Learning From Mistakes: Machine Learning Enhanced Human Expert Effort Estimates

Federica Sarro, Rebecca Moussa, Alessio Petrozziello and Mark Harman

Abstract—In this paper, we introduce a novel approach to predictive modeling for software engineering, named Learning From Mistakes (LFM). The core idea underlying our proposal is to automatically learn from past estimation errors made by human experts, in order to predict the characteristics of their future misestimates, therefore resulting in improved future estimates. We show the feasibility of LFM by investigating whether it is possible to predict the type, severity and magnitude of errors made by human experts when estimating the development effort of software projects, and whether it is possible to use these predictions to enhance future estimations. To this end we conduct a thorough empirical study investigating 402 maintenance and new development industrial software projects. The results of our study reveal that the type, severity and magnitude of errors are all, indeed, predictable. Moreover, we find that by exploiting these predictions, we can obtain significantly better estimates than those provided by random guessing, human experts and traditional machine learners in 31 out of the 36 cases considered (86%), with large and very large effect sizes in the majority of these cases (81%). This empirical evidence opens the door to the development of techniques that use the power of machine learning, coupled with the observation that human errors are predictable, to support engineers in estimation tasks rather than replacing them with machine-provided estimates.

Index Terms—Software Effort Estimation, Estimate errors, Human expert estimates, Human Bias, Human-competitive results.

1 Introduction

Software development effort estimation is a crucial activity for project planning and monitoring, specifically for ensuring that the product is delivered on time and within budget [1], [2]. Studies have shown that engineers make inaccurate effort estimations [3], [4], [5], [6], which can negatively affect the outcome of software projects leading to great losses [2], [7].

To support engineers in obtaining more accurate estimates, researchers and practitioners have attempted to devise various automated methods over the last three decades [1], [8]. However, despite the rise of automated predictive modeling, human expert judgement is still the most commonly applied strategy for software effort estimation [6], [9] and their expertise has not been fully exploited in combination with automated approaches [10], [11].

This observation motivates us to depart from received wisdom and current research practice in the predictive modeling community. In this work, we shift the focus from creating automated models able to predict software effort to creating automated models able to predict the errors made by human experts when estimating effort and using this to adjust their estimates. Rather than seeking solely to compete with (or even replace) human experts, our approach learns, not only from traditional past projects cost drivers, but also from past expert judgements, essentially building into the predictive model the ability to learn from their past estimation errors (i.e. misestimates). We name this approach *Learning From Mistakes* (LFM) as it argues that:

1) it is possible to predict the type, severity and magnitude of human experts misestimates by learning from the estimation errors they have made in the past;
2) these predictions can be usefully exploited in order to enhance future effort estimates.

In order to evaluate the feasibility and effectiveness of LFM, we carry out a thorough empirical study following best practice for the evaluation of prediction models in Software Engineering [12], [13], [14].

To address the first claim, we study the predictability of 402 human expert misestimates in terms of their type (i.e. under-/over- estimates), severity (i.e. low, medium, high), and magnitude (i.e. estimation error relative to the true effort value). If we can show that these misestimate characteristics are indeed predictable, then we can investigate whether their prediction can be used to improve future effort estimates. In particular, to address the second claim, we adjust the original human expert estimates with the predicted magnitude errors and compare the two (i.e. $original estimate$ vs. $original estimate - predicted misestimate$).

The results of our empirical study show that:

1) **Human expert misestimates are predictable**. The average classification accuracy, measured in Area Under the ROC Curve, for both the type of misestimation and its level of severity of all techniques over all datasets, is 71% and 70%, respectively. Also, the prediction of the amount of misestimation made by human experts is very close to the true amount of misestimation (i.e. the average median absolute error of all classifiers across all datasets is 0.28).
2) **This predictability can be usefully exploited**. That is, LFM enhances human experts' effort predictions obtaining estimates that are significantly better than

• *E-mail: f.sarro@ucl.ac.uk, rebecca.moussa.18@ucl.ac.uk, a.petrozziello@ucl.ac.uk, mark.harman@ucl.ac.uk*

Manuscript received xxx; revised xxx.

those provided by random guessing, human experts and traditional automated learners in 32 out of 36 cases (89%) (with large and very large effect sizes observed in 81% of these cases), and never worse in the remaining 11% cases.

The scientific contribution of these findings is the empirical evidence to support the claim that human estimation errors are, indeed, predictable and can be used to improve human experts' effort estimates. This is the first time this question has been investigated in the software engineering literature. The finding is important because it provides an entirely orthogonal deployment route for estimation technology: Instead of replacing human estimators with machine learnt estimations, it advocates for techniques that support humans in the necessary task of software development effort estimation.

In the remainder of the paper we first explain the details of our LFM approach for software effort estimation (Section 2) and then the empirical study we carried out to assess its feasibility and effectiveness (Section 3). The results of the study are discussed (Section 4) together with their relation with existing work (Section 5) and future studies (Section 6).

2 LFM FOR SOFTWARE EFFORT ESTIMATION

In this section we explain how LFM can be used in practice, by software companies, to estimate/predict the effort needed for realising software projects. LFM is useful in any scenario where the target project for which the effort is unknown but has been estimated by the expert. The aim of LFM is to support the human expert in acknowledging possible errors in their estimations and use this information to improve their estimates. In order to do that, LFM looks at past software projects and learns the errors that experts had committed when estimating the effort. This is divided into three phases that are shown in Figure 1 and are described in details below.

Phase 1. Deriving Type, Severity and Magnitude of Past Estimate Errors:

This step gathers information about historical software projects realised by the same company or by different ones (usually referred to as single-company data or cross-company data, respectively). Each of these projects is described by a set of cost drivers, such as functional size, team experience, programming languages and the actual effort (e.g. person-hours) required to realise the project, as recorded by the company employers[1]. Part of the novelty of LFM is to augment these cost drivers by adding information about human expert past misestimates. In particular, LFM computes the type, the severity and the magnitude of past misestimates by using both the human expert effort estimates and the actual effort.

More formally, given a set of past software projects , each project $p \in$, is characterised by the actual effort, ActualEff_p, which was required to complete p and by the estimated effort, EstEff_p, which was originally estimated by the expert[2]. Based on this, the type of error estimate ($\text{MisestimationType}_p$) of a project p is given by

$$\text{MisestimationType}_p = \begin{cases} \text{over-estimate}, & \text{if } \text{EstEff}_p > \text{ActualEff}_p \\ \text{under-estimate}, & \text{if } \text{EstEff}_p < \text{ActualEff}_p \end{cases}$$

The severity of the error estimate of a project p ($\text{MisestimationSeverity}_p$) is computed by ranking the past projects with respect to the Magnitude of Relative Error (MRE) and grouping these MRE calculations into different severity levels (i.e. low, medium, high), according to given thresholds as follows:

$$\text{MisestimationSeverity}_p = \begin{cases} \text{low}, & \text{if } \text{MRE}_p < \alpha \\ \text{med}, & \text{if } \alpha \leq \text{MRE}_p < \beta \\ \text{high}, & \text{if } \text{MRE}_p \geq \beta \end{cases}$$

where MRE_p measures, for a given project p, the absolute difference between the actual effort and the estimated effort (i.e. absolute residual error) relative to the actual effort:

$$\text{MRE}_p = \frac{|\text{EstEff}_p - \text{ActualEff}_p|}{\text{ActualEff}_p}$$

The number of severity levels and the associated thresholds are parameters to our approach determined by the procedures in place in a given company. In this paper we experimented with three levels of severity (i.e. low, med, high) according to the following thresholds: the first 33th percentile for the low level (i.e. 33% of projects with the lowest MRE), the 34th to 66th percentile for the med severity level, and the remaining projects for the high one (i.e. 33% of projects with highest MRE). Of course, different settings can be used without altering the formulation of LFM.

The magnitude of the misestimation (Misestimation-Magnitude) for a given project p is computed as the relative error:

$$\text{MisestimationMagnitude}_p = \frac{\text{EstEff}_p - \text{ActualEff}_p}{\text{ActualEff}_p}$$

These misestimation characteristics, together with company-specific cost drivers, will be exploited in Phase 2.

Phase 2. Predicting The Characteristics of Future Estimation Errors:

The second phase, starts with a software project for which the development effort is unknown and needs to be estimated (i.e. *Target Software Project*). LFM exploits the information gathered in Phase 1, in order to find similarities between the target project and past projects. It uses this knowledge in order to predict the characteristic of the estimate error for the target project.

Specifically, from the information gathered in Phase 1 we are able to predict the type, the severity and the magnitude of the errors that might occur when predicting the effort of a target project (tp) (i.e. $\text{MisestimationType}_{tp}$, $\text{MisestimationSeverity}_{tp}$, $\text{MisestimationMagnitude}_{tp}$).

In order to predict $\text{MisestimationType}_{tp}$, we can use a two-class (i.e. binary) classifier since MisestimationType can assume only two values (i.e. under-/over- estimates), while to predict the severity of the error (i.e.

1. More details about the cost drivers used in our experiments are provided in Section 3.2.

2. Such an effort in our study is provided for each project by the software company and it is measured as person hours.

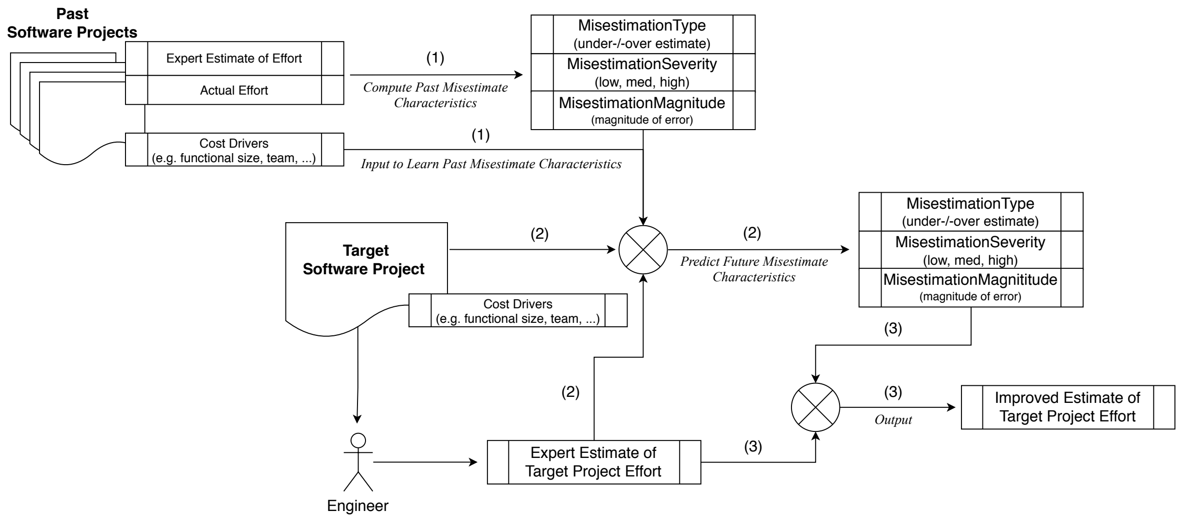

Past Software Projects
Expert Estimate of Effort
Actual Effort
Cost Drivers (e.g. functional size, team, ...)
(1) Compute Past Misestimate Characteristics
(1) Input to Learn Past Misestimate Characteristics
MisestimationType (under-/-over estimate)
MisestimationSeverity (low, med, high)
MisestimationMagnitude (magnitude of error)
Target Software Project
Cost Drivers (e.g. functional size, team, ...)
(2) Predict Future Misestimate Characteristics
MisestimationType (under-/-over estimate)
MisestimationSeverity (low, med, high)
MisestimationMagnititude (magnitude of error)
(3) Output
Improved Estimate of Target Project Effort
Expert Estimate of Target Project Effort
Engineer

Fig. 1: The Learning From Mistakes Approach (LFM). The numbers on the arrows correspond to each of the phases described in Section 2.

MisestimationSeverity$_{tp}$), a multiclass (i.e. multinomial) classifier[3] is necessary, given the severity levels defined in Phase 1. In order to predict the magnitude of the misestimates for the target project (i.e. MisestimationMagnitude$_{tp}$), and given the regression nature of the problem, any automated estimator ranging from simple regression- or analogy- based learners [16], [17] to more sophisticated ones such as those based on search-based approaches [18], [19], [20], [21] or deep-learning [22] can be applied. Obviously, different learners may exhibit different performance for different scenarios. Ultimately, the choice of the learner is a parameter of our approach.

Phase 3. Exploiting Predicted Misestimations:

The third and final phase of our approach involves exploiting the misestimations predicted for the target project. The early identification of potential under-/over- estimates and their severity can better guide human experts in understanding their predictions. On the other hand, the predicted misestimation magnitude can be used to automatically adjust human expert estimates. In the empirical study presented in this paper, we show that we can enhance the effort estimate produced by a human expert for a target project by adjusting it using the misestimation magnitude predicted during Phase 2 ($EstEff_{tp} - MisestimationMagnitude_{tp}$).

3 Empirical Study Design

In this section we explain the design of the empirical study we carried out to asses the feasibility and effectiveness of LFM. Our study follows the most recent best practices for the evaluation of prediction models in software engineering [12], [23], [24], [25], [26].

3.1 Research Questions

Our first two research questions investigate the predictability of the error made by human experts when estimating software project effort. The first research question tackles this as a classification problem, whereas the second research question treats it as a regression task. Specifically, RQ1 investigates whether we can classify the misestimation type and severity (i.e. MisestimationType and MisestimationSeverity as defined in Section 2 Phase 2), while RQ2 investigates whether we can estimate the magnitude of the error (MisestimationMagnitude as defined in Section 2 Phase 2). Our third and final research question focuses on the use of the predicted misestimations in order to adjust future human expert estimates as explained in Section 2 Phase 3. In the following we describe the way they are addressed.

RQ1. Predicting Type/Severity of Human Expert Misestimations: Can we predict the type and severity of the errors made by human experts when estimating software effort?

To address this question, we use four different machine learning techniques, namely CART, KNN, NB and RF (for more details, see Section 3.3) to classify the type and severity of human expert misestimates. In particular, we answer the following sub-questions:

RQ1.1 To what extent is the type of human expert misestimation predictable?

RQ1.2 To what extent is the severity of human expert misestimation predictable?

RQ2. Predicting the Magnitude of Human Expert Misestimations: Can we predict the magnitude of the misestimations made by human experts?
To answer this question we assess the effectiveness of four machine learners (i.e. CART, KNN, LP, RF). As a sanity check, we compare them with Random Guessing (RG).

RQ3. Enhancing Software Effort Estimates via LFM: Can software effort predictions be improved by learning from previous misestimations?

In order to address this question, we compare the prediction produced by LFM against human estimations of software development effort. For completeness, we also compare LFM with estimations obtained using traditional machine learning approaches alone. As sanity check, we

3. Multiclass classification is the problem of classifying instances into one of the more than two classes [15]. Classifying instances into one of the two classes is called binary classification. Multiclass classification should not be confused with multi-label classification, where multiple labels are to be predicted for each instance.

Dataset	Misestimation Type (RQ1.1)		Misestimation Severity (RQ1.2)			Misestimation Magnitude (RQ2)				Actual Effort (RQ3)			
	Under	Over	Low	Med	High	Min	Max	Mean	Std. Dev.	Min	Max	Mean	Std. Dev
ISBSG-C	63%	37%	33%	33%	34%	-0.93	29.49	0.54	4.26	207	46787	6574.22	10641.24
ISBSG-FP	65%	35%	33%	33%	34%	-0.96	10.83	-0.02	0.88	207	63732	5827.29	7550.27
KD	41%	59%	34%	31%	35%	-0.40	3.59	0.23	0.72	286	113930	5450.00	17723.16
KP	35%	65%	33%	33%	34%	-0.65	1.31	0.15	0.35	219	8656	2046.30	1927.96
Medical	52%	48%	32%	34%	34%	-0.83	0.07	0.46	1.42	60	10060	1530.00	1785.98
Telecom	59%	41%	35%	30%	35%	-0.49	0.61	-0.09	0.26	279	10244	2403.29	2707.80

TABLE 1: Descriptive statistics of the target variables used for each research question.

also compare LFM against RG. Therefore, we answer the following sub-questions:

RQ3.1: Does LFM provide better effort estimates than RG?

RQ3.2: Does LFM provide better effort estimates than traditional machine learners?

RQ3.3: Does LFM provide better effort estimates than human experts?

In the following we describe in detail the data (Section 3.2), the techniques (Section 3.3), the validation approach (Section 3.4), and the evaluation criteria and statistical tests (Section 3.5) used to address the above RQs. We also discuss possible threats to the validity of our empirical study (Section 3.6).

3.2 Datasets

To answer the RQs outlined in Section 3.1 we carry out a thorough empirical study using six real-world industrial datasets containing a total of 402 software projects developed by different software companies world-wide and collected up to 2018.

These datasets cover a variety of application domains (ranging from telecommunications to medical information systems), exhibit different project characteristics (e.g. technologies, tools and programming languages) and also vary in size (17 to 190 projects, 4 to 17 cost drivers depending on the dataset).

Table 2 summarises the descriptive statistics of the features of each of the datasets. We can observe that five datasets contain cost drivers based on the Function Point Analysis (FPA) [27] or COSMIC [28] functional size measurement (FSM) methods[4], which are widely used as independent variables to derive effort estimation models [36]. The `Medical` dataset instead contains features computed based on data models (e.g. number of entities), which have been shown to be useful cost drivers in previous work [35]. Moreover, all datasets contain two more features: the hours to complete a software project as estimated by a human expert (i.e. Expert Estimated Effort) and the number of hours actually required to complete it as recorded at the end of the project by the company (i.e. Actual Effort). These two features are used to compute the MisestimationType, MisestimationSeverity and the MisestimationMagnitude which are used as prediction target (i.e. dependent variable) for RQ1 and RQ2 as explained in Section 2, while the Actual Effort is used as a prediction target for RQ3. In our experiments we use both datasets consisting of projects that have been estimated by one expert (e.g., Medical) and others where the effort of different projects has been estimated by different experts (e.g., ISBSG). Descriptive statistics of the targets of each RQ are provided in Table 1.

Further details for each of the datasets are provided below to allow readers to assess whether the results we have gathered may apply to their own context.

The `ISBSG-C` and `ISBSG-FP` datasets have been extracted from the International Software Benchmarking Standards Group (ISBSG) repository release June 2018 R2 [37]. This repository contains software projects submitted by leading IT and metrics companies from around the world and has been widely used by practitioners as well as researchers for software project effort estimation studies [38]. The `ISBSG-C` dataset contains 49 projects characterised by four independent variables based on COSMIC [28] (i.e. Cosmic Entry, Cosmic Exit, Cosmic Read, Cosmic Write), Expert Estimated Effort and Actual Effort. While the `ISBSG-FP` dataset contains 190 projects characterised by six independent variables based on FP [27] (i.e. Input Count, Output Count, Enquiry Count, File Count, Interface Count, Added Count), Expert Estimated Effort and Actual Effort. We cannot disclose more details about this dataset due to a non-disclosure agreement (NDA).

The `Kitchenham` dataset contains data from both maintenance and new development software projects curated by the Computer Sciences Corporation on behalf of several client organisations [39]. This dataset contains projects spanning different products from different sources. All the projects are characterised by five independent variables based on the Function Point counts (i.e. External Input, External Output, Logical Internal, External Intergace, External Inquiry), the Expert Estimation and the Actual Effort. The effort estimates were made as part of the company's standard estimating process. In our study we considered only those software projects for which the effort estimate was made solely based on human experts' judgement for a total of 69 projects. Since these projects include both perfective maintenance and development projects, we analyse them separately, thereby obtaining two disjoint sets, namely `KD` and `KP` which contain 29 and 40 projects, respectively. More details about this dataset and its raw data can be found elsewhere [39].

4. FSM methods have obtained world-wide acceptance [1] and allow software size measurement in terms of the functionality with which users are provided. The first FSM method was FPA [27], and several variants have since been defined (e.g. MarkII and NESMA) with the aim of improving size measurement or extending the applicability domain [29]. These are all referred to as the first generation of FSM methods. COSMIC is instead a second generation FSM method having distinguishing characteristics, including its applicability to business, real-time, and infrastructure software (or their hybrids) [28], and possibility to extend its usage to other kinds of software such as Web and Mobile applications [30], [31], [32], [33], [34], [35].

The `Medical` dataset provides a set of 24 cost drivers (see Table 2 for the the full list) and effort data recorded for 77 modules of a single software system (i.e. a medical records database system built and implemented over a period of five months) [11]. Each of the modules implements a data entry/edit or reporting functionality and has associated 24 cost drivers. Since all this data was available during the module specification phase it can be used as input to a prediction system [11]. For each of the modules a single project manager's estimate of the effort, and the actual effort (both expressed in person-hours) are available.

The `Telecom` dataset contains 17 software projects developing typical administrative software, internal software development for a telecommunication company. All projects are characterised by four independent variables (i.e. Input Types, Entities, Output Types, Transactions), each representing an FP basic component [27], Expert Estimated Effort and Actual Effort. We cannot disclose more details about this dataset due to an NDA.

Dataset	Feature	Min	Max	Mean	Std. Dev.
`ISBSG-C` (49 projects)	Cosmic Entry	4	447	86.51	107.63
	Cosmic Exit	2	594	107.31	130.75
	Cosmic Read	0	545	83.88	109.24
	Cosmic Write	0	542	55.49	93.62
	Expert Estimated Effort	90	53774	7349.02	10641.24
	Actual Effort	207	46787	6574.22	9338.89
`ISBSG-FP` (190 projects)	Input count	0	2014	128.34	196.70
	Output Count	0	2760	77.34	213.85
	Enquiry Count	0	2356	105.85	189.40
	File Count	0	3196	93.69	254.20
	Interface Count	0	261	18.25	38.39
	Added Count	0	10571	331.92	815.44
	Expert Estimated Effort	80	58800	5082.94	7407.87
	Actual Effort	207	63732	5827.29	7550.27
`KD` (29 projects)	External Input	0	4701	263	731.69
	External Output	6	5265	241.40	812.53
	Logical Internal	0	1724	113.90	276.95
	External Interface	0	92	6.73	15.51
	External Inquiry	0	2925	152.40	454.82
	Expert Estimated Effort	337	79870	4586.00	12417.53
	Actual Effort	286	113930	5450.00	17723.16
`KP` (40 projects)	External Input	0	789	125.82	147.63
	External Output	0	360	81.00	95.71
	Logical Internal	0	402	61.02	89.45
	External Interface	0	614	25.52	89.72
	External Inquiry	0	618	80.50	125.81
	Expert Estimated Effort	200	8690	2038.70	1736.68
	Actual Effort	219	8656	2046.30	1927.96
`Medical` (77 projects)	Create Transactions	0	3	0.85	0.74
	Read Transactions	0	25	5.19	4.57
	Update Transactions	0	16	1.47	2.20
	Delete Transactions	0	2	0.26	0.59
	Reports Called	0	2	0.23	0.60
	Reports Produced	0	2	0.27	0.50
	Elements Reported	0	24	3.26	6.46
	Fields Calculated	0	14	0.77	2.45
	Fields Entered	0	19	5.26	3.82
	Screens Called	0	10	0.79	1.89
	Screens Displayed	0	6	1.04	0.90
	Elements Displayed	0	78	11.64	14.04
	Entities	0	22	4.27	3.72
	Entities Providing Data	0	14	4.13	3.25
	Entities Consuming Data	0	16	1.20	1.90
	Attributes	0	19	5.26	3.82
	Attributes Updated	0	69	9.42	13.26
	Attributes Consumed	0	83.00	19.81	18.69
	Links (1.1)	0	2	0.30	0.49
	Links (1.m)	0	13	3.09	3.32
	Optional Links	0	12	3.13	3.30
	Mandatory Links	0	3	0.25	0.54
	Entity Provisions	0	25	5.20	4.57
	Entity Consumptions	0	16	1.52	2.20
	Expert Estimated Effort	228	9450	1120.00	1278.82
	Actual Effort	60	10060	1530.00	1785.98
`Telecom` (17 projects)	Input types	4	858	201.71	242.15
	Entities	15	444	124.53	110.53
	Output Types	10	2322	484.88	640.69
	Transactions	7	265	51.18	59.54
	Expert Estimated Effort	450	9595	1967.29	2284.26
	Actual Effort	279	10244	2403.29	2707.80

TABLE 2: Descriptive statistics of the datasets used.

3.3 Classification and Regression Techniques

The concept of LFM is not defined by the machine learning approach used to classify the estimate errors. Therefore any technique can be used to this end and the choice is left to the practitioner.

In our empirical study, we experiment with five publicly available machine learners, namely Classification and Regression Trees (CART) [40], k-Nearest Neighhbours (KNN) [41], Naïve Bayes (NB) [42], Linear Programming (LP) [43] and Random Forest (RF) [44], all of which are well-known and widely-used by software engineering researchers and practitioners. Using such approaches avoids the risk that LFM benefits from some special or sophisticated ML technique. The results achieved with these traditional techniques can be considered as a lower bound to any more advanced technique, while their public availability supports and promotes replicability and extension of our work.

In order to address RQ1 (which involves a classification task), we use four machine learning approaches, namely CART, KNN, NB, RF, which are able to handle both binary and multiclass problems [15]. To address RQ2 and RQ3, which involve a regression task, instead, we use LP4EE (as it has been recently proposed as a robust baseline approach for prediction studies [13]) and three traditional and widely used estimation methods, namely CART, RF and KNN, which are representative of regression-based and analogy-based estimators, respectively. Moreover, as a sanity check we always compare all the approaches to RG for all RQs. For each of these techniques we use the `R` tool[5] version 3.4.1. For CART, KNN, NB and RF, we build and tune a model for each LOO training set within each dataset, and use it to predict the effort of the target observation. We use the function `trainControl` available from the `R` package `Caret`[6] version 6.0.84, which performs a search to identify machine learning settings that generalise best on the training set[7], as recommended in recent work [25], [47]. In the following we briefly describe each of the techniques.

Random Guessing (RG) is a worst case lower bound benchmark suggested to assess the usefulness of a prediction system [12]. It randomly assigns the y value of another case to the target case. More formally, it is defined as: predict a y for the target case t by randomly sampling (with equal probability) over all the remaining $n - 1$ cases and take $y = r$ where r is drawn randomly from $1 \ldots n^r = t$ [12]. Any prediction system should outperform RG since an inability to predict better than random implies that the prediction system is not using any target case information.

Classification and Regression Trees (CART) are machine learning methods used to build prediction models by recursively partitioning the data and fitting a simple prediction model within each partition [48]. The partitioning can be represented graphically with a decision tree. Decision trees where the dependent variable takes a finite set of values are called classification trees, while those where the dependent variable takes continuous values are called regression trees.

5. https://www.r-project.org
6. http://topepo.github.io/caret/index.html
7. Specifically, we used the setting `method=repeatedcv, repeats=30` (and `tuneLength = 10` for KNN). Since more advanced tuning techniques can be used [20], [45], [46], the results provided herein can be considered as a lower bound.

K-Nearest Neighbor (KNN) is an analogy-based approach that, given a target instance (i.e. a software project characterized by a vector of n features), retrieves from a case base of past projects, those instances which are relevant to the target one [49]. These relevant cases are identified by using the Euclidean distance as a similarity function, which measures the distance between the target case and the other cases based on the values for the n features of these projects. The average of the effort values of the k most similar past projects is then used as the effort predicted for the target project. If there are ties for the k-th nearest vectors, all candidates are used to compute the average. The choice of k is left to the user, in this work we experiment with different values of $k = 1, ..., 10$.

Linear Programming for Effort Estimation (LP4EE) is a baseline prediction model recently proposed to provide a robust yet easy-to-use approach for effort estimation[8] [13]. The model takes advantage of the Simplex algorithm, which deterministically minimizes an error function on a training set, and applies the learnt weights on a test set to make predictions [13]. In this paper we extend the original formulation of LP4EE [13] to handle negative-values predictions (see Appendix A). The original version is used in RQ3 (to predict the effort) whereas, the modified version is used in RQ2 (to predict the MisestimationMagnitude which can take the form of both positive and negative values).

Naïve Bayes (NB) is a statistical technique that uses the combined probabilities of the different attributes to predict the target variable, based on the principle of Maximum A Posteriori [50]. This approach is naturally extensible to the case of having more than two classes, and was shown to perform well in spite of the underlying simplifying assumption of conditional independence.

Random Forest (RF) is an ensemble technique which aggregates the predictions made by a collection of decision trees (each with a subset of the original set of attributes) [51]. Each tree infers a split of the training data based on feature values to produce a good generalization. RF can naturally handle binary or multiclass classification problems. The leaf nodes refer to either of the classes concerned.

3.4 Validation Approach

For each of the datasets in our study, we perform a Leave-One-Out (LOO) cross-validation for all RQs. Given a dataset containing n observation, one observation at time is used as target and the remaining $n - 1$ instances are used to train the model; the process is repeated n times. Thus, for each dataset, we obtained n pairs of training data and test data, and we report the results obtained on the test data by using boxplots, summary statistics, and statistical tests as detailed in Section 3.5. LOO is a deterministic approach that, unlike other cross validation techniques, does not rely on any random selection to create the training and testing sets. According to recent work [14], assessment via LOO eliminates conclusion instability caused by random sampling, making evaluations that use it more easily reproducible. However, if chronological information about the projects is available it would be preferable to adopt a time-based validation approach, because LOO may give more optimistic results than those that might realistically be achieved in practice [52]. In our study we use LOO because start and completion dates are not available for all projects.

3.5 Evaluation Criteria and Statistical Tests

In order to evaluate the performance of the techniques considered in RQ1 (i.e. CART, KNN, NB and RF) to classify misestimation types and severity we used the Area Under the ROC Curve (AUC-ROC) [53], which value ranges between 0 and 1. For a two-class problem (such as classifying error types) an AUC-ROC value of 1 represents a perfect classifier, while an area of 0.5 represents a Random (i.e. worthless) one. To evaluate the performance for a multi-class problem (such as the classifying errors' severity) we exploited the generalization to n-class classification proposed by Hand and Till [54] which extends the AUC definition to the case of more than two classes by averaging pairwise comparisons. In this case a value of 1 still represents a perfect classifier, while an area of $1/n$ represents a Random classifier, where n is the number of classes considered.

In order to compare the performance of the estimation methods analysed for RQ2 and RQ3 (i.e. CART, KNN, LP, LFM and Expert), we measured the Absolute Error (i.e. $|PredictedValue - RealValue|$), where the $RealValue$ is our target prediction variable. This target prediction variable is equal to MisestimationMagnitude for RQ2 and it is equal to ActualEffort for RQ3. Thus, for RQ2 we measure the Absolute Error between the human expert Misestimation-Magnitude and the one predicted by LFM; while to answer RQ3 we compute the absolute error between the actual effort and the effort predicted by LFM, human expert, and the considered ML. We use boxplots to visualise the difference in performance among different prediction methods and also use significance statistical tests. Both the boxplots and the statistical tests are based on these distributions.

In order to evaluate whether the differences in performance of the classifiers used in RQ1 are significant, we use the Friedman Test[9] [55]. This is a non-parametric test which works with the ranks of the techniques rather than their actual performance values, making it less susceptible to the distribution of the performance of these parametric values. The null hypothesis that is tested in our work is the following: "There is no significant difference in the AUC-ROC values obtained by the approaches compared", at a confidence limit, α, of 0.05. If the null-hypothesis is rejected, then it can be concluded that at least two of the techniques are significantly different from each other. When a significant difference is found, the Nemenyi test [56] is often recommended as a post-hoc test to identify the techniques with a statistically significant difference[9] [57]. The performance of two classifiers is thought to be significantly different if the corresponding average ranks differ by at least the critical distance (CD) [57]. The results of this test are presented in a diagram which is used to compare the performance of multiple techniques by ranking them. It consists of an axis, on which the average ranks of the methods are plotted and

8. The source code is available at https://github.com/fedsar/LP4EE

9. The `R` package `stats` (version 3.6.1) was used for the Friedman and the Wilcoxon Signed-Rank tests, the `R` package `PMCMRplus` (version 1.4.2) was used for the Nemenyi Test.

of the CD bar. The groups of classifiers whose values are significantly different, are not connected by a line.

To establish if the estimates of one method are statistically significantly better than the estimates provided by another method (RQ2 and RQ3), we compare the absolute errors they achieved for each of the datasets. In particular, to check for statistical significance, we use the Wilcoxon Signed-Rank Test [58][9], which is a safer test to apply than parametric tests, since it raises the bar for significance, by making no assumptions about underlying data distributions. In particular, we test the following Null Hypothesis: "The absolute errors provided by the prediction model P_i are not significantly lower than those provided by the prediction model P_j.", set the confidence limit, α, at 0.05 and applied the Bonferroni correction (α/K, where K is the number of hypotheses) when multiple hypotheses were tested. The Bonferroni correction is the most conservatively cautious of all corrections and its usage allows us to avoid the risk of Type I errors (i.e. incorrectly rejecting the Null Hypothesis and claiming predictability without strong evidence). In order to investigate the effect size of the Wilcoxon Signed-Rank Test results, we compute the correlation coefficient $r = \frac{Z}{\sqrt{N}}$, where Z is the standard score of the Wilcoxon test and N is the number of pair observations. Indeed, r is recommended as an effect size measure for paired non-parametric statistical significance tests [59]. The r effect is considered small $\geq$0.10, medium $\geq$0.30, large $\geq$ 0.50 and very large $\geq$ 0.70 [60], [61].

3.6 Threats to Validity

In this section we discuss the construct, conclusion, and external threats to the validity of our empirical study.

To satisfy construct validity, a study has "to establish correct operational measures for the concepts being studied" [62]. This means that the study should represent to what extent the predictor and response variables precisely measure the concepts they claim to measure [63]. Thus, the choice of the features and how to collect them represents a crucial aspect. We tried to mitigate such a threat by using real-world data previously used to empirically evaluate effort estimation methods. We mitigate threats arising from unrealistic or incorrect data usage by only considering software projects for which the cost-drivers were collected and measured before human experts made the predictions, and were never modified afterwards, so that they can be correctly used as independent variables in machine learning prediction systems. Moreover, we considered only those projects for which the human-estimated efforts were made solely based on human expert judgement (i.e. no other technique was used to support experts in their estimation). Given that our approach aims to adjust and enhance the expert's final estimate, we need to use projects where the expert's estimation is provided. However, not all projects contain also chronological information and we had to use the LOO validation, which may lead to more optimistic results with respect to a time-based validation [52].

With regards to the conclusion validity, we carefully applied the statistical tests, verifying all the required assumptions and correcting for multiple hypotheses statistical testing. We also followed recent best practice to assess prediction systems [12], [13], [64]. Moreover, we used datasets of different sizes to mitigate the threats related to the number of observations. We also used traditional ML techniques implemented in publicly available tools to allow for replications and comparisons.

To mitigate threats to external validity we used six real-world industrial datasets containing software projects related to different application domains and companies, which are thus characterised by various project and human factors such as development process, developer experience, tools and technologies used, cost drivers, time and budget constraints [65]. Although we used a set of subjects that has such a degree of diversity, we cannot claim that our results generalise beyond the subjects studied. It is worth noting that the formulation of the approach is independent from the nature of the projects. That is, the approach could potentially work with any kind of project as long as they can be characterised in terms of the same (or a subset of) cost drivers used for describing the past projects stored in the database. In our empirical study we experimented with both new development and maintenance projects, and we study both the effort of realizing entire projects and specific software modules, in order to assess the feasibility of LFM for a wide range of projects type. While we observe that LFM performs generally well with different techniques, some of them might be preferable depending on the size of the dataset. Also, results obtained with datasets of small size should be taken with caution as the number of projects might not yield to a relevant statistical analysis, as in the case of the Telecom dataset.

4 Empirical Study Results

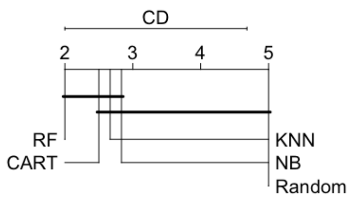
In this section we report and discuss the results we obtained carrying out the empirical study described in Section 3.

4.1 RQ1. Predicting Type/Severity of Human Expert Misestimations

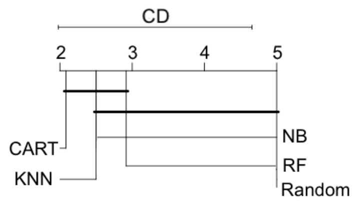
To address RQ1 we compare the performance of CART, KNN, NB and RF for predicting the type and the severity of human expert estimate errors. The accuracy results, measured by the AUC-ROC, are summarised in Table 3.

RQ1.1-Predicting human expert misestimation type: From Table 3 we can observe that all techniques outperform the random classifier (i.e. AUC-ROC < 0.5) in all of the cases studied with an average AUC-ROC across all techniques and datasets equal to 0.71. Moreover, results show that RF obtains the highest AUC-ROC values on three out of the six datasets under study (i.e. `ISBSG-FP`, `KP`, `Medical`) with AUC-ROC values ranging from 0.65 to 0.92. This conclusion is reinforced by the Friedman Test as it shows statistically significant difference (p-value $<$ 0.001) between the performance of the techniques studied (CART, KNN, NB, RF, RG). Nemenyi's Critical-Difference (shown in Figure 2a) also supports this, ranking RF first and Random last with a statistically significant difference (p-value $<$ 0.001).

RQ1.2-Predicting human expert misestimation severity: Similar observations hold when we consider the prediction of MisestimationSeverity. Results show that all techniques always provide better AUC-ROC values than random classification (i.e. AUC-ROC $>$ 0.33), with an average



(a) MisestimationType



(b) MisestimationSeverity

Fig. 2: RQ1: Critical Difference (CD) diagram of the post-hoc Nemenyi test with α = 0.05. The difference between two methods is significant if the gap between their ranks is larger than the critical distance. There is a line between two methods if the rank gap between them is smaller than the critical distance.

Target Variable	Dataset	CART	KNN	NB	RF
MisestimationType (binary class)	`ISBSG-C`	**0.75**	0.63	0.62	0.56
	`ISBSG-FP`	0.61	0.59	0.61	**0.65**
	`KD`	0.71	**0.85**	0.62	0.63
	`KP`	0.59	0.68	0.71	**0.82**
	`Medical`	0.86	0.88	0.91	**0.92**
	`Telecom`	**1.00**	0.64	0.57	0.65
MisestimationSeverity (multi-class)	`ISBSG-C`	**0.95**	0.54	0.66	0.65
	`ISBSG-FP`	0.53	**0.57**	0.53	0.56
	`KD`	**0.86**	0.69	0.79	0.72
	`KP`	0.64	**0.68**	0.62	0.60
	`Medical`	0.64	0.65	**0.73**	0.67
	`Telecom`	**1.00**	0.93	0.83	0.83

TABLE 3: RQ1: AUC-ROC values obtained by CART, KNN, NB and RF when predicting the type and the severity of human expert misestimations.

AUC-ROC value of 0.70 of all techniques across all datasets. Results also show that CART obtains the highest AUC-ROC values on three out of six datasets (ISBSG-C, KD and Telecom), whereas KNN performs best on two of the remaining datasets (ISBSG-FP and KP) and NB performs best on the remaining dataset. The Friedman Test also concludes a difference in the predictors' performance (p-value <0.001) with Nemenyi's Critical-Difference Diagram (shown in Figure 2b) ranking CART first and Random last with a statistically significant difference when comparing the two. On the other end, KNN, NB and RF rank second, third and fourth, respectively, with the gap between their ranks and Random not being larger than CD.

Therefore, in answering to RQ1 we can state that:

Answer to RQ1: The type and severity of human expert misestimations are predictable with an average AUC-ROC value of all techniques (over all datasets) being equal to 0.71 for type and 0.70 for severity.

4.2 RQ2. Predicting the Magnitude of Human Expert Misestimations

To answer RQ2, we investigate the capability of traditional regression- and analogy- based estimation approaches (i.e. CART, KNN, LP and RF) to predict the MisestimationMagnitude of human expert misestimates.

Figure 3 shows the boxplots of the distribution of the absolute prediction errors produced by CART, KNN, LP and RF as well as the sanity check, RG.

We can observe that all techniques are able to predict, with a low absolute error, the magnitude of the error committed by human experts when estimating software effort. This can be seen from the boxplots of each dataset where the median of the best technique does not exceed an absolute error of 0.25 on all datasets. Results also show that the median absolute error of all ML techniques over all datasets is also low, with an average equal to 0.28.

Figure 3 shows that all techniques outperform RG on four (out of six) datasets. Whereas, on the remaining two datasets (`ISBSG-C` and `KD`), at least two of the machine learners (namely KNN and LP) achieve better results than RG, with no technique being worse. The Wilcoxon test results (reported in Table 4) also support this conclusion as they show that all techniques are statistically significantly better than RG on four out of six datasets (i.e. `ISBSG-FP`, `KP`, `Medical`, `Telecom`), with 13% of these cases having a very large effect size, 31% having a large effect size, 50% having a medium and 6% small effect sizes. Whereas, on the `ISBSG-C` dataset KNN and LP significantly outperform RG, and for the other two techniques, the null hypothesis cannot be rejected as well as for the `KD` dataset.

These results highlight that the magnitude of misestimations is indeed predictable for all datasets considered. Moreover, for each of the datasets we can identify the best performing approach based on its median absolute errors (as shown by the bar in the boxplots - Figure 3) and the number of times it is statistically significantly better than the other ones according to the Wilcoxon test and effect size (Table 4). In RQ3 we will refer to this approach as LFM.

Based on the results above we can conclude that:

Answer to RQ2: The misestimate magnitude is highly predictable with an average value of the median absolute errors (MedAE) obtained by all techniques, across all datasets, being equal to 0.28.

4.3 RQ3. Enhancing Software Effort Estimates via LFM

To address RQ3, we investigate the capability of LFM to improve human expert estimates. Figure 4 shows the boxplots of the distributions of absolute error values obtained by LFM, human experts and the automated estimators (i.e. CART, KNN, LP and RF) when predicting the effort of software projects.

RQ3.1 LFM vs. Random Guessing (RG): The improvements achieved by LFM over RG (as shown in Figure 4) are always statistically significant ($p < 0.001$) with five very large ($r \geq 0.7$) effect sizes and a large one ($r \geq 0.5$) (see

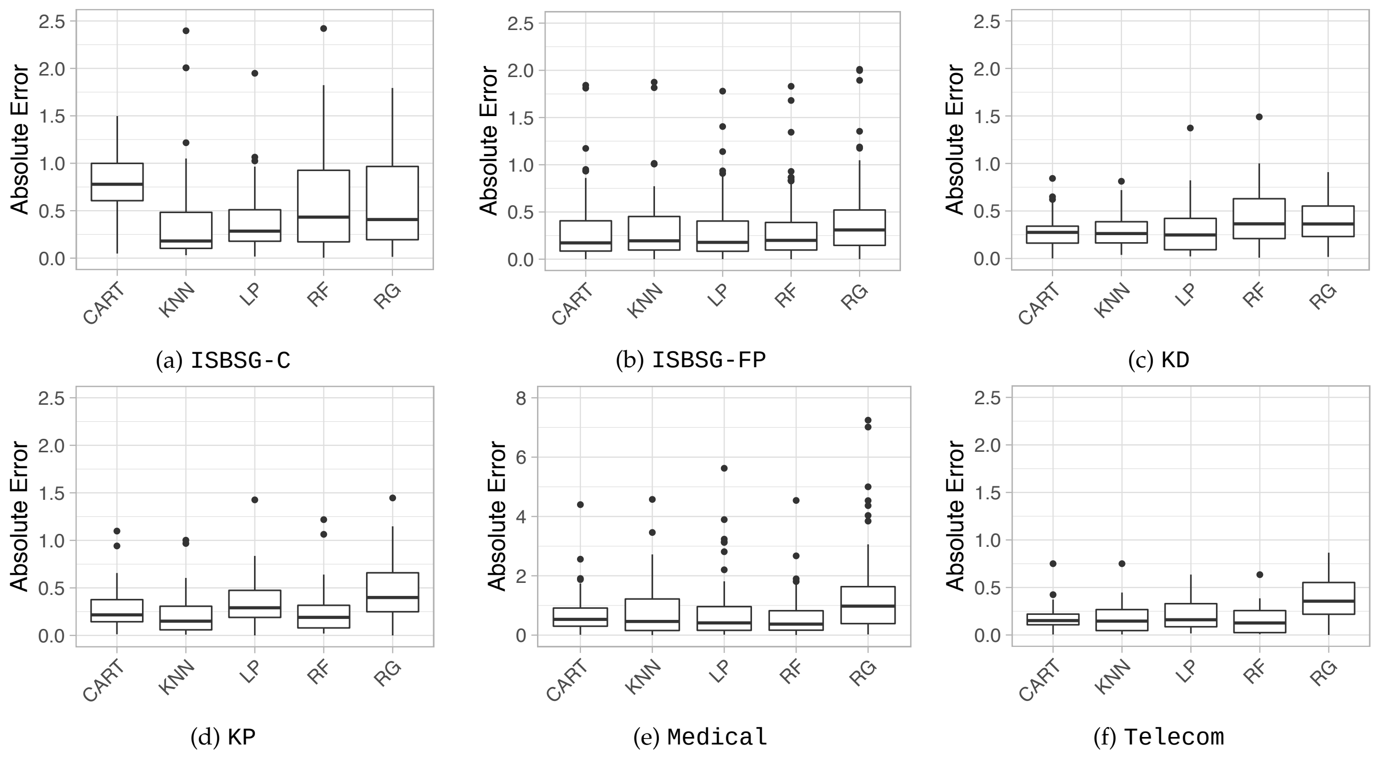


Fig. 3: RQ2. Boxplots of the absolute errors achieved by CART, KNN, LP, RF and RG when predicting the magnitude of human expert misestimations.

Table 5). Thereby LFM successfully passes our sanity check of beating Random Guessing.

RQ3.2 LFM vs. Traditional ML Estimators: The analysis of the boxplots of the absolute errors (see Figure 4) reveals that our proposed algorithm, LFM, not only outperforms RG, but it also performs better than all the other machine learning methods against which we compare it, on five (out of the six) datasets. That is, the absolute error values provided by LFM are lower than those provided by CART, KNN, LP and RF in 21 out of the 24 cases considered. These observations are confirmed by inferential statistical analysis, the results of which are presented in Table 5; the improvement of our algorithm over these four techniques is statistically significant and the effect size is very large ($r \geq 0.7$) in six of these comparisons, large ($r \geq 0.5$) in 13, medium ($r \geq 0.3$) in one and small ($r \geq 0.1$) in two of them. As for the remaining dataset (`Medical`), LFM performs statistically significantly better than CART (as shown by the statistical test), with effect size equal to 0.29.

Therefore, we can positively answer to RQ3.2: The use of LBM allows us to obtain significantly more accurate estimates than traditional ML techniques for 88% of the cases.

RQ3.3 LFM vs. Human Expert Judgement: From the boxplots in Figure 4 we observe that LFM enhances the original human expert estimates for all the datasets by providing the lowest absolute errors. Results of the Wilcoxon test (see Table 5) reveal that it achieves statistically significant better estimations (all p-values being less than 0.035) on four out of the six datasets with two of them having a medium effect size ($r \geq 0.3$) and the other two having a small one ($r \geq 0.1$). For the other two datasets (i.e. `KD` and `Telecom`), LFM obtains lower absolute error values than human expert estimates (see Figure 4), however the difference is not statistically significant on these datasets, which contain the smallest number of observation (17 and 29, respectively). Moreover, the average relative errors across all datasets obtained by LFM are up to 33% lower than those resulting from the estimations made by the human expert[10].

These results show that LFM is not only better than alternative automated techniques, but that it also has an edge over purely human expertise alone. Therefore, we can positively answer to RQ3.3: The use of LFM allows us to improve human expert estimates.

Based on the results above, we can conclude that:

Answer to RQ3: LFM improves expert judgement on all datasets with improvements that are statistically significant on four out of the six datasets studied.

5 Related Work

Previous research has been carried out to support engineers in estimating software development effort, focusing on the following aspects:

– Improving the accuracy of software effort estimates by proposing and comparing a large number of techniques such as regression and analogy-based [68],

10. MMRE should not be used as the only indicator to compare prediction models as it can be misleading (see e.g. [66], [67]), we use it only to provide a notion of the error with respect to the actual effort.

Dataset	Technique	vs. CART	vs. KNN	vs. LP	vs. RF	vs. RG
ISBSG-C	CART	-	1.000 (0.00)	1.000 (0.00)	0.862 (0.02)	0.905 (0.02)
	KNN	<0.001 (0.51)	-	0.201 (0.18)	0.001 (0.47)	0.001 (0.49)
	LP	<0.001 (0.73)	0.802 (0.04)	-	0.003 (0.43)	0.036 (0.30)
	RF	0.140 (0.21)	0.999 (0.00)	0.997 (0.00)	-	0.547 (0.09)
ISBSG-FP	CART	-	0.126 (0.11)	0.996 (0.00)	0.755 (0.02)	<0.001 (0.37)
	KNN	0.874 (0.01)	-	0.964 (0.00)	0.951 (0.00)	<0.001 (0.29)
	LP	0.004 (0.21)	0.036 (0.15)	-	0.382 (0.06)	<0.001 (0.39)
	RF	0.245 (0.08)	0.049 (0.14)	0.619 (0.04)	-	<0.001 (0.38)
KD	CART	-	<0.001 (0.74)	0.517 (0.12)	0.007 (0.50)	0.111 (0.30)
	KNN	0.999 (0.00)	-	0.710 (0.07)	0.088 (0.32)	0.221 (0.23)
	LP	0.492 (0.13)	0.297 (0.19)	-	0.008 (0.49)	0.078 (0.33)
	RF	0.994 (0.00)	0.916 (0.02)	0.992 (0.00)	-	0.703 (0.07)
KP	CART	-	0.999 (0.00)	0.032 (0.34)	0.969 (0.01)	0.005 (0.44)
	KNN	0.001 (0.53)	-	0.001 (0.54)	0.049 (0.31)	<0.001 (0.70)
	LP	0.969 (0.01)	0.999 (0.00)	-	0.999 (0.00)	0.020 (0.37)
	RF	0.032 (0.34)	0.953 (0.01)	0.001 (0.51)	-	<0.001(0.62)
Medical	CART	-	0.790 (0.03)	0.320 (0.12)	0.999 (0.00)	<0.001 (0.46)
	KNN	0.211 (0.15)	-	0.461 (0.09)	0.987 (0.00)	<0.001 (0.45)
	LP	0.682 (0.05)	0.542 (0.07)	-	0.987 (0.00)	<0.001 (0.47)
	RF	0.001 (0.39)	0.014 (0.29)	0.013 (0.29)	-	<0.001 (0.56)
Telecom	CART	-	0.663 (0.11)	0.482 (0.17)	0.897 (0.03)	0.010 (0.62)
	KNN	0.363 (0.22)	-	0.373 (0.22)	0.824 (0.05)	0.010 (0.62)
	LP	0.537 (0.15)	0.644 (0.11)	-	0.785 (0.07)	0.005 (0.67)
	RF	0.112 (0.39)	0.189 (0.32)	0.229 (0.29)	-	<0.001 (0.96)

TABLE 4: RQ2: Results of the Wilcoxon test (p-value and r effect size) comparing the absolute errors provided by CART, KNN, LP and RF vs. each other and vs. RG when predicting the human expert MisestimationMagnitude.

	vs. Expert	vs. CART	vs. KNN	vs. LP	vs. RF	vs. RG
ISBSG-C	0.002 (0.44)	<0.001 (0.66)	<0.001 (0.66)	<0.001 (0.58)	<0.001 (0.62)	<0.001 (0.75)
ISBSG-FP	0.011 (0.19)	<0.001 (0.65)	<0.001 (0.62)	<0.001 (0.55)	<0.001 (0.61)	<0.001 (0.71)
KD	0.449 (0.14)	<0.001 (0.72)	0.001 (0.60)	0.002 (0.57)	0.003 (0.55)	<0.001 (0.97)
KP	0.035 (0.33)	<0.001 (0.82)	0.001 (0.55)	0.005 (0.44)	<0.001 (0.65)	<0.001 (0.91)
Medical	0.013 (0.29)	0.014 (0.29)	0.963 (0.01)	0.971 (0.00)	0.988 (0.00)	<0.001 (0.60)
Telecom	0.132 (0.37)	<0.001 (0.96)	<0.001 (0.96)	0.002 (0.74)	<0.001 (0.96)	<0.001 (0.94)

TABLE 5: RQ3: Results of the Wilcoxon test (p-value and r effect size) comparing the absolute errors obtained by LFM vs. those obtained by human experts and traditional automatic learners (i.e. CART, KNN, LP, RF) when predicting software projects' effort.

[69], machine learning [70], ensemble [71], search-based [8], [72].

– Experimenting with, and comparing, different size measures as cost drivers (e.g. [36], [73], [74], [75], [76]).
– Experimenting with, and comparing, within- vs. -cross company data (e.g [77], [78], [79], [80]).
– Investigating estimate uncertainty (e.g. [3], [11], [21], [81]) and prediction intervals (e.g. [82], [83], [84], [85], [86], [87], [88]).
– Studying human bias in effort estimates (e.g. [89], [90], [91], [92], [93]).

A comprehensive review on the use of expert judgement, formal models and their combination can be found elsewhere [10]. In the following, we focus on those studies that have investigated human bias in predicting task duration, and specifically in predicting software development effort.

Predicting task duration has been the focus of a lot of research in different fields. Despite the different nature of tasks under examination, previous studies almost universally show that human predictions tend to be biased. For example, studies in psychology and human cognition show that humans might be subject to the phenomenon of the central tendency of judgement, which describes the tendency for humans to over-estimate small tasks and under-estimate large ones [94], as well as to the phenomenon of planning fallacy, which is the human tendency to underestimate future task duration despite knowing that previous similar tasks could not be completed on time [95]. This kind of bias has been later attributed to misremembering previous task duration (i.e. memory-bias) and using such a duration as a basis for future predictions. For example, Roy and Christenfeld [96] studied whether a systematic memory-bias has an effect, or could explain, a similar systematic bias in prediction, and showed that people tend to underestimate the duration of future events because they based their estimation on the perceived duration rather than actual duration of similar events that had occurred in the past. Subsequent studies have shown that this prediction bias can be reduced when feedback about previous task duration is provided, thus refreshing and ultimately correcting the memory (see e.g. [97]).

The principle of human bias in predictions has also been studied, from different perspectives, for the task of software effort estimation, as detailed below.

Surveys on estimation practice in the software industry found that human effort estimates are over-optimistic [3], [4] and there is a strong over-confidence in their accuracy [5]. A recent survey on agile practice also revealed that half of the respondents believe that their effort estimates on average are under/over estimated by an error of 25% or more [6].

Other studies have looked into possible reasons for bias mainly basing their investigation on statistical analysis of project characteristics and questionnaires posed to project managers [89], [91], [92], [93], [98]. Lederer and Prasad [89] found that the main cause of misestimates was from

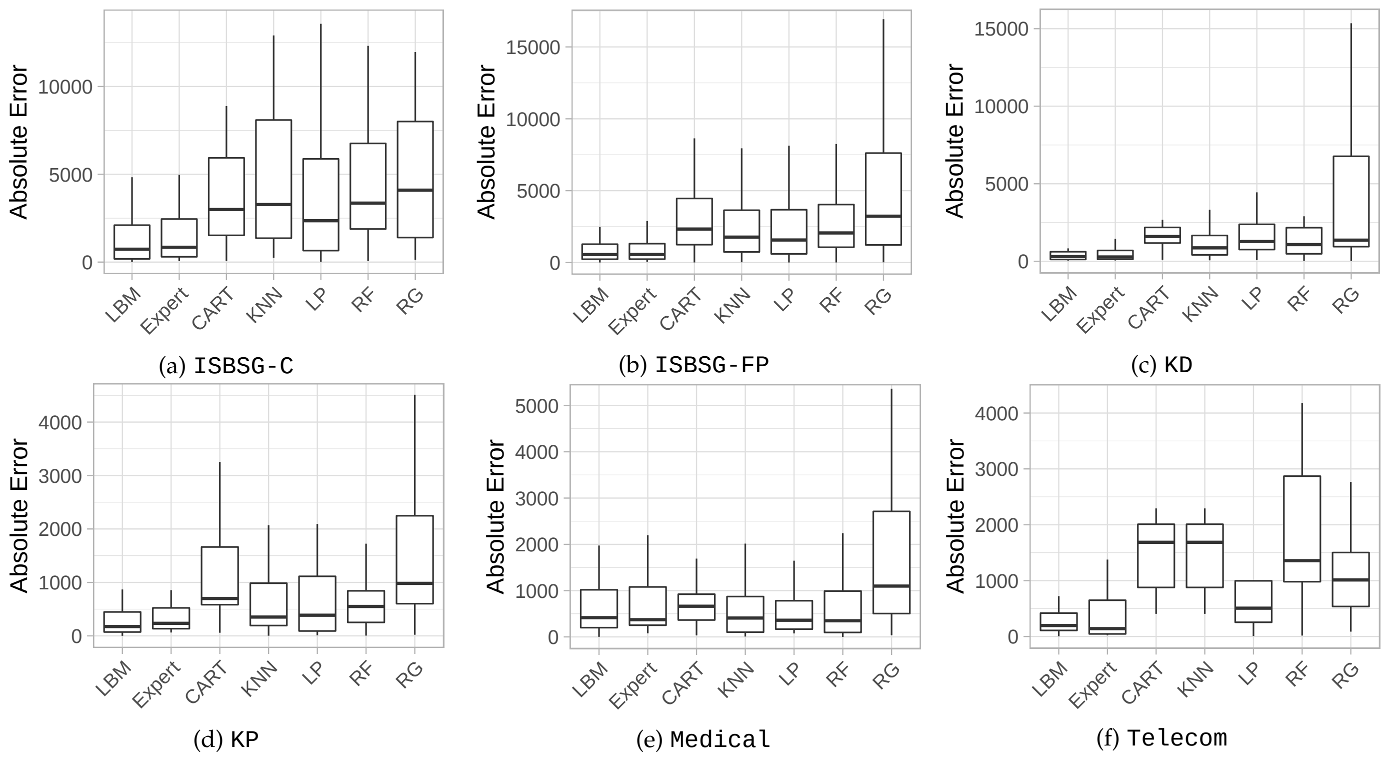


Fig. 4: RQ3. Boxplots of absolute errors obtained by LFM, Human Experts and traditional automatic estimators (i.e. CART, KNN, LP, RF) when predicting software projects' effort.

the management control side. Specifically, the lack of tasks like consideration in performance reviews as to whether estimates were met, project control comparing estimates and actual performance, and careful examination of the estimate by the management of the information system department resulted in inaccurate estimates. Whereas Gray et al. [91] used contingency table analysis, logistic regression and log-linear modeling to prove that the expert-derived effort prediction used to develop a collection of modules from a large health-care system showed systematic biases involving the size and type of the modules understudy. Jørgensen [92] investigated the accuracy and bias variation of effort estimates through the use of a regression analysis-based model. The study analyses 49 software tasks from a single organisation using collected information about variables that were believed to have an effect on accuracy or the bias of the estimates. The results highlight that several factors influence the increase of error in estimates, such as the estimates being provided by a software developer rather than a project leader or the customer prioritizing time-to-delivery as opposed to quality or cost. Jørgensen and Moløkken-Østvold [98] also studied differences in types of reasons for estimation error depending on the role of the estimators, data collection approach, and analysis technique with results showing that all three types of reasons play a major role in estimate inaccuracies. In a more recent study, Jørgensen and Grimstad [93] examined the relationship between biases resulting from effort estimates produced by software developers, and developer cultural dimensions such as the way one sees oneself, the thinking style, nationality, experience, skill, education, sex, and organizational role. Results showed that estimation bias was present along most of the studied dimensions and that there was a strong correlation between estimation bias and the developer level of interdependence.

While the aforementioned work has focused on reasons for human bias to support experts in making more accurate and realistic estimates, our study uses machine learning predictions of human bias to automatically adjust and enhance the expert's final estimate of the overall effort which, to the best of our knowledge, has not been explored yet. This concept of using error to adjust future estimates has been used by Kultur et al. [99], however it has been applied to adjust for errors resulting from machine estimates of effort. Their work propose an ensemble of neural networks with associative memory (ENNA) and take the machine learner's estimation bias into account by using KNN to retrieve past projects that are similar to the new one. The estimated bias of the nearest neighbors ensemble is calculated as the average of the differences between the actual and the estimated values for those projects. This bias is then added to the estimated effort of the new project. Results show that ENNA provides estimates that are significantly better than neural networks and regression trees. On the other hand, we propose the use of automated models in order to predict the errors made by human experts (rather than predicting the effort) when estimating effort and exploiting this information to adjust their final estimates. Rather than seeking solely to compete with (or even replace) human experts, our approach aims to use machine learners to learn, from both traditional past projects cost drivers and from past expert judgements, essentially building into the predictive model the ability to learn from their past estimation errors. Although a part of our work also proposes to adjust the final estimates by taking into account or adding an error/bias (Phase 3), the main idea behind our proposed approach is to automatically learn from and predict human expert error, and to show that these predictions can be exploited to help experts improve their final estimates.

6 Conclusions and Future Work

In this paper we showed that we can learn from past misestimations in order to improve future estimates; the Learning From Mistakes (LFM) approach.

We demonstrated the effectiveness of LFM with an empirical study involving human expert effort estimates (and related errors) from 402 industrial software projects developed by several different companies. Our results reveal that it is possible to predict the type, the severity, and the magnitude of the mistakes made by human experts when estimating software effort, and that we can successfully exploit this information to significantly improve future human expert-based estimates.

The LFM approach proposed in this paper can be applied by following these three steps:

1) Maintain a database of projects for which both the effort estimated and the one actually needed to realise the project have been recorded, together with the cost drivers characterising the project (e.g. functional size measures);
2) For any target project: use LFM to classify the type and severity of the errors human experts are likely to make when estimating the effort needed for target projects based on the information stored in the database (our findings from RQ1 show that this is possible); use LFM to quantify the estimate error for target projects based on the magnitude of human expert past misestimations (our findings from RQ2 show that this is possible);
3) Enhance the human expert effort estimates based on the past errors learned from Step 2 (our findings from RQ3 show that such an enhancement can be statistically significant as in the case of the projects we considered herein).

Our LFM approach provides two routes to support human experts. One of these is the more traditional, and widely-studied route of improved effort estimation, but the other is the less explored one of providing automatic feedback on the experts' own judgements, rather than seeking to second-guess them. For example, one can investigate whether using information gathered from past misestimations as a cost driver can enhance the accuracy of traditional automated predictive models. While, following the second route, one could investigate the way in which machine learners model human expert misestimates (e.g. do bias models exhibit similar bias trends as those observed in human predictions when estimating effort?) to further provide useful insights to the experts. Once adopted in practice, one could also observe how the use of LFM can affect human expert judgement, over time, in estimating effort, and how often LFM would need to be re-trained to promptly reflect potential difference in behaviour and to prevent model staleness.

Future work can also explore the applicability of LFM to other Software Engineering predictive tasks (e.g. bug fixing time prediction [100], customer ratings prediction [101], code review recommendation [102], software vulnerabilities predictions [103]).

Acknowledgments

This research is supported by the ERC Advanced fellowship grant EPIC (741278). The authors would like to thank Barbara Kitchenham, Magne Jørgensen, and Stephen G. MacDonell for providing part of the data used in this study.

Appendix

In this appendix we explain the mathematical formulation of the Linear Programming model we used in RQ2 to predict the MisestimationMagnitude.

Linear Programming (LP) [104] aims to achieve the best outcome from a mathematical model with a linear objective function subject to linear equality and inequality constraints. The feasible region is given by the intersection of the constraints and the Simplex (linear programming algorithm) is able to find a point in the polyhedron where the function has the smallest value (minimisation) in polynomial time.

Here, we generalize the model proposed for the effort estimation by Sarro and Petrozziello [13]. In the original implementation, the model is subject to an inequality constraint imposing that the value estimated for each of the observations in the training set has to fall in R_0^+. Here, we remove the inequality constraints allowing the model to use both positive and negative feature values as well as to optimize for both positive and negative values, as follows:

$$\begin{aligned} \text{minimise} \quad & \sum_{i=1}^{n} \Big| \sum_{j=1}^{m} a_{ij} x_j - \mathit{ActualValue}_i \Big| \\ & x_j \text{ free}, \qquad j = 1, \dots, m \end{aligned} \tag{1}$$

where a_{ij} represents the coefficient of the j^{th} feature for the i^{th} project, x_j is the value of the j^{th} feature, and $\mathit{ActualValue}_i$ is the actual effort of the i^{th} project.

Due to the non-linearity of the absolute value function, the above model has been linearised as follows:

$$\begin{aligned} \text{minimise} \quad & \sum_{i=1}^{n} t_i \\ \text{subject to} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} x_j - \mathit{ActualValue}_i - t_i \leq 0 \\ & \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} x_j - \mathit{ActualValue}_i + t_i \geq 0 \\ & x_j \text{ free}, \qquad j = 1, \dots, m \\ & t_i \text{ free}, \qquad i = 1, \dots, n \end{aligned} \tag{2}$$

Let $X_i, \forall i$ be the part of Eq. (1) wrapped in the absolute value. $\forall i$, the slack variable t_i and the following two constraints have been added to the model: $X_i \leq t_i$ and $-X_i \leq t_i$. Therefore we can have one of the following cases:

$X_i > 0$ The second constraint, $-X_i \leq t_i$, is always fulfilled as $-X_i$ is negative and t_i is implicitly ≥ 0. Since t_i is minimised by the objective function and $0 \leq X_i \leq t_i$, the first constraint, $X_i \leq t_i$, is satisfied and t_i is $abs(X)$.

$X_i < 0$ The first constraint, $X_i \leq t_i$, is always fulfilled as X_i is negative and t_i is implicitly ≥ 0. Since t_i is minimised by the objective function and $0 \leq -X_i \leq t_i$, the second constraint, $-X_i \leq t_i$, is satisfied and t_i is $abs(X)$.

$X_i = 0$ Both constraints are always fulfilled since t_i is implicitly ≥ 0. Since t_i is minimised by the objective function, $0 = X_i = t_i$. So t_i is $abs(X)$.

References

[1] L. C. Briand and I. Wieczorek, "Resource estimation in software engineering," *Encyclopedia of software engineering*, 2002.

[2] A. Trendowicz and R. Jeffery, "Software project effort estimation," *Foundations and Best Practice Guidelines for Success, Constructive Cost Model, COCOMO*, pp. 277–293, 2014.

[3] K. Molkken and M. Jörgensen, "A review of surveys on software effort estimation," in *Proc. of ISESE'03*, 2003, pp. 223–230.

[4] 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.

[5] T. M. Gruschke and M. Jørgensen, "The role of outcome feedback in improving the uncertainty assessment of software development effort estimates," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 17, no. 4, pp. 1–35, 2008.

[6] M. Usman, E. Mendes, and J. Börstler, "Effort estimation in agile software development: a survey on the state of the practice," in *Proceedings of the 19th international conference on Evaluation and Assessment in Software Engineering*, 2015, pp. 1–10.

[7] S. McConnell, *Software estimation: demystifying the black art*. Microsoft press, 2006.

[8] F. Ferrucci, M. Harman, and F. Sarro, *Search-Based Software Project Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 373–399.

[9] M. Jørgensen and T. Halkjelsvik, "Sequence effects in the estimation of software development effort," *Journal of Systems and Software*, vol. 159, p. 110448, 2020.

[10] M. Jørgensen, "Forecasting of software development work effort: Evidence on expert judgement and formal models," *International Journal of Forecasting*, vol. 23, no. 3, pp. 449–462, 2007.

[11] S. G. MacDonell and M. J. Shepperd, "Combining techniques to optimize effort predictions in software project management," *Journal of Systems and Software*, vol. 66, no. 2, pp. 91 – 98, 2003.

[12] M. J. Shepperd and S. G. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.

[13] F. Sarro and A. Petrozziello, "Linear programming as a baseline for software effort estimation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, pp. 12:1–12:28, 2018.

[14] E. Kocaguneli and T. Menzies, "Software effort models should be assessed via leave-one-out validation," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1879–1890, 2013.

[15] M. Aly, "Survey on multiclass classification methods," 2005.

[16] F. Walkerden and R. Jeffery, "An empirical study of analogy-based software effort estimation," *Empirical software engineering*, vol. 4, no. 2, pp. 135–158, 1999.

[17] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 425–438, 2011.

[18] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Using tabu search to estimate software development effort," in *Proc. of MENSURA*. LNCS 5891, Springer, 2009, pp. 307–320.

[19] F. Ferrucci, C. Gravino, R. Oliveto, F. Sarro, and E. Mendes, "Investigating tabu search for web effort estimation," in *Proc. of EUROMICRO-SEAA'10*, 2010, pp. 350–357.

[20] A. Corazza, S. D. Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "Using tabu search to configure support vector regression for effort estimation," *EMSE*, vol. 18, no. 3, pp. 506–546, 2013.

[21] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective software effort estimation," in *Proc. of the 38th International Conference on Software Engineering ICSE*, 2016, pp. 619–630.

[22] M. Choetkiertikul, H. K. Dam, T. Tran, T. T. M. Pham, A. Ghose, and T. Menzies, "A deep learning model for estimating story points," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2018.

[23] A. Arcuri and L. C. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *STVR*, vol. 24, no. 3, pp. 219–250, 2014.

[24] W. B. Langdon, J. 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.

[25] 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.

[26] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 286–295.

[27] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 639–648, 1983.

[28] A. Abran, J. Desharnais, A. Lesterhuis, B. Londeix, R. Meli, P. Morris, S. Oligny, M. O'Neil, T. Rollo, G. Rule, L. Santillo, C. Symons, and H. Toivonen. (2015) The COSMIC Functional Size Measurement Method – Measurement Manual, version 4.0.1 In http://www.cosmicon.com/portal/public/MMv4.0.1.pdf.

[29] Ç. Gencel and O. Demirörs, "Functional size measurement revisited," *ACM Trans. Softw. Eng. Methodol.*, vol. 17, no. 3, 2008.

[30] H. van Heeringen and E. Van Gorp, "Measure the functional size of a mobile app: Using the cosmic functional size measurement method," in *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*. IEEE, 2014, pp. 11–16.

[31] F. Ferrucci, C. Gravino, P. Salza, and F. Sarro, "Investigating functional and code size measures for mobile applications," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, 2015, pp. 365–368.

[32] ——, "Investigating functional and code size measures for mobile applications: A replicated study," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2015, pp. 271–287.

[33] L. De Marco, F. Ferrucci, C. Gravino, F. Sarro, S. Abrahão, and J. Gómez, "Functional versus design measures for model-driven web applications: a case study in the context of web effort estimation," in *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metric (WETSoM)*, 2012, pp. 21–27.

[34] B. Marín, O. Pastor, and A. Abran, "Towards an accurate functional size measurement procedure for conceptual models in an MDA environment," *Data Knowl. Eng.*, vol. 69, no. 5, pp. 472–490, 2010.

[35] S. Abrahão, L. D. Marco, F. Ferrucci, J. Gómez, C. Gravino, and F. Sarro, "Definition and evaluation of a COSMIC measurement procedure for sizing web applications in a model-driven development environment," *Information and Software Technology*, vol. 104, pp. 144–161, 2018.

[36] S. Di Martino, F. Ferrucci, C. Gravino, and F. Sarro, "Web effort estimation: Function point analysis vs. COSMIC," *Information and Software Technology*, vol. 72, pp. 90–109, 2016.

[37] ISBSG. (2019) The international software benchmarking standards group. [Online]. Available: http://www.isbsg.org

[38] M. Fernández-Diego and F. G. L. Guevara, "Potential and limitations of the isbsg dataset in enhancing software engineering research: A mapping review," *Information and Software Technology*, vol. 56, 2014.

[39] B. Kitchenham, S. Lawrence Pfleeger, B. McColl, and S. Eagan, "An empirical study of maintenance and development estimation accuracy," *Journal of Systems and Software*, vol. 64, no. 1, pp. 57–77, 2002.

[40] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[41] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[42] P. Langley, W. Iba, K. Thompson *et al.*, "An analysis of bayesian classifiers," in *Aaai*, vol. 90, 1992, pp. 223–228.

[43] A. Charnes and W. W. Cooper, "Programming with linear fractional functionals," *Naval Research logistics quarterly*, vol. 9, no. 3-4, pp. 181–186, 1962.

[44] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[45] 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*, vol. PP, no. 99, pp. 1–1, 2018.

[46] 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 *Proc. of PROMISE'10*, 2010, pp. 4:1–4:10.

[47] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. ACM, 2016, pp. 321–332.

[48] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, ser. Statistics/Probability Series. Belmont, California, U.S.A.: Wadsworth Publishing Company, 1984.

[49] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge university press, 2007.

[50] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., 2005.

[51] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[52] B. Sigweni, M. Shepperd, and T. Turchi, "Realistic assessment of software effort estimation models," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 1–6.

[53] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[54] D. J. Hand and R. J. Till, "A simple generalisation of the area under the roc curve for multiple class classification problems," *Machine Learning*, vol. 45, no. 2, pp. 171–186, 2001.

[55] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[56] P. Nemenyi, "Distribution-free multiple comparisons (doctoral dissertation, princeton university, 1963)," *Dissertation Abstracts International*, vol. 25, no. 2, p. 1233, 1963.

[57] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[58] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. Lawrence Earlbaum Associates, 1988.

[59] R. Rosenthal, H. Cooper, and L. Hedges, "Parametric measures of effect size," *The handbook of research synthesis*, vol. 621, pp. 231–244, 1994.

[60] J. Cohen, *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[61] J. A. Rosenthal, "Qualitative descriptors of strength of association and effect size," *Journal of social service Research*, vol. 21, no. 4, pp. 37–59, 1996.

[62] B. Kitchenham, L. Pickard, and S. Pfleeger, "Case studies for method and tool evaluation," *IEEE Software*, vol. 12, no. 4, pp. 52–62, 1995.

[63] E. Mendes, S. Counsell, N. Mosley, C. Triggs, and I. Watson, "A comparative study of cost estimation models for web hypermedia applications," *EMSE*, vol. 8, no. 23, pp. 163–196, 2003.

[64] P. A. Whigham, C. A. Owen, and S. G. Macdonell, "A baseline model for software effort estimation," *ACM TOSEM*, vol. 24, no. 3, pp. 20:1–20:11, 2015.

[65] L. C. Briand and J. Wüst, "Modeling development effort in object-oriented systems using design properties," *IEEE TSE*, vol. 27, no. 11, pp. 963–986, 2001.

[66] 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.

[67] B. Kitchenham and E. Mendes, "Why comparative effort prediction studies may be invalid," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, ser. PROMISE. New York, NY, USA: ACM, 2009.

[68] L. C. Briand and I. Wieczorek, "Software resource estimation," *Encyclopedia of Software Engineering*, pp. 1160–1196, 2002.

[69] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE TSE*, vol. 23, no. 11, pp. 736–743, 2000.

[70] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 41–59, 2012.

[71] A. Idri, M. Hosni, and A. Abran, "Systematic literature review of ensemble effort estimation," *Journal of Systems and Software*, vol. 118, no. C, pp. 151–175, 2016.

[72] F. Sarro, "Search-based predictive modelling for software engineering: How far have we gone?" in *Proceedings of the 11th International Symposium on Search-Based Software Engineering (SSBSE)*, ser. Lecture Notes in Computer Science, S. Nejati and G. Gay, Eds., vol. 11664. Springer, 2019, pp. 3–7.

[73] C. Gencel, "How to use cosmic functional size in effort estimation models?" in *Software Process and Product Measurement*. Springer, 2008, pp. 196–207.

[74] M. de Freitas Junior, M. Fantinato, and V. Sun, "Improvements to the function point analysis method: A systematic literature review," *IEEE Transactions on Engineering Management*, vol. 62, no. 4, pp. 495–506, 2015.

[75] F. Ferrucci, C. Gravino, and F. Sarro, "Conversion from ifpug fpa to cosmic: within-vs without-company equations," in *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2014, pp. 293–300.

[76] S. Di Martino, F. Ferrucci, C. Gravino, and F. Sarro, "Assessing the effectiveness of approximate functional sizing approaches for effort estimation," *Information and Software Technology*, vol. 123, p. 106308, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584920300604

[77] E. Mendes, M. Kalinowski, D. Martins, F. Ferrucci, and F. Sarro, "Cross- vs. within-company cost estimation studies revisited: an extended systematic review," in *18th International Conference on Evaluation and Assessment in Software Engineering, EASE*, 2014, pp. 12:1–12:10.

[78] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local versus global lessons for defect prediction and effort estimation," *IEEE Transactions on software engineering*, vol. 39, no. 6, pp. 822–834, 2013.

[79] E. Kocaguneli, T. Menzies, and E. Mendes, "Transfer learning in effort estimation," *Empirical Software Engineering*, vol. 20, no. 3, pp. 813–843, 2015.

[80] L. Minku, F. Sarro, E. Mendes, and F. Ferrucci, "How to make best use of cross-company data for web effort estimation?" in *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on*. IEEE, 2015, pp. 1–10.

[81] M. Jorgensen, "Realism in assessment of effort estimation uncertainty: It matters how you ask," *IEEE Transactions on Software Engineering*, vol. 30, no. 4, pp. 209–217, 2004.

[82] M. Jørgensen and D. Sjöberg, "An effort prediction interval approach based on the empirical distribution of previous estimation accuracy," *Information and Software Technology*, vol. 45, no. 3, pp. 123 – 136, 2003.

[83] M. Jørgensen and D. I. K. Sjoeberg, "An effort prediction interval approach based on the empirical distribution of previous estimation accuracy," *Information and software Technology*, vol. 45, no. 3, pp. 123–136, 2003.

[84] M. Jørgensen, "Looking back on previous estimation error as a method to improve the uncertainty assessment of benefits and costs of software development projects," in *2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 2018, pp. 19–24.

[85] K. H. Teigen and M. JØrgensen, "When 90% confidence intervals are 50% certain: On the credibility of credible intervals," *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, vol. 19, no. 4, pp. 455–475, 2005.

[86] M. Jørgensen and K. Moløkken, "Combination of software development effort prediction intervals: Why, when and how?" in *Proc. of SEKE'02*, 2002, pp. 425–428.

[87] M. Jørgensen, K. H. Teigen, and K. MoløKken, "Better sure than safe? over-confidence in judgement based software development effort prediction intervals," *Journal of Systems and Software*, vol. 70, no. 1-2, pp. 79–93, 2004.