Sampling Methods in Software Effort Estimation: An Investigation on Bias-Variance Trade-Off

Ekrem Kocaguneli, *Student Member, IEEE* Tim Menzies, *Member, IEEE* Martin Shepperd, *Member, IEEE*

Abstract—
Background: Experimental design is an important concept in software effort estimation. Many papers use different experimental selections: leave-one-out-cross-validation (LOOCV) a.k.a. N-Way, 10-Way, 3-Way etc. Also there are different justifications for the adopted strategies: The bias-variance trade-off, associated run times and so on.
Aim: The theory states that the more test sets we have (i.e. as *x* increases in *x*-Way), the variance increases and the bias decreases. However, there is no systematic investigation of this theoretical concept. In this paper, we systematically investigate whether theoretical assumptions hold for software effort datasets.
Method: We selected 20 different effort datasets and 90 different algorithms to compare different experimental settings. For each dataset, we calculated the bias and variance of every algorithm under the experimental settings of LOOCV and 3-Way.
Results: As a result of our investigation on 90 algorithms and 20 effort datasets, we saw that the theory does not hold for effort estimation. We have observed that LOOCV and 3-Way have very similar bias and variance values.
Conclusion: Seeing that LOOCV and 3-Way have almost exactly the same bias and variance values, we can conclude that for software effort estimation the bias-variance trade-off is not the main concern of experimentation. Therefore, the main concern when opting for a particular experimental strategy should be run-times and reproduction of the experiments.

Index Terms—Software Cost Estimation, Experimentation, Bias, Variance

1 Introduction

Sampling method is an important topic for software effort estimation (from now on SEE) studies and an empirical study to compare the pros and cons of different sampling methods in SEE is urgent.

The biggest research topic in SEE since 1980s is the introduction of new methods and comparing them to old ones [11]. In their comprehensive systematic review Jorgensen and Shepperd report 61% of selected SEE papers deal with that topic" [11]. This group of papers use *historical data*, i.e. and not a single one of them employs a data collection methodology.

Only generating theories from historical data entails an internal validity threat, which we would like to call *fixed-scenario-problem*. Ideally a learned theory should be applied to new scenarios to observe if the predicted effect occurs in practice. The lack of new scenarios in evaluation is defined to be the *fixed-scenario-problem* and it threats the evaluation experiments like the ones reported in [11]. Therefore, studies without a new scenario for the learned model are limited within their experimental settings.

On the other hand it is impractical to expect every study to collect new data. The mitigation to *fixed-scenario-problem* is possible by simulating the application of a method to a new situation. Sampling method (from now on SM) forms the basis of such a simulation [2], [8].

There is a wide palette of available SMs used in the literature [2], [7], [24], [32]: Leave-one-out (LOO), 10Way and 3Way are examples to the most commonly used ones. Similar to choosing colors from a palette, the choice of different SMs paints a different picture. For example, theoretically LOO results in high-variance and low-bias in the results, whereas 10Way or 3Way generate just the opposite (low-variance, high-bias) [10], [32]. The change of bias and variance (from now on $B\&V$) from one method to the other is known as $B\&V$ trade-off. Employing the wrong SM or disregard of the $B\&V$ trade-off due to particular SMs endanger the validity of a particular study.

To our surprise, in SEE domain there is no study employing a rigorous experimentation to observe the effects of different SMs. Kitchenham et al. have already identified and raised the issue of SM selection [19], [20]; however, their mentioning is more of a pointer to future work rather than an investigation. Hence, this paper is a natural follow-up of previous SEE research. Furthermore, it is the first of its kind to rigorously investigate the $B\&V$ trade-off inherent in different SMs and it concerns more than half the SEE field. In this paper, we present $B\&V$ trade-off results of 3 different SMs: LOO, 3Way and 10Way cross-validation. Our experimentation includes 90 methods applied on 20 datasets.

The experimental results showed that $B\&V$ behavior of SMs are different than the predicted: For most of the algorithms, bias and variance values are statistically the same. However, we have seen orders of magnitude differences in terms of run times, see Figure 1 for exact values. The values

• *Ekrem Kocaguneli and Tim Menzies are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. E-mail: ekocagun@mix.wvu.edu, tim@menzies.us*
• *Martin Shepperd is with the School of Information Systems Computing & Maths, Brunel University. E-mail: martin.shepperd@brunel.ac.uk*

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

SM	Run Time
LOO	$8199.945 * 5$
3Way	$8199.945 * 3$
10Way	8199.945

Fig. 1. The run times in seconds for different SMs.

of Figure 1 belong to experiments coded in MATLAB and run on a 64-bit dual-core machine. Given these findings, we recommend considering experimental concerns to choose an SM. If the main concern is the exact reproduction of the current work by another researcher, then LOO should be used. Otherwise, if the lower run times are the main concern, then we recommend 3Way or 10Way.

1.1 Contributions

The contributions of this research are summarized below:

- The first systematic investigation of *B&V* trade-off in SEE domain
- An extensive experimentation of 20 datasets and 90 algorithms
- Showing that *B&V* is not the main concern for SEE
- Recommendations based on experimental concerns:
  - For lower run-times the order of preference is: 1) 3Way, 2) 10Way, 3) LOO.
  - For reproducibility prefer LOO

2 Terminology

A typical dataset consists of a a matrix X and a vector Y. The input variables (a.k.a. features) are stored in X, where each row corresponds to an observation and each column corresponds to a particular variable. Similarly, the dependent variable is stored in a vector Y, where for each observation in X there exists a response value.

Now assume that a prediction model represented by $\hat{f}(x)$ has been learned from a training dataset τ. So as to measure the errors between the actual values in Y and the predictions given by $\hat{f}(x)$, we can make use of an error function represented by $L(Y, \hat{f}(x))$. Some examples of error functions are squared loss (given in Equation 1) or absolute loss (given in Equation 2).

$$L(Y, \hat{f}(x)) = \left(Y - \hat{f}(x)\right)^2 \quad (1)$$

$$L(Y, \hat{f}(x)) = |Y - \hat{f}(x)| \quad (2)$$

Given the assumptions that the underlying model is $Y = f(X) + \epsilon$ where $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma_\epsilon^2$, then we can come up with a derivation of the squared-error loss for $\hat{f}(X)$ [1]. The error for a point $X = x_0$ is:

$$\begin{aligned} Error(x_0) &= E\left[\left(Y - \hat{f}(x_0)\right)^2 |X = x_0\right] \\ &= \sigma_\epsilon^2 + \left(E[\hat{f}(x_0) - f(x_0)]\right)^2 \\ &\quad + E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right] \\ &= \sigma_\epsilon^2 + Bias^2(\hat{f}(x_0)) + Var(\hat{f}(x_0)) \\ &= \underbrace{IrreducableError}_{1^{st}Term} + \underbrace{Bias^2}_{2^{nd}Term} \\ &\quad + \underbrace{Variance}_{3^{rd}Term} \end{aligned}$$

In the above derivation, the explanations of the 1^{st}, 2^{nd} and 3^{rd} terms are as follows:

- The $1^{st}Term$ is the so called *"irreducable error"*, i.e. the variance of the actual model around its true mean. This variance is inevitable regardless of how well we model $f(x_0)$, only exception to that is when the actual variance is zero (when $\sigma_\epsilon^2 = 0$).
- The $2^{nd}Term$ is the square of the bias, which is the measure of how different the model estimates are fromt the *true* mean of the underlying model.
- The $3^{rd}Term$ is the variance of the estimated model. It is the expectation of the squared deviation of the estimated model from its own mean.

Furthermore, the above derivation is for an individual instance. The bias and variance values associated with an algorithm $\hat{f}(X)$ is the mean of all individual values.

Then the question becomes how the bias and variance (from now on $B\&V$) relate to different choices of the training size (*K*), i.e. the relation to cross-validation method (CV). Here we will consider two cases of CV: leave-one-out (LOO) and 3-Way. Ideally when training size is equal to the dataset size (*K=N*), we expect CV to be approximately unbiased and to have high variance, because N training sets are so similar to one another. On the other hand, for small values of *K*, say *K=N/3* as in 3-Way, we expect lower variance and a higher bias [1]. Naively put, the relationship is:

- LOO : Higher variance, lower bias
- 3-Way : Lower variance, higher bias

In an ideal case, when we plot $B\&V$ values of each individual test instances on x and y axes respectively, we expect 2 clusters:

- Upper Left: Low bias, high variance; i.e. LOO results.
- Lower right: High bias, low variance; i.e. 3Way results.

Just for the sake of clarity, a very *simple* but *ideal* case would look like Figure 2.

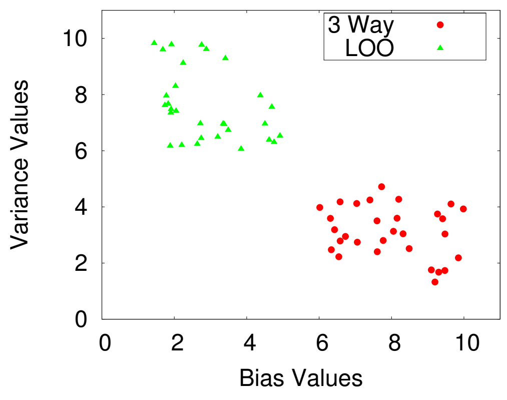


Fig. 2. A simple simulation for the ideal case of $B\&V$ relation to testing strategies.

Dataset	Used by us	Used by others
telecom	[16]	[34]
kemerer	[16]	[9], [34]
cocomo81o	[21], [28], [30]	
desharnaisL1	[21]	
cocomo81s	[21], [28], [30]	
desharnaisL3	[21]	
albrecht	[16],	[9], [26], [27], [34], [35]
cocomo81e	[3], [21], [30]	
nasa93_center_5	[21], [28], [30]	
desharnaisL2	[21]	
desharnais	[15]–[17], [21]	[14], [18], [25]–[27], [34]
maxwell		[27], [33]
sdr		[23], [36]
nasa93_center_1	[21], [28], [30]	
miyazaki94		[31]
nasa93_center_2	[21], [28], [30]	
finnish		[6], [34]
cocomo81	[21], [28], [30]	[4],
nasa93	[21], [28], [30]	
china	this study	

Fig. 3. A sample of effort estimation papers that use the data sets explored in this paper.

Method	Used by
LOO	[15], [16], [25] [17], [21], [27] [22], [34]
3-Way	[22]
10-Way	[3], [22], [28] [36]
Others (ad-hoc, 6-Way etc.)	[6], [23], [26] [30], [33], [35]

Fig. 4. Distribution of the studies in Figure 3 w.r.t. their SM. Majority of the studies use LOO. LOO is followed by ad-hoc methods, 10-Way then 3-Way.

3 Related Work

3.1 Effort Estimation

3.1.1 *Algorithmic Methods*

There are many algorithmic effort estimators. For example, if we restrict ourselves to just instance-based algorithms, Figure **??** shows that there are thousands of options just in that one sub-field.

As to non-instance methods, there are many proposed in the literature including various kinds of regression (simple, partial least square, stepwise, regression trees), and neural networks just to name a few. For notes on these non-instance methods, see §**??**.

Note that instance & non-instance-based methods can be combined to create even more algorithms. For example, once an instance-based method finds its nearest neighbors, those neighbors might be summarized with regression or neural nets [26].

3.1.2 *Non-Algorithmic Methods*

An alternative approach to algorithmic approaches (e.g. the instance-based methods of Figure **??**) is to utilize the best knowledge of an experienced expert. Expert based estimation [12] is a human intensive approach that is most commonly adopted in practice. Estimates are usually produced by domain experts based on their very own personal experience. It is flexible and intuitive in a sense that it can be applied in a variety of circumstances where other estimating techniques do not work (for example when there is a lack of historical data). Furthermore in many cases requirements are simply unavailable at the bidding stage of a project where a rough estimate is required in a very short period of time.

Jorgensen [13] provides guidelines for producing realistic software development effort estimates derived from industrial experience and empirical studies. One important finding concluded was that the *combined estimation* method in expert based estimation offers the most robust and accurate combination method, as combining estimates captures a broader range of information that is relevant to the target problem, for example combining estimates of analogy based with expert based method. Data and knowledge relevance to the project's context and characteristics are more likely to influence the prediction accuracy.

Although widely used in industry, there are still many ad-hoc methods for expert based estimation. Shepperd et al. [35] do not consider expert based estimation an empirical method because the means of deriving an estimate are not explicit and therefore not repeatable, nor easily transferable to other staff. In addition, knowledge relevancy is also a problem, as an expert may not be able to justify estimates for a new application domain. Hence, the rest of this paper does not consider non-algorithmic methods.

3.2 Bias-Variance Trade-Off

Figure 3 shows the studies used the datasets presented here.[1]

When the studies shown in Figure 3 are investigated we see that they use different testing strategies. The below table shows the distribution of these papers w.r.t. the testing strategy they use.

4 Methodology

4.1 Datasets

The description of 20 datasets used in this study are provided in Figure 5.

1. Make another table showing which methods these papers use.

				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. 5. 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 these datasets, see the appendix.

4.2 Methods

4.2.1 Ten Pre-processors

In this study, we investigate:

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

None is the simplest preprocessor- all values are unchanged.

With the **norm** preprocessor, numeric values are normalized to a 0-1 interval using Equation 3. Normalization means that no variable has a greater influence that any other.

$$normalizedValue = \frac{(actualValue - min(allValues))}{(max(allValues) - min(allValues))} \tag{3}$$

With the **log** preprocessor, all numerics are replaced with their logarithm. This **log**ging procedure minimizes the effects of the occasional very large numeric value.

Principal component analysis [2], or **PCA**, is a *feature synthesis* preprocessor that converts a number of possibly correlated variables into a smaller number of uncorrelated variables called components. The first component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

Some of the preprocessors aim at finding a subset of all features according to certain criteria such as **SFS** (sequential forward selection) and **SWR** (stepwise regression). **SFS** adds features into an initially empty set until no improvement is possible with the addition of another feature. Whenever the selected feature set is enlarged, some oracle is called to assess the value of that set of features. In this study, we used the MATLAB, *objective* function (which reports the 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).

SWR adds and removes features from a multilinear model. Addition and removal is controlled by the p-value in an F-Statistic. At each step, the F-statistics for two models (models with/out one feature) 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.

Discretizers are pre-processors that maps every numeric value in a column of data into a small number of discrete values:

- **width3bin:** This procedure clumps the data features into 3 bins, depending on equal width of all bins see Equation 4.

$$binWidth = ceiling\left(\frac{max(allValues) - min(allValues)}{n}\right) \tag{4}$$

- **width5bin:** Same as **width3bin** except we use 5 bins.
- **freq3bin:** Generates 3 bins of equal population size;
- **freq5bin:** Same as **freq3bin**, only this time we have *5* bins.

4.2.2 Nine Learners

Based on our reading of the effort estimation literature, we identified nine commonly used learners that divide into

- 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**.

Instance-based learning can be used for analog-based estimation. A large class of ABE algorithms was described in Figure **??**. Since it is not practical to experiment with the 6000 options defined in Figure **??**, we focus on two standard variants. ABE0 is our name for a very basic type of ABE that we derived from various ABE studies [14], [26], [29]. In **ABE0-xNN**, 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. We explored two different x:

- **ABE0-1NN:** Only the closest analogy is used. Since the median of a single value is itself, the estimated value in **ABE0-1NN** is the actual effort value of the closest analogy.
- **ABE0-5NN:** The 5 closest analogies are used for adaptation.

Iterative Dichotomizers seek the best attribute value $splitter$ that most simplifies the data that fall into the different splits. Each such splitter becomes a root of a tree. Sub-trees are generated by calling iterative dichotomization recursively on each of the splits. The CART iterative dichotomizer [5] is defined for continuous target concepts and its $splitters$ strive to reduce the GINI index of the data that falls into each split. In this study, we use two variants:

- **CART (yes):** This version prunes the generated tree using cross-validation. For each cross-val, an internal nodes is made into a leaf (thus pruning its sub-nodes). The subtree that resulted in the lowest error rate is returned.
- **CART (no):** Uses the full tree (no pruning).

In *Neural Nets*, or **NNet**, an input layer of project details is connected to zero or more "hidden" layers which then connect to an output node (the effort prediction). The connections are weighted. If the signal arriving to a node sums to more than some threshold, the node "fires" and a weight is propagated across the network. Learning in a neural net compares the output value to the expected value, then applies some correction method to improve the edge weights (e.g. back propagation). Our **NNet** uses three layers.

This study also uses four *regression methods*. **LReg** is a simple linear regression algorithm. Given the dependent variables, this learner calculates the coefficient estimates of the independent variables. **SWreg** is the stepwise regression discussed above. Whereas above, **SWreg** was used to select features for other learners, here we use **SWreg** as a learner (that is, the predicted value is a regression result using the features selected by the last step of **SWreg**). Partial Least Squares Regression (**PLSR**) as well as Principal Components Regression (**PCR**) are algorithms that are used to model a dependent variable. 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 independent variables are different. **PCR** generates new independent variables to explain the observed variability in the actual ones. 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**, on the other hand, 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.

4.3 Experiments

5 Results

When we calculated the $B\&V$ values for 90 algorithms (the algorithms in Comba paper) on various datasets, we were unable to observe the behavior of Figure 2, i.e. we did not observe two distinct clusters at predicted $B\&V$ zones. On the contrary, we observed that both $B\&V$ values are close to one another for LOO and 3Way, i.e. the two clusters mostly overlap. Also, the *ideal* or *predicted* lowness and highness for $B\&V$ values were not visible too. The actual $B\&V$ values were both high, regardless of the testing strategy. In Figure **??**, Figure 6, Figure 7 the $B\&V$ plots of 90 algorithms (i.e. 90 circles for 3-Way and 90 triangles for LOO) for Nasa93, Cocomo81 and Desharnais datasets are to be seen. All the values reported in these figures are logged. Also note that the axes in these figures are not scaled, because the differences are so small that scaling the axes makes it difficult to observe the behavior of $B\&V$. See in these figures, how the *ideal* behavior of $B\&V$ differs from the *actual* case for software effort datasets. We have conducted these experiments on many more datasets and the results are pretty much the same: 1) No ideal behavior for 3-Way and LOO; 2) 3-Way and LOO $B\&V$ values overlap.

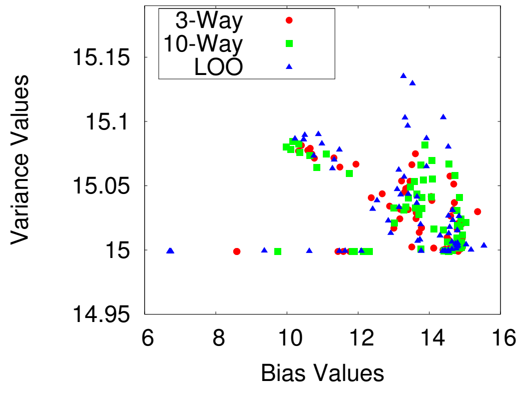


Fig. 6. $B\&V$ values for Cocomo81.

The plot of sorted $B\&V$ values of Figure 8 are given in Figure 10 and Figure **??**.

5.1 Conclusions

No difference between bias and variance.

References

[1] *The Elements of Statistical Learning*. Springer, July 2003.
[2] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2004.

dataset		bias		variance	
		3Way	10Way	3Way	10Way
cocomo81	LOO	43.33	82.22	56.67	80.00
	3Way		21.11		40.00
cocomo81o	LOO	91.11	100.00	75.56	93.33
	3Way		90.00		63.33
cocomo81e	LOO	67.78	88.89	54.44	77.78
	3Way		35.56		18.89
cocomo81s	LOO	62.22	86.67	55.56	74.44
	3Way		32.22		34.44
nasa93	LOO	81.11	90.00	62.22	75.56
	3Way		58.89		60.00
nasa93_center_1	LOO	94.44	94.44	46.67	84.44
	3Way		81.11		46.67
nasa93_center_2	LOO	84.44	95.56	76.67	91.11
	3Way		57.78		42.22
nasa93_center_5	LOO	86.67	96.67	70.00	87.78
	3Way		71.11		41.11
desharnais	LOO	100.00	100.00	91.11	93.33
	3Way		100.00		81.11
desharnaisL1	LOO	100.00	100.00	91.11	92.22
	3Way		97.78		85.56
desharnaisL2	LOO	98.89	100.00	91.11	93.33
	3Way		94.44		68.89
desharnaisL3	LOO	94.44	100.00	60.00	100.00
	3Way		85.56		43.33
sdr	LOO	52.22	64.44	28.89	62.22
	3Way		20.00		16.67
albrecht	LOO	98.89	100.00	78.89	93.33
	3Way		77.78		50.00
finnish	LOO	100.00	100.00	91.11	92.22
	3Way		100.00		84.44
kemerer	LOO	92.22	100.00	77.78	85.56
	3Way		82.22		57.78
maxwell	LOO	94.44	100.00	81.11	88.89
	3Way		82.22		64.44
miyazaki94	LOO	76.67	93.33	52.22	77.78
	3Way		50.00		35.56
telecom	LOO	100.00	100.00	91.11	95.56
	3Way		100.00		70.00

Fig. 8. Percentage of ties. For every highlighted cell, the percentage of ties w.r.t. the dataset size is given. LOO, 3Way and 10Way are represented by the letters a, b and c respectively.

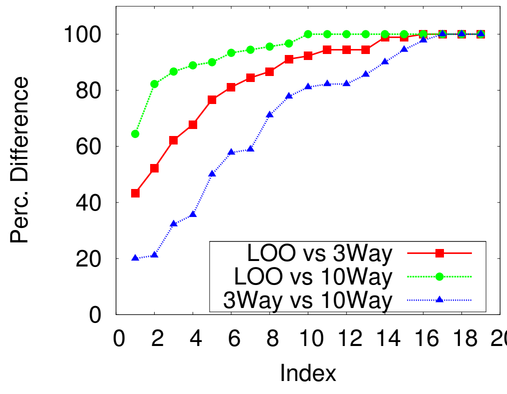


Fig. 9. Sorted bias values of LOO, 3Way and 10Way. Actual values are given in Figure 8.

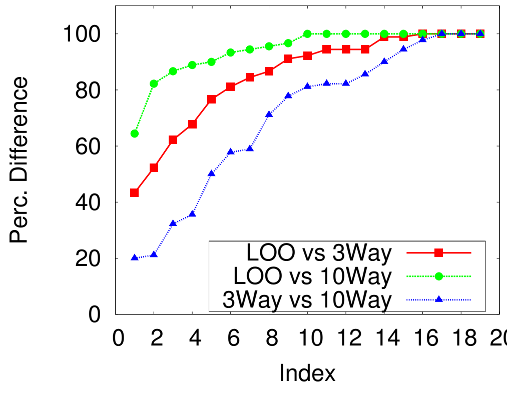


Fig. 10. Sorted bias values of LOO, 3Way and 10Way. Actual values are given in Figure 8.

[3] 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.

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

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

[6] L. C. Briand, K. El Emam, D. Surmann, I. Wieczorek, and K. D. Maxwell. An assessment and comparison of common software cost estimation modeling techniques. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 313–322, New York, NY, USA, 1999. ACM.

[7] J. Demsar. Statistical Comparisons of Clasifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[8] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 2006.

[9] G. R. Finnie, G. E. Wittig, and J.-M. Desharnais. A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*, 39(3):281 – 289, 1997.

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer, 2 edition, 2008.

[11] M. Jø rgensen and M. Shepperd. A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering*, pages 33–53, Jan. 2007.

[12] M. Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1-2):37–60, 2004.

[13] M. Jorgensen. Practical guidelines for expert-judgment-based software effort estimation. *Software, IEEE*, 22(3):57–63, 2005. 0740-7459.

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

[15] J. Keung. Theoretical maximum prediction accuracy for analogy-based software cost estimation. In *Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific*, pages 495 –502, 3-5 2008.

[16] J. Keung and B. Kitchenham. Experiments with analogy-x for software cost estimation. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 229 –238, 26-28 2008.

[17] J. W. Keung, B. A. Kitchenham, and D. R. Jeffery. Analogy-x: Providing statistical inference to analogy-based software cost estimation. *IEEE Trans. Softw. Eng.*, 34(4):471–484, 2008.

[18] C. Kirsopp and M. Shepperd. Making inferences with small numbers of training sets. *Software, IEE Proceedings*, 149, 2002.

[19] 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*, pages 1–5, New York, NY, USA, 2009. ACM.

[20] B. Kitchenham, E. Mendes, and G. H. Travassos. Cross versus Within-Company Cost Estimation Studies: A Systematic Review. *IEEE Trans. Softw. Eng.*, 33(5):316–329, 2007.

[21] E. Kocaguneli, G. Gay, Y. Yang, T. Menzies, and J. Keung. When to use data from other projects for effort estimation. In *ASE '10: To Appear In the Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, New York, NY, USA, 2010.

[22] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung. Exploiting the Essential Assumptions of Analogy-based Effort Estimation. *To Appear in IEEE Trans. Softw. Eng*, 2011.

[23] 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*, pages 330–338, New York, NY, USA, 2008.

[24] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 2008.

[25] J. Li and G. Ruhe. Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+. *Empirical Software Engineering*, 13(1):63–96, 2008.

[26] 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*, 82:241–252, 2009.

[27] Y. Li, M. Xie, and G. T. A study of the non-linear adjustment for analogy based software cost estimation. *Empirical Software Engineering*, pages 603–643, 2009.

[28] K. Lum, T. Menzies, and D. Baker. 2cee, a twenty first century effort estimation methodology. In *International Society of Parametric Analysis Conference (ISPA / SCEA)*, May 2008.

[29] 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*, 8(2):163–196, 2003.

[30] T. Menzies, Z. Chen, J. Hihn, and K. Lum. Selecting best practices for effort estimation. *IEEE Trans. Softw. Eng.*, 32:883–895, 2006.

[31] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki. Robust regression for developing software estimation models. *J. Syst. Softw.*, 27(1):3–16, 1994.

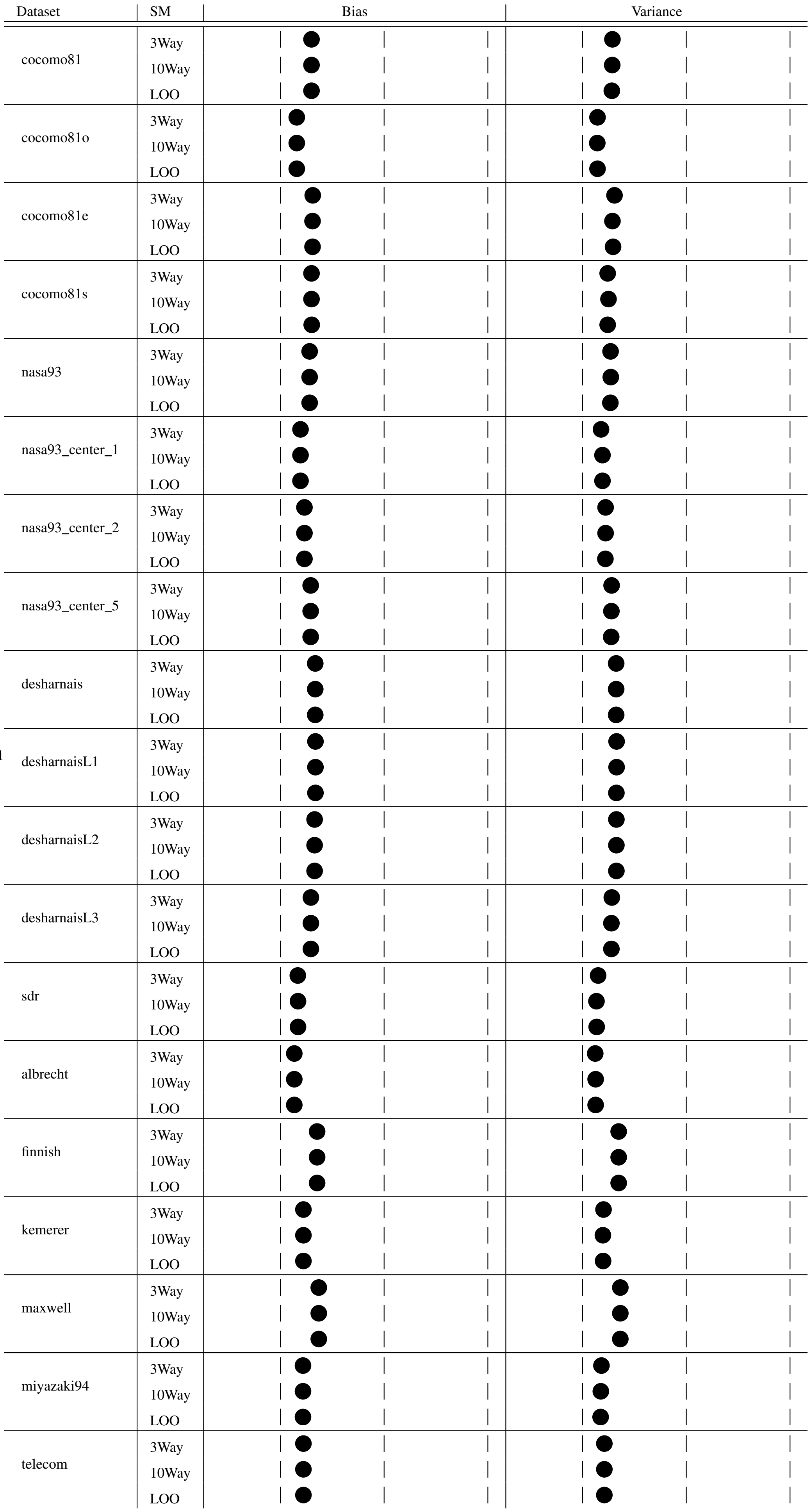
[32] G. Seni and J. Elder. *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*. Morgan and Claypool Publishers, 2010.

[33] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris. Software productivity and effort prediction with ordinal regression. *Information and Software Technology*, 47(1):17 – 29, 2005.

[34] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Trans. Softw. Eng.*, 23(11):736 –743, nov 1997.

[35] M. Shepperd, C. Schofield, and B. Kitchenham. Effort estimation using analogy. In *Proceedings of the 18th International Conference on Software Engineering*, pages 170 –178, 25-29 1996.

[36] B. Turhan, O. Kutlubay, and A. Bener. Evaluation of feature extraction methods on software cost estimation. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 497 –497, 20-21 2007.



1

Fig. 7. $B\&V$ values in quartiles.