, grouped by (a) tuner and (b) machine learning technique. These statistics are calculated by counting the results of each test: a *win* if the test determines a difference ( $p < 0.05$ ) and multi-objective has a higher metric, a *loss* if the test determines a difference and single-objective has a higher metric, and a *tie* if the test determines no difference ( $p \geq 0.05$ ). In total, multi-objective tuning scored 60 wins, 247 ties, and 173 losses in  $SA$ , and 159 wins, 268 ties, and 53 losses on  $SD$ .

All the multi-objective tuners had similar statistics. Regarding accuracy, the tuners maintained accuracy in 17 to 27 cases (out of 50 cases), and were able to improve it in 1 to 10 cases. However, they incurred in loss of accuracy in 9



(a) Rank on SA.



(b) Rank on SD.

Fig. 2: Distribution of dataset ranks for each SVR and CART model.

Table 3: Win-tie-loss statistics for multi-objective vs. single objective tuning.  
 (a) Hyper-parameter tuning (b) Machine learning technique

Tuner	SA			SD			Model	SA			SD		
	W	T	L	W	T	L		W	T	L	W	T	L
1+1	6	25	9	20	18	2	SVR	2	14	32	29	15	4
BO	9	21	10	13	22	5	Log+SVR	21	10	17	37	6	5
CGA	1	19	20	11	27	2	CART	4	44	0	8	37	3
DE	1	21	18	6	29	5	Log+CART	6	40	2	10	38	0
DG	4	17	19	12	21	7	RR	9	25	14	7	33	8
FS	1	27	12	12	21	7	Log+RR	0	23	25	15	24	9
GA	1	17	22	10	27	3	CP+RR	2	31	15	10	34	4
HB	6	18	16	18	18	4	Log+CP+RR	2	13	33	12	29	7
HS	9	20	11	14	21	5	VT+RR	10	29	9	15	26	7
PSO	5	20	15	13	21	6	Log+VT+RR	4	18	26	16	26	6
RS	10	21	9	16	21	3							
TS	7	21	12	14	22	4							
	60	247	173	159	268	53		60	247	173	159	268	53

to 22 cases (44% of cases in the worst scenario). The four tuners with highest accuracy from section 4.1 were those with the least improvement on accuracy. The CGA, DE, FS, and GA tuners registered only one win and up to 22 losses. Regarding their accuracy, each tuner improved in 6 to 20 cases, maintained it in 18 to 29 cases, and reduced it in 2 to 7 cases. Similarly, CGA, DE, FS, GA, and DG tuners obtained 12 or less wins. In general, tuners that were already good at optimizing models gained less from multi-objective optimization.

A difference in multi-objective performance is seen when analyzing the statistics by model. Multi-objective tuning has its largest effect on the Log+SVR model, with 21 improvements on accuracy and 37 improvements on stability, from the total 48 cases. However, there is potential hazard in 17 cases where multi-objective optimization resulted in loss. When applied to Log+SVR, the 1+1 and TS tuners scored 3 wins and 1 tie in accuracy, and 4 wins in stability, out of 4 cases. Similarly, the BO, HB, HS, PSO, and RS tuners obtained 4 wins for stability, but resulted in loss of accuracy in certain scenarios. GA, CGA, DG, and FS obtained the least gains from multi-objective optimization when applied to Log+SVR. We again observe that the best single-objective performers gained less from multi-objective optimization.

In the case of CART and Log+CART models, multi-objective optimization achieved more stable results. For both models, the tuners achieved 10 wins, 84 ties, and 2 losses on *SA*, and 18 wins, 75 ties, and 3 losses on *SD*. Although not particularly effective in improving either accuracy or stability, applying multi-objective tuning can be a net positive, as it would rarely result on either metric lowering and can potentially improve both metrics in certain cases.

The six RR and SVR model did not particularly benefit from multi-objective tuning, as accuracy was reduced in a maximum of 33 cases. With the exception of VT+RR, the amount of losses in *SA* outnumbered the amount of wins. The inverse can be observed for *SD*, with as high as 29 wins for multi-objective tuners. However, given that SVR and RR-based models had already some of the lowest accuracy scores in this study, we do not recommend multi-objective tuning for these models.

The metrics we obtained for the multi-objective tuners was an average of the Pareto front. In a real application, one of the hyper-parameter solutions in the front would be selected to construct the final model. Thus, multi-objective optimization has the aggregated value of producing multiple equally viable solutions.

Multi-objective hyper-parameter tuning had different effects depending on the model. For Log+SVR, tuning can boost stability at a potential loss of accuracy. In CART and Log+CART models, multi-objective tuning maintained or improved the accuracy and stability, except in 5 cases. Multi-objective tuning can also improve the stability of SVR and RR models, but it can lower the accuracy of the model.

## 5 Conclusion

In this study we evaluated the impact of multi-objective hyper-parameter tuning on accuracy and stability. Twelve single-objective and multi-objective tuners were compared over 5 machine learning algorithms, 2 data transformations, and 4 datasets.

The results of the study show that single-objective hyper-parameter tuning improves both the accuracy and stability on CART and SVR models, but had no effect on the studied ridge regression models. The most effective models, in both terms of stability and accuracy, were combinations of Log+SVR tuned using genetic algorithm, compact genetic algorithm, differential evolution, and flash. Multi-objective optimization offers a trade-off between accuracy and stability depending on the model. For Log+SVR, multi-objective tuning only improved the performance of the non-highest ranking tuners, such as random search. For models that use CART, multi-objective tuning resulted in improved or maintained accuracy and stability, with very few cases in which the multi-objective model performed worse. For SVR and ridge regression models, multi-objective tuning may provide improvements on stability, but at the potential cost of degrading their already low accuracy. Multi-objective optimization does offer multiple solutions that are equally viable, and a researcher or practitioner can select the hyper-parameter values that better suit their needs.

Future work includes exploring more datasets in the SEE literature, as well as replicating this study on further machine learning algorithms and data transformation methods. We are also interested in studying the effect of ‘meta-tuning’ the hyper-parameter tuners. For example, study how does a genetic algorithm tuner perform when parameters such as mutation and crossover rates are altered.

## References

1. Agrawal, A., Fu, W., Chen, D., Shen, X., Menzies, T.: How to “dodge” complex software analytics. *IEEE Transactions on Software Engineering* (2019)
2. Agrawal, A., Yang, X., Agrawal, R., Shen, X., Menzies, T.: Simpler hyperparameter optimization for software analytics: Why, how, when? *arXiv preprint arXiv:2008.07334* (2020)
3. Albon, C.: *Machine learning with python cookbook: Practical solutions from pre-processing to deep learning.* ” O’Reilly Media, Inc.” (2018)
4. Ali, A., Gravino, C.: Improving software effort estimation using bio-inspired algorithms to select relevant features: An empirical study. *Science of Computer Programming* **205**, 102621 (2021)
5. Azzeh, M., Nassif, A.B., Banitaan, S., Almasalha, F.: Pareto efficient multi-objective optimization for local tuning of analogy-based estimation. *Neural Computing and Applications* **27**(8), 2241–2265 (2016)
6. Boehm, B.W.: Software engineering economics. *IEEE transactions on Software Engineering* (1), 4–21 (1984)
7. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: *Classification and regression trees.* CRC press (1984)
8. Dejaeger, K., Verbeke, W., Martens, D., Baesens, B.: Data mining techniques for software effort estimation: a comparative study. *IEEE transactions on software engineering* **38**(2), 375–397 (2011)
9. Fu, W., Menzies, T., Shen, X.: Tuning for software analytics: Is it really necessary? *Information and Software Technology* **76**, 135–146 (2016)
10. Fu, W., Nair, V., Menzies, T.: Why is differential evolution better than grid search for tuning defect predictors? *arXiv preprint arXiv:1609.02613* (2016)
11. González-Ladrón-de Guevara, F., Fernández-Diego, M., Lokan, C.: The usage of isbsg data fields in software effort estimation: A systematic mapping study. *Journal of Systems and Software* **113**, 188–215 (2016)
12. Langdon, W.B., Dolado, J., Sarro, F., Harman, M.: Exact mean absolute error of baseline predictor, marp0. *Information and Software Technology* **73**, 16–18 (2016)
13. Langsari, K., Sarno, R.: Optimizing effort and time parameters of cocomo ii estimation using fuzzy multi-objective pso. In: *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI).* pp. 1–6. *IEEE* (2017)
14. Lederer, A.L., Prasad, J.: Causes of inaccurate software development cost estimates. *Journal of systems and software* **31**(2), 125–134 (1995)
15. Malgonde, O., Chari, K.: An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering* **24**(2), 1017–1055 (2019)
16. Menzies, T., Shepperd, M.: *Special issue on repeatable results in software engineering prediction* (2012)
17. Mittas, N., Angelis, L.: Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Transactions on software engineering* **39**(4), 537–551 (2012)
18. Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S.: Finding faster configurations using flash. *IEEE Transactions on Software Engineering* **46**(7), 794–811 (2018)
19. Oliveira, A.L., Braga, P.L., Lima, R.M., Cornélio, M.L.: Ga-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *information and Software Technology* **52**(11), 1155–1166 (2010)



20. Palaniswamy, S.K., Venkatesan, R.: Hyperparameters tuning of ensemble model for software effort estimation. *Journal of Ambient Intelligence and Humanized Computing* pp. 1–11 (2020)
21. Sarro, F., Petrozziello, A., Harman, M.: Multi-objective software effort estimation. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). pp. 619–630. IEEE (2016)
22. Scott, A.J., Knott, M.: A cluster analysis method for grouping means in the analysis of variance. *Biometrics* pp. 507–512 (1974)
23. Shalev-Shwartz, S., Ben-David, S.: Understanding machine learning: From theory to algorithms. Cambridge university press (2014)
24. Shepperd, M.: Software project economics: a roadmap. In: 2007 Future of Software Engineering. pp. 304–315. IEEE Computer Society (2007)
25. Shepperd, M., MacDonell, S.: Evaluating prediction systems in software project estimation. *Information and Software Technology* **54**(8), 820–827 (2012)
26. Song, L., Minku, L.L., Yao, X.: The impact of parameter tuning on software effort estimation using learning machines. In: Proceedings of the 9th international conference on predictive models in software engineering. pp. 1–10 (2013)
27. Song, L., Minku, L.L., Yao, X.: The potential benefit of relevance vector machine to software effort estimation. In: Proceedings of the 10th International Conference on Predictive Models in Software Engineering. pp. 52–61 (2014)
28. Song, L., Minku, L.L., Yao, X.: Software effort interval prediction via bayesian inference and synthetic bootstrap resampling. *ACM Transactions on Software Engineering and Methodology (TOSEM)* **28**(1), 1–46 (2019)
29. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering* **45**(7), 683–711 (2018)
30. Tawosi, V., Sarro, F., Petrozziello, A., Harman, M.: Multi-objective software effort estimation: A replication study. *IEEE Transactions on Software Engineering* (2021)
31. Villalobos-Arias, L., Quesada-López, C., Guevara-Coto, J., Martínez, A., Jenkins, M.: Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation. In: Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'20). ACM (2020)
32. Villalobos-Arias, L., Quesada-López, C., Martínez, A., Jenkins, M.: Hyperparameter tuning of classification and regression trees for software effort estimation. In: Trends and Applications in Information Systems and Technologies: Volume 3 9. pp. 589–598. Springer International Publishing (2021)
33. Villalobos-Arias, L., Quesada-López, C., Martínez, A., Jenkins, M.: Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: un mapeo de literatura. *Revista Ibérica de Sistemas e Tecnologías de Informação (E42)*, 305–318 (2021)
34. Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C.: Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology* **54**(1), 41–59 (2012)
35. Xia, T., Krishna, R., Chen, J., Mathew, G., Shen, X., Menzies, T.: Hyperparameter optimization for effort estimation. arXiv preprint arXiv:1805.00336 (2018)
36. Yang, L., Shami, A.: On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* **415**, 295–316 (2020)

## Appendix H

# Paper 6: Comparative study of Random Search Hyper-Parameter Tuning for Software Effort Estimation

**Reference** Villalobos-Arias, L., Quesada-López, C. (2021). Comparative study of Random Search Hyper-Parameter Tuning for Software Effort Estimation. In Proceedings of the 17th International Conference on Predictable Models and Data Analytics in Software Engineering. To be published.

**Status** Camera-ready version submitted. Awaiting publication.

# Comparative study of Random Search Hyper-Parameter Tuning for Software Effort Estimation

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

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

## ABSTRACT

Empirical studies on software effort estimation have employed hyper-parameter tuning algorithms to improve model accuracy and stability. While these tuners can improve model performance, some might be overly complex or costly for the low dimensionality datasets used in SEE. In such cases a method like random search can potentially provide similar benefits as some of the existing tuners, with the advantage of using low amounts of resources and being simple to implement. In this study we evaluate the impact on model accuracy and stability of 12 state-of-the-art hyper-parameter tuning algorithms against random search, on 9 datasets of the PROMISE repository and 4 sub-datasets from the ISBSG R18 dataset. This study covers 2 traditional exhaustive tuners (grid and random searches), 6 bio-inspired algorithms, 2 heuristic tuners, and 3 model-based algorithms. The tuners are used to configure support vector regression, classification and regression trees, and ridge regression models. We aim to determine the techniques and datasets for which certain tuners were 1) more effective than default hyper-parameters, 2) more effective than random search, 3) which model(s) can be considered “the best” for which datasets. The results of this study show that hyper-parameter tuning was effective (increased accuracy and stability) in 862 (51%) of the 1,690 studied scenarios. The 12 state-of-the-art tuners were more effective than random search in 95 (6%) of the 1,560 studied (non-random search) scenarios. Although not effective in every dataset, the combination of flash tuning, logarithm transformation and support vector regression obtained top ranking in accuracy on the highest amount (8 out of 13) of datasets. Hyperband tuned ridge regression with logarithm transformation obtained top ranking in accuracy on the highest amount (10 out of 13) of datasets. We endorse the use of random search as a baseline for comparison for future studies that consider hyper-parameter tuning.

## CCS CONCEPTS

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

*PROMISE'21, August 19–20, 2021, Athens, Greece*

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

<https://doi.org/10.1145/1122445.1122456>

## KEYWORDS

hyper-parameter tuning, software effort estimation, machine learning, empirical study

## ACM Reference Format:

Leonardo Villalobos-Arias and Christian Quesada-López. 2021. Comparative study of Random Search Hyper-Parameter Tuning for Software Effort Estimation. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'21), August 19–20, 2021, Athens, Greece*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/1122445.1122456>

## 1 INTRODUCTION

Software effort estimation (SEE) is a software engineering activity that deals with the estimation of the effort, in terms of time required per person (person-hours), for the completion of a software engineering project [6]. An effort estimate is helpful for other activities in the early SE process, such as planning, budgeting, risk analysis, and improvement analysis [5]. For this reason, accurate and consistent effort estimates are a key issue for software development organizations, and over- and underestimates can result in respectively missed business opportunities or threats to project delivery [20].

Machine learning (ML) has been researched for many years now in SEE as a way to improve the accuracy of effort estimates [10]. To further bolster the prediction capacity of a ML model, researchers employ a series of techniques, including data pre-processing, feature and project selection, cross-validation, and hyper-parameter tuning. Previous research in SEE has identified that the selection of techniques affects model performance, but so far no study has found an “optimal” scheme. Song et al. [32] have determined one factor that can affect model accuracy is the configuration of hyper-parameters in the model. Moreover, a study by Fu et al. [14] shows evidence that tuning can change conclusions on which models are better than others.

Hyper-parameter tuning algorithms are a way to automatically determine appropriate hyper-parameter values [23, 32]. Research on search-based software engineering (SBSE) has determined that tuning can improve the accuracy of a machine learning scheme [25, 32], as well as making their results more stable [35, 36]. However, hyper-parameter tuning can be costly. An exhaustive tuner like grid search can explore thousands of parameter configurations, which proportionally elevates the computational cost. In a previous experiment, we determined that a simple random search algorithm that samples 60 hyper-parameter combinations can perform as well as an exhaustive grid search on more than 4,000 hyper-parameter values for SEE data [38]. Agrawal et al. [2] suggest that the complexity

of datasets in software engineering is related to the efficacy of certain tuning algorithms. Simple learners and tuners can be effective for optimization on datasets with low intrinsic dimensionality (i.e. datasets that can be compressed with little information loss). Because of this, random search may be an effective hyper-parameter tuning algorithm for SEE, while at the same time being simple to implement and quick to execute.

In previous research, we have evaluated 13 hyper-parameter tuning algorithms on the ISBSG 2018 dataset with respect to their improvements on classification and regression trees (CART), support vector regression (SVR), and ridge regression (RR) models in 4 sub-datasets from the ISBSG R18 repository [37, 38]. This paper extends our previous evaluation to 9 PROMISE SEE datasets, and focuses on the comparison of random search against the other 12 state-of-the-art tuners.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 describes the study design. Section 4 shows the results. Section 5 outlines the conclusions.

## 2 RELATED WORK

This section reviews some of the studies in SEE and related SE areas that use hyper-parameter tuning algorithms. Table 1 shows the tuners we encountered in this literature that we evaluate in this study, including the used library or repository. We include some additional tuners from a recent literature study by Yang et al. [43]. We categorized these tuners into 4 groups: exhaustive, model-based, heuristic, and bio-inspired. The 2 exhaustive tuners are simple methods that do not consider the qualities of the search space to determine the next exploration point. The 3 model-based tuners represent the explored points and their performance with a predictive model, which then are used to predict the most promising candidates for exploration. The 2 heuristic tuners are search-based methods that use a set of rules to determine, based on the results of previous exploration, the new point(s) in the search space to visit. The 6 bio-inspired algorithms follow behavior found in nature, such as natural selection, and implement it in the form of a search algorithm.

In software engineering literature, the predilection of many studies is to employ traditional, exhaustive tuning algorithms. Minku [25] proposed a method for online effort estimation based on grid search, which contemplates tuning of the base model as well as tuning of a clustering algorithm. Their results highlight that tuning improved the predictive accuracy of online SEE models. Ertuğrul et al. [13] evaluated grid search applied to 9 different SEE models, showing that CART achieved the lowest mean average error. Tantithamthavorn et al. [35, 36] researched the impact of grid search regarding its impact to accuracy, stability, and training time; as well as parameter sensitivity. Song et al. [32] evaluate the impact of hyper-parameter tuning by studying model performance under the best, the worst, and the default hyper-parameter values.

In SEE and other related research areas in software engineering, researchers have started to propose, adapt, and evaluate non-exhaustive, heuristic hyper-parameter tuners. These have demonstrated to be viable alternatives to the traditional exhaustive algorithms like grid search. Palaniswamy et al. [27] studied particle swarm optimization and genetic algorithms as hyper-parameter tuners to improve the performance of ensemble learners. The Dodge tuning algorithm was proposed by Agrawal et al. [1, 2]. Dodge prioritizes points in the search space by avoiding redundant configurations (i.e. searching near parameters that do not substantially improve accuracy) and endorses space around the best candidates. Xia et al. [41] perform a study on the prediction of software project health in which they evaluate the performance of differential evolution tuned CART against untuned models. Their results show that tuned CART can consistently achieve less than 10% prediction error. Nair et al. [26] propose using machine learning algorithms to perform hyper-parameter tuning by introducing their flash algorithm. Flash utilizes a regression tree model to represent the explored hyper-parameters and their accuracy, and using said model to predict which configuration candidate should be evaluated next. In the area of software performance, a study by Chen et al. [8] adapted the Bayesian optimization method and a predictive tree model for optimization of compiler flags.

This study further investigates the effect of these heuristic tuners in SEE, evaluating them in the PROMISE SEE datasets.

## 3 EXPERIMENTAL DESIGN

In this section, we detail the research questions, datasets, cross-validation, evaluation metrics, machine learning algorithms, and hyper-parameter tuners used in this study.

### 3.1 Research Questions

The main objective of this study is to evaluate random search hyper-parameter tuning against 12 state-of-the-art algorithms on their impact of accuracy and stability of SEE models. We propose the following research questions to guide this research:

**RQ1:** Which tuners improve model accuracy and maintain stability with respect to default hyper-parameters?

**RQ2:** Which tuners improve model accuracy and maintain stability with respect to random search?

**RQ3:** Which model(s) can be considered the best for which datasets?

The focus of these questions is on the effectiveness of the hyper-parameter tuning algorithms. While tuning can potentially improve a SEE model, it requires at substantially more time to find the optimal hyper-parameter values. For this to be worthwhile, we would expect, at least, a significant increase in prediction accuracy of the model. In addition, previous studies have reported that another important characteristic of prediction models in software engineering is stability [35, 36]. We thus define the concept of an effective tuner in this study as a method that can significant increase model accuracy while maintaining (or increasing) the model stability.

### 3.2 Datasets

We evaluate the hyper-parameter tuners in 9 datasets from the PROMISE (also known as SEACRAFT) public repository as well as 4 sub-datasets from the ISBSG release 2018 (ISBSG R18) repository.

<sup>1</sup><https://scikit-learn.org/stable/>

<sup>2</sup>[https://github.com/arenmax/effort\\_oil\\_2019](https://github.com/arenmax/effort_oil_2019)

<sup>3</sup><https://github.com/amritbhanu/Dodge>

<sup>4</sup><https://github.com/FacebookResearch/Nevergrad>

<sup>5</sup><https://100.github.io/Solid/>

<sup>6</sup><https://github.com/zygmuntz/hyperband>

<sup>7</sup><https://pypi.org/project/geneticalgorithm/>

**Table 1: Hyper-parameter tuners evaluated in this study.**

Tuner	Code	Ref	Type	Description	Implementation
Grid search	GS	[4]	Exhaustive	Evaluates each possible combinations of hyper-parameters values in the search space.	scikit-learn <sup>1</sup>
Random search	RS	[4]	Exhaustive	Randomly samples 60 combinations of hyper-parameter values from the search space.	scikit-learn <sup>1</sup>
Flash	FS	[26]	Model	Represents all explored solutions using a regression tree, and determines the best possible candidate by predicting their performance.	OIL <sup>2</sup>
Dodge	DG	[1]	Model	Dodge represents visited solutions using a tree, and prioritizes “interesting” points by avoiding solutions within $\epsilon$ performance of previous solutions.	Dodge <sup>3</sup>
Bayesian optimization	BO	[31]	Model	Represents all explored solutions using a Bayesian model, and determines the best possible candidate by predicting their performance.	Nevergrad <sup>4</sup>
Tabu search	TS	[16]	Heuristic	TS explores a potential solution and its neighbors using a series of transformations, and employs a taboo list to exclude recently visited solutions.	Solid <sup>5</sup>
Hyperband	HB	[22]	Heuristic	HB is based on the concept of halving a space, with each iteration discarding the worst half of possible solutions.	Hyperband <sup>6</sup>
Particle swarm optimization	PSO	[17]	Bio	PSO simulates the behavior of a flock of birds by having each individual congregate closer to the most fit individual each iteration.	Nevergrad <sup>4</sup>
Harmony search	HS	[15]	Bio	Mimicking the search for musical harmony, HS improvises a solution each iteration based off a list of the best harmonies encountered.	Solid <sup>5</sup>
Genetic algorithm	GA	[12]	Bio	Each iteration of the algorithm eliminates a percentage of the least fit individuals, and new members of the population are generated based off the survivors.	PyPi <sup>7</sup>
1+1 GA	1+1	[9]	Bio	A variant of GA in which only one individual survives and one is generated each iteration.	Nevergrad <sup>4</sup>
Compact GA	CGA	[18]	Bio	A variant of GA that represents the population as a probabilistic vector to reduce memory use.	Nevergrad <sup>4</sup>
Differential evolution	DE	[34]	Bio	DE keeps a frontier comprised by the most fit individuals, and new individuals are generated by randomly combining three members of the frontier.	OIL <sup>2</sup>

To measure the complexity of these datasets, we employed the *intrinsic dimensionality* measure and calculation algorithm by Levina et al. [21].

**3.2.1 PROMISE.** Table 2 shows the datasets used in this study, their number of projects, number of features, number of categorical features, names of predictor features, name of the target feature, and L1 and L2 intrinsic dimensionality. We use the 9 datasets and preprocessing rules used in the study by Xia et al. [42]. This selection comprises some of the most used datasets in hyper-parameter tuning studies for SEE [39]. In the same line of Xia et al., we have chosen to remove features that are 1) irrelevant to the effort estimation task, 2) unavailable at the time of estimation, and 3) highly correlated or overlap with each other. For desharnais, we removed 4 projects that had missing values: rows 38, 44, 66, and 75.

**3.2.2 ISBSG R18.** We extend the design of our previous studies [37, 38], in which we employed the International Software Benchmarking Standards Group Development & Enhancement 2018 Repository

Release 1 (ISBSG R18) dataset. Similarly to Xia et al. [42], we removed features that are 1) irrelevant to the effort estimation task, 2) unavailable at the time of estimation, and 3) highly correlated or overlap with each other, and 4) have 25% or more missing values. In addition, we selected only those projects that had: a) function point type IFPUG 4+ or COSMIC, b) data point quality A or B, c) function point quality A or B if function point type is IFPUG 4+, d) development type is “new development”, and e) projects with recorded effort values for the development team. Lastly, we splitted the dataset in 2 sub-sets by function point type: IFPUG 4+ or COSMIC. Two variants were studied for each set: using total function points, and using separate basic functional components. All datasets contain the following 13 features, plus their specific function point features: Application Group, Architecture, Case Tool Used (except COSMIC), Development Methodologies (except COSMIC), Development Platform, Industry Sector, Intended Market, Language Type, Max Team Size, Team Size Group, Used Methodology, and Year of Project. The target feature was Normalized Work Effort Level

Table 2: PROMISE datasets used in this study.

Dataset	Proj.	Features		Predictors	Target	Dim.	
		N	Cat.			L1	L2
albrecht	24	4	0	Input, Output, Inquiry, File	Effort	5.4	8.0
china	499	6	0	Input, Output, Enquiry, File, Interface, Resource	Effort	6.2	6.2
desharnais	77	5	2	TeamExp, ManagerExp, Transactions, Entities, PointsAdjust	Effort	3.4	5.4
finnish	38	4	0	hw, at, FP, co	dev.eff.hrs	5.4	5.4
isbsg10	37	11	10	Data_Quality, UFP, IS, DP, LT, PPL, CA, FS, RS, Recording_Method, FPS	s_effort	6.9	3.4
kemerer	15	3	0	AdjFP, Hardware, Language	EffortMM	5.4	3.4
kitchenham	145	3	1	Client.code, Project.type, Adjusted.function.points	Actual.effort	3.4	3.4
maxwell	62	23	0	App, Har, Db, Ifc, Source, Telonuse, Nlan, T01, T02, ..., T15, Size	Effort	3.8	3.4
miyazaki94	47	6	0	SCRN, FORM, FILE, ESCRN, EFORM, EFILE	Effort	5.4	5.4

Table 3: ISBSG R18 datasets used in this study.

Dataset	Proj.	Features		FP	Dim.	
		N	Cat.		L1	L2
ifpug_total	821	14	10	Functional Size	4.8	3.4
ifpug_bfc	821	18	10	Input, Output, Enquiry, File, Interface	5.4	5.4
cosmic_total	168	12	8	Functional Size	4.3	2.5
cosmic_bfc	168	15	8	Entry, Exit, Read, Write	3.4	5.4

1. Table 3 summarizes the four ISBSG subsets used in this study, including their function point type features.

### 3.3 Cross-validation

For PROMISE datasets, we used 100 times out-of-sample bootstrap sampling as a cross-validation method [11]. For a dataset of size  $N$ , an out-of-sample bootstrap takes  $N$  samples with replacement from the original data as the train set, and uses those projects not included as the test set. On average, the test set contains 36.8% of the original projects. This process is repeated 100 times, producing that many pairs of train and test sets. Out-of-sample bootstrap is used instead of  $k$ -fold cross-validation, as it generates more stable results when having less samples, as well as being recommended for highly skewed datasets [36].

Following our previous design, we employ 10 times 10-fold cross-validation for the ISBSG R18 datasets. We randomly divide the data into 10 groups or folds. One fold is selected to become a test set, and the remaining 9 are combined to become the train set; repeating until every fold has been used once as the test set. This process is repeated an additional 100 times, for a total of 100 pairs of train and test sets.

For each scheme (combination of machine learning algorithm, tuner, and data transformation), the models were tuned on the train set. Because the tuners do not have access to the test set, a sub-partition was performed using  $k$ -fold cross-validation. The value of  $k$  was 5 for PROMISE sets and 10 for ISBSG sets. Once the optimal

hyper-parameter values are determined, the model is re-trained with these parameters on the complete train set. Lastly, the model is used to predict on the test set. The predicted effort values are used, along with the actual effort, to compute the evaluation metrics, for a total of 100 metrics per scheme and dataset.

### 3.4 Evaluation Metrics

We measured the *accuracy* of the SEE models using the standardized accuracy (*SA*) [30]. *SA* is calculated as  $1 - MAR/MAR_{p_0}$ , where *MAR* is the mean of a between predicted and actual effort values.  $MAR_{p_0}$  is the *MAR* of a baseline model, for which we use the adjusted random guessing algorithm by Langdon *et al.* [19].

The *stability* of these models was measured using a metric we named standardized stability (*SD*), which is a modified version of the stability ratio by Tantithanthavorn *et al.* [35, 36] to function in the same scale as *SA*. Moreover, instead of comparing against default parameters, we calculate the stability ratio with respect to the baseline estimator  $p_0$ . The *SD* is calculated as  $1 - SdAR/SdAR_{p_0}$ , where *SdAR* is the standard deviation of the differences between predicted and actual effort values. We use the baseline model  $p_0$  for both *SA* and *SD*.

Both *SA* and *SD* are ratios that show the percentage of improvement in accuracy and stability with respect to the baseline. A *SA* value of 0 indicates that the model has performance that is as good as random guessing. A model that obtains an *SA* score of 1 is 100% more accurate than random guessing. Conversely, negative *SA* or *SD* scores indicate that the model is *worse* than randomly guessing.

### 3.5 Machine Learning Algorithms

We employed five machine learning models in this experiment: classification and regression trees (CART) [7], support vector regression (SVR) [40] and three variants of ridge regression (RR) [10]. For RR, we employed: ridge regression (no feature selection), ridge regression with percentile correlation [29] feature selection (CP+RR), and ridge regression with variance threshold [3] feature selection (VT+RR). Table 4 shows the hyper-parameter search space. Tuners that explore continuous ranges for numerical features (genetic algorithms, for example), instead use the minimum and maximum values. The hyper-parameter ranges and values were selected based on suggestions of previous studies [24, 33].



**Table 4: Hyper-parameter search space.**

Model	Hyper-parameters and values
CART	min_samples_leaf = {1, 2, 3, ..., 20}; max_depth = {1, 2, 3, ..., 20}; min_impurity_decrease = ( $10^x$ , $x = \{-5, -4.5, -4, \dots, 0\}$ )
SVR	$C = 1, 5, 15, 30, \{50, 100, \dots, 450\}, \{500, 1000, \dots, 15000\}$ ; kernel = {rbf, sigmoid}; $\gamma = (10^x, x = \{-3, -2.5, -2, \dots, -0.5\})$ ; auto, scale; $\epsilon = 10^x, x = \{-3, -2.5, -2, \dots, -0.5\}$
RR	$\alpha = 1, \{5, 10, \dots, 45\}, \{50, 75, \dots, 500\}$
VT+RR	threshold = 0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5; $\alpha$ from RR
CP+RR	percentile = {10, 20, 30, ..., 90}; $\alpha$ from RR

We employed the logarithm (Log) transformation in this experiment. The five machine learning models were evaluated using no data transformation, and using the Log transformation before training. For example, two variants of the SVR model were evaluated: no transformation SVR (SVR), and Log transformation plus SVR (Log+SVR). Thus in total, this experiment evaluated 10 models.

### 3.6 Hyper-parameter Tuners

We employ a total of 13 tuning techniques in this study, plus default hyper-parameters. These are the 13 algorithms presented on table 1. All tuners, with the exception of grid search, were configured to explore a maximum of 60 hyper-parameter combinations. Future studies could compare 60-sample random search against tuning algorithms with higher budgets. Flash was configured to have a population (sample) size of 2880 with an initial sample of 5. Dodge was configured with  $n_1 = 12$  and  $n_2 = 48$  (i.e. 60 iterations), and  $\epsilon = 0.01$ . Bayesian optimization was configured with value of  $\alpha = 0.0001$ . Tabu search was used with a tabu list size of 5, a neighborhood size of 5, and a maximum of 12 steps. For Hyperband,  $\eta$  was set to 3. PSO was configured with a population size of 20. For Harmony search, the parameters were configured as *memory\_size* = 10, *memory\_considering\_rate* = 9, *pitch\_adjustment\_rate* = 0.05, *fret\_width* = 0.01, and *max\_steps* = 50 (the implementation does not count the initial memory as part of *max\_steps*). The genetic algorithm was configured with 14 iterations and a population size of 5, a mutation rate of 0.3, and a crossover rate of 0.8, and a surviving population of 0.2 (1 individual). Because of this, GA explores a total of 61 hyper-parameters: 5 initial values plus 4 per iteration, times 14. Differential evolution uses a mutation rate of 0.8, crossover rate of 0.7, a population size of 6, and a total of 10 iterations. The 1+1 and compact genetic algorithms are used with parameters. Grid search and random search do not have parameters, excepting the exploration budget.

The (non grid search) hyper-parameter tuners used in this study were configured to explore a maximum of 60 values in the search space. This budget was chosen based on the study by Bergstra and Bengio [4], in which random search for hyper-parameter tuning obtained better results with respect to grid search when the exploration budget was 64 or more. According to Zheng [44], if 5% of

the hyper-parameter space contains near-optimal solutions, a random sample of 60 values has a 95% chance to contain one or more optimal parameters. However, the results presented in this study are limited to techniques configured to explore 60 hyper-parameter values. Additional studies are required to confirm the impact of tuning in stability and accuracy using different exploration budgets.

## 4 RESULTS

In this section we answer the research questions presented in section 3.1 by presenting the results obtained from our experiment. The obtained metrics and results of this experiment is publicly available in <http://tiny.cc/tuningvsrandom>. This paper presents a summary of these results.

### 4.1 RQ1: Tuning vs. Default

To answer the first research question on whether tuning can improve accuracy and maintain stability with respect to default hyper-parameters, we evaluated the improvement in performance provided by the 13 tuners. Specifically, we performed a Wilcoxon sign-rank test (0.05 significance) with Holm-Bonferroni corrections (for the total number of tuners) on the obtained SA and SD scores against default hyper-parameters for each model, dataset, and tuner; resulting in a total of 1,690 evaluations. These results are labeled as one of three cases: wins, where the tuner has a significantly *better* score than default; tie, where the tuned and default have *similar* scores; and loss, where the model has a significantly *worse* score than default. Because tuning requires additional resources, we would consider tuning effective if it *increases* accuracy (win in SA), and is able to *increase or maintain* stability (win or tie in SD). Otherwise, we consider tuning not effective. Table 5 shows for each tuner, model and dataset the amount of scenarios (i.e. combinations of the other two factors) in which tuning was effective, with the total amount of scenarios in the bottom row.

**Table 5: Effectiveness of tuning against default hyper-parameters.**

Tuner	Model	Dataset
GA	70	Log+SVR 120 cosmic total 130
HB	70	Log+RR 112 ifpug total 93
RS	70	Log+CP+RR 104 maxwell 90
1+1	69	CP+RR 93 miyazaki94 80
CGA	69	Log+VT+RR 88 kitchenham 74
FS	68	Log+CART 73 cosmic bfc 73
BO	67	VT+RR 71 desharnais 56
HS	66	RR 69 albrecht 54
GS	65	SVR 66 ifpug bfc 52
PSO	65	CART 66 kernerer 49
TS	63	china 46
DE	62	isbsg10 42
DG	58	finnish 23
	130	169 130

The studied tuners were effective in 862 (51%) out of the 1,690 evaluated scenarios. On the non-effective scenarios, 46 combinations of model and dataset resulted in no tuner being able to significantly increase accuracy. If we could always select an effective algorithm, tuning would be effective for 84 out of 130 model  $\times$  dataset combinations, or 65% of the studied cases. For the remaining 35% cases, none of the tuning approaches can improve upon default hyper-parameters.

Hyper-parameter tuning was effective in datasets with L1 dimensionality less than 5: cosmic total, maxwell, cosmic bfc, ifpug total, desharnais, and kitchenham. Particularly, tuning was effective in the cosmic total dataset regardless of the used tuner or the base model. The effectiveness of tuning in the other low-dimensionality datasets depends on particular combinations of base model and tuner. With the exception of miyazaki94, the amount of scenarios in which tuning is effective is less for datasets with dimensionality 5 or greater. This observation is consistent with a previous study of tuning in software engineering by Agrawal et al. [2], in which they report that dodge hyper-parameter tuning achieved better results in datasets with dimensionality 3 or less.

Out of the 10 studied models, tuning was more consistently effective for Log+SVR. The 13 studied tuners were effective in 9 of the studied datasets: albrecht, cosmic bfc, cosmic total, finnish, ifpug bfc, ifpug total, isbsg10, kitchenham, and maxwell. The 13 tuners failed to improve performance of Log+SVR in the miyazaki94 and kemerer datasets. Particularly for miyazaki94, tuning *decreased* the accuracy of the model with respect to default parameters. For the china dataset, only CGA was effective. For the desharnais dataset, both CGA and RS were effective.

There were 46 combinations of dataset and model (out of 130) in which none of the 13 tuners were effective. These included 7 datasets for RR and VT+RR (albrecht, china, cosmic bfc, finnish, ifpug bfc, isbsg10, kitchenham), 7 datasets for SVR (china, desharnais, isbsg10, kemerer, kitchenham, maxwell, miyazaki94), 5 datasets for Log+CART (albrecht, finnish, isbsg10, kemerer, miyazaki94), 4 datasets for Log+VT+RR (china, finnish, ifpug bfc, ifpug total), 4 datasets for CART (albrecht, finnish, isbsg10, kemerer), 4 datasets for CP+RR (desharnais, ifpug bfc, ifpug total, kemerer), 3 datasets for Log+RR (china, finnish, ifpug bfc), 3 datasets for Log+CP+RR (desharnais, finnish, ifpug bfc), and 2 datasets for Log+SVR (kemerer, miyazaki94). Moreover, there was no tuner and model combination that was effective across all datasets. Similarly, there were 53 such cases in which the 13 tuners were effective.

**RQ1:** there was no tuner that always improved model accuracy and maintained stability, and are effective depending on the base model and dataset. There is some overlap in the performance of tuners, as in 53 cases every tuner was effective, and in 46 cases every tuner was ineffective. In other words, the choice of tuner was relevant in only 31 model  $\times$  dataset combinations. Tuning was generally more effective in scenarios where the dataset had low dimensionality (<5).

## 4.2 RQ2: Tuning vs. Random

To answer the second research question on whether tuning can improve accuracy and maintain stability with respect to random search, we compared the performance of random search against

the other 12 studies tuners. We performed an analysis similar to RQ1, for which we obtained the win-tie-loss values of a Wilcoxon sign-rank test with Holm-Bonferroni corrections on the *SA* and *SD* scores of the 12 tuners compared to random search. Similarly, we considered the tuners *more effective* than random if they can *increase* accuracy, and *increase or maintain stability*. Table 6 shows for each tuner, model and dataset the amount of scenarios (i.e. combinations of the other two factors) in which tuning was effective, with the total amount of scenarios in the bottom row.

**Table 6: Effectiveness of tuning against random search.**

Tuner	Model	Dataset
CGA	21	Log+SVR 25 albrecht 21
GA	13	CP+RR 17 maxwell 19
TS	11	RR 13 desharnais 16
HB	10	SVR 10 ifpug total 14
HS	8	CART 8 cosmic total 13
FS	6	VT+RR 8 finnish 13
1+1	5	Log+CART 6 miyazaki94 12
DE	5	Log+RR 5 ifpug bfc 9
DG	5	Log+VT+RR 3 china 8
PSO	5	Log+CP+RR 0 kemerer 7
GS	4	isbsg10 5
BO	2	kitchenham 3
		cosmic bfc 1
	130	156 120

The studied tuners were more effective than random search in only 95 out of the 1,560 studied non-random search scenarios (12 tuners  $\times$  10 models  $\times$  13 datasets), or 6% of all cases. If we assume a practical scenario, in which we evaluated and select the most effective tuner, tuning is more effective than random search in 35 combinations of model and dataset out of 130 (27%). In the remaining 73% of combinations, random search is as effective (or more) than the other tuners.

The 4 tuners which outperformed random search in the highest amount of scenarios were CGA, GA, TS, and HB. The first two correspond with variants of genetic algorithms, and other three are heuristic-based methods.

Grid search, one of the most used tuners in SEE literature, was more effective than random search in only 4 (3%) of all scenarios. Three of these cases were on the ISBSG R18 datasets. Many of the studies that utilize Grid Search on the traditional SEE datasets could switch to random search and achieve similar results. Moreover, the best tuners were more effective than random search in only 17% of all cases.

**RQ2:** there was no tuner that could outperform random search in all scenarios. The studied hyper-parameter methods were more effective than random search in only 6% of the studied model and dataset combinations. This shows that random search is a viable tuning algorithm for SEE datasets.



### 4.3 RQ3: Best Tuners

To answer the third research question on whether a particular model and tuner combination can be considered the overall “best” technique in all of the studied datasets. To achieve this, we ranked the evaluated models using the Scott-Knott algorithm [28]. The Scott-Knott method uses hierarchical clustering to rank each technique in equivalence groups, so that techniques within each group have similar performance. This analysis was performed for the (Box-cox transformed) accuracy and stability of each dataset, totaling 13 accuracy ranks and 13 stability ranks assigned to each model+tuner. Table 7 shows the 10 most accurate and the 10 most stable models across the studied datasets. To determine this, we counted the amount of times (N) that the model obtained the top ranking for each dataset. In the case of ties, we selected the technique with higher SA or SD, averaged across all datasets.

**Table 7: Top 10 techniques for SA and SD across all datasets.**

Accuracy			Stability		
Model	N	SA	Model	N	SD
FS+Log+SVR	8	0.48	HB+Log+RR	10	0.61
DE+Log+SVR	7	0.48	HB+Log+VT+RR	9	0.61
GA+Log+SVR	7	0.48	GA+Log+RR	9	0.61
CGA+Log+SVR	7	0.47	HS+Log+VT+RR	9	0.61
TS+CP+RR	7	0.45	HS+Log+RR	9	0.60
CGA+RR	7	0.45	HB+Log+CP+RR	9	0.60
GA+RR	7	0.45	PSO+Log+RR	9	0.60
PSO+CP+RR	7	0.45	GA+Log+CP+RR	9	0.60
DG+CP+RR	7	0.45	HS+Log+CP+RR	9	0.60
HB+RR	7	0.45	GA+Log+VT+RR	9	0.60

For accuracy, there was no technique that ranked first in all datasets, and thus none can be considered the overall “best”. However, there are techniques that obtained top ranking in more datasets than others. The flash-tuned SVR model with log transformation (FS+Log+SVR) ranked top in 8 out of 13 datasets, with a minimum SD of 0.31, a mean SD of 0.48, and a maximum SD of 0.67. The second, third, and fourth top SA techniques according to the ranking in table 7 corresponded with Log+SVR with DE, GA, and CGA tuning. The models ranked 5 to 10 correspond with tuned RR and CP+RR models.

The top-ranked RR models obtained the top rank in the datasets where the best model did not. FS+Log+SVR ranked top in some of the datasets with high amount of features (ISBSG R18 subsets and isbsg10), and the RR models did so on the smaller datasets (desharnais, kitchenham, maxwell, miyazaki94). Thus, these two approaches are complementary, and generally cover each other’s weaknesses on SEE.

For stability, there was no technique that ranked first in all cases, and thus none can be considered the overall “best”. However, all top 10 (and up to the top 63) models correspond with ridge regression models. The HB+Log+RR model ranked top in 10 out of 13 datasets, with a minimum SD of 0.51, a mean SD of 0.61, and a maximum SD of 0.87. Moreover, HS+Log+RR obtained second ranking on the other 3 datasets (albrecht, kitchenham, maxwell).

**RQ3:** No model and tuner combination can be considered the best overall. The technique that was ranked first in accuracy in the highest amount of datasets was FS+Log+SVR. Other combinations of genetic algorithm Log+SVR also obtained good accuracy scores. The technique that was ranked first in stability in the highest amount of datasets was HS+Log+RR.

## 5 CONCLUSION

In this study we evaluated the impact on model accuracy and stability of 12 state-of-the-art hyper-parameter tuning algorithms against random search. These tuners adjusted the parameters of 10 machine learning models, which were trained on 9 datasets of the PROMISE repository and 4 sub-datasets from the ISBSG R18 dataset. This involved the tuning, training and evaluation of 1,690 models; or 169,000 models if we account for the 100 repetitions of bootstrap cross-validation.

The results obtained in the study indicate that hyper-parameter tuning might not be effective in all circumstances, and that the elevated cost of tuning might not necessarily improve a model. We however determined that tuning was effective in 65% of the studied situations if an appropriate tuner was selected. The effectiveness of tuning was higher on datasets with intrinsic dimensionality less than 5.

This study portrayed random search as a viable search algorithm for SEE. We covered 12 state-of-the-art tuners, and our results show that these methods were more effective than random search in only 6% of the studied model and dataset combinations. If the best tuner is selected, the state-of-the-art tuners are more effective than random search in 27% of the studied model and dataset combinations. Of the studied tuners, genetic and heuristic-based tuners had the highest amount of scenarios that outperformed random search.

There was no “best” model for the studied datasets. The results obtained in this study suggest that the most effective tuning method depended on the model and data. The FS+Log+SVR model was ranked first in accuracy in the highest amount of datasets, and that HB+Log+RR was ranked first in stability in the highest amount of datasets.

Our primary recommendation for future studies, SEE practitioners, and generally any user of hyper-parameter tuning is to “look before you leap”. That is, analyze and determine how complex is a dataset before utilizing a costly and exhaustive algorithm. We advise SEE studies to select and evaluate a set of hyper-parameter tuning algorithms and determine which one was effective for their particular dataset and model. For this end, we particularly recommend the flash, (compact) genetic algorithm, differential evolution, and hyperband algorithms as effective hyper-parameter tuners. We also endorse the use of random search as a baseline method tuner. Just as random guessing became a baseline predictor for the standardized accuracy metric in SEE, perhaps random guessing can become the baseline tuner.

For future work, we contemplate the replication this study on further machine learning algorithms and data transformation methods. We are also interested in extending this study to other datasets in the software engineering literature. Another venue is studying the effect of ‘tuning’ the hyper-parameter tuners, as well as comparing random search against tuners with higher budgets.

## ACKNOWLEDGMENTS

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

## REFERENCES

- [1] Amritanshu Agrawal, Wei Fu, Di Chen, Xipeng Shen, and Tim Menzies. 2019. How to “DODGE” Complex Software Analytics. *IEEE Transactions on Software Engineering* (2019).
- [2] Amritanshu Agrawal, Xueqi Yang, Rishabh Agrawal, Rahul Yedida, Xipeng Shen, and Tim Menzies. 2021. Simpler Hyperparameter Optimization for Software Analytics: Why, How, When. *IEEE Transactions on Software Engineering* (2021), 1–1. <https://doi.org/10.1109/TSE.2021.3073242>
- [3] Chris Albon. 2018. *Machine learning with python cookbook: Practical solutions from preprocessing to deep learning*. O'Reilly Media, Inc.
- [4] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, Feb (2012), 281–305.
- [5] Barry Boehm, Chris Abts, and Sunita Chulani. 2000. Software development cost estimation approaches—A survey. *Annals of software engineering* 10, 1-4 (2000), 177–205.
- [6] Barry W Boehm. 1984. Software engineering economics. *IEEE transactions on Software Engineering* 1 (1984), 4–21.
- [7] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and regression trees*. CRC press.
- [8] Junjie Chen, Ningxin Xu, Peiqi Chen, and Hongyu Zhang. 2021. Efficient Compiler Autotuning via Bayesian Optimization. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1198–1209.
- [9] Duc-Cuong Dang and Per Kristian Lehre. 2016. Self-adaptation of mutation rates in non-elitist populations. In *International Conference on Parallel Problem Solving from Nature*. Springer, 803–813.
- [10] Karel Dejaeger, Wouter Verbeke, David Martens, and Bart Baesens. 2011. Data mining techniques for software effort estimation: a comparative study. *IEEE transactions on software engineering* 38, 2 (2011), 375–397.
- [11] Bradley Efron. 1983. Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American statistical association* 78, 382 (1983), 316–331.
- [12] AP Engelbrecht. 2005. Fundamentals of computational swarm intelligence. *England, John Wiley & Sons Ltd* (2005), 5–129.
- [13] Egemen Ertugrul, Zakir Baytar, Çağatay Çatal, and Ömer Can Muratlı. 2019. Performance tuning for machine learning-based software development effort prediction models. *Turkish Journal of Electrical Engineering & Computer Sciences* 27, 2 (2019), 1308–1324.
- [14] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135–146.
- [15] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. 2001. A new heuristic optimization algorithm: harmony search. *simulation* 76, 2 (2001), 60–68.
- [16] Fred Glover and Manuel Laguna. 1998. Tabu search. In *Handbook of combinatorial optimization*. Springer, 2093–2229.
- [17] XC Guo, JH Yang, CG Wu, CY Wang, and YC Liang. 2008. A novel LS-SVMs hyperparameter selection based on particle swarm optimization. *Neurocomputing* 71, 16-18 (2008), 3211–3215.
- [18] Georges R Harik, Fernando G Lobo, and David E Goldberg. 1999. The compact genetic algorithm. *IEEE transactions on evolutionary computation* 3, 4 (1999), 287–297.
- [19] William B Langdon, Javier Dolado, Federica Sarro, and Mark Harman. 2016. Exact mean absolute error of baseline predictor, MARP0. *Information and Software Technology* 73 (2016), 16–18.
- [20] Albert L Lederer and Jayesh Prasad. 1995. Causes of inaccurate software development cost estimates. *Journal of systems and software* 31, 2 (1995), 125–134.
- [21] Elizaveta Levina and Peter J Bickel. 2005. Maximum likelihood estimation of intrinsic dimension. In *Advances in neural information processing systems*. 777–784.
- [22] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research* 18, 1 (2017), 6765–6816.
- [23] Gang Luo. 2016. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics* 5, 1 (2016), 18.
- [24] Onkar Malgonde and Kaushal Chari. 2019. An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering* 24, 2 (2019), 1017–1055.
- [25] Leandro L Minku. 2019. A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. *Empirical Software Engineering* (2019), 1–52.
- [26] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding faster configurations using flash. *IEEE Transactions on Software Engineering* 46, 7 (2018), 794–811.
- [27] Sampath Kumar Palaniswamy and R Venkatesan. 2020. Hyperparameters tuning of ensemble model for software effort estimation. *Journal of Ambient Intelligence and Humanized Computing* (2020), 1–11.
- [28] Andrew Jhon Scott and M Knott. 1974. A cluster analysis method for grouping means in the analysis of variance. *Biometrics* (1974), 507–512.
- [29] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.
- [30] Martin Shepperd and Steve MacDonell. 2012. Evaluating prediction systems in software project estimation. *Information and Software Technology* 54, 8 (2012), 820–827.
- [31] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.
- [32] Liyan Song, Leandro L Minku, and Xin Yao. 2013. The impact of parameter tuning on software effort estimation using learning machines. In *Proceedings of the 9th international conference on predictive models in software engineering*. 1–10.
- [33] Liyan Song, Leandro L Minku, and Xin Yao. 2014. The potential benefit of relevance vector machine to software effort estimation. In *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*. 52–61.
- [34] Rainer Storn and Kenneth Price. 1997. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11, 4 (1997), 341–359.
- [35] Chakkril Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering*. 321–332.
- [36] Chakkril Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2018. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering* 45, 7 (2018), 683–711.
- [37] Leonardo Villalobos-Arias, Christian Quesada-López, Jose Guevara-Coto, Alexandra Martínez, and Marcelo Jenkins. 2020. Evaluating Hyper-Parameter Tuning using Random Search in Support Vector Machines for Software Effort Estimation. In *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'20)*. ACM.
- [38] Leonardo Villalobos-Arias, Christian Quesada-López, Alexandra Martínez, and Marcelo Jenkins. 2021. Hyper-Parameter Tuning of Classification and Regression Trees for Software Effort Estimation. In *Trends and Applications in Information Systems and Technologies: Volume 3 9*. Springer International Publishing, 589–598.
- [39] Leonardo Villalobos-Arias, Christian Quesada-López, Alexandra Martínez, and Marcelo Jenkins. 2021. Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: un mapeo de literatura. *Revista Ibérica de Sistemas e Tecnologías de Informação E42* (2021), 305–318.
- [40] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. 2012. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology* 54, 1 (2012), 41–59.
- [41] Tianpei Xia, Wei Fu, Rui Shu, and Tim Menzies. 2020. Predicting Project Health for Open Source Projects (using the DECART Hyperparameter Optimizer). *arXiv preprint arXiv:2006.07240* (2020).
- [42] Tianpei Xia, Rahul Krishna, Jianfeng Chen, George Mathew, Xipeng Shen, and Tim Menzies. 2018. Hyperparameter optimization for effort estimation. *arXiv preprint arXiv:1805.00336* (2018).
- [43] Li Yang and Abdallah Shami. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415 (2020), 295–316.
- [44] Alice Zheng. 2015. Evaluating machine learning models: a beginner’s guide to key concepts and pitfalls. (2015).

# Bibliography

- [1] B. W. Boehm, "Software engineering economics," *IEEE transactions on Software Engineering*, no. 1, pp. 4–21, 1984.
- [2] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches—a survey," *Annals of software engineering*, vol. 10, no. 1-4, pp. 177–205, 2000.
- [3] A. L. Lederer and J. Prasad, "Causes of inaccurate software development cost estimates," *Journal of systems and software*, vol. 31, no. 2, pp. 125–134, 1995.
- [4] M. Nasir, "A survey of software estimation techniques and project planning practices," in *Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06)*, pp. 305–310, IEEE, 2006.
- [5] A. Abran, *Software project estimation: the fundamentals for providing high quality information to decision makers*. John Wiley & Sons, 2015.
- [6] B. Baskeles, B. Turhan, and A. Bener, "Software effort estimation using machine learning methods," in *2007 22nd international symposium on computer and information sciences*, pp. 1–6, IEEE, 2007.
- [7] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Transactions on software engineering*, vol. 33, no. 1, pp. 33–53, 2006.
- [8] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens, "Data mining techniques for software effort estimation: a comparative study," *IEEE transactions on software engineering*, vol. 38, no. 2, pp. 375–397, 2011.

- [9] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Information and Software Technology*, vol. 54, no. 1, pp. 41–59, 2012.
- [10] S. K. Sehra, Y. S. Brar, N. Kaur, and S. S. Sehra, "Research patterns and trends in software effort estimation," *Information and Software Technology*, vol. 91, pp. 1–21, 2017.
- [11] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [12] L. L. Minku, "A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation," *Empirical Software Engineering*, pp. 1–52, 2019.
- [13] T. Menzies and M. Shepperd, "Special issue on repeatable results in software engineering prediction," 2012.
- [14] L. Song, L. L. Minku, and X. Yao, "The impact of parameter tuning on software effort estimation using learning machines," in *Proceedings of the 9th international conference on predictive models in software engineering*, p. 9, ACM, 2013.
- [15] M. Jørgensen, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, vol. 70, no. 1-2, pp. 37–60, 2004.
- [16] E. Mendes and B. Kitchenham, "Further comparison of cross-company and within-company effort estimation models for web applications," in *10th International Symposium on Software Metrics, 2004. Proceedings.*, pp. 348–357, IEEE, 2004.
- [17] C. Mair and M. Shepperd, "The consistency of empirical comparisons of regression and analogy-based software project cost prediction," in *2005 International Symposium on Empirical Software Engineering, 2005.*, pp. 10–pp, IEEE, 2005.
- [18] M. Shepperd, "Software project economics: a roadmap," in *2007 Future of Software Engineering*, pp. 304–315, IEEE Computer Society, 2007.
- [19] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 316–329, 2007.

- [20] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pp. 321–332, IEEE, 2016.
- [21] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, no. 1, p. 18, 2016.
- [22] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "Using tabu search to configure support vector regression for effort estimation," *Empirical Software Engineering*, vol. 18, no. 3, pp. 506–546, 2013.
- [23] L. Song, L. L. Minku, and X. Yao, "The potential benefit of relevance vector machine to software effort estimation," in *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, pp. 52–61, ACM, 2014.
- [24] M. Hosni, A. Idri, A. Abran, and A. B. Nassif, "On the value of parameter tuning in heterogeneous ensembles effort estimation," *Soft Computing*, vol. 22, no. 18, pp. 5977–6010, 2018.
- [25] E. Ertuğrul, Z. Baytar, Ç. Çatal, and Ö. C. Muratli, "Performance tuning for machine learning-based software development effort prediction models," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 27, no. 2, pp. 1308–1324, 2019.
- [26] L. L. Minku and X. Yao, "Ensembles and locality: Insight on improving software effort estimation," *Information and Software Technology*, vol. 55, no. 8, pp. 1512–1528, 2013.
- [27] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyperparameter optimization," in *Advances in neural information processing systems*, pp. 2546–2554, 2011.
- [28] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Transactions on Software Engineering*, 2018.
- [29] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

- [30] V. K. Bardsiri, D. N. A. Jawawi, S. Z. M. Hashim, and E. Khatibi, "A pso-based model to increase the accuracy of software development effort estimation," *Software Quality Journal*, vol. 21, no. 3, pp. 501–526, 2013.
- [31] P. L. Braga, A. L. Oliveira, and S. R. Meira, "A ga-based feature selection and parameters optimization for support vector regression applied to software effort estimation," in *Proceedings of the 2008 ACM symposium on Applied computing*, pp. 1788–1792, ACM, 2008.
- [32] A. Abran, *Software metrics and software metrology*. John Wiley & Sons, 2010.
- [33] C. Quesada-López, J. Murillo-Morera, and M. Jenkins, "Un estudio comparativo de técnicas de minería de datos y aprendizaje máquina para la estimación del esfuerzo utilizando puntos de función," *Revista Ibérica de Sistemas e Tecnologías de Informação*, no. E17, pp. 595–609, 2019.
- [34] M. Jorgensen, "Experience with the accuracy of software maintenance task effort prediction models," *IEEE Transactions on software engineering*, vol. 21, no. 8, pp. 674–681, 1995.
- [35] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, vol. 33, no. 1, pp. 2–13, 2006.
- [36] H. Zhang and X. Zhang, "Comments on" data mining static code attributes to learn defect predictors"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 635–637, 2007.
- [37] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'" ," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 637–640, 2007.
- [38] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE transactions on software engineering*, vol. 37, no. 3, pp. 356–370, 2010.
- [39] N. Mittas and L. Angelis, "Ranking and clustering software cost estimation models through a multiple comparisons algorithm," *IEEE Transactions on software engineering*, vol. 39, no. 4, pp. 537–551, 2012.

- [40] J. Keung, E. Kocaguneli, and T. Menzies, “Finding conclusion stability for selecting the best effort predictor in software effort estimation,” *Automated Software Engineering*, vol. 20, no. 4, pp. 543–567, 2013.
- [41] M. Shepperd and S. MacDonell, “Evaluating prediction systems in software project estimation,” *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.
- [42] W. B. Langdon, J. Dolado, F. Sarro, and M. Harman, “Exact mean absolute error of baseline predictor, marp0,” *Information and Software Technology*, vol. 73, pp. 16–18, 2016.
- [43] J. J. Dolado, D. Rodriguez, M. Harman, W. B. Langdon, and F. Sarro, “Evaluation of estimation models using the minimum interval of equivalence,” *Applied Soft Computing*, vol. 49, pp. 956–967, 2016.
- [44] L. Lavazza and S. Morasca, “On the evaluation of effort estimation models,” in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pp. 41–50, ACM, 2017.
- [45] S. D. Conte, H. E. Dunsmore, and Y. Shen, *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., 1986.
- [46] B. A. Kitchenham, L. M. Pickard, S. G. MacDonell, and M. J. Shepperd, “What accuracy statistics really measure,” *IEEE Proceedings-Software*, vol. 148, no. 3, pp. 81–85, 2001.
- [47] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, “A simulation study of the model evaluation criterion mmre,” *IEEE transactions on software engineering*, vol. 29, no. 11, pp. 985–995, 2003.
- [48] I. Myrtveit, E. Stensrud, and M. Shepperd, “Reliability and validity in comparative studies of software prediction models,” *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380–391, 2005.
- [49] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, “How effective is tabu search to configure support vector regression for effort estimation?,” in *Proceedings of the 6th international conference on predictive models in software engineering*, pp. 1–10, 2010.

- [50] M. Azzeh, "Software effort estimation based on optimized model tree," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pp. 1–8, 2011.
- [51] A. Agrawal, T. Menzies, L. L. Minku, M. Wagner, and Z. Yu, "Better software analytics via " duo": Data mining algorithms using/used-by optimizers," *arXiv preprint arXiv:1812.01550*, 2018.
- [52] A. Agrawal, W. Fu, D. Chen, X. Shen, and T. Menzies, "How to " dodge" complex software analytics," *IEEE Transactions on Software Engineering*, 2019.
- [53] V. Nair, Z. Yu, and T. Menzies, "Flash: A faster optimizer for sbse tasks," *arXiv preprint arXiv:1705.05018*, 2017.
- [54] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using flash," *IEEE Transactions on Software Engineering*, vol. 46, no. 7, pp. 794–811, 2018.
- [55] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?," *Information and Software Technology*, vol. 76, pp. 135–146, 2016.
- [56] W. Fu, V. Nair, and T. Menzies, "Why is differential evolution better than grid search for tuning defect predictors?," *arXiv preprint arXiv:1609.02613*, 2016.
- [57] M. Azzeh, A. B. Nassif, and S. Banitaan, "A better case adaptation method for case-based effort estimation using multi-objective optimization," in *2014 13th International Conference on Machine Learning and Applications*, pp. 409–414, IEEE, 2014.
- [58] J. H. Wu and J. W. Keung, "Utilizing cluster quality in hierarchical clustering for analogy-based software effort estimation," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 1–4, IEEE, 2017.
- [59] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The promise repository of empirical software engineering data," 2012.
- [60] ISBSG, "The international software benchmarking standards group." <https://www.isbsg.org/>, 2018. Accessed: 2020-07-07.
- [61] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.



- [62] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update,” *Information and Software Technology*, vol. 64, pp. 1–18, 2015.
- [63] I. Sommerville, “Software engineering 9th edition,” *ISBN-10137035152*, 2011.
- [64] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.
- [65] T. Menzies, L. Williams, and T. Zimmermann, *Perspectives on data science for software engineering*. Morgan Kaufmann, 2016.
- [66] D. Zhang, Y. Dang, J.-G. Lou, S. Han, H. Zhang, and T. Xie, “Software analytics as a learning case in practice: Approaches and experiences,” in *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, pp. 55–58, ACM, 2011.
- [67] D. Zhang, S. Han, Y. Dang, J.-G. Lou, H. Zhang, and T. Xie, “Software analytics in practice,” *IEEE software*, vol. 30, no. 5, pp. 30–37, 2013.
- [68] M. Harman, S. A. Mansouri, and Y. Zhang, “Search based software engineering: A comprehensive analysis and review of trends techniques and applications,” *Department of Computer Science, King’s College London, Tech. Rep. TR-09-03*, p. 23, 2009.
- [69] M. Harman and B. F. Jones, “Search-based software engineering,” *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.
- [70] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, “Search based software engineering: Techniques, taxonomy, tutorial,” in *Empirical software engineering and verification*, pp. 1–59, Springer, 2010.
- [71] M. Harman and A. Mansouri, “Search based software engineering: Introduction to the special issue of the IEEE transactions on software engineering,” *IEEE Transactions on Software Engineering*, no. 6, pp. 737–741, 2010.
- [72] I. Fister Jr, X.-S. Yang, I. Fister, J. Brest, and D. Fister, “A brief review of nature-inspired algorithms for optimization,” *arXiv preprint arXiv:1307.4186*, 2013.

- [73] M. Harman, “The relationship between search based software engineering and predictive modeling,” in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, pp. 1–13, 2010.
- [74] J. J. Dolado, “A validation of the component-based method for software size estimation,” *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 1006–1021, 2000.
- [75] J. J. Dolado, “On the problem of the software cost function,” *Information and Software Technology*, vol. 43, no. 1, pp. 61–72, 2001.
- [76] C. Kirsopp, M. J. Shepperd, and J. Hart, “Search heuristics, case-based reasoning and software project effort prediction,” 2002.
- [77] X. Guo, J. Yang, C. Wu, C. Wang, and Y. Liang, “A novel ls-svms hyperparameter selection based on particle swarm optimization,” *Neurocomputing*, vol. 71, no. 16-18, pp. 3211–3215, 2008.
- [78] A. Arcuri and G. Fraser, “Parameter tuning or default values? an empirical investigation in search-based software engineering,” *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.
- [79] A. Arcuri and G. Fraser, “On parameter tuning in search based software engineering,” in *International Symposium on Search Based Software Engineering*, pp. 33–47, Springer, 2011.
- [80] M. F. Bosu and S. G. Macdonell, “Experience: Quality benchmarking of datasets used in software effort estimation,” *Journal of Data and Information Quality (JDIQ)*, vol. 11, no. 4, p. 19, 2019.
- [81] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki, “Robust regression for developing software estimation models,” *Journal of Systems and Software*, vol. 27, no. 1, pp. 3–16, 1994.
- [82] M. Shepperd and C. Schofield, “Estimating software project effort using analogies,” *IEEE Transactions on software engineering*, vol. 23, no. 11, pp. 736–743, 1997.
- [83] C. Mair, M. Shepperd, and M. Jørgensen, “An analysis of data sets used to train and validate cost prediction systems,” in *ACM SIGSOFT software engineering notes*, vol. 30, pp. 1–6, ACM, 2005.

- [84] S. Amasaki, “Replicated analyses of windowing approach with single company datasets,” in *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*, pp. 14–17, ACM, 2011.
- [85] M. D. Prabhakar, “Prediction of software effort using artificial neural network and support vector machine,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 3, 2013.
- [86] M. Fernández-Diego and F. González-Ladrón-De-Guevara, “Potential and limitations of the isbgs dataset in enhancing software engineering research: A mapping review,” *Information and Software Technology*, vol. 56, no. 6, pp. 527–544, 2014.
- [87] F. González-Ladrón-de Guevara, M. Fernández-Diego, and C. Lokan, “The usage of isbgs data fields in software effort estimation: A systematic mapping study,” *Journal of Systems and Software*, vol. 113, pp. 188–215, 2016.
- [88] B. A. Kitchenham and E. Mendes, “A comparison of cross-company and within-company effort estimation models for web applications,” in *Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering, Edinburgh, Scotland, UK*, pp. 47–55, 2004.
- [89] L. L. Minku and S. Hou, “Clustering dycom: an online cross-company software effort estimation study,” in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 12–21, 2017.
- [90] J. Murillo-Morera, C. Quesada-López, C. Castro-Herrera, and M. Jenkins, “A genetic algorithm based framework for software effort prediction,” *Journal of Software Engineering Research and Development*, vol. 5, no. 1, p. 4, 2017.
- [91] L. Minku, F. Sarro, E. Mendes, and F. Ferrucci, “How to make best use of cross-company data for web effort estimation?,” in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–10, IEEE, 2015.
- [92] A. G’eron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O’Reilly Media, 2019.
- [93] A. Halevy, P. Norvig, and F. Pereira, “The unreasonable effectiveness of data,” *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

- [94] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.
- [95] Y. Saeys, I. Inza, and P. Larrañaga, "A review of feature selection techniques in bioinformatics," *bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.
- [96] C. Albon, *Machine learning with python cookbook: Practical solutions from pre-processing to deep learning*. " O'Reilly Media, Inc.", 2018.
- [97] B. Efron, "Estimating the error rate of a prediction rule: improvement on cross-validation," *Journal of the American statistical association*, vol. 78, no. 382, pp. 316–331, 1983.
- [98] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2016.
- [99] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959, 2012.
- [100] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn," in *ICML workshop on AutoML*, vol. 9, Citeseer, 2014.
- [101] A. Engelbrecht, "Fundamentals of computational swarm intelligence," *England, John Wiley & Sons Ltd*, pp. 5–129, 2005.
- [102] A. L. Oliveira, P. L. Braga, R. M. Lima, and M. L. Cornélio, "Ga-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation," *information and Software Technology*, vol. 52, no. 11, pp. 1155–1166, 2010.
- [103] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE transactions on evolutionary computation*, vol. 3, no. 4, pp. 287–297, 1999.
- [104] D.-C. Dang and P. K. Lehre, "Self-adaptation of mutation rates in non-elitist populations," in *International Conference on Parallel Problem Solving from Nature*, pp. 803–813, Springer, 2016.

- [105] B. Doerr, H. P. Le, R. Makhmara, and T. D. Nguyen, “Fast genetic algorithms,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 777–784, 2017.
- [106] E. Hazan, A. Klivans, and Y. Yuan, “Hyperparameter optimization: A spectral approach,” *arXiv preprint arXiv:1706.00764*, 2017.
- [107] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.
- [108] F. Glover and M. Laguna, “Tabu search,” in *Handbook of combinatorial optimization*, pp. 2093–2229, Springer, 1998.
- [109] Z. W. Geem, J. H. Kim, and G. V. Loganathan, “A new heuristic optimization algorithm: harmony search,” *simulation*, vol. 76, no. 2, pp. 60–68, 2001.
- [110] R. Storn and K. Price, “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces,” *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.
- [111] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A novel bandit-based approach to hyperparameter optimization,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.
- [112] I. Sabuncuoglu and M. Bayiz, “Job shop scheduling with beam search,” *European Journal of Operational Research*, vol. 118, no. 2, pp. 390–412, 1999.
- [113] E. S. Jun and J. K. Lee, “Quasi-optimal case-selective neural network model for software effort estimation,” *Expert Systems with Applications*, vol. 21, no. 1, pp. 1–14, 2001.
- [114] C. M. Bishop, “Pattern recognition,” *Machine learning*, vol. 128, no. 9, 2006.
- [115] J.-k. Lee and K.-T. Kwon, “Software cost estimation using svr based on immune algorithm,” in *2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, pp. 462–466, IEEE, 2009.
- [116] S. H. S. Moosavi and V. K. Bardsiri, “Satin bowerbird optimizer: A new optimization algorithm to optimize anfis for software development effort estimation,” *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 1–15, 2017.

- [117] R. L. Glass, “The software-research crisis,” *IEEE Software*, vol. 11, no. 6, pp. 42–47, 1994.
- [118] B. Kitchenham, S. Charters, D. Budgen, P. Brereton, M. Turner, S. Linkman, M. Jørgensen, E. Mendes, and G. Visaggio, “Guidelines for performing systematic literature reviews in software engineering,” tech. rep., Technical report, Ver. 2.3 EBSE Technical Report. EBSE, 2007.
- [119] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.
- [120] L. Villalobos-Arias, C. Quesada-López, A. Martínez, and M. Jenkins, “Técnicas de ajuste de hiperparámetros de algoritmos de aprendizaje automático para la estimación de esfuerzo: un mapeo de literatura,” *Revista Ibérica de Sistemas e Tecnologías de Informação*, no. E42, pp. 305–318, 2021.
- [121] R. d. A. Araújo, S. Soares, and A. L. Oliveira, “Hybrid morphological methodology for software development cost estimation,” *Expert Systems with Applications*, vol. 39, no. 6, pp. 6129–6139, 2012.
- [122] E. Kocaguneli, T. Menzies, and J. W. Keung, “Kernel methods for software effort estimation,” *Empirical Software Engineering*, vol. 18, no. 1, pp. 1–24, 2013.
- [123] D. Azhar, P. Riddle, E. Mendes, N. Mittas, and L. Angelis, “Using ensembles for web effort estimation,” in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 173–182, IEEE, 2013.
- [124] H. Hota, R. Shukla, and S. Singhai, “Predicting software development effort using tuned artificial neural network,” in *Computational Intelligence in Data Mining-Volume 3*, pp. 195–203, Springer, 2015.
- [125] M. Hosni, A. Idri, A. B. Nassif, and A. Abran, “Heterogeneous ensembles for software development effort estimation,” in *2016 3rd International Conference on Soft Computing & Machine Intelligence (ISCFMI)*, pp. 174–178, IEEE, 2016.
- [126] M. Hosni, A. Idri, and A. Abran, “Investigating heterogeneous ensembles with filter feature selection for software effort estimation,” in *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, pp. 207–220, 2017.

- [127] L. Villalobos-Arias, C. Quesada-López, J. Guevara-Coto, A. Martínez, and M. Jenkins, “Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation,” in *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 31–40, 2020.
- [128] L. Villalobos-Arias, C. Quesada-López, A. Martínez, and M. Jenkins, “Hyper-parameter tuning of classification and regression trees for software effort estimation,” in *Trends and Applications in Information Systems and Technologies: Volume 3 9*, pp. 589–598, Springer International Publishing, 2021.
- [129] L. Villalobos-Arias, C. Quesada-López, J. Murillo-Morera, and M. Jenkins, “Hyper-parameter tuning using genetic algorithms for software effort estimation,” in *Proceedings of the 16th Iberic Conference on Information and Technology Systems*, 2021.
- [130] L. Villalobos-Arias, C. Quesada-López, A. Martínez, and M. Jenkins, “Multi-objective hyper-parameter tuning for software effort estimation,” in *Proceedings of the 24th Ibero-American Conference on Software Engineering*, To be published.
- [131] L. Villalobos-Arias and C. Quesada-López, “Comparative study of random search hyper-parameter tuning for software effort estimation,” in *Proceedings of the 17th International Conference on Predictable Models and Data Analytics in Software Engineering*, To be published.
- [132] A. J. Scott and M. Knott, “A cluster analysis method for grouping means in the analysis of variance,” *Biometrics*, pp. 507–512, 1974.