On the Value of Ensemble Effort Estimation

Ekrem Kocaguneli, *Student Member*, *IEEE*, Tim Menzies, *Member*, *IEEE*, and Jacky W. Keung, *Member*, *IEEE*

Abstract—*Background*: Despite decades of research, there is no consensus on which software effort estimation methods produce the most accurate models. *Aim:* Prior work has reported that, given M estimation methods, no single method consistently outperforms all others. Perhaps rather than recommending *one* estimation method as *best*, it is wiser to generate estimates from *ensembles of multiple estimation methods*. *Method:* Nine learners were combined with 10 preprocessing options to generate $9 \times 10 = 90$ *solo methods*. These were applied to 20 datasets and evaluated using seven error measures. This identified the best n (in our case $n = 13$) solo methods that showed stable performance across multiple datasets and error measures. The top 2, 4, 8, and 13 solo methods were then combined to generate 12 *multimethods*, which were then compared to the solo methods. *Results:* 1) The top 10 (out of 12) multimethods significantly outperformed *all* 90 solo methods. 2) The error rates of the multimethods were significantly less than the solo methods. 3) The ranking of the best multimethod was remarkably stable. *Conclusion:* While there is no best single effort estimation method, there exist best combinations of such effort estimation methods.

Index Terms—Software cost estimation, ensemble, machine learning, regression trees, support vector machines, neural nets, analogy, k-NN

1 Introduction

Correctly estimating the effort required to develop software is of vital importance. Over or underestimation of software development effort can lead to undesirable results:

- Underestimation results in schedule and budget overruns, which may cause project cancellation.
- Overestimation hinders the acceptance of promising ideas, thus threatening organizational competitiveness.

A practitioner when encountering the literature would almost certainly strike closed (i.e., fixed-parameter) models in the first instance, e.g., COCOMO [1], FPA [2], SLIM [3]. Further possible models to encounter range from

- simple regression [4],
- to analogy-based methods [5],
- to complex combinations of techniques such as Corazza et al.'s [6] use of tabu search to configure support vector machines or Menzies et al.'s [7] approach that tries hundreds of different methods.

Not only is the current literature confusing and voluminous [8], some results suggest that it may be impossible to assess which effort estimators are the best. Shepperd et al. [9] warn that when comparing M estimation methods, the ranking of any one method may change if the conditions are changed. Hence, they argue that it is fundamentally impossible to offer a definitive ranking such that $method_1$ is better than $method_2$.

This paper revisits the Shepperd et al. results and offers a more optimistic conclusion. We find that if we *combine* the estimates from *multiple estimators*, then those combined methods perform *better than any single estimator*. We agree with Shepperd et al. that there may be no single best effort estimator. However, there may be *best ensembles of methods*.

This should not be a surprising result. Many researchers argue that, in theory, best estimates come from combinations of multiple predictions. For example, Jorgensen advises that, for expert-based estimation, it is best to generate estimates from multiple methods [10]. Researchers in machine learning concur, e.g., Seni et al. report that averaging the estimates from *many methods* often does better than using *any solo method* [11]. Similar conclusions are offered by researchers in the fields of statistics [12] and machine learning [13], [14].

Current empirical results [15], [16], [17] in software engineering report the exact *opposite* effect than that predicted by Jorgensen, Seni et al., and others [10], [11], [12], [13], [14].

- Kocaguneli et al. failed to improve estimation by averaging across the predictions of 14 estimators [15].
- Baker could not improve estimation accuracy by *boosting* ensembles [16].[1]

In that respect our results are different from the prior empirical studies on ensembles. Whereas previous studies on the ensembles (through different strategies) report that ensembles are not statistically better than single learners, our study reports that (through the right strategy) ensembles can outperform single learners. This paper resolves this contradiction between theory and experimental results. To the best of our knowledge, this is the first such result in the effort

- E. Kocaguneli and T. Menzies are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506. E-mail: ekocagun@mix.wvu.edu, tim@menzies.us.
- J.W. Keung is with the Department of Computing, Hong Kong Polytechnic University, Room PQ714, Mong Man Wai Building, Hung Hom, Kowloon, Hong Kong, China. E-mail: jacky.keung@comp.polyu.edu.hk.

Manuscript received 23 Dec. 2010; revised 12 Oct. 2011; accepted 30 Oct. 2011; published online 10 Nov. 2011.
Recommended for acceptance by D. Sjøberg.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2010-12-0381.
Digital Object Identifier no. 10.1109/TSE.2011.111.

1. Boosting is an ensemble example that generates a series of methods where $method_i$ focuses on the cases that were most difficult for $method_{i-1}$ [18].

estimation literature that is supported by extensive experimentation.

It will be argued that the mistake made by Baker et al. and Kocaguneli et al. was to assume that *all* solo methods are candidates for combination into multimethod ensembles. This is not the case. Solo methods can be sorted into a minority of *superior* methods and a majority of *inferior* methods. We show below that, in the majority case, solo methods are outperformed by multimethods built from the superior set.

We therefore offer the following advice for a successful ensemble of methods:

1. Try a large number of methods, among which there are at least some good methods (shown to have good performance by prior work).
2. Sort the methods using the evaluation methods discussed in this paper. Discard all but the best solo methods.
3. Build ensembles from the remaining solo methods.

This research makes the following contributions:

- a novel scheme for ensembling the best solo methods, whose product is successful results (unlike previous research) concerning multimethods applied on effort data,
- an evaluation method for the stability of methods,
- stable multimethods that outperform *all* solo methods.

This paper is structured as follows: Section 2 discusses related work. Section 3 summarizes the learners, preprocessing options, and solo/multimethods used in this study. Our methodology is explained in Section 4, which generates the results of Section 5. The threats to the validity of our results are reviewed in Section 6. We provide a discussion of our work in Section 7 and conclude with Section 8.

2 Related Work

2.1 Software Effort Estimation Methods

Software effort estimation (from now on SEE) can be defined as the process of estimating the total effort necessary to complete a software project [19]. According to the extensive systematic review by Jorgensen and Shepperd, developing new models is the biggest research topic in SEE since 1980s [8]. Therefore, there are many SEE models that have been proposed over the years and a taxonomy is necessary to classify such a large corpus. Myrtveit et al. define taxonomy as an explanation of a concept by highlighting the similarities and differences between that particular concept and the others [20].

There exist a number of different taxonomies [20], [21]. Briand and Wieczorek report that there is no agreement on the best taxonomy and define all proposed taxonomies to be subjective [21]. For example, Menzies et al. divide SEE methods into two groups: model-based and expert-based [7]. According to this taxonomy, model-based methods use some algorithm(s) to summarize old data and to make predictions for the new data. Expert-based methods make use of human expertise which is possibly supported by process guidelines and/or checklists.

Myrtveit et al. use a different taxonomy where they propose a dataset dependent differentiation between methods [20]. According to that taxonomy the methods are divided into two:

- sparse-data methods that require few or no historical data, e.g., expert-estimation [10];
- many-data approaches where a certain amount of historical data is a must, e.g., functions and arbitrary function approximations (such as classification and regression trees).

Shepperd et al. propose a 3-class taxonomy [5]: 1) expert-based estimation, 2) algorithmic models, and 3) analogy. According to this taxonomy, expert-based models target the consensus of human experts through some process. Jorgensen define expert-based methods as a human-intensive process of negotiating the estimate of a new project and arriving at a consensus [10]. There are formal methods proposed for expert-based estimation like Delphi [22]. However, Shepperd and Cartwright note in another study that it is mostly the case that companies follow an informal process for expert-based estimation [9]. Algorithmic models include the adaptation of a formula to local data. Prominent examples of these methods are COCOMO [1] and function points [23]. Analogy-based methods find similar past projects, then adapt the effort values of them for the current project that is to be estimated.

In [24], Jorgensen defines some guidelines for generating realistic software effort estimates. An important finding in Jorgensen's study (which parallels our findings) is that *combining estimations* coming from different sources (e.g., from experts and instance-based learners) captures a broader range of information related to the estimation problem. Hence, multisource predictions offer the most robust and accurate estimator.

2.2 Ranking Instability

One long-standing issue with software effort estimation is *ranking instability*. Ideally, given M methods, we can definitively rank them in terms of their predictive accuracy:

$$M_1 \geq M_2 \geq M_3 \geq \ldots$$

This ideal has yet to be seen in the literature. To the contrary, Shepperd and Kadoda report that there is no certainty in that sort [20], [25]. For example, in the experiments of [25], a large number of synthetic datasets were generated (from distributions found in one real-world dataset). As they changed the conditions of their experiments, Shepperd et al. found that no method was consistently best across every condition. Specifically, they found that the performance of a method was dependent on the dataset, the random number generator used to select train and test sets, and the evaluation method used to assess model accuracy. Hence, they concluded that it was not feasible to sort methods into some definitive order. Extending their work, we say that when M methods are sorted to find a rank r_i for method M_i, then if experimental conditions are changed, then that rank will change by an amount $\delta r > 0$.

Recently, researchers have had access to more methods than that used by Shepperd et al. For example, Menzies et al. [26] studied 158 methods. While that study was over a very limited dataset (just two old COCOMO datasets), their preliminary results prompted this study (where we work with 20 datasets). In a result consistent with Shepperd et al., Menzies et al. found that as we changed the random numbers used to generate train/test sets, then all 158 methods showed some $\delta r > 0$, i.e., their precise ranking changed. However, they also found a small number of methods with two interesting properties:

1. Their ranks were very high.
2. Their δr values were very small.

Property #1 means that some methods performed comparatively very well and Property #2 means that high ranking persisted across multiple experimental conditions. Hence, if we look at enough methods, it may be possible to rank methods even when they exhibit $\delta r > 0$. Accordingly, in the results reported below, we will report not only the rank of each method, but also their associated δr. The methods we recommend will be those with *high ranks and low* δr.

2.3 Ensemble of Methods

A standard machine learning technique is to try multiple methods on the available data, then recommend the one that performs the best [27]. Many effort estimation papers apply this technique to demonstrate that (say) their preferred new method is superior to those proposed in prior work.

Ensemble learning takes a different approach. Rather than choosing *one* method, ensembles build multiple predictors, where estimates coming from different learners are combined through particular mechanisms, e.g., voting of individual learner estimates on the final prediction [28]. Before continuing any further we need to clear a terminology difference. From now on the term *"learner"* refers to a stand-alone algorithm without any supplemental pre or postprocessing step (e.g., *k*-NN, neural nets, etc.), whereas *"solo method"* will refer to an algorithm supplemented with a preprocessing option (e.g., logging+*k*-NN, discretization+ neural nets, etc.).

Ensembles are useful since any particular learner comes with its own assumptions [27]. These assumptions may be best suited to different parts of the training data [27], [28], [29]. In ensembles, methods can augment each other, i.e., a method patches errors made by another method. For example, when reducing estimated mean-squared-error, multimethods attain smaller or equal error rates than single methods [11].

It is a recommended practice to combine solo methods that have different characteristics [27], [29], [30]. There are many techniques to attain different-characteristic multimethods. The first way is through representation of the data. The multimethod structure may be based on unirepresentation (all learners use the same representation of data) or multirepresentation (different learners use different representations) [27]. Examples of such strategies are the use of different feature sets [31], [32] or different training sets [33].

The second way is through architectural methodologies. Bagging (Bootstrap Aggregating) and boosting are among the most common examples of that approach [27], [34]. In bagging, n-many solo methods are independently applied on n-many different training samples, where each training sample is selected via bootstrap sampling [27] with replacement. Boosting on the other hand arranges solo methods in a sequential manner: Each solo method pays more attention to the instances on which the previous method was unsuccessful. Boosting is reported to be considerably better than bagging [34], [35], [36], but has trouble in handling noisy datasets [34], [36].

2.4 Ensemble of Methods in SE

Ensemble methods are widely used in data mining (see the literature survey of [11]). Nevertheless, in the software domain, ensemble of methods (multimethods) have not been reported to be superior to solo methods in terms of prediction accuracy [15], [17]. In other words, ensembles fall short of providing a statistically significant increase in the prediction accuracy values over the solo methods. For particular error measures that were used to evaluate the prediction accuracy see the work of Kocaguneli et al. and Khoshgoftaar et al. [15], [17]. Khosgoftaar et al. [17] question the performance of different multimethod schemes under different scenarios in the domain of software quality. They use different combinations of 17 learners induced on seven datasets and report that multimethods do not yield a significant increase in accuracy.

Kocaguneli et al. [15] replicated the work of Khoshgoftaar et al. [17] in the domain of software effort estimation. They exploited a combination of 14 methods applied on three software effort estimation datasets. Their conclusion was similar to that of the replicated study [17]. The application of multimethods under different scenarios did not provide a significant increase in the estimation accuracy. Similarly, in the study of Vinaykumar and Ravi [37], different learners were employed in two types of ensembles, but only one of them was reported to be successful.

Another example of ensembles is the ensemble of *single-type* learners, where multiple versions of a single learner are combined. Pahariya et al. use linear combinations of genetic algorithms [38], where they report improvements over single-learners. Kultur et al. report improvements through collections of neural networks [39]. Unless a learner is supplemented with a preprocessing or a postprocessing option, then the learners in the ensemble are the copy of one another and have the same biases/assumptions. Unlike an ensemble of single-type learners, all the multimethods reported in our study are the combination of a learner augmented with pre/postprocessing options. There are also some applications of ensemble methods in effort estimation so as to process datasets. In [40], Twala et al. use multiple imputation techniques to handle missing data and in [41] Khoshgoftaar et al. make use of learner ensembles as a filter to improve the data quality.

Our ensemble is different from the above. We take care to prune inferior solo methods *before* building the ensemble. As shown below, this leads to very effective effort estimators.

3 Effort Estimation Methods

The effort estimation methods studied in this paper fall into two groups: *solo* methods and multimethods. *Solo* methods

TABLE 1
The Summary Table for the *Solo Methods*

Pre-processing Options		**Learners**	
Abbreviation	**Explanation**	**Abbreviation**	**Explanation**
norm	Normalization	ABE0-1NN	Basic ABE with 1 nearest neighbor
log	Taking natural logarithm	ABE0-5NN	Basic ABE with 5 nearest neighbors
PCA	Principal Component Analysis	SWReg	Stepwise Regression
SFS	Sequential Forward Selection	CART (yes)	Classification and Regression Tree with pruning
SWReg	Stepwise Regression	CART (no)	Classification and Regression Tree without pruning
width3bin	Discretize into 3 bins based on equal width	NNet	Neural Net with two hidden layers
width5bin	Discretize into 5 bins based on equal width	LReg	Simple linear regression
freq3bin	Discretize into 3 bins based on equal frequency	PCR	Principal components regression
freq5bin	Discretize into 5 bins based on equal frequency	PLSR	Partial least squares regression
none	Apply no pre-processor		

This table provides a list of abbreviations as well as their explanations for preprocessing options and learners used in this research.

are some combination of a *preprocessing option* and a *learner*. Multimethods are combinations of solo methods.

3.1 90 Solo Methods

In our experiments, we used 10 different preprocessing options and 9 learners. These were selected on two criteria:

- Learners must come from the SE effort estimation literature; e.g., [4], [5], [8], [39], [42], [43], [44], [45], [46].
- Learners must make different assumptions about the data.

This second criteria is based on data-mining theory that recommends using different learners that fail under different circumstances [27], [28], [29], [47]. For example, ABE methods assume that similar instances of the dataset have similar dependent variable values whose translation into SEE domain is that similar projects have similar effort values [44]. Decision tree inducers adopt a divide and conquer approach and assume that decisions on the prediction can be made through a sequence of tests/decisions that usually involve one feature at a time [34]. Artificial neural networks assume that data can be modeled through different geometries of directed graphs, where nodes represent the neurons and the connecting edges represent the weights applied on the output of the neurons [48]. With the arrival of each new instance, the effect is propagated through neurons and edges. Linear regression assumes that the trend seen in data can be formulated through a linear model [49]. Stepwise regression is built on the assumption that the set of features that maximize the F-value (evaluates if the variables in the model are significantly related to the independent variable) should be used in the final model [50].

Ensembles work best when one member of the ensemble patches the mistakes made by other methods of the ensemble. We hence used 10 preprocessing options:

- three *simple preprocessing options*: **none, norm, and log**,
- one *feature synthesis* method: **PCA**,
- two *feature selection* methods: **SFS** and **SWreg**,
- four *discretization* methods: Based on equal frequency/width

and 9 learners

- two *iterative dichotomizers*: **CART(yes), CART(no)**,
- a *neural net*: **NNet**,
- four *regression methods*: **LReg, PCR, PLSR, SWReg**,
- two *instance-based* learners: **ABE0-1NN, ABE0-5NN**.

Note that "ABE" is short for analogy-based effort estimation. ABE0-kNN is a standard ABE with execution steps of

- *normalization* of data to zero-one interva,
- a *euclidean* distance measure,
- estimates generated using the *k nearest neighbors*.

For a brief list of the learners and preprocessors, see Table 1; for their detailed descriptions, see the Appendix.

Combining 10 preprocessing options and 9 learners results in $10 * 9 = 90$ solo methods. A summary table for abbreviations of the methods and preprocessing options is provided in Table 4. Note that some of the combinations are less plausible than the others, e.g., 1) **norm & ABE0**, as ABE0 already has a normalization mechanism in it, 2) **different discretizations & CART**, as CART already has an inherent discretization mechanism, 3) **SWReg & SWReg**, as using the learner itself as a preprocessor would be too cumbersome. However, we included them in our analysis because they would find their appropriate rankings when compared with better methods. Furthermore, this brute-force analysis with all possible (but not necessarily plausible) preprocessing and learner combinations has to be carried out only once to find the better performing solo methods that are to be ensembled into multimethods.

3.2 Multimethods

Multimethods combine two or more solo methods. Many combination schemes have been proposed in the literature [27], [29], [30], [43]. Complex combination schemes include bagging [51], boosting [18], or random forests [52], [53]. Simpler methods include computing the mean, median, or inverse-ranked weighted mean (IRWM [43], see Fig. 1) of estimates coming from *n-many* solo methods. Our aim is not to investigate complex schemes, but to observe how multimethods perform compared to solo methods on effort datasets. Hence, we adopt simple schemes (mean, median, and IRWM).

In IRWM, the final estimates from M methods $e_1, e_2, ..., e_m$ that have been ranked $r_1, r_2, ..r_m$ is a weighted sum by the ranks of all methods in the ensemble. The top and bottom-ranked methods of m methods get a weight of m and 1 (respectively). More generally, a method with rank r_i gets a weight of $m+1-r_i$. The final estimate in IRWM is hence $\left(\sum_i (m+1-r_i) \cdot e_i\right) / \left(\sum_i i\right)$.

Fig. 1. IRWM. Generalized from [43].

4 Methodology

4.1 Multiple Error Measures

This section describes several performance measures used in this research. All the performance measures listed here have the property that we can find at least one publication proposing their use for effort estimation.

Error measures comment on the success of a prediction. For example, the absolute residual (AR) is the difference between the predicted and the actual values:

$$AR_i = |x_i - \hat{x}_i|, \tag{1}$$

(where $x_i, \hat{x}_i$ are the actual and predicted values, respectively, for test instance i). MAR is the mean of individual AR values.

The Magnitude of Relative Error measure, a.k.a. MRE, is a very widely used evaluation criterion for selecting the best effort estimator from a number of competing software prediction models [44], [54]. MRE measures the error ratio between the actual effort and the predicted effort. It can be expressed as the following equation:

$$MRE_i = \frac{|x_i - \hat{x}_i|}{x_i} = \frac{AR_i}{x_i}. \tag{2}$$

A related measure is MER (Magnitude of Error Relative to the estimate [54]):

$$MER_i = \frac{|x_i - \hat{x}_i|}{\hat{x}_i} = \frac{AR_i}{\hat{x}_i}. \tag{3}$$

A summary of MRE can be derived as the Mean Magnitude of Relative Error (MMRE) or Median Magnitude of Relative Error (MdMRE), which can be calculated as follows, respectively:

$$MMRE = \frac{\sum_{i=1}^{n} MRE_i}{n}, \tag{4}$$

$$MdMRE = median(MRE_1, MRE_2, \ldots, MRE_n). \tag{5}$$

A common alternative error measure is PRED(25), which can be defined as the percentage of predictions falling within 25 percent of the actual values

$$PRED(25) = \frac{100}{N} \sum_{i=1}^{N} \begin{cases} 1, & \text{if } MRE_i \leq \frac{25}{100}, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

For example, $\mathrm{PRED}(25) = 50\%$ implies that half of the estimates fall within 25 percent of the actual values [44].

There are many other error measures including Mean Balanced Relative Error (MBRE) and the Mean Inverted Balanced Relative Error (MIBRE) studied by Foss et al. [54]:

$$MBRE_i = \frac{|\hat{x}_i - x_i|}{min(\hat{x}_i, x_i)}, \tag{7}$$

$$MIBRE_i = \frac{|\hat{x}_i - x_i|}{max(\hat{x}_i, x_i)}. \tag{8}$$

Interpreting these error measures without any statistical test may be misleading. A recent discussion about this issue can be found in [55]. To evaluate our results subject to a statistical test, we make use of the so-called *win-tie-loss* statistics. Win-tie-loss statistics employ a Wilcoxon nonparametric statistical hypothesis test with 95 percent confidence. Wilcoxon is more robust than the Student's t-test as it compares the sums of ranks, unlike the Student's t-test, which may introduce spurious findings as a result of outliers in the given datasets. Ranked statistical tests like the Wilcoxon are also useful if it is not clear that the underlying distributions are Gaussian [56].

```
if WILCOXON($E_i$, $E_j$, 95) says they are the same then
    $tie_i = tie_i + 1$;
    $tie_j = tie_j + 1$;
else
    if better($E_i$, $E_j$) then
        $win_i = win_i + 1$
        $loss_j = loss_j + 1$
    else
        $win_j = win_j + 1$
        $loss_i = loss_i + 1$
    end if
end if
```

Fig. 2. Comparing methods (i, j).

We stored the performance of every method w.r.t. seven error measures over 20 datasets. This enabled us to collect *win-tie-loss* statistics using the algorithm of Fig. 2. In Fig. 2, we first check if two distributions i, j are statistically different according to the Wilcoxon test (95 percent); if they are not, then we increment tie_i and tie_j. If the distributions are statistically different, we update win_i, win_j, and $loss_i, loss_j$ after comparing their error measures. The *better* function in the *if* statement of Fig. 2 varies according to the performance criteria. For some error measures such as MMRE and MdMRE, better means lower values, i.e., lower means and medians, respectively. However, for PRED(25), better means higher PRED(25) values.

Note that 20 datasets have considerable size differences, e.g., the *China* dataset contains almost half of all the records used in this study. However, the above-mentioned performance measures are derived separately for every dataset.

4.2 Experimental Conditions

Recall the results of Shepperd and Kadoda [25] and Menzies et al. [26]: Different experimental conditions can change the rank of an effort estimator. Hence, it is important to study not just the rank of an estimator, but also how well that method performs across multiple experimental conditions such as

1. *error measures* that measure a method's performance,
2. *comparison summaries* used to report the performance of many methods over many datasets,
3. *datasets* used in the experiments.

The *error measures* used in this study, as defined above, are MAR, MMRE, MdMRE, MMER, PRED(25), MBRE, MIBRE.

As to *comparison summaries*, the following procedure was repeated for each error measure. Each of our 90 methods was compared to 89 others using the procedure of Fig. 2. In our procedure, we sum the *wins*, *losses*, *ties* of Fig. 2 and we rank our methods by that sum. That is, for an estimation method to be highly ranked, it must perform comparatively well across *all* error measures.

The results of those comparisons are contained in the *win*, *tie*, *loss* counters of Fig. 2. These comparisons can be summarized through many ways:

TABLE 2
The 1,198 Projects Used in This Study Come from 20 Datasets

				Historical Effort Data					
Dataset	**Features**	**Size**	**Description**	**Units**	**Min**	**Median**	**Mean**	**Max**	**Skew**
cocomo81	17	63	NASA projects	months	6	98	683	11400	4.4
cocomo81e	17	28	Cocomo81 embedded projects	months	9	354	1153	11400	3.4
cocomo81o	17	24	Cocomo81 organic projects	months	6	46	60	240	1.7
cocomo81s	17	11	Cocomo81 semi-detached projects	months	5.9	156	849.65	6400	2.64
nasa93	17	93	NASA projects	months	8	252	624	8211	4.2
nasa93_center_1	17	12	Nasa93 projects from center 1	months	24	66	139.92	360	0.86
nasa93_center_2	17	37	Nasa93 projects from center 2	months	8	82	223	1350	2.4
nasa93_center_5	17	40	Nasa93 projects from center 5	months	72	571	1011	8211	3.4
desharnais	12	81	Canadian software projects	hours	546	3647	5046	23940	2.0
desharnaisL1	11	46	Projects in desharnais that are developed with Language1	hours	805	4035.5	5738.9	23940	2.09
desharnaisL2	11	25	Projects in desharnais that are developed with Language2	hours	1155	3472	5116.7	14973	1.16
desharnaisL3	11	10	Projects in desharnais that are developed with Language3	hours	546	1123.5	1684.5	5880	1.86
sdr	22	24	Turkish software projects	months	2	12	32	342	3.9
albrecht	7	24	Projects from IBM	months	1	12	22	105	2.2
finnish	8	38	Software projects developed in Finland	hours	460	5430	7678.3	26670	0.95
kemerer	7	15	Large business applications	months	23.2	130.3	219.24	1107.3	2.76
maxwell	27	62	Projects from commercial banks in Finland	hours	583	5189.5	8223.2	63694	3.26
miyazaki94	8	48	Japanese software projects developed in COBOL	months	5.6	38.1	87.47	1586	6.06
telecom	3	18	Maintenance projects for telecom companies	months	23.54	222.53	284.33	1115.5	1.78
china	18	499	Projects from Chinese software companies	hours	26	1829	3921	54620	3.92
		Total: 1198							

Indentation in column one denotes a dataset that is a subset of another dataset. For notes on these datasets, see the Appendix.

1. number of losses,
2. number of wins,
3. number of wins-losses.

Demsar [57] reports that there is no generally accepted method of *comparison summarization*. Hence, we compute δr from the changes in the ranks seen when we move across all three summarization methods:

- First, we generate rankings of estimators using *number of losses* (repeated for all error measures) via a leave-one-out procedure (where each example becomes a test and the remaining data are used for training).
- Next, we compare that rank to other ranks generated by other comparisons summaries (again, repeated for all error measures).
- The maximum rank change for a method over all the dimensions is the δr value for that method.

Finally, we run our rig over multiple datasets. Our experiments use the 20 publicly available effort estimation datasets in the PROMISE repository (see http://promisedata.org/?cat=14). The names and properties of these datasets are provided in Table 2. Note that some projects in the Desharnais dataset contain missing values and we used mean imputation [58] to handle missing values. See the Appendix for more explanatory notes.

Combining the above, we can see that the experiments of this paper are an extensive analysis of different conditions for effort estimation experiments. Given 89 comparisons among solo methods, 7 error measures, and 20 datasets, then each method appears in $89 \times 7 \times 20 = 12{,}460$ comparisons.

4.3 Focus on Superior Methods

Fig. 3 is a plot that shows the success of the solo methods through ranking and the variability in that ranking. The x-axis of Fig. 3 shows the ranking of the 90 solo methods according to number of losses over all seven error measures and 20 datasets. The most successful methods have the lowest number of total losses, whereas the least successful ones have the highest number of total losses. Then solo methods are ranked on the x-axis starting from the best one. Therefore, better methods appear on the left-hand-side of that figure (so the top-ranked method appears at position $x = 1$).

The ranking of methods is identified by the x-axis of Fig. 3, whereas the variability in that ranking is given by the y-axis. The y-axis of Fig. 3 shows the maximum *changes*, δr, seen for each method as we compare the ranks across number of losses, number of wins, and number of wins-losses. In other words, a solo method has different rankings according to its win, $loss$, or $win - loss$ values and the related number seen on the y-axis is the biggest difference between its best and worst rankings. In a result consistent with Shepperd et al., all methods have $\delta r > 0$. However, the good news is that the top-ranked methods have a very low δr. That is, even if these top-ranked methods jumped rank by their maximum δr, then they would still be performing better than most of the other 90 methods.

In signal processing, it is standard practice to segment data based on the region of maximal change [59]. An inspection of Fig. 3 shows that the region of maximal δr occurs after $X = 13$. This is an interesting division since the region $1 \leq X \leq 13$ contains methods with high rank and low δr. We call these methods *superior* and the rest *inferior*. The list of the top 13 methods is shown in Table 3.

Fig. 3 is investigated further in a related paper that discusses the relative ranking concept of the solo methods [60]. The names of the top 13 *superior* solo methods of Fig. 3 are listed in Table 3. When we look at Table 3, we see that none of the superior solo methods try to fit one model to all the data:

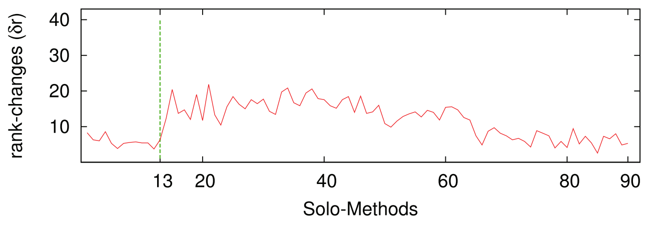


Fig. 3. Methods and the associated changes, i.e., δr values. Note the sudden increase in δr values after $X = 13$. We call methods in the region $1 \leq X \leq 13$ *superior.*

TABLE 3
Ranking of the Top 13 *Superior* Solo Methods and Related δr Values

rank	δr	pre-processing option	learner
1	8	norm	CART (yes)
2	6	norm	CART (no)
3	6	none	CART (yes)
4	9	none	CART (no)
5	5	log	CART (yes)
6	4	log	CART (no)
7	5	SWReg	CART (yes)
8	6	SWReg	CART (no)
9	6	SFS	CART (yes)
10	5	SFS	CART (no)
11	5	SWReg	ABE0-1NN
12	4	log	ABE0-1NN
13	5	SWReg	ABE0-5NN

These solo methods are combined in various ways to form 12 multimethods.

- The CART regression tree learner appears at ranks 1 through 10 of Table 3. Each branch of a regression tree defines one context in which an estimate may be different.
- Analogy-based estimation (ABE) appears at ranks 11, 12, 13. ABE builds a different model for each test instance (using the test instance's kth nearest neighbors).

Solo methods are not the focus of this paper. Hence, we move on to discuss multimethods.

4.4 Build Ensembles

To form multimethods, we build ensembles using the top M solo methods in the sort order of Table 3. In this study, we use $M \in \{2, 4, 8, 13\}$. To generate an estimate, we ask all members of an ensemble to offer a prediction. These are combined in one of three ways: mean, median, and IRWM.

These multimethods are then ranked alongside the solo methods in the same manner as Fig. 3. This gives us the comparison of 90 *solo methods* + 12 multimethods = 102 *methods*. Every method is compared to 101 others with respect to seven error measures and over 20 datasets. Therefore, the maximum number of comparisons for any method now becomes $101 \times 7 \times 20 = 14{,}140$. To the best of our knowledge, this is the one of the most extensive effort estimation experiments yet reported in the literature (and for extensive nonexperimental studies, see [8], [61]).

5 Results

Fig. 4 shows the rank of our 102 methods. As before, the x-axis ranks the methods according to number of losses and the y-axis shows the δr of each method. Table 4 shows the 102 methods, sorted in the same way as the x-axis of Fig. 4.

Two aspects of these result are worth commenting on:

- The top $X = 10$ methods (marked by a dashed line) are all multimethods. The remaining multiple methods appear at ranks 14, 15, 18. That is, in the majority case ($\frac{10}{12} = 83\%$), combinations of methods perform better that any solo method. Further, in all cases ($\frac{12}{12} = 100\%$), they are ranked higher than the majority of other methods.
- The multimethod at $X = 1$ also has the lowest δr of any method in this study. This method generated estimates using the IRWM value of the 13 top-ranked solo methods.

Note that the second result is exactly the "ensembles are better" result as might be predicted by the researchers mentioned in the introduction [10], [11], [12], [13], [14]. We speculate that prior SE researchers who failed to find that

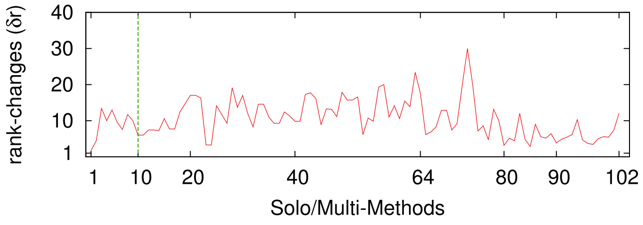


Fig. 4. Rank changes of solo and multimethods. Region $1 \leq X \leq 10$ contains 10 out of 12 multimethods. See that **Top13/Mean** at $X = 1$ has a δr of 1, i.e., it outperforms all other methods w.r.t. seven different error measures and 20 datasets.

TABLE 4
Detailed Preprocessing Option and Learner Combinations and Related δr Values

rank	δr	pre-proc.	learner	rank	δr	learner	
1	1	Top13	Irwm	52	17	norm	SWReg
2	4	Top13	Mean	53	6	none	SWReg
3	13	Top13	Median	54	11	freq3bin	5NN
4	10	Top2	Mean	55	10	width3bin	CART (yes)
5	13	Top4	Mean	56	19	width3bin	CART (no)
6	10	Top2	Median	57	20	PCA	5NN
7	8	Top4	Median	58	11	PCA	NNet
8	12	Top8	Median	59	14	none	NNet
9	10	Top2	Irwm	60	11	width5bin	SWReg
10	6	Top4	Irwm	61	15	width3bin	5NN
11	6	norm	CART (yes)	62	14	SWReg	NNet
12	7	norm	CART (no)	63	23	SFS	NNet
13	7	none	CART (yes)	64	18	width5bin	1NN
14	7	none	CART (no)	65	6	SWReg	LReg
15	11	Top8	Irwm	66	7	none	LReg
16	8	log	CART (yes)	67	8	norm	PLSR
17	8	log	CART (no)	68	13	width5bin	5NN
18	12	Top8	Mean	69	13	norm	LReg
19	15	SWReg	CART (yes)	70	7	freq5bin	1NN
20	17	SWReg	CART (no)	71	9	freq3bin	CART (yes)
21	17	SWReg	1NN	72	20	freq3bin	CART (no)
22	16	SFS	CART (yes)	73	30	PCA	1NN
23	3	SFS	CART (no)	74	20	freq3bin	1NN
24	3	log	1NN	75	7	width3bin	SWReg
25	14	SWReg	5NN	76	9	log	SWReg
26	12	PCA	PLSR	77	5	width5bin	PLSR
27	9	none	PLSR	78	13	log	PCR
28	19	SWReg	PCR	79	10	log	PLSR
29	14	PCA	PCR	80	3	width3bin	1NN
30	17	none	PCR	81	5	width3bin	PLSR
31	12	SFS	1NN	82	4	width5bin	PCR
32	8	PCA	CART (yes)	83	12	norm	PCR
33	15	PCA	CART (no)	84	5	width3bin	LReg
34	15	SFS	5NN	85	3	width3bin	PCR
35	11	norm	1NN	86	9	freq5bin	PCR
36	9	none	1NN	87	6	freq5bin	SWReg
37	9	freq5bin	CART (yes)	88	5	width5bin	LReg
38	12	freq5bin	CART (no)	89	6	freq3bin	PCR
39	11	freq5bin	5NN	90	4	freq3bin	PLSR
40	10	SFS	LReg	91	5	freq5bin	PLSR
41	10	width5bin	CART (yes)	92	5	log	LReg
42	17	width5bin	CART (no)	93	6	freq3bin	SWReg
43	18	SWReg	PLSR	94	10	freq5bin	LReg
44	16	SFS	PLSR	95	5	width5bin	NNet
45	9	SFS	PCR	96	4	norm	NNet
46	13	norm	5NN	97	3	width3bin	NNet
47	13	PCA	SWReg	98	5	log	NNet
48	11	none	5NN	99	6	freq3bin	NNet
49	18	SWReg	SWReg	100	5	freq5bin	NNet
50	16	log	5NN	101	7	freq3bin	LReg
51	16	SFS	SWReg	102	12	PCA	LReg

Methods are sorted by the sum of their losses seen in all performance measures and all datasets. The method with fewest losses is ranked #1 and is ***Top13/Irwm****. At the other end of the scale, the method with the most losses is ranked #102 and is* ***PCA/LReg****. Note that for this table ABE0-1NN and ABE0-5NN are abbreviated to 1NN and 5NN, respectively.*

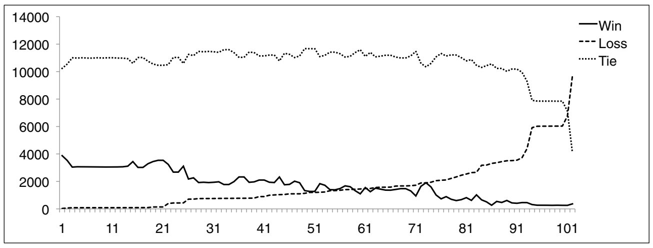


Fig. 5. The sum of win, tie, and loss values for all methods of Fig. 4 (over all error measures and all datasets). Since one method is compared to 101 other methods, over 7 error measures and 20 datasets, the sum of win, tie, and loss values is $101 \times 7 \times 20 = 14{,}140$.

"ensembles are better" did not prune away inferior solo methods before building an ensemble. One word of caution here is that although we use a deterministic sampling method (LOOCV) in this research, some algorithms contain probabilistic processes (e.g., the back-pruning process of **CART (yes)**). This may create small alterations in the performances of the methods (solo and multi) using these algorithms. Hence, it may be the case that one gets a slight variation of Table 4; however, the general picture (solo methods outperformed by multimethods) remains the same.

Better yet, as shown in Fig. 4, this largest ensemble at rank $X = 1$ had the lowest δr seen in any of our 102 methods ($\delta r = 1$). This result underscores the main message of this paper: The method that scored the best (and had the greatest stability across different experimental conditions) was the one that used the highest number of *superior* solo methods.

Fig. 5 shows the sum of win, tie, and $loss$ values for the methods of Fig. 4. Every method of Fig. 4 is compared to 101 other methods, over seven error measures and 20 datasets, so the maximum value that either one of the win, tie, $loss$ statistics can attain is $101 \times 7 \times 20 = 14{,}140$. Notice in Fig. 5 that except for the low performing methods on the right hand-side, the tie values are in the 10,000-12,000 band. Therefore, they would not be so informative as to differentiate the methods, so we consult win and $loss$ statistics. There is a considerable difference between the best and the worst methods in terms of win and $loss$ values (in the extreme case it is close to 4,000). In a way Fig. 5 is a sanity check of Fig. 4 because it shows that the rankings reported in Fig. 4 are due to considerable win and $loss$ value differences between the high (left hand-side) and low (right hand-side) performing methods.

Other results offer yet more evidence of the superiority of multimethods over solo methods. Fig. 6 sorts the MdMRE values of all the solo methods and all the multimethods:

- Multimethods generated lower MdMRE values than the solo methods.
- While some of the solo methods have alarmingly large errors (as observed from the steep right-hand-side of the dashed line in that figure), note that *none* of the 12 best multimethods have large MdMRE values.

That is, multimethods are far less prone to incorrect effort estimates than solo methods.

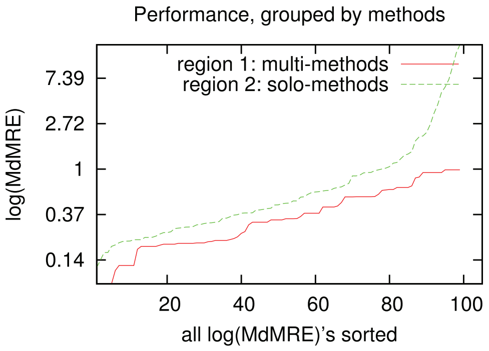


Fig. 6. The spectrum of MdMRE values for two regions of methods: solo methods and multimethods. Here we keep the order of methods the same as those given in Table 4 and divide those methods into two regions. Notice how multimethods attain the lowest MdMRE scores.

6 Threats to Validity

External validity questions whether the results can be generalized outside the specifications of a study [62]. To ensure external validity, this paper has studied a large number of projects. For example, Table 4 of [61] lists the total number of projects used by a sample of other studies. The median value of that sample is 186, i.e., one-sixth of the 1,198 projects used here. In terms of external validity, this report has higher validity than a standard effort estimation study (which might apply one error measure to a handful of datasets). An ideal case for our work or other similar studies would be to request datasets with specific properties that would best suit the experimental concerns and that would support the random sampling of data in experimentation. However, SEE datasets are neither easy to find nor easy to collect. Therefore, this stands as an external validity issue and calls for more effort in SEE data collection.

It is clear that this study has not explored a full range of effort estimation methods. Future work is required to repeat this study using the top performing solo methods found here and possibly more. Nevertheless, given the extent of the experimentation reported here, this paper offers much support for the general claim that when generating estimates, it is wise to *first* sort and prune all available solo methods, and *then* try combinations of the best solo methods. Another validity issue to mention is that none of the learners has been exhausted via fine-tuning. Therefore, future work is required to exhaust all the parameters of every learner to use their best version. However, exhaustive fine-tuning of a learner through some heuristic may be as comprehensive as a paper in its own right, e.g., the tabu search heuristic proposed by Corazza et al. [6].

Construct validity (i.e., face validity) asks if we are measuring what we actually intended to measure [63]. Previous studies have concerned themselves with the construct validity of different performance measures for effort estimation (e.g., [54]). To make sure that we do not favor a particular conclusion due to limited number of performance measures, we used seven error measures and questioned the stability of our conclusions. We empirically showed the surprising result that multimethods are stable across a range of performance criteria. The surprise factor comes from the fact that solo methods and successful multimethods have different rank change values. Unlike most of the solo methods with high rank changes w.r.t different performance measures (as high as 30), the top-10 multimethods are quite *stable and successful*, with the highest rank change of 13 and the lowest rank change of 1.

In terms of *internal validity*, there is one dimension of experimental conditions not explored above. This paper used leave-one-out to assess methods. An alternate experimental condition would be to use N-way cross-validation where examples are divided into B bins and each bin is tested on a model learned from the remaining bins. In theory, not using cross-validation is a threat to the validity of our results since we did not check if our results were stable across both leave-one-out and cross-validation. According to Hastie et al. [12, p. 242], leave-one-out testing generates:

- lower bias estimates than cross-validation (since cross-validation must learn from fewer examples),
- higher variance estimates than cross-validation (since leave-one-out conducts many more tests).

Similar assertions are made by Kohavi [13] and Breiman and Spector [14].

Elsewhere [64], we have documented the surprising result that, at least for the datasets of Table 2 , there is very little difference in the bias and variance values generated for leave-one-out and cross-validation. We conjecture that the standard comments on "bias versus variance tradeoff" for "leave-one-out versus cross-validation" needs to be revisited for the small effort datasets seen in effort estimation.

Since leave-one-out and cross-validation have similar bias/variance profiles, we are free to select either method. This paper uses leave-one-out for the following reasons. Leave-one-out is a deterministic procedure that can be exactly repeated by any other researcher with access to a particular dataset. The same cannot be said about cross-validation since it divides the data into bins using

- a random number generator,
- some stratification heuristics. Stratification attempts to maintain the same class distributions in each bin as in the entire dataset). Stratification is a heuristic procedure since it may encounter issues that have no singular solution (e.g., how to space rare outliers among the bins).

Note that without knowledge of the stratification heuristics and without access to an identical random number generator, it can be difficult for researcher "A" to reproduce the cross-validation results of researcher "B."

7 Discussion

7.1 Learning Curve

One of our observations is that ensembles are more trustworthy (less ranking instability). For someone with a strong background in machine learning algorithms, the number of learners to combine is not an issue as he/she has already gone through the learning curve. However, from a practitioner's point of view, there is a cost-benefit tradeoff between

- the cost of learning new learners and
- the additional performance benefit.

TABLE 5
The Learning Curve of the Top-13 *Superior* Solo Methods

rank	pre-processor	learner	# of additions	total learned
1	norm	CART (yes)	1 [CART (yes)]	1
2	norm	CART (no)	1 [CART (no)]	2
3	none	CART (yes)	0	2
4	none	CART (no)	0	2
5	log	CART (yes)	0	2
6	log	CART (no)	0	2
7	SWReg	CART (yes)	1 [SWR]	3
8	SWReg	CART (no)	0	3
9	SFS	CART (yes)	1 [SFS]	4
10	SFS	CART (no)	0	4
11	SWReg	ABE0-1NN	1 [ABE-kNN]	5
12	log	ABE0-1NN	0	5
13	SWReg	ABE0-5NN	0	5

The column "# of additions" shows how many algorithms (learner or preprocessor) are needed for the transition to the current row from the previous one and the required algorithm for that transition is given in brackets. The column "total learned" shows how many algorithms need to be learned until the current row. The assumption is that common numeric methods such as normalizing and taking the natural logarithm of numbers are known in advance.

We acknowledge the fact that building an ensemble model from scratch may be too challenging for practitioners without prior machine learning experience. However, our industry-collaborated project experience shows that once the model is built by researchers, its adoption/implementation by practitioners is a pretty straightforward process [65], [66], [67], [68]. This section provides an in depth discussion and alternative solutions for such practitioners.

The best multimethod reported in this paper requires combination of the top-13 solo methods. The learning curve associated with the top-13 solo methods is given in Table 5. From Table 5, we see that a practitioner willing to use multimethods at all has to learn at least two learners (CART(yes) and CART(no)). Not included in Table 5 are the numeric manipulations such as normalizing an array of numbers and taking logarithm, mean, or median of these numbers. Before learning the algorithms of Table 5, a practitioner will need to learn these numeric operations, then learning just two learners in fact enables him/her to use up until the sixth best performing solo method. By using the top-6 solo methods, we can build the fourth, fifth, sixth, seventh, ninth, and 10th best methods of Table 4. Also note that aside from the algorithms of Table 5, a practitioner will most likely require learning the simple formula of IRWM.

In summary, building successful multimethods is not necessarily a difficult process for a practitioner. See in Table 5 that building the best method reported here requires learning five algorithms (learner or preprocessor) and learning only two learners enables someone to build four very successful multimethods. Therefore, our recommendations to practitioners who are willing to use multimethods but lack the knowledge of machine learning algorithms are

- Start with initial two learners and build the associated multimethods.
- See the performance of the current multimethods.
- Build new multimethods only if you are not pleased with the performance of the current ones.

The matter of mutual SE knowledge transfer between research and industry is not restricted to the discussion we provided in this section. Practitioners willing to learn further on that issue are strongly recommended to read [69].

7.2 Ensemble and Accuracy

Writing in the field of marketing, Armstrong reports in his 2007 study that the multimethod forecasts outperform single-method forecasts [70]. He cites 31 studies where multiple source prediction consistently outperforms single source prediction by 3.4 to 23.4 percent (average = 12.5 percent). It is expected to see that a meaningful combination method such as ours results in superior performance. On the other hand, it is impossible to cover all meaningful combination strategies in a single paper. Our study adopts only one of the many proposed successful combination methods [71]. Citing from one of our reviewers: "Work by Hogarth [72] demonstrates that a reduced intercorrelation between the prediction sources may be just as important as finding the prediction sources with highest expected accuracy (excluding the poor methods) when combining predictions." For this research, it means that understanding the interdependencies between different solo methods could lead to clearer definitions of when to combine which particular solo methods. In return, such a strategy could yield more robust multimethods. For example, a very promising future work (as indicated by the cited reviewer) would be to include the best neural network model into multimethods.

In the ideal case, different multimethods would perform optimally under different combination schemes. For example, when combining a high number of solo methods, it would be better to use a scheme that would catch the central tendency and be able to handle extreme values, e.g., median. However, our findings do not support that implication. In other words, our best performing multimethod includes a high number of solo methods and its combination scheme is IRWM. That discrepancy between implications and results is a familiar concept in the forecasting literature: It is difficult to give robust guidelines as to how to combine the methods in an optimum way [73]. Investigation of robust combination schemes and their implications would be a good future direction to our study.

8 Conclusion

There are many effort estimation methods and little guidance on how to choose between them. Prior results from Shepperd et al. are pessimistic about the consistency of the rankings of different methods. If M methods are ranked in an experiment, then that ranking can be changed by altering the conditions of that experiment (e.g., the data used in training or the performance measure used).

This paper confirms the inconsistency of the rankings effect reported by Shepperd et al.: In Figs. 3 and 4 it was seen that a method's ranking can change by some amount δr. Nevertheless, our conclusions are more optimistic than those of Shepperd et al. While some methods have very large δr, others do not. In fact, for the solo methods shown in Fig. 3, the better methods have smaller δr. Better yet, when we combined the highest ranked solo methods, we found that the top-ranked multimethod had almost zero δr. Therefore, we recommend multimethods since, as shown in Fig. 4:

- The multimethods consistently outperform most, if not all, of the solo methods.
- The performance of the multimethods is more trustworthy (has the smallest ranking instability).
- Also, as shown in Fig. 6, the multimethods avoid the problem of very large errors seen with other methods.

The success of our results raises the question: Why were prior experiments in ensembles for effort estimation [15], [16] so unsuccessful? Reading from the prior effort estimation literature, we offer the following hypothesis: *Many effort estimation methods are inferior* so combinations of inferior members will also be inferior. In Fig. 3, for example, while there is evidence for 13 useful methods (in the range $1 \leq X \leq 13$), the remaining methods have large δr or fall into the worse ranks. That is, of the 90 solo methods we have explored, there is no evidence for the value of $\frac{77}{90} = 86\%$ of the methods. Without a ranking like Fig. 3 to guide method selection, it is therefore probable that researchers will try combining weaker methods.

New methods are constantly being invented (e.g., see [6], [39], [74]) so we make no claim that the 90 methods explored here cover the space of all possible effort estimators. We also make no claim that one study can cover all the preprocessing options there is in the literature and our study is no exception. Our study covers a total of 10 options for preprocessing; however, it does not cover the effects of noise or outlier removal. The investigation of the effects of noise/outlier removal on method performance would in fact be an interesting future direction to this study.

Finally, we can offer an answer to the vexing question: *What is the best effort estimator?* Simply put, we have no evidence suggesting that any solo method is some universal, best in all circumstances, effort estimator. However, if no solo method is always the best, some ensemble of solo methods may offer consistently better performance. We have seen that 9 out of 12 multimethods outperformed all others. Better yet, our top-ranked multimethod has the greatest stability among any of the 102 methods explored in this study. Hence, this study proposes multimethods as the technique that finds the best learner.

Appendix

Note: The content of the following appendices comes from [60]. Although the work in [60] and this paper use the same base learners and preprocessing options, they address two very different research ideas. In [60], our concern is an investigation of the common SEE methods in retrospect, i.e., we reevaluate the past work in SEE and seek for stable conclusions in methods and datasets. We use the results of this investigation to comment on the methods and on the datasets used extensively in SEE.

On the other hand, in this paper we investigate new and robust estimation methods, i.e., the ensembles. This research is built around a similar code base (the implementation of learners and preprocessing options), yet it uses the knowledge we got by observing old methods to propose a completely new direction of building a novel scheme: *multimethods.*

Appendix A: Data

All the data used in this study are available either at http://promisedata.org/data or through the authors. As shown in

Table 2 , we use a variety of different datasets in this research. The standard **COCOMO** datasets (cocomo*, nasa*) are collected with the COCOMO approach [1].

The **Desharnais** dataset contains software projects from Canada. It is collected with function points approach.

SDR contains data from projects of various software companies in Turkey. SDR is collected by Softlab, the Bogazici University Software Engineering Research Laboratory [75].

The **Albrecht** dataset consists of projects completed at IBM in the 1970s and details are given in [2].

The **Finnish** dataset originally contained 40 projects from different companies and data were collected by a single person. The two projects with missing values are omitted here; hence we use 38 instances. More details can be found in [76].

The **Kemerer** is a relatively small dataset with 15 instances, whose details can be found in [77].

The **Maxwell** dataset comes from the finance domain and is composed of Finnish banking software projects. Details of this dataset are given in [78].

The **Miyazaki** dataset contains projects developed in COBOL. For details see [79].

Telecom contains projects which are enhancements to a United Kingdom telecommunication product and details are provided in [44].

The **China** dataset includes various software projects from multiple companies developed in China.

Note that two of these datasets (Nasa93c2, Nasa93c5) come from different development centers around the United States. Another two of these datasets (Cocomo81e, Cocomo81o) represent different kinds of projects. The Cocomo81e "embedded projects"are those developed within tight constraints (hardware, software, operational,...); The Cocomo81o "organic projects" come from small teams with good experience working with less than rigid requirements.

Appendix B: Preprocessing Options

Simple numeric techniques. This category lists the preprocessors that entail mere numeric alterations of actual values rather than application of certain algorithms.

- **norm.** *"norm"* represents the application of normalization on the data. The data are normalized to 0-1 interval according to (9).
- **log.** Take the natural logarithm of the independent variables:

$$normalizedValue = \frac{(actualValue - min(allValues))}{(max(allValues) - min(allValues))}. \tag{9}$$

Feature synthesis.*"PCA"* stands for principal component analysis [58]. PCA is a dimension alteration mechanism, where an n-dimensional space is altered into another n-dimensional space.

Feature selection. Some of the preprocessors aim at finding a subset of all features according to certain criteria. This subset is supposed to produce a feature subset that will ultimately increase the estimation accuracy. Under this category, we have SFS and SWREg.

Sequential forward selection (**"SFS"**) is a feature selection mechanism in which features are added into an initially empty set until no improvement is possible with the addition of another feature. The so-called *improvement* is defined through an *objective* function. In our implementation based on Matlab, the *objective* function is the mean-squared-error of a simple linear regression on the training set. One caution to be made here is that exhaustive search algorithms over all features can be very time consuming (2^n combinations in an n-feature dataset); therefore SFS works only in the forward direction (no backtracking).

"SWReg" stands for stepwise regression, which can be defined as a systematic method for adding and removing features from a multilinear model. Removal and addition of features depend on their statistical significance in regression. Our SWReg implementation that is developed by using Matlab starts execution with a preliminary model, then it compares the performances of smaller and larger models. At each step a potential feature is to be decided for addition or removal. Addition and removal are based on the p-value in an F-statistic. In a particular step, the F-statistics for two models (models with and without the feature that is being questioned in that step) are calculated. Provided that the feature was not in the model, the null hypothesis is: "Feature would have a zero coefficient in the model, when it is added." If the null hypothesis can be rejected, then the feature is added to the model. As for the other scenario (i.e., feature is already in the model), the null hypothesis is: "Feature has a zero coefficient." If we fail to reject the null hypothesis, then the term is removed.

Note that stepwise methods are locally optimal and may not necessarily be globally optimal. In other words, depending on the initial model and the order of features for inclusion-exclusion, the algorithm may result in different final models, hence different performances.

Discretization:

- **width3bin.** This procedure bins each one of the data features into three bins, depending on equal width of all bins. The bin-width for a general n-bin procedure is given in (10). In our 3-bin case, once we know the bin-width, we assign each feature value to either 1, 2, or 3, depending on which particular bin the value is in:

$$binWidth = ceiling\left(\frac{(max(allValues) \ - \ min(allValues))}{n}\right). \tag{10}$$

- **width5bin.** Exactly same as *"width3bin"* except that this time we have five bins instead of 3.
- **freq3bin.** *"freq3bin"* means binning each feature into three bins, depending on the equal frequency count (equal number of instances in each bin). Similar to previously mentioned binning methods, in this method each feature value is assigned to 1, 2, or 3. For that, all values of a particular feature are sorted in ascending order. Then, first $ceiling(numberOf(allInstances)/3)$ instances are assigned 1, the second

$ceiling(numberOf(allInstances)/3)$ instances are assigned 2, and so on.

- **freq5bin.** The same as *"freq3bin"* but with five bins.

Others. The option(s) that cannot be categorized into the aforementioned categories are mentioned here. Application of no preprocessing to the data is also a choice. **"none"** represents the choice of no preprocessor selection.

Appendix C: Learners

Instance-based learners. The analogy-based estimation method that we call ABE0-xNN refers to a very basic type of ABE that we defined through our readings of various ABE studies [43], [74], [80]. In ABE0-xNN, features are first normalized to the 0-1 interval, then the distance between the test and train instances is measured according to euclidean distance function, x nearest neighbors are chosen from the training set, and finally, for finding estimated value (a.k.a adaptation procedure), the median of x nearest neighbors is calculated. Therefore, when we say ABE0-xNN, all the design options including the choice of *x-many* closest analogies become explicit to a reader. In our experimentation we use two different x values (i.e., two different analogy values):

- **ABE0-1NN:** Only the closest analogy is used.
- **ABE0-5NN:** The five closest analogies are found and used for adaptation.

Iterative dichotomization. Under this category, we used classification and regression trees (*CART*) as described by Breiman et al. [81] and developed it using Matlab routines. CART is a nonparametric technique and it can work both for classification and regression type of problems; in our case it is the latter. When planning to construct a CART, there are a couple of points to consider. First is the selection of splits: There are a variety of solutions such as misclassification rate, information (or entropy), or GINI index. CART uses the GINI index to calculate the impurity of a tree [81]. Second, the decision on when to stop further splits: The implementation allows you to specify a threshold value or use the default value. We have used default threshold values in the implementation (for further details please refer to Matlab documentation). Last, assigning a class to each terminal node. Each test instance results in a terminal node and therefore different test instances resulting in the same terminal node are given the same predicted value. For regression, the predicted value in a terminal node is a fit on the independent variables of the instances in that node.

We have used two types of CART, which are given below:

- **CART (yes).** In this version, the pruning is on, meaning that the training data are used in a cross-validation process and, for each cross-validation run, some internal nodes are randomly chosen as the leaf nodes and their subnodes are removed. Finally, the subtree that resulted in the lowest error rate is returned. The returned subtree is a suboptimal solution and it is locally optimum.
- **CART (no).** The subtrees of CART will not be considered and the full tree will be used.

Two-layered neural net. *"Neural Nets"* (NNet) are memoryless structures that can be defined as universal approximators [82], meaning that: 1) They store information coming from training data as weights during training and, after that phase, they do not need training data and 2) they can approximate any function, depending on how complex the NNet structure is. NNets basically have a three layer structure: input layer, hidden layer, and the output layer. Depending on the complexity of the problem, one can model more complex functions with an NNet by increasing the number of hidden layers in the structure. In our implementation, we used a simple NNet structure with two hidden layers, one input layer, and one output layer.

Regression methods. Under this category, we list our regression-based learners. **LReg** stands for linear regression. Given the dependent variables, this learner calculates the coefficient estimates of the independent variables.

Partial least squares regression (**PLSR**) as well as principal components regression (**PCR**) are algorithms that are used to model a dependent variable and we have used Matlab routines to implement those functions in our experiments. While modeling an independent variable, they both construct new independent variables as linear combinations of original independent variables. However, the ways in which they construct the new independent variables are different:

- **PCR.** This method generates new independent variables to explain the observed variability in the actual ones. However, while generating new variables, the dependent variable is not considered at all. In that respect, PCR is similar to selection of *n-many* components via PCA (the default value of components to select is 2, so we used it that way) and applying linear regression.
- **PLSR.** Unlike PCR, PLSR considers the independent variable and picks up the *n-many* of the new components (again with a default value of 2) that yield lowest error rate. Due to this particular property of PLSR, it usually results in a better fitting.

SWReg. In the algorithms section, we use SWReg as a regression method on the *selected-out* independent variables to explain the dependent variable.

Acknowledgments

This research is funded in part by the US National Science Foundation, CISE, project #0810879.

References

[1] B.W. Boehm, *Software Engineering Economics.* Prentice Hall, 1981.
[2] A. Albrecht and J. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Eng.,* vol. 9, no. 6, pp. 639-648, Nov. 1983.
[3] L.H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, within Budget.* Yourdon Press, 1992.
[4] B. Boehm, C. Abts, and S. Chulani, "Software Development Cost Estimation Approaches—A Survey," *Annals of Software Eng.,* vol. 10, pp. 177-205, 2000.
[5] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort Estimation Using Analogy," *Proc. 18th Int'l Conf. Software Eng.,* pp. 170-178, 1996.
[6] 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?" *Proc. Sixth Int'l Conf. Predictive Models in Software Eng.,* 2010.

[7] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting Best Practices for Effort Estimation," *IEEE Trans. Software Eng.,* vol. 32, no. 11, pp. 883-895, Nov. 2006.

[8] M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Trans. Software Eng.,* vol. 33, no. 1, pp. 33-53, Jan. 2007.

[9] M. Shepperd and M. Cartwright, "Predicting with Sparse Data," *IEEE Trans. Software Eng.,* vol. 27, no. 11, pp. 987-998, Nov. 2001.

[10] M. Jorgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software,* vol. 70, pp. 37-60, 2004.

[11] G. Seni and J. Elder, *Ensemble Methods in Data Mining: Improving Accuracy through Combining Predictions.* Morgan and Claypool Publishers, 2010.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction,* second ed. Springer, 2008.

[13] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," *Proc. 14th Int'l Joint Conf. Artificial Intelligence,* pp. 1137-1143, 1995.

[14] L. Breiman and P. Spector, "Submodel Selection and Evaluation in Regression. The X-Random Case," *Int'l Statistical Rev.,* vol. 60, no. 3, pp. 291-319, Dec. 1992.

[15] E. Kocaguneli, Y. Kultur, and A. Bener, "Combining Multiple Learners Induced on Multiple Data Sets for Software Effort Prediction," *Proc. Int'l Symp. Software Reliability Eng.,* 2009.

[16] D. Baker, "A Hybrid Approach to Expert and Model-Based Effort Estimation," master's thesis, Lane Dept. of Computer Science and Electrical Eng., West Virginia Univ., https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443, 2007.

[17] T.M. Khoshgoftaar, P. Rebours, and N. Seliya, "Software Quality Analysis by Combining Multiple Projects and Learners," *Software Quality Control,* vol. 17, no. 1, pp. 25-49, 2009.

[18] Y. Freund and R. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Computer and System Sciences,* vol. 55, pp. 119-139, 1997.

[19] J.W. Keung, "Theoretical Maximum Prediction Accuracy for Analogy-Based Software Cost Estimation," *Proc. 15th Asia-Pacific Software Eng. Conf.,* http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4724583, pp. 495-502, 2008.

[20] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and Validity in Comparative Studies of Software Prediction Models," *IEEE Trans. Software Eng.,* vol. 31, no. 5, pp. 380-391, May 2005.

[21] L. Briand and I. Wieczorek, *Resource Modeling in Software Engineering,* second ed. Wiley, 2002.

[22] G. Rowe and G. Wright, "The Delphi Technique as a Forecasting Tool: Issues and Analysis," *Int'l J. Forecasting,* vol. 15, pp. 351-371, 1999.

[23] K. Atkinson and M. Shepperd, "The Use of Function Points to Find Cost Analogies," *Proc. Fifth European Software Cost Modelling Meeting,* 1994.

[24] M. Jorgensen, "Practical Guidelines for Expert-Judgment-Based Software Effort Estimation," *IEEE Software,* vol. 22, no. 3, pp. 57-63, May/June 2005.

[25] M. Shepperd and G.F. Kadoda, "Comparing Software Prediction Techniques Using Simulation," *IEEE Trans. Software Eng.,* vol. 27, no. 11, pp. 1014-1022, Nov. 2001.

[26] T. Menzies, O. Jalali, J. Hihn, D. Baker, and K. Lum, "Stable Rankings for Different Effort Models," *Automated Software Eng.,* vol. 17, http://dx.doi.org/10.1007/s10515-010-0070-z, pp. 409-437, 2010.

[27] E. Alpaydin, "Techniques for Combining Multiple Learners," *Proc. Eng. Intelligent Systems Conf.,* vol. 2, pp. 6-12, 1998.

[28] T.G. Dietterich, "Ensemble Methods in Machine Learning," *Proc. First Int'l Workshop Multiple Classifier Systems,* pp. 1-15, 2000.

[29] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas, "On Combining Classifiers," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 20, no. 3, pp. 226-239, Mar. 1998.

[30] K. Ali, "On the Link Between Error Correlation and Error Reduction in Decision Tree Ensembles," technical report, Dept. of Information and Computer Science, Univ. of California, 1995.

[31] T.K. Ho, J.J. Hull, and S.N. Srihari, "Decision Combination in Multiple Classifier Systems," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 16, no. 1, pp. 66-75, Jan. 1994.

[32] S. Günter and H. Bunke, "Feature Selection Algorithms for the Generation of Multiple Classifier Systems and their Application to Handwritten Word Recognition," *Pattern Recognition Letters,* vol. 25, no. 11, pp. 1323-1336, 2004.

[33] T.K. Ho, "Random Decision Forests," *Proc. Third Int'l Conf. Document Analysis and Recognition,* p. 278, 1995.

[34] H. Zhao and S. Ram, "Constrained Cascade Generalization of Decision Trees," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 6, pp. 727-739, June 2004.

[35] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation.* Morgan Kaufmann, 2000.

[36] E. Bauer and R. Kohavi, "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants," *Machine Learning,* vol. 36, http://dx.doi.org/10.1023/A:1007515423169, pp. 105-139, 1999.

[37] M.C.K. Vinaykumar and V. Ravi, "Software Cost Estimation Using Soft Computing Approaches," *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques,* pp. 499-518, 2010.

[38] J. Pahariya, V. Ravi, and M. Carr, "Software Cost Estimation Using Computational Intelligence Techniques," *Proc. World Congress Nature Biologically Inspired Computing,* pp. 849-854, Dec. 2009.

[39] Y. Kultur, B. Turhan, and A.B. Bener, "ENNA: Software Effort Estimation Using Ensemble of Neural Networks with Associative Memory," *Proc. 16th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.,* pp. 330-338, 2008.

[40] B. Twala, M. Cartwright, and M. Shepperd, "Ensemble of Missing Data Techniques to Improve Software Prediction Accuracy," *Proc. 28th Int'l Conf. Software Eng.,* http://doi.acm.org/10.1145/1134285.1134449, pp. 909-912, 2006.

[41] T.M. Khoshgoftaar, S. Zhong, and V. Joshi, "Enhancing Software Quality Estimation Using Ensemble-Classifier Based Noise Filtering," *Intelligent Data Analysis,* vol. 9, http://portal.acm.org/citation.cfm?id=1239046.1239048, pp. 3-27, Jan. 2005.

[42] K. Lum, T. Menzies, and D. Baker, "2CEE, A Twenty First Century Effort Estimation Methodology," *Proc. ISPA/SCEA Joint Ann. Conf. Training Workshop,* pp. 12-14, 2008.

[43] E. Mendes, I.D. Watson, C. Triggs, N. Mosley, and S. Counsell, "A Comparative Study of Cost Estimation Models for Web Hypermedia Applications," *Empirical Software Eng.,* vol. 8, no. 2, pp. 163-196, 2003.

[44] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Trans. Software Eng.,* vol. 23, no. 11, pp. 736-743, Nov. 1997.

[45] C. Chang, "Finding Prototypes for Nearest Neighbor Classifiers," *IEEE Trans. Computers,* vol. 23, no. 11, pp. 1179-1185, Nov. 1974.

[46] A. Venkatachalam, "Software Cost Estimation Using Artificial Neural Networks," *Proc. Int'l Joint Conf. Neural Networks,* pp. 987-990, 1993.

[47] J. Ghosh, "Multiclassifier Systems: Back to the Future," *Proc. Third Int'l Workshop Multiple Classifier Systems,* pp. 1-15, 2002.

[48] H. Park and S. Baek, "An Empirical Validation of a Neural Network Model for Software Effort Estimation," *Expert Systems with Applications: An Int'l J.,* vol. 35, no. 3, pp. 929-937, 2008.

[49] K. Pillai and V. Sukumaran Nair, "A Model for Software Development Effort and Cost Estimation," *IEEE Trans. Software Eng.,* vol. 23, no. 8, pp. 485-497, Aug. 1997.

[50] E. Mendes and N. Mosley, "Further Investigation into the Use of CBR and Stepwise Regression to Predict Development Effort for Web Hypermedia Applications," *Proc. Int'l Symp. Empirical Software Eng.,* 2002.

[51] L. Breiman, "Bagging Predictors," *Machine Learning,* vol. 24, pp. 123-140, 1996.

[52] Y. Jiang, B. Cukic, and T. Menzies, "Cost Curve Evaluation of Fault Prediction Models," *Proc. 19th Int'l Symp. Software Reliability Eng.,* pp. 197-206, 2008.

[53] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, pp. 5-32, 2001.

[54] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A Simulation Study of the Model Evaluation Criterion MMRE," *IEEE Trans. Software Eng.,* vol. 29, no. 11, pp. 985-995, Nov. 2003.

[55] B. Kitchenham and E. Mendes, "Why Comparative Effort Prediction Studies May be Invalid," *Proc. Fifth Int'l Conf. Predictor Models in Software Eng.,* pp. 1-5, 2009.

[56] J. Kliijnen, "Sensitivity Analysis and Related Analyses: A Survey of Statistical Techniques," *J. Statistical Computation and Simulation,* vol. 57, nos. 1-4, pp. 111-142, 1997.

[57] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," *J. Machine Learning Research,* vol. 7, pp. 1-30, 2006.

[58] E. Alpaydin, *Introduction to Machine Learning.* MIT Press, 2004.

[59] A. Oppeneheim, A. Wilsky, and S. Hamid, *Signals and Systems.* Prentice Hall, 1996.

[60] J. Keung, E. Kocaguneli, and T. Menzies, "A Ranking Stability Indicator for Selecting the Best Effort Estimator in Software Cost Estimation," *Automated Software Eng.,* http://menzies.us/pdf/11draftranking.pdf, 2011.

[61] B. Kitchenham, E. Mendes, and G.H. Travassos, "Cross Versus Within-Company Cost Estimation Studies: A Systematic Review," *IEEE Trans. Software Eng.,* vol. 33, no. 5, pp. 316-329, May 2007.

[62] D. Milic and C. Wohlin, "Distribution Patterns of Effort Estimations," *Proc. 30th EUROMICRO Conf.,* 2004.

[63] C. Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers.* Blackwell Publisher, 2002.

[64] E. Kocaguneli and T. Menzies, "Software Effort Models Should Be Assessed via Leave-One-Out Validation," in preparation, http://bit.ly/biasVarDraft, 2011.

[65] A. Nelson, T. Menzies, and G. Gay, "Sharing Experiments Using Open-Source Software," *Software: Practice and Experience,* vol. 41, no. 3, pp. 283-305, 2011.

[66] A. Bakir, E. Kocaguneli, A. Tosun, A. Bener, and B. Turhan, "Xiruxe: An Intelligent Fault Tracking Tool," *Proc. Int'l Conf. Artificial Intelligence and Pattern Recognition,* 2009.

[67] A. Tosun, A. Bener, and E. Kocaguneli, "BITS: Issue Tracking and Project Management Tool in Healthcare Software Development," *Proc. Int'l Conf. Software Eng. and Knowledge Eng.,* 2009.

[68] E. Kocaguneli, A. Tosun, A. Bener, B. Caglayan, and B. Turhan, "PREST: An Intelligent Software Metrics Extraction, Analysis and Defect Prediction Tool," *Proc. Int'l Conf. Software Eng. and Knowledge Eng.,* 2009.

[69] T. Menzies, C. Bird, T. Zimmermann, W. Schulte, and E. Kocaganeli, "The Inductive Software Eng. Manifesto: Principles for Industrial Data Mining," *Proc. Int'l Workshop Machine Learning Technologies in Software Eng.,* 2011.

[70] J.S. Armstrong, "Significance Tests Harm Progress in Forecasting," *Int'l J. Forecasting,* vol. 23, no. 2, pp. 321-327, 2007.

[71] J.S. Armstrong, *Principles of Forecasting: A Handbook for Researchers and Practitioners.* Kluwer Academic, 2001.

[72] R.M. Hogarth, "A Note on Aggregating Opinions," *Organizational Behavior and Human Performance,* vol. 21, no. 1, pp. 40-46, 1978.

[73] R.M. Hogarth and H. Kunreuther, "Pricing Insurance and Warranties: Ambiguity and Correlated Risks," *The GENEVA Papers on Risk and Insurance—Theory,* vol. 17, no. 1, pp. 35-60, 1992.

[74] Y. Li, M. Xie, and T. Goh, "A Study of Project Selection and Feature Weighting for Analogy Based Software Cost Estimation," *J. Systems and Software,* vol. 82, pp. 241-252, 2009.

[75] A. Bakir, B. Turhan, and A. Bener, "A New Perspective on Data Homogeneity in Software Cost Estimation: A Study in the Embedded Systems Domain," *Software Quality Control J.,* vol. 18, pp. 57-80, http://dx.doi.org/10.1007/s11219-009-9081-z, 2009.

[76] B. Kitchenham and K. Känsälä, "Inter-Item Correlations among Function Points," *Proc. 15th Int'l Conf. Software Eng.,* http://dl.acm.org/citation.cfm?id=257572.257677, pp. 477-480, 1993.

[77] C. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Comm. ACM,* vol. 30, no. 5, pp. 416-429, May 1987.

[78] K.D. Maxwell, *Applied Statistics for Software Managers.* Prentice Hall, 2002.

[79] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki, "Robust Regression for Developing Software Estimation Models," *J. Systems and Software,* vol. 27, no. 1, pp. 3-16, 1994.

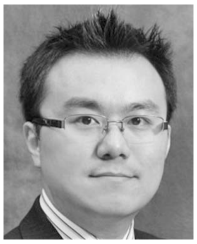
[80] G. Kadoda, M. Cartwright, and M. Shepperd, "On Configuring a Case-Based Reasoning Software Project Prediction System," *Proc. UK CBR Workshop Cambridge,* pp. 1-10, 2000.

[81] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[82] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks,* vol. 2, no. 5, pp. 359-366, 1989.

Ekrem Kocaguneli received the MSc and BS degrees in computer engineering from Bogazici University. He is working toward the PhD degree in computer science and electrical engineering at West Virginia University. His main research interests include software effort estimation, artificial intelligence applications in empirical software engineering, and intelligent tools to aid software processes. He is a student member of the IEEE and ACM.

Tim Menzies is an associate professor in computer science and electrical engineering at West Virginia University (WVU) and the author of more than 200 refereed publications. At WVU, he has been a lead researcher on projects for the NSF, NIJ, DoD, NASA's Office of Safety and Mission Assurance, as well as SBIRs and STTRs with private companies. He teaches data mining and artificial intelligence. He is the cofounder of the PROMISE conference series. In 2012, he will be the cochair of the program committee for IEEE Automated Software Engineering conference. He is a member of the IEEE.



Jacky W. Keung is an assistant professor in the Department of Computing at the Hong Kong Polytechnic University (PolyU). Prior to joining PolyU, he was a senior research scientist in the software engineering research group at NICTA. He also holds an academic position in the School of Computer Science and Engineering at the University of New South Wales. His current research interests include software engineering for cloud computing, machine learning, software measurement, cost estimation, quality control and risk management, and software process improvement. He is a member of the IEEE.