The Journal of Systems and Software 86 (2013) 1879–1890

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Software effort models should be assessed via leave-one-out validation

Ekrem Kocaguneli*, Tim Menzies

CSEE, West Virginia University, Morgantown, USA

ARTICLE INFO

Article history:
Received 9 April 2012
Received in revised form 16 February 2013
Accepted 18 February 2013
Available online 15 March 2013

Keywords:
Software cost estimation
Prediction system
Bias
Variance

ABSTRACT

Context: More than half the literature on software effort estimation (SEE) focuses on model comparisons. Each of those requires a *sampling method* (SM) to generate the train and test sets. Different authors use different SMs such as leave-one-out (LOO), 3Way and 10Way cross-validation. While LOO is a deterministic algorithm, the N-way methods use random selection to build their train and test sets. This introduces the problem of *conclusion instability* where different authors rank effort estimators in different ways.
Objective: To reduce conclusion instability by removing the effects of a sampling method's random test case generation.
Method: Calculate bias and variance (*B*&*V*) values following the assumption that a learner trained on the whole dataset is taken as the true model; then demonstrate that the *B*&*V* and runtime values for LOO are similar to N-way by running 90 different algorithms on 20 different SEE datasets. For each algorithm, collect runtimes, *B*&*V* values under LOO, 3Way and 10Way.
Results: We observed that: (1) the majority of the algorithms have statistically indistinguishable *B*&*V* values under different SMs and (2) different SMs have similar run times.
Conclusion: In terms of their generated *B*&*V* values and runtimes, there is no reason to prefer N-way over LOO. In terms of reproducibility, LOO removes one cause of conclusion instability (the random selection of train and test sets). Therefore, we depreciate N-way and endorse LOO validation for assessing effort models.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

The largest research topic in software effort estimation (hereafter, SEE) is the introduction and evaluation of new, and empirical comparison of, prediction methods. In a comprehensive systematic review Jørgensen and Shepperd report more than 60% of the reviewed SEE papers deal with this topic (Jørgensen and Shepperd, 2007).

Assessing new prediction methods is complicated by small sample sizes of the training data. Valerdi (2011) and Hihn (Menzies et al., 2006) offer the rule of thumb that there should be five to ten rows of training data per attribute. Most effort estimation data sets are smaller than that: for example, five recent effort estimation publications (Mendes et al., 2003; Auer et al., 2006; Baker, 2007; Kocaguneli et al., 2012; Li et al., 2009) use data sets with dozens of attributes but only a handful of rows (median values of the number of rows are 13, 15, 31, 33, 52, respectively).

As a result, prediction models tend to be over fitted to the particulars of the training data used in particular studies. This leads to the problem of *conclusion instability*; i.e. different studies make different conclusions regarding what is the "best" effort estimator. Myrtveit et al. (2005) and Shepperd and Kadoda (2001) studied a large number of synthetic data sets (generated from distributions found in a real-world data set). They found that, as they changed the conditions of their experiments, no method was consistently best across every condition. Specifically, the performance of a method depends on:

1. The dataset;
2. The evaluation method used to assess model accuracy;
3. The generation method used to build training and test sets.

Elsewhere, we have addressed points #1 and #2 (Keung et al., 2012; Kocaguneli et al., 2011): If R_i is the rank of method M_j within a set of M methods, then we use δr to denote the maximum rank change of that method as we alter the evaluation method and the data set. Our own experiments confirmed Shepperd's previous work; i.e. that $\delta r \neq 0$. Hence, we cannot say for certain that a method ranked at R_i is always better than another ranked at R_{i+1}. However, we have found that if we analyzed enough methods using enough

* Corresponding author.
E-mail addresses: kocaguneli@gmail.com (E. Kocaguneli), tim@menzies.us (T. Menzies).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.02.053

Method	Used by
LOO	[7, 17–23]
Others (ad-hoc, 6-Way etc.)	[3, 8, 24–27]
10-Way	[7, 28–30]
3-Way	[7, 31, 32]

Fig. 1. SEE papers that use different SEE methods.

evaluation methods, then the ranking variability is much smaller than the number of methods; i.e. for large M, $\delta r \ll |M|$. That is, given 90 methods, we can say that (a) the top 30 are better than the bottom 30; (b) the value of the methods ranked 31–59 is unknown; so (c) we should focus on those methods in the top third (Keung et al., 2012). Other experiments have confirmed the superiority of those top ranked methods (Kocaguneli et al., 2011).

This paper focuses on the third point listed above; i.e. the *generation method used to build training and test sets*. SEE research uses historical data to estimate future performance. The induced predictor is tested on data that is not used in generating the predictor. In practice, this means using some sampling method (SM) to divide historical data into:

- Training data that the prediction system can learn from
- Unseen or test validation data that is used to assess predictive accuracy.

SEE validation studies adopt different SMs such as leave-one-out (LOO), 3Way and 10Way (Demsar, 2006; Alpaydin, 2004; Lessmann et al., 2008; Seni and Elder, 2010) (the details of these SMs are given in Section 2.3). As shown in Fig. 1, there is no consensus in the SEE literature as to which SM should be used to evaluate new predictors. Note that, in that figure, many researchers use some variant of N-way. We argue that the use of N-way contributes significantly to the conclusion instability problem. The randomization step of N-way makes the results virtually unrepeatable since a sequence of random numbers are usually very different when generated by different algorithms implemented in different languages running different toolkits on different platforms. This means that an N-way analysis incurs the problem of point #3, discussed above.

On the other hand, a LOO analysis is deterministic and repeatable since, given access to the same data, it is possible to generate identical train and test sets. However, there are two problems with LOO: the *high variance* and *long runtimes*:

- *High variance:* In theory, as discussed below, the results of a LOO analysis will have a higher variance and lower bias compared to an N-way study. This introduces a complication into the analysis; e.g. using LOO it will be harder to distinguish the performance of different methods since the performance of those methods will exhibit a wider variability.
- *Long runtimes:* A 3-way analysis of 1000 examples will require the construction of three effort models. On the other hand, a LOO analysis of the sample examples will require the construction of 1000 effort models. If the effort model is slow to generate (e.g. some genetic algorithm exploring all subsets of possible attributes Li et al., 2009) then 1000 repeats is impractically slow.

We show in this paper that neither of these problems are necessarily an issue:

- For 20 data sets from the PROMISE repository, we show that the bias and variance (B&V) of results generated by LOO is statistically indistinguishable from 10-way and 3-way. That is, while in theory LOO and N-way studies generate different results, in practice, those differences are insignificant.
- LOO conducts many repeated calculations over the same data sets. If those calculations are cached and re-used later in the LOO, then the runtimes for LOO become close to those of N-way.

A likely objection to these results would be: Why should one of the three SMs, in particular LOO, be preferred over the others if they are indistinguishable in terms of bias, variance and run times? That is a very valid objection. If the experimental conditions require researchers to use randomization in their use of SMs (e.g. N-way), then they should do so. On the other hand, if the experimental conditions allow researchers to pick up any SM; then, unlike N-Way SMs, LOO would let another research team to exactly replicate a prior work. Furthermore, the results of LOO would allow us to see per instance estimation performance. In other words, if one research team shares their estimates for each and every instance of a data set, the other research team can not only compare aggregate performance measures (such as MMRE, MdMRE and Pred(25)), but also can compare how good their estimate is for each instance. This type of a replication process would help immensely in addressing the conclusion instability. A recent special issue in Empirical Software Engineering Journal targeted the issue of conclusion instability (Menzies and Shepperd, 2012). All the papers submitted to that special issue unanimously accepted that conclusion instability is a pressing issue and needs to be handled. In this research, we target one aspect of the instability problem and provide empirical evidence as to why LOO should be preferred over N-Way SMs.

The rest of this paper is structured as follows. First, we present some background notes on effort estimation and the definitions of bias and variance (note that B&V will be used as an acronym for "bias and variance" from now on). Second, we describe an experiment comparing bias and variance. Third, we explore the runtimes associated with LOO vs. N-way. Our conclusion will be that there is no reason from the effort estimation community to suffer with N-way studies:

- The deterministic nature of LOO studies makes them more repeatable.
- They are not necessarily slower than N-way.
- Nor do they generate different biases and variances.

2. Background

2.1. Effort estimation

Effort estimation is the activity of predicting the amount of effort required to complete a software development project (Keung, 2008). Estimation activities are carried out through:

- algorithmic methods
- non-algorithmic methods

Algorithmic methods learn a model from historical data and pass new projects through that model to generate their estimates. The number of proposed algorithmic methods and associated variants easily exceed tens of thousands. Fig. 3 of Keung et al. (2012) shows that for analogy-based effort estimation (which is just one branch of algorithmic methods), likely combinations are more than 6000. Some other examples to algorithmic methods are: various kinds of regression (simple, partial least square, stepwise, regression trees), neural networks and instance-based algorithms, just to name a few. In Appendix A we provide the algorithmic methods used in this study.

Non-algorithmic methods utilize the best knowledge of experienced human experts. Such non-algorithmic methods, a.k.a. expert-based estimation, is defined to be a human intensive approach that is most commonly adopted in practice (Jørgensen, 2004). In expert-based variants, estimates are produced by domain experts based on their very own personal experience. On one hand, these methods are flexible and intuitive as they can be applied in a variety of circumstances where other estimating techniques do not work. For example, when there is no historical data or the requirements of a project are unavailable at the initial stages, a rough estimate in a very short period of time can be provided by expert estimates. On the other hand – regardless of the efforts to establish guidelines for expert-based methods (Jørgensen, 2004) – there are still many *ad hoc* methods used in practice. Shepperd et al. (1996) do not consider expert based estimation as an empirical method, since 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. Lastly, from an experimental point of view SMs do not make sense for expert estimates, because expert estimates are based on the expert's personal experience rather than different divisions of train/test sets. Hence, the rest of this paper excludes non-algorithmic methods from the discussion of bias and variance.

2.2. Defining bias and variance

A typical SEE dataset consists of 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.

The variable we try to predict (effort value) is stored in vector Y, and the independent variables that define software projects are stored in the columns of the matrix X (i.e. each column of X is a vector corresponding to the values of an independent variable). Let us assume that an element of the vector Y, $y_0 \in \mathbb{R}$, is the dependent variable value corresponding to an instance vector (row) of X, $x_0 \in \mathbb{R}^n$, where n is the number of independent features. Also let us assume that these two are related to one another in the following manner (Hastie et al., 2008):

$$y_0 = f(x_0) + \epsilon \tag{1}$$

where;

- $f(x_0)$ is the true model that we cannot know, but try to model using an estimation method, e.g. linear regression;
- and ϵ is assumed to be a normally distributed error term with zero mean and a variance of $Var(\epsilon) = \sigma_\epsilon^2$, i.e. $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$

Although we cannot know the true model $f(x_0)$, it is possible to model it using an estimation method like linear regression trained on the training data (which is generated by an SM). The estimated model is represented with $\hat{f}(x_0)$ and it is learned using the training data. Note that the ϵ parameter is used as the noise term. In other words, if we had infinite data to learn the true model, we would perfectly model the true model, i.e. $f(x_0) = \hat{f}(x_0)$; in this case, there would be no bias or variance. However, there would still be noise associated with the actual data itself, which is represented with ϵ and there would be a so-called "irreducible error" due to noise, which would be the $Var(\epsilon) = \sigma_\epsilon^2$. So as to measure the error between an actual value, y_0, and the corresponding prediction given by $\hat{f}(x_0)$, we can make use of an error function represented by $L(y_0, \hat{f}(x_0))$. Some examples of error functions are squared loss and absolute loss. Following the previous literature (Hastie et al., 2008; Seni and Elder, 2010; Molinaro et al., 2005), we will make use of the squared error loss function, which is provided in Eq. (2).

$$L(y_0, \hat{f}(x_0)) = (y_0 - \hat{f}(x_0))^2 \tag{2}$$

If we model $f(x_0)$ using $\hat{f}(x_0)$ that is trained on the training data, the error is defined as:

$$\begin{aligned} Error(x_0) &= E\left[\left(y_0 - \hat{f}(x_0)\right)^2 | x_0\right] \\ &= \sigma_\epsilon^2 + \left(E[\hat{f}(x_0) - f(x_0)]\right)^2 + E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right] \\ &= \sigma_\epsilon^2 + Bias^2(\hat{f}(x_0)) + Var(\hat{f}(x_0)) \end{aligned}$$

where,

- The 1*stTerm*, σ_ϵ^2, is the irreducible error;
- The 2*ndTerm*, $Bias^2(\hat{f}(x_0))$, is the square of bias;
- The 3*rdTerm*, $Var(\hat{f}(x_0))$, is the variance.

The explanations of these terms are as follows:

- The 1*stTerm* is the so-called *"irreducible error"*, i.e. the variance of the actual model around its true mean. Since we cannot possibly know the variation of the actual data, recall that we used the $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ assumption. This variance is inevitable regardless of how well we model $f(x_0)$.
- The 2*ndTerm* is the square of the bias. Bias is defined to be the measure of how different the model estimates are from the *true* mean of the underlying model, i.e. $(E[\hat{f}(x_0) - f(x_0)])$. Hence, the second term of the derivation turns out to be the square of the bias.
- The 3*rdTerm* is the variance of the estimated model. It is the expectation of the squared deviation of the estimated model from its own mean.

Seni and Elder warns about the calculation of bias that it *cannot* be computed but can be used as a helpful theoretical concept (see p. 23 of Seni and Elder, 2010). The biggest handicap towards calculation of bias comes from the fact that we can never know the true model (Seni and Elder, 2010; Molinaro et al., 2005), unless it was designed as a mathematical model in the first place. Then we cannot derive the bias (see the true model, $f(x_0)$, in the derivation). That is a critical problem that needs to be handled, if we are to make empirical investigations on the *B*&*V* trade-off in SEE. Without concrete definitions of "bias"; hence the "true model", *B*&*V* discussions regarding sampling methods in SEE will be nothing more than expert opinions.

To handle that problem, we need to make assumptions regarding the true model. A successful application of such an assumption is provided by Molinaro et al.: A learner trained on the whole dataset is taken as the true model (Molinaro et al., 2005). This approach is quite useful as it replaces the *unknown* true model with a *known*, mathematically definable model; thereby, enabling the bias derivation. In our experimentation we used the option provided by Molinaro et al. (2005).

2.3. In theory, SMs affect results

We can group the SMs used in SEE literature into two main groups: Leave-one-out (LOO) and N-Way. The difference between these SMs is as follows:

LOO:

Take one instance at a time as the test set
Build the learner on the remaining $N-1$ instances (training set)
Use the model to estimate for the test set.

N-Way:
Randomize order of rows in data
Divide dataset into β bins of size close or equal to N/β, where N is the number of instances in a data set and β is the number of bins, e.g. in 3Way, $\beta = 3$
Use each subset as the test set and the remaining subsets as the training set
Repeat this procedure multiple times: Hall and Holmes (2003) recommend ten repeats of a 10-way study.

An important question associated with SMs is how *B&V* relate to different choices of the training size (K). To answer this question, we make two observations. Given a data set D of fixed size, and test and training data sets $D = train \cup test$, then:

- The training set grows progressively smaller from LOO to 10way to 3way.
- The test set grows progressively larger from LOO to 10way to 3way.

The first observation effects the bias and the second observation effects the variance. To see that, recall that any induction algorithms seeks for a target concept in some training data. As a training set gets smaller, it becomes less likely to contain examples that describe the target. Hence, the induction algorithm will "miss" the target and the resulting model will be biased (its predictions will deviate away from true model values, i.e. $f(x_0)$). That is, in theory, bias will increase from LOO to 10way to 3way. In the seminal work of Kitchenham et al., the effects of SMs on *B&V* is also discussed and they foresee the importance of experimental investigation of *B&V* values on SEE data sets, which is followed in our research.

On the other hand, as the training set shrinks, the test set grows. To understand the effect of test set size N on the variance, recall that variance is the difference between each prediction and the mean of all the predictions (μ); i.e.

$$Var(\hat{f}(x_0)) = E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right] = \frac{\sum_i^N (X_i - \mu)}{N}.$$

Note that $\lim_{N\to 0} Var(\hat{f}(x_0)) = \infty$; i.e. smaller tests sets can have larger variance. Hence, in theory, variance will increase from 3way to 10way to LOO.

In summary, according to the above discussion, we would expect:

- LOO: High variance, low bias (see upper left of Fig. 2)
- 3Way: Low variance, high bias (see lower right of Fig. 2)
- 10Way: Values between LOO and 3Way (see center of Fig. 2)

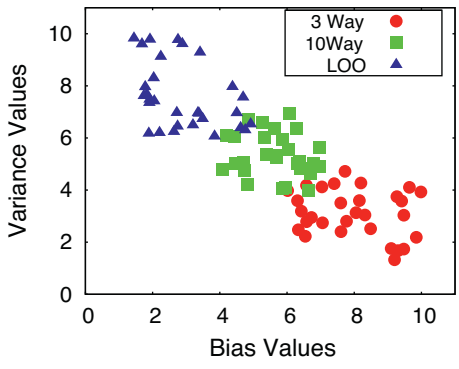

Fig. 2. A simple simulation for the *"expected"* case of *B&V* relation to testing strategies.

The results of this paper can be expressed with respect to Fig. 2: the empirical results (reported below) cannot distinguish between the *B&V* of LOO, 3Way, and 10Way.

3. Experiment1: comparing bias and variance

This section describes an experiment to compare *B&V* for LOO and N-way.

3.1. Algorithms: pre-processors and learners

This study uses *10 different pre-processors* × *9 learners = 90 algorithms*. The selection is based on two criteria:

- Learners and pre-processors must come from SEE literature; (e.g. Lum et al., 2008; Mendes et al., 2003; Jørgensen and Shepperd, 2007; Shepperd et al., 1996; Kultur et al., 2008; Shepperd and Schofield, 1997; Chang, 1974; Venkatachalam, 1993; Boehm et al., 2000).
- Learners must make different assumptions about the data.

This second criteria is based on data-mining theory that different learners are built on different assumptions, hence they have different biases (Kittler et al., 1998; Alpaydin, 1998; Dietterich, 2000; Ghosh, 2002).

We hence used 10 pre-processors:

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

and 9 learners:

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

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

- *Normalization* of data to zero-one interval;
- A *Euclidean* distance measure;
- Estimates generated using the *k nearest neighbors*.

For detailed descriptions of all these learners, see Appendix A.

3.2. Experiments

3.2.1. Generate true model f(x)

Each algorithm is trained on each entire dataset and the estimates are stored as the values of $f(x)$. The values of $f(x)$ will be used for *B&V* calculations.

3.2.2. Get estimates

Let A_i ($i \in \{1, 2, \ldots, 90\}$) be one of the 90 algorithms and let D_j ($j \in \{1, 2, \ldots, 20\}$) be one of the 20 datasets. Also let SM_k ($k \in \{1, 2, 3\}$) be one of the 3 SMs. In this step every A_i is run on every D_j subject to every SM_k. In other words every $A_i \times D_j \times SM_k$ combination is exhausted, and related predictions are stored to be used for *B&V* calculations.

3.2.3. Calculate B&V values

The $f(x)$ values and predictions coming from $A_i \times D_j \times SM_k$ runs are used to calculate the B&V. At the end of this step, we have separate B&V values for every $A_i \times D_j \times SM_k$. Another interpretation is that for every algorithm-dataset combination ($A_i \times D_j$) we have 3 values of B&V (1 for each SM_i).

3.2.4. Statistical check on B&V values

In this step we check if the B&V values for every $A_i \times D_j$ combination are statistically different from one another (checks are based on Mann–Whitney at 95% confidence interval). This way we can see if the run of an algorithm on a single dataset subject to different SMs generate significantly different B&V values. Since we have 3 different SMs, for every $A_i \times D_j$ there are 3 different tuples to look at: LOO vs. 3Way; LOO vs. 10Way; 3Way vs. 10Way. For each tuple we ask Mann–Whitney if the B&V values coming from $A_i \times D_j$ are different under the SM's of that tuple. We note down whether they are not statistically different (i.e. they "tie") or not. After processing all the SM tuples for all $A_i \times D_j$, we can see what percent of the 90 algorithms generated statistically same B&V values for different SM tuples and for different datasets. The pseudo-code for this process is given in Fig. 3.

3.3. Datasets

There is at least one study in SEE using one or more of the 20 datasets used in our study (see Fig. 4). Therefore, the results presented here are based on a large corpus and concern a number of previously published SEE studies. The description of 20 datasets used in this study are provided in Fig. 5. These datasets are available at https://code.google.com/p/promisedata/.

In terms of geography, our datasets are very diverse. Three of these data sets (nasa93_center_1, nasa93_center_2, nasa93_center_5) come from different development centers around the United States. Other data sets come from around the world:

- The desharnais dataset includes Canadian software projects,
- cocomo81 and nasa93 include projects developed in the United States,
- sdr, contains projects of various software companies in Turkey (Bakir et al., 2009).

Some other data sets (cocomo81e, cocomo81o, cocomo81s) represent different kinds of projects (embedded, organic and semi-detached respectively) developed by different team sizes and under different constraints (Boehm, 1981). Sdr contains data from recent projects of various software companies in Turkey. Sdr is collected by Softlab, the Bogazici University Software Engineering Research Laboratory (Bakir et al., 2009). The albrecht data set consists of projects completed in IBM in the 1970s (Albrecht and Gaffney, 1983). The finnish data set contains 40 projects from different companies and was collected by a single person in the 90s (Kitchenham and Känsälä, 1993). The two projects with missing values are omitted here; hence we use 38 instances. Kemerer is a rather small dataset with 15 instances, whose details can be found in Kemerer (1987). Maxwell is another relatively new data set (projects from late 90s early 00s) that comes from finance domain and is composed of Finnish banking software projects. Details are given in Maxwell (2002). Miyazaki contains projects developed in COBOL (Miyazaki et al., 1994). Telecom contains projects which are enhancements to a U.K. telecommunication product (Shepperd and Schofield, 1997).

Note also in Fig. 5, the skewness of the effort values (up to 6.06): The datasets are extremely heterogeneous with as much as 60-fold variation. There is also some divergence in the features used to describe the datasets:

Dataset	Used by
telecom	[19, 23]
kemerer	[19, 23, 52]
cocomo81o	[3, 21, 29]
desharnaisL1	[21]
cocomo81s	[3, 21, 29]
desharnaisL3	[21]
albrecht	[8, 19, 22, 23, 27, 52]
cocomo81e	[3, 21, 28]
nasa93_center_5	[3, 21, 29]
desharnaisL2	[21]
desharnais	[8, 17–23, 53, 54]
maxwell	[22, 26]
sdr	[25, 30]
nasa93_center_1	[3, 21, 29]
miyazaki94	[50]
nasa93_center_2	[3, 21, 29]
finnish	[23, 24]
cocomo81	[3, 21, 29, 45]
nasa93	[3, 21, 29]
china	this study

Fig. 4. A sample of effort estimation papers that use the data sets explored in this paper.

```
for D_j, where j ∈ 1...20 do
  for A_i, where i ∈ 1...90 do
    for Tupple, where Tupple ∈ {LOOvs3Way; LOOvs10Way; 3Wayvs10Way} do
      if Mann-Whitney(Bias values of A_i for D_j from SM's in Tupple, 95%) says the same
      then
        Mark method as "tied" for bias values
      else
        Mark method as "not-tied" for bias values
      end if
      if Mann-Whitney(Var. values of A_i for D_j from SM's in Tupple, 95%) says the same
      then
        Mark method as "tied" for variance values
      else
        Mark method as "not-tied" for variance values
      end if
    end for
  end for
end for
```

Fig. 3. Comparing B&V values coming from different $A_i \times D_j$ combinations under different SM tuples. This comparison helps us see what percentage of 90 methods "tie" w.r.t. B&V values.

Dataset	Features	Size	Description	Historical Effort Data				
				Units	Min	Median	Max	Skewness
cocomo81	17	63	NASA projects	months	6	98	11400	4.4
cocomo81e	17	28	Embedded projects	months	9	354	11400	3.4
cocomo81o	17	24	Organic projects	months	6	46	240	1.7
cocomo81s	17	11	Semi-detached projects	months	5.9	156	6400	2.64
nasa93	17	93	NASA projects	months	8	252	8211	4.2
nasa93_center_1	17	12	Projects from center 1	months	24	66	360	0.86
nasa93_center_2	17	37	Projects from center 2	months	8	82	1350	2.4
nasa93_center_5	17	40	Projects from center 5	months	72	571	8211	3.4
desharnais	12	81	Canadian software projects	hours	546	3647	23940	2.0
desharnaisL1	11	46	Projects developed in Language1	hours	805	4035.5	23940	2.09
desharnaisL2	11	25	Projects developed in Language2	hours	1155	3472	14973	1.16
desharnaisL3	11	10	Projects developed in Language3	hours	546	1123.5	5880	1.86
sdr	22	24	Turkish software projects	months	2	12	342	3.9
albrecht	7	24	Projects from IBM	months	1	12	105	2.2
finnish	8	38	Projects developed in Finland	hours	460	5430	26670	0.95
kemerer	7	15	Large business applications	months	23.2	130.3	1107.3	2.76
maxwell	27	62	Projects from banks in Finland	hours	583	5189.5	63694	3.26
miyazaki94	8	48	Projects developed in COBOL	months	5.6	38.1	1586	6.06
telecom	3	18	Telecom maintenance projects	months	23.54	222.53	1115.5	1.78
china	18	499	Chinese software projects	hours	26	1829	54620	3.92
		Total: 1198						

Fig. 5. The 1198 projects used in this study come from 20 data sets. Indentation in column one denotes that indented dataset is a subset of another one.

- While data sets have some effort values in common (measured in terms of man-months or man-hours), no other feature is shared by all data sets.
- The COCOMO and NASA data sets (i.e. cocomo81, cocomo81e, cocomo81o, cocomo81s, nasa93, nasa93_center_1, nasa93_center_2, nasa93_center_5) use the features defined by Boehm (1981); 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.

Also note that SEE data sets come from specific contexts and organizations, which are influential on the characteristics of the data sets. As can be seen in Fig. 5, the data sets used in this study come from a variety of different contexts and organizations operating in different countries. One aspect of the data sets coming from different contexts is that we are able to observe the behavior of *B&V* related to different SM's on very different data sets. Another aspect is that it is difficult to make a statement about the quality of each data set. Although these data sets have been used extensively in SEE community, investigation of data quality issues is an important future direction to this study. Furthermore, replication of this study on proprietary data sets may provide further evidence.

Meta-analysis methods like the investigation of the effect size as a complementary step on the statistical tests is a promising direction. For example, the literature review of Kampenes et al. (2007) present a good framework on the application of meta-analysis techniques and shows how to identify the effect sizes in software engineering studies. Meta analysis of the results coming from SEE data sets may be beneficial for practitioner audiences, since – as also pointed out by Kampenes et al. (2007) – statistically significant differences in comparison of two populations may have a very limited effect. Such a meta-analysis deserves a study on its own right and may be followed as a future work to this research.

3.4. Results

After calculating the *B&V* values for 90 algorithms on all the datasets, we were unable to observe the behavior of Fig. 2, i.e. we did not observe three clusters at predicted *B&V* zones. On the contrary, we observed that *B&V* values associated with different SMs were very close to one another.

For example, see in Fig. 6 the mean *B&V* values of 90 algorithms for china data set. Note that different SMs are represented with different symbols and for every SM there are 90 symbol occurrences corresponding to 90 algorithms. The *B&V* values associated with each SM overlap, instead of forming separate clusters. Also, the *expected* relative low and high *B&V* values of SMs (see Fig. 2 for expected low and high) were not visible too. Unlike the expected behavior, the actual *B&V* values were both high, regardless of the utilized SM.

We have conducted these experiments on all the datasets and generated Fig. 6 for every dataset. However, the results are the same:

1. The *expected* behavior was not found LOO, 3-Way and 10Way;
2. The different SMs did not form distinct clusters (as witnessed by the overlapping *B&V* values of LOO, 3-Way and 10Way in Fig. 6).

There is insufficient space to repeat Fig. 6 for every dataset. Hence, we summarized these *B&V* values in terms of quartile charts of Fig. 7. Fig. 7 shows every dataset in a separate row, which is then divided into 3 sub-rows. Sub-rows correspond to 3 different SMs and they show the related *B&V* quartile charts separately. In every quartile chart, the median (represented with a dot), 25*th* quartile

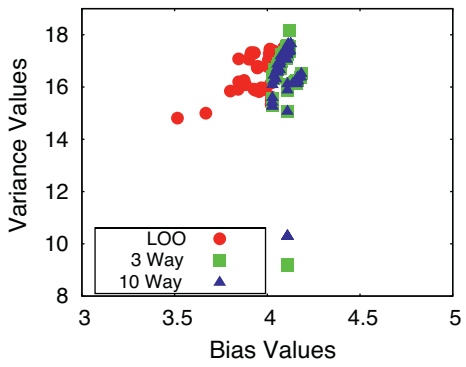

Fig. 6. *B&V* values for china data set (shown values are the natural logarithm of the actual values).

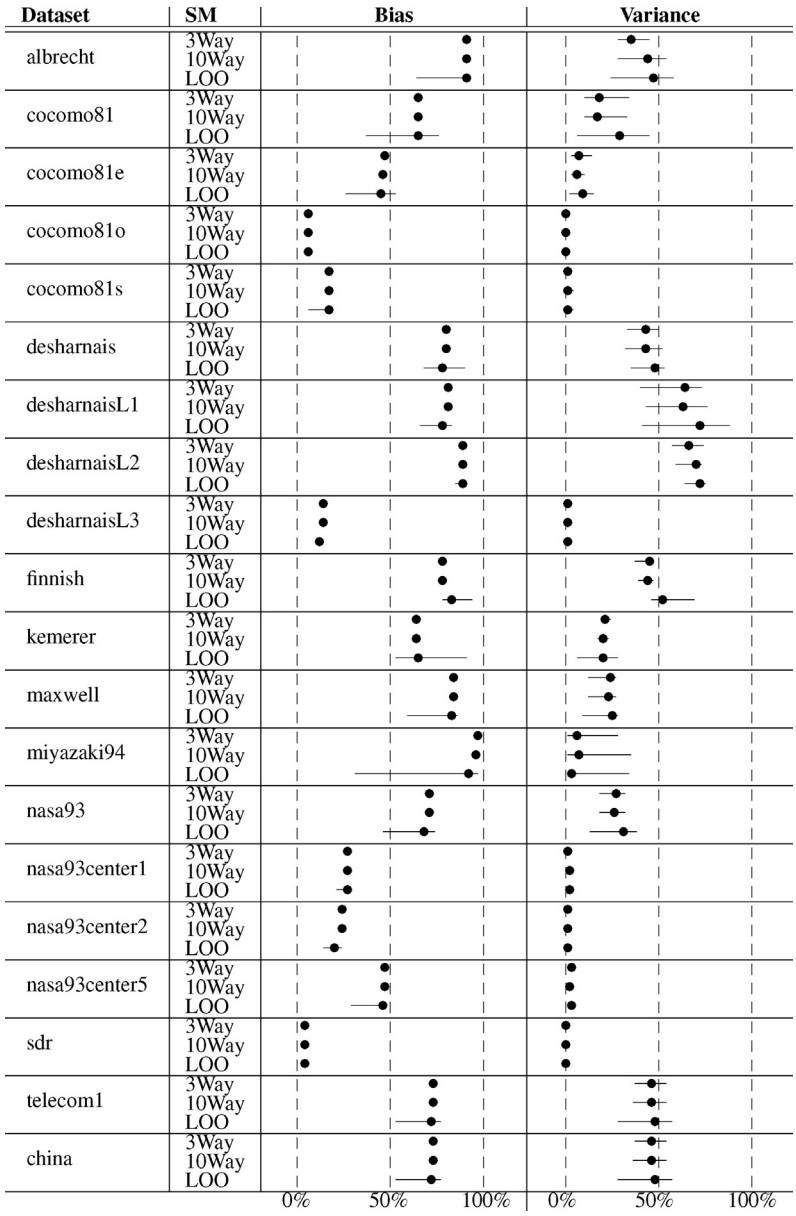

Fig. 7. *B&V* values in quartiles for all datasets. Black dots denote median values. Horizontal lines denote the inter-quartile range (25–75 percentile band). In many results, the inter-quartile range is so small that it disappears behind the median dots. For the purposes of display, all the bias values were normalized min to max (of each dataset row), 0–100 (ditto with variance). The key observation from this result is that within each group of SMs, the *B&V* are very similar.

(the left horizontal line-end) and the 75*th* quartile (the right horizontal line-end) are shown. Note that all of Fig. 6 appears as the last 3 rows of Fig. 7.

Fig. 8 shows what percent of 90 algorithms "tied" w.r.t. to Mann–Whitney; i.e. difference in their *B&V* values were statistically indifferent. See Fig. 3 for the pseudo-code of the comparison of *B&V* values w.r.t. Mann–Whitney. Every cell of Fig. 8 reports the percentage of methods (out of 90) that "tied" for particular SM tuples (LOO vs. 3Way, LOO vs. 10Way, 3Way vs. 10Way) under different datasets.

The distributions of Fig. 8 are summarized in Fig. 9: note the high number of ties. That is, measured in the number of ties as witnessed by Mann–Whitney, these results were the same more often than not.

In order to better explore the deltas between our treatments, we applied a 1-way ANOVA analysis. 1-way ANOVA test takes a vector of output values (bias and variance one at a time) and a factor, which in our case is the sampling method. The *p*-value yielded by this 1-way ANOVA tests the null hypothesis that all samples are drawn from populations with the same mean. If a *p*-value is near zero, then this casts doubt on the null hypothesis, i.e. at least one of the sample means is significantly different than the others. The *p*-value for bias values is: 0.107, which is a border value for a significance level of 99%. Similarly, *p*-value for the variance values is: 0.348 for a significance level of 99%.

Due to: (1) the border *p*-value of ANOVA and (2) the fact that ANOVA assumes sample distributions are Gaussian, which is hardly the case for the observed *B&V* values. To see the non-Gaussian behavior of *B&V* values, refer to figures provided in the following link: http://goo.gl/TLNbg, where *B&V* values are plotted in the form of 10-bin equal-width histograms for each SM and dataset. We also performed a Friedman test (which is a rank-based non-parametric test) followed by a multiple comparison test. Friedman's test is appropriate when the assumptions of a parametric test do not hold

dataset		bias		variance	
		3Way	10Way	3Way	10Way
cocomo81	LOO	53	61	61	61
	3Way	-	90	-	61
cocomo81o	LOO	74	81	70	73
	3Way	-	91	-	50
cocomo81e	LOO	62	61	48	50
	3Way	-	86	-	53
cocomo81s	LOO	59	58	48	48
	3Way	-	40	-	28
nasa93	LOO	47	46	60	64
	3Way	-	90	-	71
nasa93_center_1	LOO	72	74	40	43
	3Way	-	92	-	50
nasa93_center_2	LOO	56	54	49	58
	3Way	-	72	-	48
nasa93_center_5	LOO	58	61	66	61
	3Way	-	94	-	69
desharnais	LOO	81	81	80	78
	3Way	-	100	-	92
desharnaisL1	LOO	79	79	79	79
	3Way	-	100	-	83
desharnaisL2	LOO	89	90	77	82
	3Way	-	99	-	79
desharnaisL3	LOO	79	80	60	71
	3Way	-	94	-	47
sdr	LOO	39	40	39	37
	3Way	-	2	-	22
albrecht	LOO	80	82	73	80
	3Way	-	80	-	69
finnish	LOO	83	83	83	83
	3Way	-	100	-	83
kemerer	LOO	70	64	56	53
	3Way	-	100	-	77
maxwell	LOO	67	73	87	88
	3Way	-	92	-	70
miyazaki94	LOO	38	46	30	38
	3Way	-	64	-	48
telecom	LOO	84	82	78	72
	3Way	-	100	-	76
china	LOO	40	46	53	52
	3Way	-	80	-	58

Fig. 8. Percentage of algorithms for which *B&V* values coming from different SMs are the same (according to Mann–Whitney at 95). Note the very high percentage values, meaning that for the majority of the algorithms different SMs generate values that are not statistically different.

		Percentile		
	Comparison	25th	50th (median)	75th
Bias	LOO vs 3Way	53	67	79
	LOO vs 10Way	58	69	81
	3way vs 10Way	80	91	99
Variance	LOO vs 3Way	48	60	77
	LOO vs10way	48	61	77
	3Way vs 10Way	50	64	76

Fig. 9. Percentiles of number of ties from Fig. 8.

and when we are interested in the effects of treatments (represented by columns) under study (Hollander and Wolfe, 1999). In our study columns represent the *B&V* values (separately) coming from 3 different SM's. The Friedman test compares the means of multiple groups to test the hypothesis that *"they are all the same"*, vs. they are not all the same. When this differentiation is too general, i.e. when we want to see further information regarding which pairs of means are different and which are not, we follow the Friedman test with a rank-based multiple comparison procedure (Hochberg and Tamhane, 1987). The confidence level used for the multiple comparison test is 99%. Based on the selected confidence interval, multiple comparison test calculates the span of confidence interval in terms of ranks around the mean rank value of each SM. The expectation is that for two sample means to be statistically different, their span of confidence interval around the mean rank should form disjoint sets. For actual *B&V* values see the csv file at http://goo.gl/ZGMBk. In this link you will also find a "readme" file explaining the contents. The code to generate Fig. 10 is given in this link too. For further implementation details regarding *friedman* and *multcompare* functions of MATLAB, refer to Hochberg and Tamhane (1987) and Hollander and Wolfe (1999) as well as related Mathworks tutorials.

Fig. 10 shows the results of the multiple comparison test. The *x*-axis of this figure shows the average ranks (according to bias and variance values, separately) corresponding to different sampling methods. The *y*-axis shows the ID's of the sampling methods (1 is for LOO, 2 is for 3Way and 3 is for 10Way). The mean ranks of each sampling method is represented with a symbol and an interval around the symbol. The interval – so-called comparison interval – shows the span of the confidence interval for each SM (the selected confidence interval here is 99%). Two means are statistically different from one another if their comparison intervals are disjoint. As can be seen in Fig. 10, none of the SM's has a disjoint comparison interval, i.e. none of the SM's is significantly different.

4. Experiment2: reducing the run-times for LOO

The total execution time of the experimentation is associated with a particular implementation method, i.e. different implementations of the same algorithm will have different run times. Therefore, we used standard MATLAB functions in this study: All methods except ABE0-1NN and ABE0-5NN, and all pre-processors except discretizers are found in MATLAB libraries.

The run times are also expected to be greatly affected by particular SMs. Each SM dictates a different number of times a learner is trained. The training-time of a learner is much greater than the testing-time since, once a learner is trained, the prediction for a particular test instance can be very quick. Below are the number of training times required for each SM on a *single* dataset:

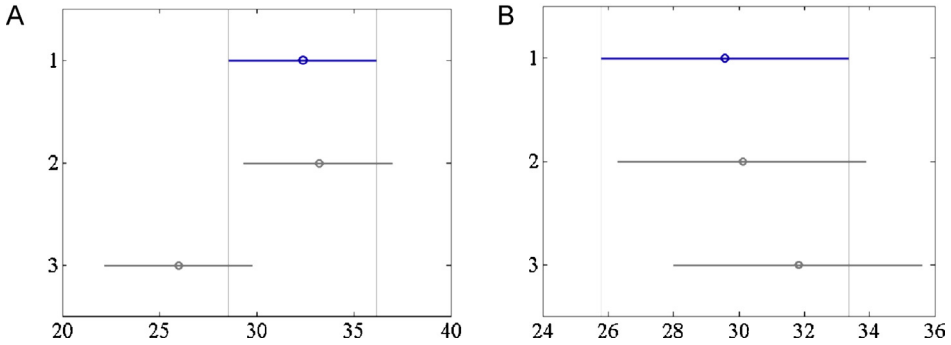

Fig. 10. Friedman test followed by multiple-comparison test. The *x*-axis shows the average ranks, whereas the *y*-axis shows the ID's of the SM's: 1 is for LOO, 2 is for 3Way and 3 is for 10Way. Two means are statistically different from one another if their comparison intervals are disjoint. None of the SM's has a disjoint comparison interval.

SM	Run time
LOO	9, 960
3Way	9, 360
10Way	10, 140

Fig. 11. The run times in seconds. The expected order of SMs from fastest to lowest (3Way, LOO, 10Way) holds, however the difference is in the order of minutes.

- **LOO:** *N trains* where *N* is the dataset size.
- **3Way:** 10repeats × 3bins = 30trains
- **10Way:** 10repeats × 10bins = 100trains

For 20 datasets in this study (a total of 1198 instances), we expect the following training times:

- **LOO:** 1198 trains.
- **3Way:** 30 trains/dataset ×20 datasets = 600 trains
- **10Way:** 100 trains/dataset ×20 datasets = 2000 trains

From above number of training times, we expect 3Way to be the fastest SM, followed by LOO and 10Way.

In Fig. 11 we see run times for 20 datasets. Our expectation that 3Way would be the fastest SM followed by LOO, then 10Way holds. However, the difference is much smaller than expected. Although there are orders of magnitude differences between SMs in terms of train-times, the run time difference is limited to a couple of minutes. Therefore, run time is not a critical factor in the choice of SMs.

One word of caution is that ABE0-*k*NN variants are the slowest methods in our experiments and if such slow methods are to be employed with LOO, then they need to be implemented carefully. Initially we coded ***ABE0 − 1NN*** and ***ABE0 − 5NN*** without regard to optimization: i.e. for every test instance, its distance to all the training instances were calculated from scratch. This way of brute-force implementation skyrockets the run times associated with LOO. The solution is to calculate the distance matrix of a dataset only once and cache that for future uses in LOO. The run time of the cached implementation for LOO – as given in Fig. 11 – is 9,960 s, whereas the brute-force implementation is 25,920 s. So the caching strategy decreases the run time of LOO by orders of magnitude.

5. Discussion

The research presented in this study is an empirical investigation of the SMs that are widely employed in SEE studies. An experimental investigation of the SMs with regard to their trade-off between multiple factors such as *B*&*V*, run-times and ease of replication is an imminent issue for SEE. For example, a recent special issue in Empirical Software Engineering Journal discusses the problem of conclusion instability in software engineering (Menzies and Shepperd, 2012). The issues related to data sampling and how various SMs are used to sample data are listed as one of the likely culprits of the conclusion instability problem. Another critical note to make is that the manuscripts submitted by industry and academia researchers unanimously mention that the instability of the proposed results in software engineering is a fundamental problem to tackle.

In this paper, we investigated different aspects of SMs on SEE data sets such as *B*&*V*, run-times and ease of replication. A practitioner investigating the SEE literature is likely to find a dauntingly large space of studies, conducted on different data sets and employing different SMs and sometimes with contradictory recommendations. In other words, SEE also suffers from the instability problem observed in software engineering (Menzies and Shepperd, 2012). We believe investigating the reasons behind the instability problem is required, particularly when the practitioner audiences are concerned. For example, for companies willing to optimize their SEE methodologies, experimentation turns out to be a critical factor. Currently, to the best of our knowledge, there is no experimental evaluation of the effects of different SMs on SEE data sets. This study serves providing this experimental evaluation.

Prior to experimental evaluation, our intuition was that LOO suffers from over fitting the training data. Because, LOO uses the largest training set in comparison to other SMs, which makes it susceptible to the noise within the training data. As a result, we expected to see the reflection of the over fitting on the error rate, hence on the *B*&*V* values (recall that *B*&*V* are associated with the squared error).

Although, we did not observe the expected behavior of different SMs on a large corpus of public data sets, the practitioners should be cautious about the importance of locality in software engineering data sets (Menzies et al., 2011; Bettenburg et al., 2012). Locality, i.e. that the properties of locally related projects differ from that of the global space is also an important property of SEE data sets. Therefore, the observed behavior of *B*&*V* values may change for the local proprietary space of a particular organization. Particularly the fact that LOO may result in over fitting for certain data sets should be kept in mind. A practitioner making use of the information provided in this manuscript may think that her/his local domain changes considerably, in comparison to the organizations from which the data sets of Fig. 5 are collected. In such a case, she/he should use the proposed experimentation in this study so as to re-evaluate the trade-offs between sampling methods.

6. Conclusions

To the best of our knowledge, in the field of SEE, this is the first empirical investigation on *B*&*V* and runtime trade-off inherent in different SMs.

Our experimentation investigates a large space of 90 algorithms and 20 datasets. The results present the surprising finding that *B*&*V* values in SEE domain behave quite different than the expected:

- Measured in terms of *B*&*V*, different SMs are not statistically different.
- Similarity of SMs also persists in terms of the run times. See in Fig. 11 that the biggest run time difference is between 3Way and 10Way, which is 780 s (13 min) or only a 7% difference in runtimes between the methods. However, note that some coding techniques(e.g. caching distance results in ***ABE0 − kNN*** variants) can significantly lower LOO run times.

Thus we see the contributions of this paper as:

- 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* trade-off and run times of SMs are not the main concerns for SEE
- Recommendation based on experimental concerns:
  - For reproducibility, we prefer LOO since this avoids non-deterministic selection of train and test sets.

Of course our results are not devoid of *validity threats*. To be able to calculate the bias values an assumption regarding the true model is required to be made (since we cannot know the true model). Following Molinaro et al. we used the assumption that a learner trained on the whole dataset may be taken as the true model (Molinaro et al., 2005). Another threat is the implementation of the algorithms. Although we used standard functions from MATLAB libraries, there is still considerable amount of code into which standard functions were embedded. Therefore, run-times will be

different in other implementations. However, since all SMs are run on the same code-base, the relative position of SMs in terms of run-times would remain the same. Another validity threat concerning the run-times is the particular machine on which the experiments are run. Similar to implementation, different machines will yield different run-times, but the relative position of SMs will remain the same.

The choice of SMs in this work depends on the literature using one or more of datasets used here. On the other hand, different choices of SMs may also have an effect on the results. While we acknowledge this issue, we note that this study, based on 90 algorithms and 20 datasets is far more extensive than the majority of the SEE studies.

We finish this study with recommendations based on our empirical findings:

- Different SMs only introduce a negligible *B*&*V* difference,
- SMs have similar run times,
- The main factor to consider when opting for an SM is the reproduction of experiments.

3Way and 10Way (and any other*N*-Way method) use stochastic selection for train and test sets. This can result in conclusion instability when one researcher tries to reproduce the results of another. Hence, we recommend LOO over 3Way and 10Way for SEE studies.

The clear future direction of this work is to explain the curious observation that bias and variance are very similar in different SM methods. It is as though some factor *other* than the selected SM is a forcing function generating a ceiling effect where all the *B*&*V* values are very high. Our suspicion is that our data sets are so small, and their internal variability is so high, that the topology of their separation is a factor more important than the SM. Recently, we have had much success in mapping and exploiting that topology (Kocaguneli et al., 2011).

Appendix A. Prediction methods

This study uses 90 algorithms, which are product of 10 pre-processors combined with 9 learners. The details of the learners as well as the pre-processors are provided below.

A.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 Eq. 3. Normalization means that no variable has a greater influence than any other.

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

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

Principal component analysis (Alpaydin, 2004), 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 multi-linear 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.

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 Eq. (4).

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

- **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.

A.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 analogy-based estimation (ABE). Since it is not practical to experiment with the all ABE variants, we focus on two standard variants. ABE0 is our name for a very basic type of ABE that we derived from various ABE studies (Mendes et al., 2003; Li et al., 2009; Kadoda et al., 2000). In **ABE0-*k*NN**, features are firstly normalized to 0–1 interval, then the distance between test and train instances is measured according to Euclidean distance function, k nearest neighbors are chosen from the training set and finally for finding estimated value (a.k.a adaptation procedure) the median of k nearest neighbors is calculated. We explored two different k:

- **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 (Breiman et al., 1984) 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-validation, an internal node is made into a leaf (thus pruning its sub-nodes). The sub-tree 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. As a pre-processor **SWreg** is 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 in MATLAB implementation, 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.

References

Albrecht, A., Gaffney, J., 1983. Software function, source lines of code and development effort prediction: a software science validation. IEEE Transactions on Software Engineeruing 9, 639–648.

Alpaydin, E., 1998. Techniques for combining multiple learners. In: Proceedings of Engineering of Intelligent Systems.

Alpaydin, E., 2004. Introduction to Machine Learning. MIT Press.

Auer, M., Trendowicz, A., Graser, B., Haunschmid, E., Biffl, S., 2006. Optimal project feature weights in analogy-based cost estimation: improvement and limitations. IEEE Transactions on Software Engineering 32, 83–92.

Baker, D., 2007. A Hybrid Approach to Expert and Model-based Effort Estimation. Master's Thesis. Lane Department of Computer Science and Electrical Engineering, West Virginia University.

Bakir, A., Kocaguneli, E., Tosun, A., Bener, A., Turhan, B., 2009. Xiruxe: an intelligent fault tracking tool. In: AIPR'09: International Conference on Artificial Intelligence and Pattern Recognition, Orlando.

Bettenburg, N., Nagappan, M., Hassan, A.E., 2012. Think locally, act globally: improving defect and effort prediction models. In: MSR, pp. 60–69.

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

Boehm, B., Abts, C., Chulani, S., 2000. Software development cost estimation approaches—a survey. Annals of Software Engineering 10, 177–205.

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

Chang, C.L., 1974. Finding prototypes for nearest neighbor classifiers. IEEE Transactions on Computers, 1179–1185.

Demsar, J., 2006. Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7, 1–30.

Dietterich, T.G.,2000. Ensemble methods in machine learning. In: MCS'00: Proceedings of the First International Workshop on Multiple Classifier Systems. Springer-Verlag, London, UK, pp. 1–15.

Ghosh, J.,2002. Multiclassifier systems: back to the future. In: MCS'02: Proceedings of the Third International Workshop on Multiple Classifier Systems. Springer-Verlag, London, UK, pp. 1–15.

Hall, M.A., Holmes, G., 2003. Benchmarking attribute selection techniques for discrete class data mining. IEEE Transactions on Knowledge and Data Engineering 15, 1437–1447.

Hastie, T., Tibshirani, R., Friedman, J., 2008. The Elements of Statistical Learning, 2nd ed. Springer.

Hochberg, Y., Tamhane, A.C., 1987. Multiple Comparison Procedures. John Wiley and Sons, Inc., Hoboken, NJ.

Hollander, M., Wolfe, D.A., 1999. Nonparametric Statistical Methods. John Wiley and Sons, Inc., Hoboken, NJ.

Jørgensen, M., 2004. A review of studies on expert estimation of software development effort. Journal of Systems and Software 70, 37–60.

Jørgensen, M., Shepperd, M., 2007. A systematic review of software development cost estimation studies. IEEE Transactions on Software Engineering 33, 33–53.

Kadoda, G., Cartwright, M., Shepperd, M., 2000. On configuring a case-based reasoning software project prediction system. In: UK CBR Workshop, Cambridge, UK, Citeseer, pp. 1–10.

Kampenes, V.B., Dybå, T., Hannay, J.E., Sjøberg, D.I.K., 2007. A systematic review of effect size in software engineering experiments. Information & Software Technology 49, 1073–1086.

Kemerer, C.F., 1987. An empirical validation of software cost estimation models. Communications of the ACM 30, 416–429.

Keung, J., 2008. Theoretical maximum prediction accuracy for analogy-based software cost estimation. In: APSEC'08: 15th Asia-Pacific Software Engineering Conference, pp. 495–502.

Keung, J., Kocaguneli, E., Menzies, T., 2012. Finding conclusion stability for selecting the best effort predictor in software effort estimation. Automated Software Engineering, 1–25, http://dx.doi.org/10.1007/s10515-012-0108-5.

Kitchenham, B., Känsälä, K., 1993. Inter-item correlations among function points. In: ICSE'93: Proceedings of the 15th international Conference on Software Engineering, pp. 477–480.

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

Kocaguneli, E., Menzies, T., Keung, J., 2011. On the value of ensemble effort estimation. IEEE Transactions on Software Engineering, 1.

Kocaguneli, E., Menzies, T., Bener, A., Keung, J.W., 2012. Exploiting the essential assumptions of analogy-based effort estimation. IEEE Transactions on Software Engineering 38, 425–438.

Kultur, Y., Turhan, B., Bener, A.B.,2008. 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. ACM, New York, NY, USA, pp. 330–338.

Lessmann, S., Baesens, B., Mues, C., Pietsch, S., 2008. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Transactions on Software Engineering 34, 485–496.

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

Lum, K., Menzies, T., Baker, D., 2008. 2CEE, a twenty first century effort estimation methodology. In: ISPA/SCEA'08: Joint Conference of International Society of Parametric Analysts and Society of Cost Estimating and Analysis, pp. 12–14.

Maxwell, K.D., 2002. Applied Statistics for Software Managers. Prentice Hall, PTR.

Mendes, E., Watson, I., Triggs, C., Mosley, N., Counsell, S., 2003. A comparative study of cost estimation models for web hypermedia applications. Empirical Software Engineering 8, 163–196.

Menzies, T., Shepperd, M., 2012. Special issue on repeatable results in software engineering prediction. Empirical Software Engineering 17, 1–17.

Menzies, T., Chen, Z., Hihn, J., Lum, K., 2006. Selecting best practices for effort estimation. IEEE Transactions on Software Engineering 32, 883–895.

Menzies, T., Butcher, A., Marcus, A., Zimmermann, T., Cok, D.R., 2011. Local vs. global models for effort estimation and defect prediction. In: ASE, pp. 343–351.

Miyazaki, Y., Terakado, M., Ozaki, K., Nozaki, H., 1994. Robust regression for developing software estimation models. Journal of Systems and Software 27, 3–16.

Molinaro, A.M., Simon, R., Pfeiffer, R.M., 2005. Prediction error estimation: a comparison of resampling methods. Bioinformatics (Oxford, England) 21, 3301–3307.

Myrtveit, I., Stensrud, E., Shepperd, M., 2005. Reliability and validity in comparative studies of software prediction models. IEEE Transactions on Software Engineering 31, 380–391.

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

Shepperd, M., Kadoda, G.F., 2001. Comparing software prediction techniques using simulation. IEEE Transactions on Software Engineering 27, 1014–1022.

Shepperd, M., Schofield, C., 1997. Estimating Software Project Effort Using Analogies. IEEE Transactions on Software Engineering 23, 736–743.

Shepperd, M., Schofield, C., Kitchenham, B.,1996. Effort estimation using analogy. In: ICSE'96: Proceedings of the 18th International Conference on Software Engineering. IEEE Computer Society, Washington, DC, USA, pp. 170–178.

Valerdi, R., 2011. Heuristics for systems engineering cost estimation. IEEE Systems Journal 5, 91–98.

Venkatachalam, A.R., 1993. Software cost estimation using artificial neural networks. In: Proceedings of International Joint Conference on Neural Networks, pp. 987–990.

Ekrem Kocaguneli received his PhD degree in Computer Science from West Virginia University. He also holds an MSc and BSc degrees in Computer Engineering from Bogazici University. His main research interests are prediction problems concerning software, artificial intelligence and machine learning applications in empirical software engineering and intelligent tools to aid software processes.

Dr. Tim Menzies is a Prof. in CSEE at WVU and the author of over 200 referred publications. At WVU, he has been a lead researcher on projects for NSF, NIJ, DoD, NASA's Office of Safety and Mission Assurance, as well as SBIRs and STTRs with private companies. Tim is the co-founder of the PROMISE conference series devoted to reproducible experiments in SE. In 2012, served as the co-chair of the program committee for the IEEE ASE conference.