On the Value of Ensemble Effort Estimation

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

Abstract—
Background: There are still no clear principles for designing a single *best* effort estimation model. Given M models and X estimation problems, no model was reported to work best on all problems. Hence, it seems misguided to ever recommed *one* estimation model as *the best*.
Aim: We seek to test the following speculation: Perhaps combining the recomendations of *mulitple estimation models* into ensembles produces the best estimates.
Method: Nine learning algorithms were combined with ten pre-processors to generate $9 \times 10 = 90$ *solo-methods* which were then applied on 20 data sets and evaluated w.r.t. 7 error measures. The best n (in our case $n = 13$) solo-methods that proved stable accross datasets and error measures were set aside to be used in ensembles (from now on *multi-methods*). The top 2,4,8 and 13 solo-methods were combined via mean, median and inverse weighted error to get 12 multi-methods. These multi-methods were then compared to the solo-methods (all comparisons use a Wilcoxon test, 95% confidence).
Results: The best 9 (out of 12) multi-methods significantly out-performed *all* 90 solo-models.
Conclusion: While there is no best single effort estimation method, there exists best combinations of multiple effort estimation methods.

Index Terms—Software Cost Estimation, Analogy, *k*-NN

1 Introduction

Over or under-estimation of software development effort can lead to undesirable results. For example under-estimation may result in schedule and budget overruns, which may eventually lead to cancellation of the projects. Over-estimations on the other hand may hinder the acceptance of promising projects, which challenges the competitiveness of an organization. Hence, precise estimation of effort is vitally important yet extremely challenging for organizations.

After decades of research, although it is hard to propose a silver-bullet solution for software effort estimation, we can identify a number of *chronic* problems. A subset of these chronic problems that are addressed in this research are:

- Lack of consistently best learners
- Instability in reported conclusions

We use nine learning algorithms combined with 10 pre-processors, which amounts to $10 \times 9 = 90$ solo-methods. Solo-methods are combined in 12 different ways to form ensembles, which will be called as multi-methods from now on. In total *90 solo-methods + 12 multi-methods = 102 methods* methods are used and they are applied on 20 different effort estimation datasets. To the best of our knowledge, in terms of methods and datasets this is one of the most comprehensive effort estimation studies to be reported (see Table 4 of [1]). In addition, in software engineering (SE) there is a limited number of studies regarding the ensemble of solo-methods [2], [3] whose findings condemn the use of multi-methods.

In this research a novel multi-method scheme is proposed and unlike previous research we report that multi-methods consistently outperform solo-methods. The above statement is in disagreement with at least 2 prior conclusions. The first disagreement is conclusion instability: The collection of previous studies has resulted in a wide-variety of conlusions that are reported to be instable or contradictory [4]. After exploring a large space of datasets, methods as well as error measures, the results in this study show that it is possible to report stable conclusions. The second disagreement is in the performance of multi-methods: Unlike previous research, we favor the use of multi-methods on SE data, provided that suitable ensemble methods are applied.

The contributions of this research can be summarized as follows:

- Successful results (unlike previous research) on multi-methods applied on effort data
- A novel scheme for ensembling only the best solo-methods into multi-methods
- A method of evaluating stability of methods
- Stable multi-methods that outperform *all* solo-methods
- A wide study of 20 datasets and 102 methods

Below is the list of research questions that were defined to guide in this work. Those questions will be revisited in §6.1 and detailed answers will be provided.

- RQ1: What are the effects of using multi-methods?
- RQ2: Why do not ensemble methods work in SE?
- RQ3: What can the experiments that use 102 methods over 20 datasets tell us about stability issues?
- RQ4: What is the best effort estimator?

- *Ekrem Kocaguneli is with the Department of Computer Engineering, Bogazici University. E-mail: ekrem.kocaguneli@boun.edu.tr*
- *Tim Menzies is with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: tim@menzies.us*
- *Jacky W. Keung is with the Department of Computing, The Hong Kong Polytechnic University. E-mail: Jacky.Keung@comp.polyu.edu.hk*

This research is funded in part by NSF,CISE, project #0810879

- RQ5: What are the implications of our results regarding software effort datasets?

The rest of this paper is structured as follows. We continue with the motivation behind this research in §2. In §8 we provide background information. Then in §4 we summarize the methods (algorithms, pre-processors and ensemble schemes) that we used. We explain our research methodology in §5, and we provide our results in §6. The threats to the validity of our results are given in §7 Finally we conclude the paper with a discussion in §9.

2 Motivation

This research is based on the lessons learned from previous work [2], [3], [5]–[7]. The work in SE domain reports that multi-methods perform poorly [2], [3]. On the other hand machine learning (ML) community perspective is that multi-methods are successful learners in practical [7], [8] as well as in theoretical settings [5]–[7].

The contradicting results urged us to investigate the reasons behind them. One of the differences we identified were the schemes used to build multi-methods. When constructing a multi-methods, it is particularly useful if there is diversity in the learning process [5], [8]. This can be achieved by 3 factors: 1) different learners, 2)different feature subsets and 3) different training sets. In this section, we suffice to say that in SE literature, these factors were not deeply investigated. We provide a deeper discussion on that issue in §9.

Another motivating factor for us came during the preliminary analysis. In Figure 1 cells represent the performance of methods (y-axis) on individual instances (x-axis) in terms of MRE (for MRE see §5). Different MRE intervals are represented with different shades of gray and we see that all intervals can occur along a line $x = i$ where $i \in \{1, ..., 102\}$. This tells us that best estimate for a project is given by different methods. We can make use of this fact by combining the estimations of *stably-successful* solo-methods. More details on the definition and combination of *stably-successful* solo-methods will be provided in §5.

3 Research Questions

Below is the list of research questions that we investigated. We will revisit each question in §6.1 and provide detailed answers.

- RQ1: What are the effects of using multi-methods?
- RQ2: What are the reasons behind the particular effects of multi-methods?
- RQ3: What can the experiments that use 102 methods over 20 datasets tell us about stability issues?
- RQ4: What is the best effort estimator?
- RQ5: What are the implications of our results regarding software effort datasets?

4 Methods

In our experiments, we used 10 different pre-processors and 9 learners. For the selection of the learners/pre-processors in our research, we focused on 2 considerations. Firstly we wanted our learners to be practical and to map the field of software effort estimation literature [4], [9]–[16]. Our second consideration was to form a collection in which learners with different assumptions and different biases would come together. Because, learning is an ill-posed problem and ensemble of learners with similar decisions do not make much contribution [5]. In other words, with finite amount of data every learner is supposed to fail under certain conditions and hence it is recommended to use different learners that fail under different circumstances [5]–[8].

Pre-processors we chose are:

- Three *simple preprocessors*: **none, norm, and log**;
- One *feature synthesis* method: **PCA**;
- Two *feature selection* methods: **SFS** and **SWreg**;
- Four *discretization* methods: divided on equal frequency/width.

Learners are:

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

A summary table for abbreviations of the methods and pre-processors is provided in Figure 2. For detailed descriptions please refer to Appendix B.

4.1 Solo/Multi-Methods

Models in effort estimation can be grouped as elaboration of single algorithms (i.e. finding the optimum parameters) or combination of these algorithms with various pre-processors. We define a *solo-method* as the combination of a pre-processor with an algorithm. In the previous section we have defined 9 algorithms and 10 pre-precossers. We combine these two sets to form *9 algorithms X 10 pre-processors = 90 solo-methods*.

We define *multi-methods* as the combination of at least two solo-methods (by definition multi-methods include multiple algorithms and pre-processors). There is a wide range of combination schemes proposed [5], [8], [10], [17]. It can be a simple scheme such as taking mean, median or inverse-ranked weighted mean (IRWM [10]) of estimates coming from *n-many* solo methods. Alternatively it can be a more complex procedure such as bagging [18], boosting [19] or random forests [20], [21]. Our aim is not to investigate complex schemes, but to observe how multi-methods perform compared to solo-methods on effort datasets. Therefore, we adopt simple schemes (mean, median and IRWM) and leave investigation of complex combinations to future work.

5 Methodology

5.1 Datasets

Among many, two very important characteristics that software effort estimation studies should have are:

- repeatability, i.e. reported experiments should be repeatable by other researchers [22], [23]
- and external validity, i.e. proposed solutions should not be applicable to a broad setting.

In the hearth of these characteristics lie the datasets. If the datasets are kept confidential, then it is almost impossible to

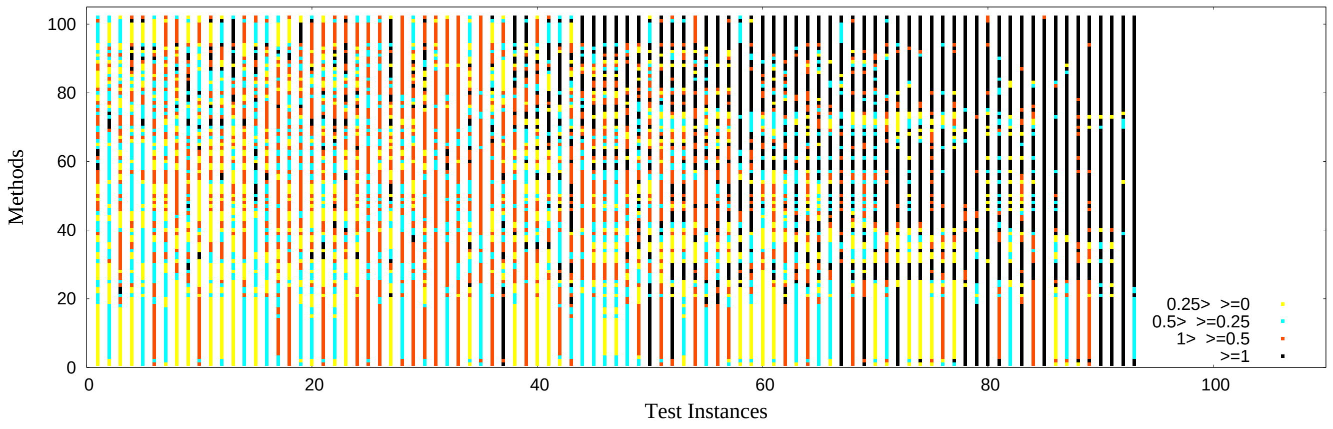

Fig. 1. The MRE values of all the methods for Nasa93. The lines along y-axis represent how the MRE values of a single instance change with respect to different methods. The fact that lines are mostly multi-colored means that different instances are best estimated by different methods. The order of methods in y-axis is given in Figure 9 and the instances on the x-axis are sorted from the lowest sum of MRE over 102 methods to the highest.

Pre-Processors		**Algorithms**	
Abbreviation	**Explanation**	**Abbreviation**	**Explanation**
norm	Normalization	$\star$ ABE0-1NN	Basic ABE with 1 nearest neighbor
log	Taking natural logarithm	$\star$ ABE0-5NN	Basic ABE with 5 nearest neighbors
PCA	Principal Component Analysis	SWReg	Stepwise Regression
SFS	Sequential Forward Selection	CART-On	Classification and Regression Tree with pruning
SWReg	Stepwise Regression	$\star$ CART-Off	Classification and Regression Tree without pruning
width3bin	Discretize into 3 bins based on equal width	$\star$ NNet	Neural Net with one hidden layer
width5bin	Discretize into 5 bins based on equal width	SLReg	Simple linear regression
freq3bin	Discretize into 3 bins based on equal frequency	PCR	Principal components regression
freq5bin	Discretize into 5 bins based on equal width	PLSR	Partial least squares regression
none	Apply no pre-processor		

Fig. 2. The summary table for the *solo-methods*. This table provides a list of abbreviations as well as their explanation for pre-processors and learners used in this research. The naming convention in this table is provided throughout the paper. $\star$ indicates that previous work [2], [3] used a similar method in their study.

replicate this study and improve the research. If they are public yet very limited in number, then the results are vulnerable to the issues of generalizability.

Our aim is to conduct a sound study in terms of generalizability and to promote replication of the reported results. In our experiments, we used 20 publicly available datasets and to the best of our knowledge this is one of the highest number of datasets to be reported in an effort estimation study. The names and properties of these datasets are provided in Figure 3, explanatory notes are given in Appendix A.

5.2 Error Measures

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 [13], [24]. MRE measures the error ratio between the actual effort and the predicted effort and can be expressed as the following equation:

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

A related measure is MER (Magnitude of Error Relative to the estimate [24]):

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

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

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

$$MdMRE = median(allMRE_i) \tag{5}$$

A common alternative error measure is PRED(25), which can be defined as the percentage of predictions falling within 25% 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, PRED(25)=50% implies that half of the estimates are falling within 25% of the actual values [13].

				Historical Effort Data					
Dataset	**Features**	**Size**	**Description**	**Units**	**Min**	**Median**	**Mean**	**Max**	**Skewness**
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 Chines software companies	hours	26	1829	3921	54620	3.92
		Total: 1198							

Fig. 3. The 1198 projects used in this study come from 20 data sets. Indentation in column one denotes a dataset that is a subset of another dataset. For notes on this datasets, see Appendix A.

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. [24]:

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

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

Interpreting these error measures without any statistical test may be misleading. A recent discussion about this issue can be found in [23]. To evaluate our results subject to a statistical test, we make use of so called *win-tie-loss* statistics. Win-tie-loss statistics employ a Wilcoxon non-parametric statistical hypothesis test with 95% confidence. Wilcoxon is more robust than the Student's *t*-test as it compares the sums of ranks, unlike 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 [25].

We stored the performance of every method w.r.t. 7 error measures over 20 datasets. This enabled us to collect *win-tie-loss* statistics using the algorithm of Figure 4. In Figure 4, we first check if two distributions i, j are statistically different according to the Wilcoxon test (95%); 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.

5.3 Experimental Design

The experimental design of our research is twofold: Finding the order of solo-methods that are to be combined into multi-methods and comparing performance of solo-methods to that of multi-methods.

The first part aims at finding the *stably-successful* solo-methods. For that purpose we consider 2 criteria: 1) Success and 2) stability. To find the successful solo-methods, the procedure of Figure 4 is repeated seven times (once for AR,

```
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 compare($E_i$) , median($E_j$) favors $i$ 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. 4. Comparing algorithms (*i*,*j*) according to performance measures E_i and E_j.

MRE, MER, MdMRE, MMRE, PRED(25), and MIBRE). Given 89 comparisons, seven performance measures and 20 datasets, the maximum number of losses for any method is $89 \times 7 \times 20 = 12,460$. 90 solo-methods are then sorted by their total number of losses. The solo-method, with fewest losses is ranked #1 and the next one is ranked #2 and so on, which we would like to call *initial-ordering*.

As for the stability we take a look at the difference between initial-ordering and the order given by each individual error measure. In Figure 5, we report the mean of maximum rank changes for each solo-method w.r.t. their initial ordering:

- Each error measure defines its own ordering of solo-methods w.r.t. its win, $loss$ or $win - loss$ values.
- Maximum rank change is the maximum absolute difference between either of these orderings and the initial-ordering.
- Then, mean of maximum rank changes coming from 7 error measures gives us Figure 5.

A line drawn parallel to x-axis at $y = 10$ gives methods, whose mean rank change is less/more than 10. See in Figure 5 that $y = 10$ line divides all methods into 3 regions: a (from method 1 to 13), b (from method 14 to 64) and c (from method 65 to 90). Region a contains *"successful"* and *stable* solo-methods that we will use for generating multi-methods. The list of *stably-successful* 13 methods is given in Figure 6.

To form multi-methods, the estimates of top 2, 4, 8 and 16 solo-methods are combined by taking the mean, median and IRWM of them. With this scheme, we have *4 (2, 4, 8, 13 solo-methods) x 3 (mean, median, IRWM) = 12 multi-methods.*

In the second part of our experimental design we include both solo-methods as well as multi-methods in the procedure of Figure 4. This gives us the comparison of *90 solo-methods + 12 multi-methods = 102 methods*. Every method is compared to 101 others with respect to seven error measures and over 20 datasets. Therefore, the maximum number of losses for any method now becomes $101 \times 7 \times 20 = 14,140$.

5.4 Discussion on Testing Strategy

An important point to consider in experimentation is the testing strategy, i.e. how to divide the dataset into test and train sets. The common methods for testing strategy are leave-one-out (LOO) and n-way cross-validation. As long as effort estimation literature is concerned, there are vast amount of studies employing either one of these testing strategies. Choosing either one of these methods theoretically bear different types of bias/variance trade offs. To see these trade-offs we have compared the bias/variance values of LOO and n-way on 20 datasets. This issue deserves a study on its own right that we will share in the near future. Here we want to suffice by saying that we have not seen enough statistical difference between bias/variance values of LOO and n-way. As an example to that scenario, Figure 7 shows the bias and variance values generated by solo-methods under LOO and 3-Way (provided that they are statistically different). For Nasa93, the number of methods that generated significantly different bias and variance values are 29 and 30 respectively. Even in that case, notice how bias and variance values of LOO and 3-Way overlap, i.e. even if they are statistically different, they are still too close to favor any of the testing strategies. Therefore, to compare the performance of solo and multi-methods we wanted to use as many instances as possible and we opted for LOO: Given N projects in a dataset, then $\forall N_i \in N$, use N_i as the test set and the remaining $N-1$ projects as the training set.

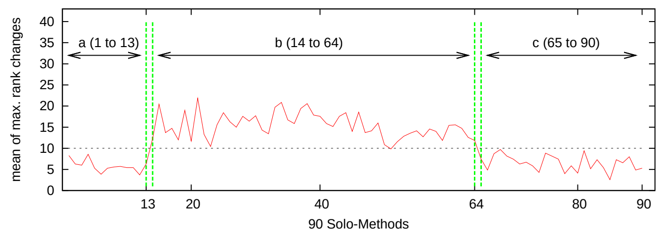

Fig. 5. Algorithms and the mean of their maximum rank changes w.r.t. initial-ordering. Mean rank change of smaller than 10 divides 90 methods into 3 regions. Region a consists of high-ranked stably-successful methods, whereas region çontains low-ranked but still stable methods. Region b on the other hand shows middle-ranked and non-stable methods.

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

Fig. 6. Rank of top-13 *stably-successful* solo-methods. These solo-methods are combined in various ways to form 12 multi-methods.

6 Results

We first evaluate our methods according to their total *loss* values over all error measures and datasets. For that purpose, the procedure of Figure Figure 4 was repeated for every error measures and dataset. Then 102 methods were sorted by their total number of losses over all datasets, i.e. we expect the best method to lose the least when compared to the others The resulting order is provided in Figure 9. The method, with fewest losses (**Top13/Mean**) is ranked #1 and the one with the most losses (**PCA/LReg**) is ranked #102 . According to this order, 9 out of 12 multi-methods appear as the best 9 methods.

When providing our results and answering the research questions, we would like to keep the ordering of methods as well as datasets fixed. This helps us better track the changes in different experimental settings. We will use the order provided in Figure 9 for that. In a similar fashion we also sorted the datasets according to total number losses. Figure 8 presents the sorted datasets from the lowest to the highest number of

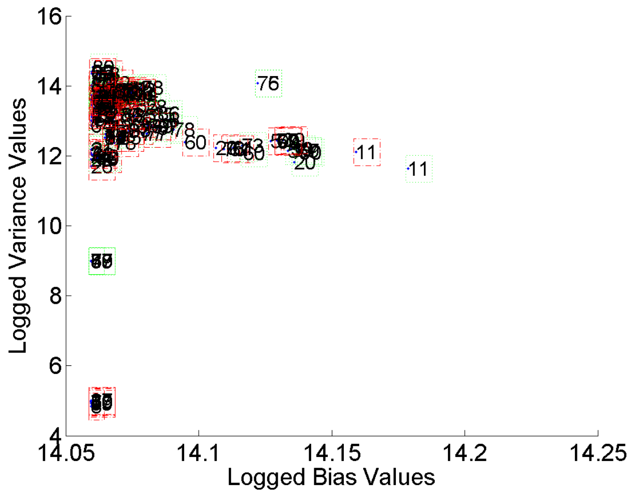

Fig. 7. The logged bias/variance values of LOO and 3-Way cross-validation for Nasa93. Significantly different bias and variance values were 29 and 30 (respectively) out of 90 methods. Here all 90 bias/variance values are shown, and it is no surprise that they are significantly very similar: Notice how most of them overlap.

losses over all error measures.

rank	dataset
1	telecom
2	kemerer
3	cocomo81o
4	cocomo81s
5	desharnaisL1
6	desharnaisL3
7	albrecht
8	desharnaisL2
9	cocomo81e
10	nasa93_center_5
11	desharnais
12	maxwell
13	sdr
14	nasa93_center_1
15	miyazaki94
16	nasa93_center_2
17	finnish
18	cocomo81
19	nasa93
20	china

Fig. 8. Rank of datasets according to least number of total losses over all methods for all performance measures. This ranking is kept fixed in all graphs that use the datasets in one of its axises.

Now that we know the order of methods according to sum of $loss$ values over all datasets and error measures, we are interested in a more detailed analysis. In Figure 10, each x,y intercept corresponds to a method applied on a dataset. An intercept shows the result of $win - loss$ values of a method as a percentage of 707 comparisons. Each method compared to 101 other methods using seven performance measures: $101 \times 7 = 707$ comparisons. So the highest $win - loss$ value is 707, where a method wins all comparisons. The y-axis shows 102 methods (sorted in the rank order of Figure 9) and the x-axis shows 20 datasets (sorted in the rank order of Figure 8).

When we look at Figure 10 we can see that there is a considerable uniformity on top ($y = 80$ and upper) and bottom ($y = 24$ and lower) parts. This tells us that good (bottom part) and bad (upper part) performing methods consistently appear within certain percentages of $win - loss$ values. Also note that including line $y = 9$ and below (i.e. top performing methods) are the multi-methods. We can observe that multi-methods are among the successful methods, however it is difficult to distinguish them from other successful methods (in region $11 \leq y \leq 24$). Therefore, we will resort to the method of Figure 5.

To provide a more detailed alternative perspective of Figure 10 we group methods on the y-axis into 3 region and observe the MdMRE values of each chunk. The first region includes the top 9 methods (9 multi-methods) of Figure 9. The second region includes methods from $11 \leq y \leq 80$, where most of the average-performing solo-methods are located and finally the third region ($81 \leq y \leq 102$) contains the low-performing methods. In Figure 11 we show the sorted MdMRE values from lowest to highest for each region. For better visualization, we scaled the MdMRE values by taking their

rank	pre-processor	learner	rank	pre-processor	learner
1	Top13	Mean	52	freq3bin	ABE05NN
2	Top13	Irwm	53	SWReg	SWReg
3	Top2	Mean	54	norm	SWReg
4	Top4	Mean	55	none	SWReg
5	Top2	Median	56	PCA	ABE05NN
6	Top4	Median	57	width3bin	ABE05NN
7	Top8	Median	58	SFS	SWReg
8	Top2	Irwm	59	width5bin	ABE01NN
9	Top4	Irwm	60	width5bin	SWReg
10	norm	CART (yes)	61	freq5bin	ABE01NN
11	norm	CART (no)	62	freq3bin	ABE01NN
12	none	CART (yes)	63	PCA	ABE01NN
13	none	CART (no)	64	width5bin	ABE05NN
14	Top8	Irwm	65	PCA	NNet
15	Top13	Median	66	none	NNet
16	log	CART (yes)	67	SWReg	NNet
17	log	CART (no)	68	freq3bin	CART (yes)
18	Top8	Mean	69	freq3bin	CART (no)
19	SWReg	ABE01NN	70	SFS	NNet
20	SWReg	CART (yes)	71	norm	PLSR
21	SWReg	CART (no)	72	none	LReg
22	log	ABE01NN	73	SWReg	LReg
23	SFS	CART (yes)	74	norm	LReg
24	SFS	CART (no)	75	width3bin	SWReg
25	SWReg	ABE05NN	76	log	SWReg
26	SFS	ABE01NN	77	width5bin	PLSR
27	SFS	ABE05NN	78	log	PCR
28	norm	ABE01NN	79	log	PLSR
29	PCA	PLSR	80	width5bin	PCR
30	none	ABE01NN	81	width3bin	PLSR
31	none	PLSR	82	norm	PCR
32	SWReg	PCR	83	width3bin	ABE01NN
33	width5bin	CART (yes)	84	width3bin	PCR
34	width5bin	CART (no)	85	freq5bin	PCR
35	PCA	PCR	86	width3bin	LReg
36	freq5bin	CART (yes)	87	freq5bin	SWReg
37	freq5bin	CART (no)	88	freq5bin	PLSR
38	freq5bin	ABE05NN	89	width5bin	LReg
39	none	PCR	90	freq3bin	PCR
40	PCA	CART (yes)	91	freq3bin	PLSR
41	PCA	CART (no)	92	log	LReg
42	norm	ABE05NN	93	freq3bin	SWReg
43	none	ABE05NN	94	freq5bin	LReg
44	SFS	LReg	95	width5bin	NNet
45	SWReg	PLSR	96	width3bin	NNet
46	log	ABE05NN	97	norm	NNet
47	SFS	PLSR	98	log	NNet
48	SFS	PCR	99	freq5bin	NNet
49	PCA	SWReg	100	freq3bin	NNet
50	width3bin	CART (yes)	101	freq3bin	LReg
51	width3bin	CART (no)	102	PCA	LReg

Fig. 9. Detailed algorithm combinations, sorted by the sum of their losses seen in all performance measures and all data sets. The algorithm with fewest losses is ranked #1 and is **Top13/Mean**. At the other end of the scale, the algorithm with the most losses is ranked #102 and is **PCA/LReg**.

log's. In Figure 11 it is clear that the 1^{st} region has a lower MdMRE rate and note that all methods of the 1^{st} band were multi-methods.

In Figure 11 we can see how the performance of solo and multi-methods behave, when grouped into regions. However, evaluation based on performance only may be misleading, in the sense that the order of methods observed w.r.t. one error measure may be different to the ordering of another error measure. Therefore, we expect successful methods to be consistent as well, i.e. we expect the ranks not to change from one measure to another measure.

Figure 12 shows the mean of maximum rank changes for all the methods in accordance with the procedure of Figure 5. In Figure 12 we observe that band1 (region of multi-methods) as well as band3 (region of low-performing methods)

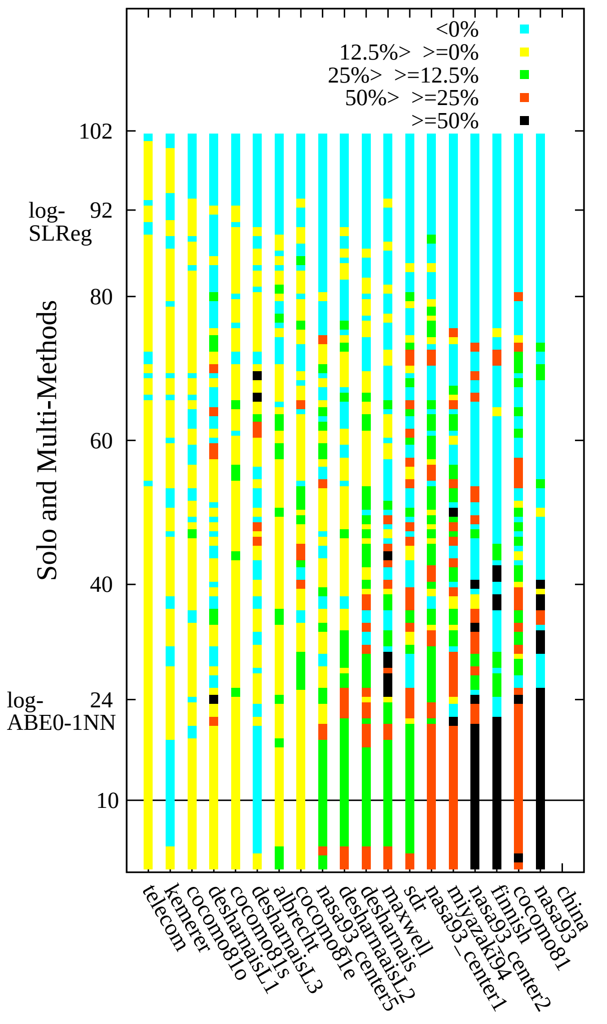

Fig. 10. Win-Loss Percentage. Number of wins minus losses ($win - loss$) seen in 102 methods and 20 datasets, expressed as a percentage of the maximum wins minus minimum losses possible for one method in one dataset. The maximum number of wins for one dataset is: *(102 methods - itself)*7 performance measures = 101*7 = 707* and minimum number of losses possible is 0, so *12.5%* is *707*0.125*, *25%* is *707*0.25* and so on. The x-axis is the 20 datasets and the y-axis is all 102 the methods. The order of algorithms is the same as those given in Figure 9. Below the line extending from the left to right at $y = 9$ all correspond to multi-methods. Note that for some algorithms, the loss values are greater than win values, in which case $win - loss$ has a negative value.

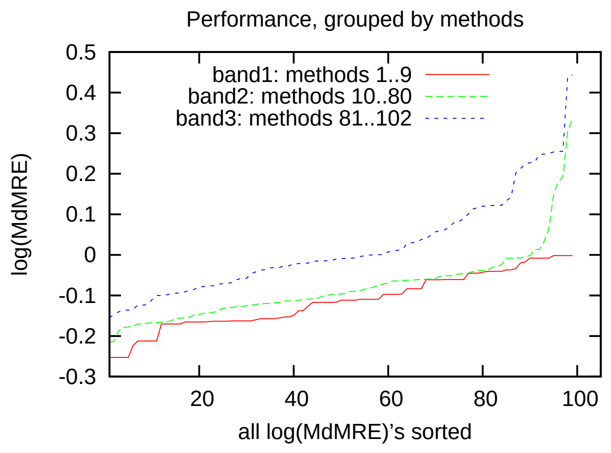

Fig. 11. Spectrum of MdMRE values for 3 bands of methods. Here we keep the order of methods same as those given in Figure 9 and divide those methods into 3 regions. Notice how the first region (all multi-methods) attain the lowest MdMRE scores.

have relatively lower mean-change values. The mean-change amount for band2 on the other hand can go as high as 30. This scenario is similar to that of Figure 5: High performers (in that case multi-methods) and low-performers are more stable than average-performing methods.

One of the most important reading of Figure 5 is that the most successful method **Top13/Mean**, has a mean-change of 1, which means that among all datasets, and across all error measures, **Top13/Mean** consistently performed as the best method. When Figure 12 is interpreted together with the previous performance plots, we can say that multi-methods behave stably and perform consistently better than solo-methods accross multiple datasets and multiple performance criteria.

6.1 Answers to Research Questions

RQ1: What are the effects of using multi-methods? Given the stably-successful solo-methods were discovered first and combined into multi-methods, we saw that multi-methods have more successful and more stable results. Among 102 methods top 9 methods were multi-methods, , i.e. most robust and accurate estimates were generated by multi-methods. The best method was **Top13/Mean** and had a mean rank-change of 1, which means that it was stable accross all error measures as well as datasets. Therefore, the observed effect of using multi-methods on effort data can be summarized as generating lower

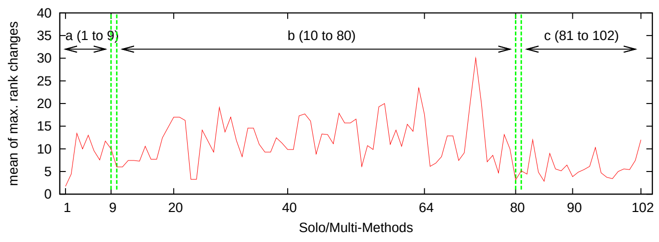

Fig. 12. Mean rank changes of solo and multi-methods. Region *a* contains 9 out of 12 multi-methods. See that **Top13/Mean** at $x = 1$ on x-axis has a mean rank change of 1, i.e. it outperforms all other methods w.r.t. 7 different error measures and 20 datasets. Notice also that bands *a* and *c* have mean rank changes around 10, whereas band *c* can go as high as 30.

error rates and having more stable results.

Why do not ensemble methods work in SE? Reading from prior effort estimation literature, there are 2 reasons we can propose: 1) The combination scheme and 2) pre-processors. Prior work on SE data have used all solo-methods in an *as is* manner, disregarding the success and stability criteria. Our combination scheme for multi-methods however first discovers the stably-successful solo-methods and combines only those that perform better than the rest. Secondly, in previous work no data pre-processors were employed (see Figure 2 where common methods/learners with previous work are marked). However, fundamental diversity factors in a multi-method scheme are different feature subsets and different training sets [5], [8]. Different pre-processors are able to provide these factors. We also have to say that to provide *exact* answers, a comperative study between prior and currently reported work is required.

RQ3: What can the experiments that use 102 methods over 20 datasets tell us about stability issues? We acknowledge and fully agree with prior work [24], [26], [27] that there is a big region of unstable learners (see high mean rank-changes in Figure 5 and Figure 12). However, with this study we have seen that **Top13/Mean** performed better than the rest of the algorithms accross 7 error measures and 20 datasets. Therefore, there are possible approaches such as multi-methods that could cure the problem of instable results in software effort estimation.

RQ4: What is the best effort estimator? Simply put we have no evidence suggesting a solo-method as the best learner. Instead, we applied the idea of ensemble of learners. We have seen that 9 out of 12 multi-methods outperformed all others and **Top13/Mean** stably outperformed all the rest. This study suggests multi-methods as the path to the best learner and for the datasets at hand **Top13/Mean** occurs as the best learner. However, we highlight the fact that we require more datasets and replications of this study to solidify this claim.

RQ5: What are the implications of our results regarding software effort datasets? It is true that we can propose an ordering of datasets depending on the total $loss$ values of methods on each dataset (Figure 8). Also it is true according to Figure 10 that some datasets are highly challenging (telecom, kemerer, cocomo81o do not let any method to attain more than 12.5% $win-loss$ ratio) whereas others are easier for a number of methods (see black bars in Figure 10 for nasa93_center_2, finnish, nasa93, china datasets). However, this is a rather broad topic that requires a study on its own right and our findings are hints for future work rather than strong claims.

7 Threats to Validity

Construct validity (i.e. face validity) asks if we are measuring what we actually intended to measure [28]. Previous studies have concerned themselves with the construct validity of different performance measures for effort estimation (e.g. [24]). To make sure that we do not favor a particular conclusion due to limited number of performance measure, we used 7 error measures and questioned the stability of our conclusions. We showed empirically the surprising result that multi-methods are stable across a range of performance criteria.

External validity questions whether the results can be generalized outside the specifications of a study [29]. To ensure external validity, this paper has studied a large number of projects. For example, Table 4 of [1] list 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 1198 projects used here.

It is clear that this study has not explored the full range of effort estimation algorithms. Clearly, future work is required to repeat this study using the top performing solo-methods found here and possibly more. Having cast doubts on the external validity of our algorithms, we hasten to add that we selected algorithms that have been extensively studied in the literature [13] as well as have been applied on the commonly available datasets. That is, we assert that these results should apply to much of the current literature on effort estimation.

8 Related Work

The previous work in literature related to our research can be grouped into 3 categories: Software effort estimation, ensemble methods in SE and ensemble of methods in other domains. The following sub-sections provide background information for each research category.

8.1 Effort Estimation

In this subsection we first make a sketch of empirical research studies on cost estimation in the last 15 years and comment on the techniqes employed by these studies. Then we biefly discuss the stability issues arising from previous results.

8.1.1 Algorithmic and Non-algorithmic Methods

There are many reported algorithmic methods in the literature. Various kinds of regression (simple, partial least square, stepwise, regression trees), neural networks and instance-based algorithms are just a few examples to algorithmic methods. Our research alone uses 9 different algorithmic methods; for more notes on our methods see §B.2. There is even a wider space associated with the options of those algorithms. For example, only the space of options associated with instance based algorithms exceed thousands [30].

The alternative to algorithmic approaches is the non-algorithmic methods. Non-algorithmic methods, a.k.a. expert-based estimation can be defined as a human-intensive process of negotiating the estimate of a new project and arriving at a consensus [31]. There are formal methods proposed for expert-based estimation like Delphi [32]. However, it is mostly the case that companies follow an informal process for expert-based estimation [33]. In [34] Jorgensen defines some guidelines so as to generate realistic software effort estimates. An important finding in Jorgensen's study, which is in parallel with our findings is that *combining estimations* coming from different sources (e.g. from experts and instance-based learners) offers the most robust and accurate combination method, as they capture a broader range of information related to the estimation problem.

8.1.2 Conclusion Instability

A *chronic* problem in software effort estimation that we address in this research is the matter of instability. An important work on that issue was conducted by Shepperd et. al., where they compare various methods and question the links between accuracy, prediction system and different datasets [27]. The experimentation in [27] uses very large datasets simulated from an actual dataset in accordance with underlying distributions. Their conclusions are: 1) no existing method is consistently the best, 2) performance of a method is dependent on the dataset and 3) it is not feasible to determine the best method. Along the same line with that study is [26], where Menzies et. al. used 158 learners and found out that the precise rankings changed according to the random number seeds used to generate train/test sets. However, an interesting finding was that four methods consistently outperformed the other 154 across all datasets, across 5 different random number seeds, and across three different evaluation criteria.

Reading from these previous studies, our opinion is that it is appropriate to revisit the conclusion instability problem. Another reason for us to revisit that problem is the availability of many more public software effort estimation datasets for further stability research. 20 datasets which have become avaliable in the last year at the PROMISE repository of reusable SE data[1] are listed in Figure 3.

8.2 Ensemble of Methods

Although most of the available estimation methods may perform acceptably good, it is usually the case that one method does not always perform the best. In that case the general approach suggested is to try different methods and choose the one that best explains the data [5]. However, ensemble of learners, i.e multi-methods take a different approach and they construct a combination scheme for solo-methods [6]. The start point of multi-methods is that each paradigm comes with its own assumptions [5] and it is likely that the patterns that violate these assumptions do not overlap [5], [6], [8]. Therefore, in an appropriate combination scheme these methods can complement one another to a certain extent and attain better estimation performance. For example under the assumption of mean-squared-error it has been proven that multi-methods attain smaller or equal error rates than single-methods [35].

It is a recommended practice to combine solo-methods that have different characteristics [5], [8], [17]. There are a couple of different ways to attain different-characteristic multi-methods. The first way is through representation of the data. The multi-method structure may be based on uni-representation (all learners use same representation of data) or multi-representation (different learners use different representations) [5]. Examples to such strategies are the use of different feature sets [36], [37] or different training sets [38]. The second way is through architectural methodologies. For example the solo-methods may work in parallel or they may work one after another)(bagging and boosting) [5]. In the former architecture, solo-methods provide their predictions and a combining mechanism consolidates these predictions. In the latter case solo-methods form a multi-stage scheme of learners.

1. http://promisedata.org/data

8.3 Ensemble of Methods in SE

The multi-method studies that used combination of different learners in SE reports pessimistic results. For example in [2] Khosgoftaar et. al. question the performance of different multi-method schemes under different scenarios in the domain of software quality. They use different combinations of 17 learners induced on 7 datasets and report that multi-methods induced on single datasets do not yield a significant increase in prediction accuracy. In [3] Kocaguneli et. al. replicate [2] in the domain of software effort estimation. They exploit combination of 14 methods applied on 3 software effort estimation datasets. The conclusion in [3] is similar to that of [2]: The application of multi-methods under different scenarios does not provide a significant increase in the estimation accuracy. In [39], different learners were employed in two types of committees where one was reported to be successful. Baker et. al. experiments with various ensemble settings (expert-based with model based, boosting, bagging) in [40] and reports very poor performance of ensembles on COCOMO-based datasets. One example to ensembles is the committee of *single-type* learners, where multiple versions of a single algorithm are combined. Pahariya et. al. use linear combinations of genetic algorithms in [41], where they report improvements over single-learners. In [12] Kultur et. al. report improvements through ensemble of neural networks. However, these single-learner based methods fall into the category of solo-methods (since there is only one type of learner) in this work. There are also some applications of ensemble methods in effort estimation so as to process datasets: In [42], Twala et. al. use multiple imputation techniques to handle missing data and in [43] Khoshgoftaar et. al. make use of learner ensembles as a filter to improve the data quality.

9 Discussion

When constructing a multi-methods scheme, it is particularly useful if there is diversity in the learning process [5], [8]. This can be achieved by 3 factors: 1) different learners, 2)different feature subsets and 3) different training sets. We can say that in SE literature, these factors were not deeply investigated. In this research we included these concepts into our scheme of multi-methods and the results proved to be successful.

Also the performance of multi-methods reported in prior studies [2], [3] are not encouraging to further question this area, at least for SE. Therefore, our research serves the purpose of providing extensive results in a SE area that are not very much questioned. We also hope that the promising results reported in this paper would prompt other researchers to further question the multi-method applications.

References

[1] B. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 316–329, 2007, member-Kitchenham, Barbara A.

[2] 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.

[3] E. Kocaguneli, Y. Kultur, and A. Bener, "Combining multiple learners induced on multiple datasets for software effort prediction," in *International Symposium on Software Reliability Engineering (ISSRE)*, 2009, student Paper.

[4] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, 2007.

[5] *Techniques for Combining Multiple Learners*. ICSC Press, 1998.

[6] T. G. Dietterich, "Ensemble methods in machine learning," pp. 1–15, 2000.

[7] J. Ghosh, "Multiclassifier systems: Back to the future," in *MCS '02: Proceedings of the Third International Workshop on Multiple Classifier Systems*. London, UK: Springer-Verlag, 2002, pp. 1–15.

[8] J. Kittler, I. C. Society, M. Hatef, R. P. W. Duin, and J. Matas, "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 226–239, 1998.

[9] K. Lum, T. Menzies, and D. Baker, "2cee, a twenty first century effort estimation methodology," *ISPA / SCEA*, pp. 12 – 14, 2008.

[10] 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 Engineering*, vol. 8, no. 2, pp. 163–196, 2003.

[11] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *ICSE '96: Proceedings of the 18th international conference on Software engineering*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 170–178.

[12] Y. Kultur, B. Turhan, and A. B. Bener, "Enna: software effort estimation using ensemble of neural networks with associative memory," in *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, New York, NY, USA, 2008, pp. 330–338.

[13] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, 1997.

[14] C. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Trans. on Computers*, pp. 1179–1185, 1974.

[15] A. Venkatachalam, "Software cost estimation using artificial neural networks," in *Proceedings of international joint conference on neural networks*, 1993, pp. 987–990.

[16] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches - a survey," *Annals of Software Engineering*, vol. 10, pp. 177–205, 2000.

[17] K. Ali, "On the link between error correlation and error reduction in decision tree ensembles," Tech. Rep., 1995.

[18] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[19] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, 1997.

[20] Y. Jiang, B. Cukic, and T. Menzies, "Cost curve evaluation of fault prediction models," in *ISSRE '08: Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*, 2008, pp. 197–206.

[21] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[22] G. Gay, T. Menzies, B. Cukic, and B. Turhan, "How to build repeatable experiments," in *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2009, pp. 1–9.

[23] B. Kitchenham and E. Mendes, "Why comparative effort prediction studies may be invalid," in *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2009, pp. 1–5.

[24] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Transactions on Software Engineering*, 2003.

[25] J. Kliijnen, "Sensitivity analysis and related analyses: a survey of statistical techniques," *Journal Statistical Computation and Simulation*, vol. 57, no. 1–4, pp. 111–142, 1997.

[26] T. Menzies, O. Jalali, J. Hihn, D. Baker, and K. Lum, "Stable rankings for different effort models," *Automated Software Engineering*, no. 4, December 2010.

[27] M. Shepperd and G. F. Kadoda, "Comparing software prediction techniques using simulation," *IEEE Trans. Software Eng*, vol. 27, no. 11, pp. 1014–1022, 2001.

[28] C. Robson, "Real world research: a resource for social scientists and practitioner-researchers," *Blackwell Publisher Ltd*, 2002.

[29] D. Milic and C. Wohlin, "Distribution patterns of effort estimations," in *Euromicro*, 2004.

[30] E. Kocaguneli, "Better methods for configuring case-based reasoning systems," Master's thesis, 2010.

[31] M. Jø rgensen, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, vol. 70, pp. 37–60, 2004.

[32] Rowe and Wright, "The delphi technique as a forecasting tool: issues and analysis," *International Journal of Forecasting*, vol. 15, 1999.

[33] M. Shepperd and M. Cartwright, "Predicting with sparse data," *IEEE Trans. Softw. Eng.*, vol. 27, no. 11, pp. 987–998, 2001.

[34] M. Jorgensen, "Practical guidelines for expert-judgment-based software effort estimation," *IEEE Softw.*, vol. 22, no. 3, pp. 57–63, 2005.

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

[36] T. K. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 1, pp. 66–75, 1994.

[37] S. Günter and H. Bunke, "Feature selection algorithms for the generation of multiple classifier systems and their application to handwritten word recognition," *Pattern Recogn. Lett.*, vol. 25, no. 11, pp. 1323–1336, 2004.

[38] T. K. Ho, "Random decision forests," in *ICDAR '95: Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1)*. Washington, DC, USA: IEEE Computer Society, 1995, p. 278.

[39] M. C. K. Vinaykumar, 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.

[40] D. Baker, "A hybrid approach to expert and model-based effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007, available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443.

[41] J. Pahariya, V. Ravi, and M. Carr, "Software cost estimation using computational intelligence techniques," pp. 849 –854, 2009.

[42] B. Twala, M. Cartwright, and M. Shepperd, "Ensemble of missing data techniques to improve software prediction accuracy," pp. 909–912, 2006. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134449

[43] T. M. Khoshgoftaar, S. Zhong, and V. Joshi, "Enhancing software quality estimation using ensemble-classifier based noise filtering," *Intell. Data Anal.*, vol. 9, pp. 3–27, January 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1239046.1239048

[44] 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 Journal*, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11219-009-9081-z

[45] A. Bakir, "Classification based cost estimation model for embedded software," Master's thesis, Bogazici University, 2008.

[46] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

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

[48] Y. Li, M. Xie, and T. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," *Journal of Systems and Software*, vol. 82, pp. 241–252, 2009.

[49] G. Kadoda, M. Cartwright, and M. Shepperd, "On configuring a case-based reasoning software project prediction system," *UK CBR Workshop, Cambridge, UK*, pp. 1–10, 2000.

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

[51] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

Appendix A
Data Used in This Study

With the exception of ISBSG-Banking and SDR, all the data used in this study is available at http://promisedata.org/data or from the authors. As shown in Figure 3, our data includes:

- Data from the International Software Benchmarking Standards Group (ISBSG);

- The Desharnais and Albrecht data sets;
- SDR, which is data from projects of various software companies from Turkey. SDR is collected from Softlab, the Bogazici University Software Engineering Research Laboratory repository [44];
- And the standard COCOMO data sets (Cocomo*, Nasa*).

Projects in ISBSG dataset can be grouped according to their business domains. In previous studies, breakdown of ISBSG according to business domain has also been used [45]. Among different business domains we selected banking due to:

1. Banking domain includes many projects whose data quality is reported to be high (ISBSG contains projects with missing attribute values).
2. ISBSG Banking domain is the dataset we have analyzed and worked for a long time due to our hands on experience in building effort estimation models in banking industry.

We will denote the banking domain subset of ISBSG as "ISBSG-Banking".

Note that two of these data sets (Nasa93c2, Nasa93c5) come from different development centers around the United States. Another two of these data sets (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.

Note also in Figure Figure 3, the skewness of our effort values (2.0 to 4.4): our datasets are extremely heterogeneous with as much as 40-fold variation. There is also some divergence in the features used to describe our data:

- While our data includes some effort value (measured in terms of months or hours), no other feature is shared by all data sets.
- The Cocomo* and NASA* data sets all use the features defined by Boehm [46]; e.g. analyst capability, required software reliability, memory constraints, and use of software tools.
- The other data sets use a wide variety of features including, number of entities in the data model, number of basic logical transactions, query count and number of distinct business units serviced.

Appendix B
Learners and Pre-Processors

B.1 Pre-Processors

- **None** is the simplest preprocessor- all values are unchanged.
- ***Simple numeric techniques:*** This category lists the preprocessors that entail mere numeric alterations of actual values rather than aplication of certain algorithms.
  - **norm:** *"norm"* represents the application of normalization on the data. The data is normalized to 0-1 interval according to Equation 9:
  - **log:** The natural logarithm of the independent variables are taken. Similar to all pre-processing procedures, the dependent variable is excluded from the pre-processing method.

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

- ***Feature synthesis:***
  "PCA" stands for principal component analysis [47]. PCA is widely used as a dimension alteration mechanism. The alteration in PCA occurs in the form of changing an *n*-dimensional space into another *n*-dimensional space. PCA lets the user know which particular features are most influential in the new *n*-dimensional space, hence user can choose to use only the most influential features. Therefore, it is also common to see the usage of PCA like a dimensionality reduction mechanism. However, in our study we prefer to use PCA only to change the *n*-dimensional space (not the number of features).
- ***Feature selection:*** Some of the pre-processors aims 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.
  - **SFS:** 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 forward direction (no backtracking).
  - **SWReg:** SWReg stands for stepwise regression, which can be defined as a systematic method for adding and removing features from a multilinear model based. 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 is based on the p-value in an F-statsitic. 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. Basically the steps followed by the method are as follows:

∗ Fit an initial model.
∗ If some features have low p-values (lower than a pre-defined threshold, which is chosen to be 0.05 in our study), i.e. if it is likely that they would have non-zero coefficients when added to the model, then add the one with the lowest p-value, then repeat this procedure. If all terms have higher p-values than the threshold value, then proceed with the next step.
∗ If some of the features have p-values, which are greater than an exit threshold (the exit threshold we used in our study is 0.10 *AND* $max(pValuesOfPreviousStep)$), then remove the feature with the highest p-values. Remember that having a high p-value means that it is likely that we will fail to reject the null hypothesis of having a zero coefficient.
∗ Exit when none of the above steps improves the model.

One caution we need to make here is 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 3 bins, depending on equal width of all bins. The bin-width for a general n-bin procedure is given in Equation 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.

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

  - **width5bin:** Exactly same as *"width3bin"* except that this time we have 5 bins instead of 3.
  - **freq3bin:** *"freq3bin"* means binning each feature into 3 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 is 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:** Exactly same as *"freq3bin"*, only this time we have 5 bins.
- ***Others:*** The option(s) that cannot be categorized into afore mentioned categories are mentioned here.
  - **none:** Application of no pre-processing to the data is also a choice. **"none"** represents the choice of no pre-processor selection.

The term called *solo-method* that we use in our research is a pair of *pre-precessor* and *learner*. Each pairing of a pre-processor with a learner defines a unique *solo-method*. Hence, the application of a *solo-method* on a dataset is a two-fold procedure. In the first phase, the pre-processor is applied on the dataset and the *processed* dataset is then passed to the learner. In the second phase, learner trains and on the training set and generates the estimates for the test set. This section describes the building-blocks of *solo-methods* in greater detail.

B.2 Learners

- ***Instance-based learners:*** When we refer to instance-based learners (either ABE methods or nearest-neighbor based methods), there are various design options to be decided and if not clearly stated those decisions remain vague to the reader. The analogy based estimation method that we call ABE0-*x*NN refers to a very basic type of ABE that we defined through our readings of various ABE studies [10], [48], [49]. In ABE0-*x*NN, features are firstly normalized to 0-1 interval, then the distance between test and train instances is measured according to Euclidean distance function, *x* nearest neighbors are chosen from 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. Since the median of a single value is itself, the estimated value in ABE0-1NN becomes the actual effort value of the closest analogy in the training set.
  - **ABE0-5NN:** Five closest analogies are found and used for adaptation.
- ***Iterative dicotomization:*** Under this category, we use classification and regression trees (*CART*) as described by Breiman et. al. [50] and developed it using MATLAB routines. CART is a non-parametric technique and it can work both for classification and regression type of problems, in our case it is the latter. There are in fact a variety of approaches for designing CART. When planning to construct a CART, there are a couple of points to consider:
  - Selection of splits
  - Decision on when a node is pure enough (when to stop splitting any further, i.e. finding the terminal node)
  - Assigning a class to each terminal node

  Firstly, the selection of splits: There are again a variety of solutions that are used to decide the selection of splits such as misclassification rate, information (or entropy)

or GINI index. CART uses the GINI index to calculate the impurity of a tree [50]. Secondly, 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). Lastly, assigning class to each terminal node: Since our problem in effort estimation is a regression problem, we have no classes but actual numeric values in the terminal nodes. 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 2 types of CART, which are given below:
  - **CART-On:** In this version, the pruning is on, meaning that the training data is used in a cross validation process and for each cross validation run, some internal nodes are chosen as the leaf nodes and their sub-nodes are removed. Finally, the sub-tree that resulted in the lowest error rate is returned. The returned sub-tree is a sub-optimal solution and it is locally optimum.
  - **CART-Off:** The sub-trees of CART will not be considered and the full tree will be used.
- ***One-layered neural net:*** *"Neural Nets"* (from now on NNet) are memoryless structures that can be defined as universal approximators [51], 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 3 layer structure: Input layer, hidden layer and the output layer. Depending on the complexity of the problem, one can model more complex functions with a NNet by increasing the number of hidden layers in the structure. In our implementation based on MATLAB, we used a simiple NNet structure with 1 hidded layer, 1 input layer and 1 output layer.
- ***Regression methods:*** Under this category, we list our regression-based learners.
  - **SLReg:** SLReg stands for simple multiple linear regression. Given the dependent variables, this learner calculates the coefficient estimates of the independent variables.
  - **PCR and PLSR:** 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 they construct the new indepent 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. However, the question of which method is more parsimonious still remains context dependent.
  - **SWReg:** We have previously defined SWReg and mentioned that it was a special regression algorithm. We have also stated how it can be used as a feature selection algorithm. In the algorithms section, we use SWReg as a regression method on the *selected-out* independent variables to explain the dependent variable.