Software Effort Estimation and Conclusion Stability

Tim Menzies, *Member, IEEE,* Omid Jalali, Jairus Hihn, Dan Baker, and Karen Lum

***Abstract*— This paper revisits the *conclusion instability* problem identified by Kitchenham, Foss, Myrtveit et.al.; i.e. conclusions regarding which software effort estimation method is "best" is highly contingent on (1) the evaluation criteria and (2) the subset of the data used in the evaluation. Using non-parametric methods (the Mann-Whitney U test), we show how to avoid conclusion instability. This paper reports a study that ranked 158 effort estimation methods via three different evaluation criteria and hundreds of different randomly selected subsets. The same four methods were ranked higher than the other 154 methods *regardless of which evaluation criteria or data subset was applied*. Hence, we recommend non-parametric evaluation to evaluate *and* prune effort estimation methods. More specifically, when learning effort estimators from COCOMO-style data, we find that manual stratification defeats many complex algorithmic methods. However, we can do better than manual stratification by augmenting Boehm's local calibration method with simple linear-time row and column pruning pre-processors. We also advise *against* model trees, linear regression, exponential time feature subset selection, and (unless the data is sparse) methods that average the estimates of nearest neighbors. To the best of our knowledge, this report is the first to offer stable conclusions regarding effort estimation across such a wide range of methods.**

***Index Terms*— COCOMO, effort estimation, data mining, evaluation, Mann-Whitney U test, non-parametric tests.**

I. INTRODUCTION

Software effort estimates are often wrong. Initial estimates may be incorrect by a factor of four [1] or even more [2]. As a result, the allocated funds may be inadequate to develop the required project. In the worst case, over-running projects are canceled, wasting the entire development effort. For example, in 2003, NASA canceled the CLCS system after spending hundreds of millions of dollars on software development. The project was canceled after the initial estimate of $206 million was increased to between $488 million and $533 million [3]. On cancellation, approximately 400 developers lost their jobs [3].

While the need for better estimates is clear, there exists a very large number of effort estimation methods [4], [5] and no good criteria for selecting between them. Few studies empirically compare all these techniques. What is more usual are narrowly focused studies (e.g. [2], [6], [7], [8]) that test, say, linear regression models in different environments.

Kitchenham et.al. [9], Foss et.al. [10] and Myrtveit et.al. [11] (hereafter, KFM) have doubted the practicality of comparatively assessing L different learners processing D data sets. The results of such a comparison, they argue, vary according to the sub-sample of the data being processed and the applied evaluation criteria. Foss et.al. comment that it

> *. . . is futile to search for the Holy Grail: a single, simple-to-use, universal goodness-of-fit kind of metric, which can be applied with ease to compare (different methods). [10, p993]*

Tim Menzies, Omid Jalali, and Dan Baker are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, USA: `tim@menzies.us`, `ojalali@mix.wvu.edu`, `danielryanbaker@gmail.com`.

Jairus Hihn and Karen Lum at with NASA's Jet Propulsion Laboratory: `jhihn@mail3.jpl.nasa.gov`, `karen.t.lum@jpl.nasa.gov`.

The research described in this paper was carried out at West Virginia University and the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the US National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not constitute or imply its endorsement by the US Government.

Download: `http://menzies.us/pdf/07stability.pdf`.

Manuscript received July 31, 2007; revised XXX, XXXX.

Methodologically, KFM's *conclusion instability* is highly problematic. Unless we can *rank* methods and *prune* inferior methods, we will soon be overwhelmed by a growing number of (possibly useless) effort estimation methods. New open source data mining toolkits are appearing with increasing frequency such as the R project[1], Orange[2], and the WEKA [12]. Such tools tempt researchers to over-elaborate their effort estimation tools. For example, our own COSEEKMO tool [13] takes nearly a day to run its 158 methods. Much of that execution is wasted since, as shown below, 154 of those methods are superfluous.

The rest of this paper presents the ranking and pruning results that culled 154 COSEEKMO methods. Rather than seeking *the* best method, we will seek a *small set* of methods that perform better than the rest. COSEEKMO contains such a *best* set of four methods. Further, in a result that is a counter-example to the KFM studies, the same set of four methods is *best* in studies using three different evaluation criteria and hundreds of different randomly selected subsets.

We explain our differences from the KFM study as follows. The root cause of conclusion instability is a very small number of estimates with very large errors. If these *outliers* fall into some of the subsets, then those subsets will have dramatically different performance results; i.e. will exhibit conclusion instability. *Non-parametric statistics* such as the U test proposed by Mann and Whitney [14] mitigate the outlier problem. The U test uses *ranks*, not precise numeric values. For example, if treatment A generates $N_1 = 5$ values {5,7,2,0,4} and treatment B generates $N_2 = 6$ values {4,8,2,3,6,7}, then these sort as follows:

Samples	A	A	B	B	A	B	A	B	A	B	B
Values	0	2	2	3	4	4	5	6	7	7	8

On ranking, averages are used when values are the same:

Samples	A	A	B	B	A	B	A	B	A	B	B
Values	0	2	2	3	4	4	5	6	7	7	8
Ranks	1	2.5	2.5	4	5.5	5.5	7	8	9.5	9.5	11

[1]`http://www.r-project.org/`
[2]`http://www.ailab.si/orange/`

Note that, when ranked in this manner, the largest value (8 in this case) gets the same rank even if it was ten to a hundred times larger. That is, such rank tests are less susceptible to large outliers. Hence, we can make stable conclusions regarding the *best* COSEEKMO methods.

The rest of this paper is structured as follows. First, we demonstrate the conclusion instability problem, which we explain in terms of large outliers. Next, we justify the use of the U test and present an example of its use. This will be followed by a description of the data and methods used in our study.

The conclusion from our study will be that non-parametric assessment of effort estimation models does not completely resolve the conclusion instability problem. As predicted by KFM, we do not find a single "best" estimation method that works for all data sets. However, it significantly reduces the conclusion instability problem to the point where it is possible to categorically reject many estimation methods. For example, our study finds four methods that are always better than 154 others, regardless of (a) which data subset was used or (b) what evaluation criteria was applied.

To the best of our knowledge, this paper is the first report of stable conclusions in effort estimation.

II. The Conclusion Instability Problem

A. Symptoms of Instability

KFM caution that, historically, ranking estimation methods has been done quite poorly. Based on an analysis of two (non-COCOMO) data sets as well as simulations over artificially generated data set, Foss et.al. and Myrtveit et.al. concluded that numerous commonly used methods such as the mean MRE[3] are unreliable measures of estimation effectiveness. Also, the conclusions reached from these standard measures can vary wildly depending on which subset of the data is being used for testing [11].

Figure 1 demonstrates conclusion instability. It shows two experimental runs. In each run, 30 times, effort estimate models were built for our 19 subsets using two methods. Each time, an effort model was built from a randomly selected 90% of the data. Results are expressed in terms of the difference in mean MRE between the two subsets; e.g. in Run #1, method1 had a much larger mean MRE than method2.

After Run #1, the results endorse method2 since that method either (a) did better (lower errors) as method1 or (b) had similar performance to method1. However, that conclusion is not stable. Observe in Run #2 that:

- The improvements of method2 over method1 disappeared in subsets 1,2,3,7, and 11.
- Worse, in subsets 1,2, and 11 method1 performed dramatically better than method2.

The deviations seen in 30 repeats of the above procedure were quite large: within each data set, the standard deviation on the MREs were $\{median, max\} = \{150\%, 649\%\}$ [13]. Port (personal communication) has proposed a bootstrapping method to determine the true performance distributions of

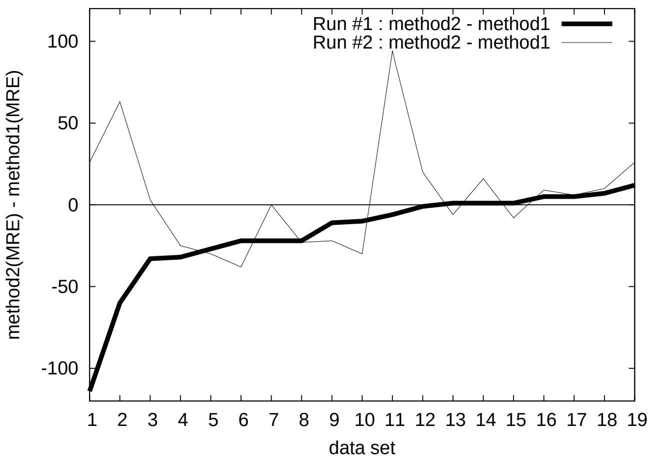

Fig. 1. Results of 2 different runs of COSEEKMO comparing two methods using mean MRE values. Points on the Y-axis show the difference in mean relative error (MRE) between method1 and method2. Lower values endorse method2 since, when such values occur, method2 has a lower error than method1.

COSEEKMO's methods. That method would require 10^2 to 10^3 re-samples and, given COSEEKMO's current runtimes, it would take 10^2 to 10^3 days to terminate.

One troubling result from the Figure 1 study is that the number of training examples was *not* connected to the size of standard deviation. A pre-experimental intuition was that the smaller the training set, the worse the prediction instability. On the contrary, we found small and large instability (i.e. MRE standard deviation) for both small and large training sets [13]. That is, instability cannot be tamed by further data collection. Rather, the data must be processed and analyzed in some better fashion (e.g. U test described below).

These large instabilities explain the contradictory results in the effort estimation literature. Jorgensen reviews fifteen studies that compare model-based to expert-based estimation. Five of those studies found in favor of expert-based methods; five found no difference; and five found in favor of model-based estimation [4]. Such diverse conclusions are to be expected if models exhibit large instabilities in their performance.

B. Diagnosing the Cause

The thin line of Figure 2 is drawn by sorting the relative error[4] (RE) seen in four of the subsets studied in Figure 1. Observe that while most of the actual RE values are nearly zero, an *infrequent* number (on the right hand side) are *extremely large* (up to 8000 in the second plot). Such large *spikes* in RE result when the predicted values are much larger than the actual values and result from (1) noise in the data or (2) a training set that learns an overly steep exponential function for the effort model.

The size of the spikes in Figure 2 are remarkable. Research papers typically report RE values in the range $0 \leq RE \leq 3$ [13]. Such values are completely dwarfed by errors in the range of 8000 such as those seen in the second plot of Figure 2

[3]MRE = magnitude of relative error $= abs(actual - predicted)/actual$.

[4]RE = $(predicted - actual)/actual$

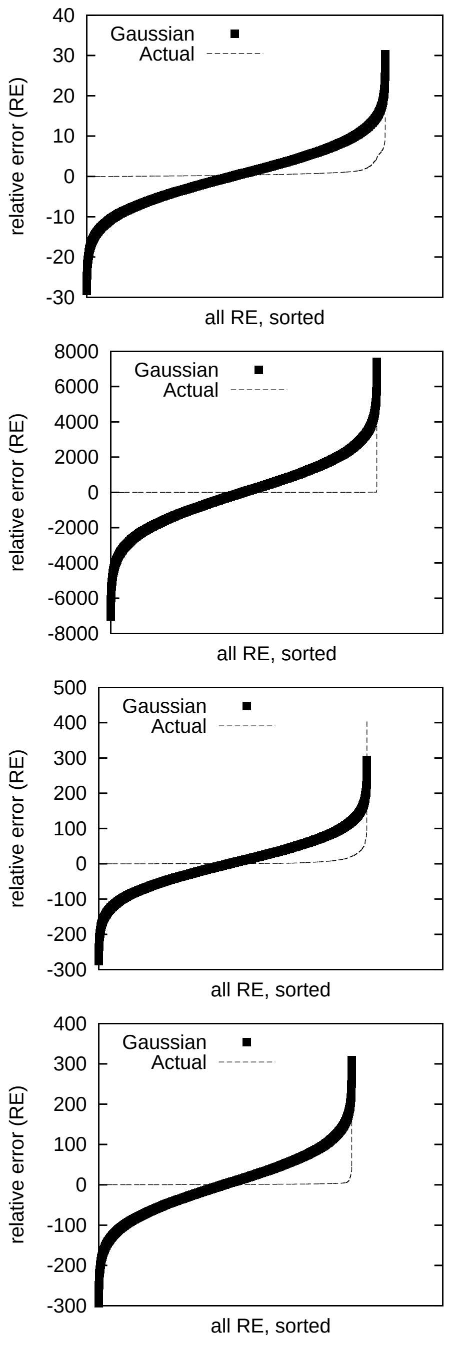

Fig. 2. Relative errors seen in COSEEKMO's experiments on four data sets. Thin lines show the actual values. Thick lines show a Gaussian distribution that uses mean and standard deviation of the actual values. From top to bottom, the plots are of: *(top)* NASA ground systems; NASA software written around 1975; NASA embedded software (i.e. software developed within tight hardware, software, and operational constraints); *(bottom)* some FORTRAN-based software systems.

(hence, most of the thin lines in Figure 2 are flat). These very large, but infrequent, outliers explain conclusion instability:

- Large outliers can make mean calculations highly misleading. A single large outlier can make the mean value far removed from the median[5].
- For data sets with only a small number of outliers (e.g. Figure 2), the conclusions reached from different subsets can be very different, depending on the absence or presence of the infrequent outliers.

[5]Median: the value below which 50% of the values fall.

Figure 2 also illustrates how poorly standard methods assess the performance of effort estimation data. Demsar [15] offers a definition of *standard methods* in data mining. In his study of four years of proceedings from the *International Conference on Machine Learning*, Demsar found that the standard method of comparative assessment were t-tests over some form of repeated sub-sampling such as cross-validation, separate subsets, or randomized re-sampling. Such t-tests assume that the distributions being studied are Gaussian and, as shown by the thick line of Figure 2, effort estimation results can be highly non-Gaussian. These thick lines show a Gaussian cumulative distribution function computed from the means and standard deviations of the actual RE values (the thin lines). For example, the Gaussian approximation to the *actual values* of

$$\{1.1, 1.3, 1.5, 1.7, 2.1, 2.3, 2.7, 800\}$$

has a mean of 101.6 and a standard deviation of 282.2. Observe how poorly such Gaussian distributions represent the actual RE values:

- There exists orders of magnitude differences between the *actual* plots (the thin lines) and the *Gaussian* approximations (the thick lines).
- The Gaussian goes negative while none of our effort estimation methods assume that it takes less than no time to build software.

C. Fixing Instability

The problem of comparatively assessing L learners run on multiple sub-samples of D data sets has been extensively studied in the data mining community. T-tests that assume Gaussian distributions are strongly deprecated. For example, Demsar [15] argues that non-Gaussian populations are common enough to require a methodological change in data mining.

After reviewing a wide range of comparisons methods[6], Demsar advocates the use of the 1945 Wilcoxon [16] signed-rank test that compares the ranks for the positive and negative differences (ignoring the signs). Writing five years earlier, Kitchenham et.al. [9] comment that the Wilcoxon test has its limitations. Demsar's report offers the same conclusion, noting that the Wilcoxon test requires that the sample sizes are the same. To fix this problem, Demsar augments Wilcoxon with the Friedman test.

One test not studied by Demsar is Mann and Whitney's 1947 modification [14] to Wilcoxon rank-sum test (proposed along with his signed-rank test). We prefer this test since:

- The Mann-Whitney U test does not require that the sample sizes are the same. So, in a single U test, learner L_1 can be compared to all its rivals.

[6]Paired t-tests with and without the use of geometric means of the relative ratios; binomial tests with the Bonferroni correction; paired t-tests; ANOVA; Wilcoxon; Friedman

The sum and median of A's ranks is

$$\begin{aligned} sum_A &= 1 + 2.5 + 5.5 + 7 + 9.5 = 25.5 \\ median_A &= 5.5 \end{aligned}$$

and the sum and median of B's ranks is

$$\begin{aligned} sum_B &= 2.5 + 4 + 5.5 + 8 + 9.5 + 11 = 40.5 \\ median_B &= 6.75 \end{aligned}$$

The U statistic is calculated from $U_x = sum_x - (N_1(N_2 + 1))/2$:

$$\begin{aligned} U_A = 25.5 - 5 * 6/2 = 10.5 \\ U_B = 40.5 - 6 * 7/2 = 19.5 \end{aligned}$$

These can be converted to a Z-curve using:

$$\begin{aligned} \mu &= (N_1 N_2)/2 &&= 516.4 \\ \sigma &= \sqrt{\frac{N_1 N_2 (N_1 + N_2 + 1)}{12}} &&= 5.477 \\ Z_A &= (U_A - \mu)/\sigma &&= -0.82 \\ Z_B &= (U_B - \mu)/\sigma &&= 0.82 \end{aligned}$$

(Note that Z_A and Z_B have the same absolute value. In all case, these will be the same, with opposite signs.)
If $abs(Z) < 1.96$ then the samples A and B have the same median rankings (at the 95% significance level). In this case, we add one to both $ties_A$ & $ties_B$. Otherwise, their median values can be compared, using some domain knowledge. In this work, *lower* values are better since we are comparing errors. Hence:

- If $median_A < median_B$ add 1 to both $wins_A$ & $losses_B$.
- Else if $median_A > median_B$ add 1 to both $losses_A$ & $wins_B$.
- Else, add 1 to both $ties_A$ & $ties_B$.

Fig. 3. An example of the Mann-Whitney U test.

- The U test does not require any post-processing (such as the Friedman test) to conclude if the median rank of one population (say, the L_1 results) is greater than, equal to, or less than the median rank of another (say, the $L_2, L_3, .., L_x$ results).

Figure 3 shows the U test for the two treatments A and B discussed in the introduction. The test concludes that these treatments are not statistically different (at the 95% significance level). As defined in Figure 3, this test counts the $wins$, $ties$, and $losses$ for A and B (where A and B are single or groups of methods). Since we seek methods that can be rejected, the value of interest to us is how often methods *lose*.

III. Experiments

A. Data

This paper is based on 19 subsets from two sources. $COC81$[7] comes from Boehm's 1981 text on effort estimation. $NASA93$[8] comes from a study funded by the Space Station Freedom Program. $NASA93$ contains data from six different NASA centers including the Jet Propulsion Laboratory. For details on this data, see the appendix. In terms of conclusion instability across data subsets, the important feature to note is that our data comes from two sources with demonstrably different properties. In 20 repeats of 90% sampling of the data, the coefficients learned by linear regression for $NASA93$ were found to have a much larger variation than $COC81$ [17].

[7]See "coc81" at `http://promisedata.org/repository`.
[8]See "nasa93" at `http://promisedata.org/repository`.

The data available to this study was in the COCOMO-I format [1] that dictates the set of possible features. An alternative is the case-based reasoning (CBR) approach used by Shepperd [18] and others [19]. CBR accepts data with any set of features. COCOMO-I was chosen since we could not access a large enough set of CBR-style data sets with arbitrary sets of features. Also, unlike other effort estimators such as PRICE-S [20], SLIM [21], or SEER-SEM [22], COCOMO is a public domain model with published data and baseline results [23].

In 2000, Boehm et.al. updated the COCOMO-I model [24]. After the update, numerous features remained the same:

- Effort is assumed to be exponential on model size.
- Boehm et.al. recommends a procedure called *local calibration* (described below) for tuning generic COCOMO to a local situation.
- Boehm et.al. advises that effort estimates can be improved via *manual stratification* (described later); i.e. use domain knowledge to select relevant past data.

At the 2005 COCOMO forum, there were some discussions about relaxing the security restrictions around the COCOMO-II data set. To date, those discussions have not progressed. Since other researchers do not have access to COCOMO-II, this paper will only report results from COCOMO-I.

B. Experimental Procedure

Each of the 19 subsets of $COC81$ and $NASA93$ were expressed as a table of data $P * F$. The table stored *project* information in P rows and each row included the *actual* development effort. In the 19 subsets of $COC81$ and $NASA93$ used in the study, $20 \leq P \leq 93$. The upper bound of this range ($P = 93$) is the largest data set's size. The lower bound of this range ($P = 20$) was selected based on experiments described elsewhere [13]. For details on these data sets, see the appendix.

The table also has F columns containing the project *features* $\{f_1, f_2, ...\}$. The features used in this study come from Boehm's COCOMO-I work (described in the appendix) and include items such as lines of code (KLOC), schedule pressure (sced), analyst capability (acap), etc.

To build an effort model, the rows of each table were divided at random into a $Train$ and $Test$ set (and $|Train| + |Test| = P$). COSEEKMO's different methods are then applied to the $Train$ set to generate a model. This model was then used on the $Test$ set. In order to compare this study with our work [13], we use the same $Test$ set size as the COSEEKMO study; i.e. $|Test| = 10$.

Effort models were assessed via three evaluation criteria:

- AR: absolute residual; $abs(actual - predicted)$;
- MRE: magnitude of relative error; $\frac{abs(predicted - actual)}{actual}$;
- MER: magnitude of error relative to estimate; $\frac{abs(actual - predicted)}{predicted}$;

Note that, according to conclusion instability, there should be instability in how methods are ranked by AR, MER, and MRE.

For the sake of statistical validity, the above procedure was repeated 20 times for each of the 19 subsets of $COC81$ and $NASA93$. Each time, a different seed was used to generate

method = name	row pruning	column pruning	learner
a = LC	✘	✘	LC = Boehm's local calibration
b = COCOMIN + LC	✔automatic $O(P^2)$	✘	local calibration
c = COCOMIN + LOCOMO + LC	✔automatic $O(P^2)$	✔automatic $O(F{\cdot}log(F)+F)$	local calibration
d = LOCOMO + LC	✘	✔automatic $O(F{\cdot}log(F)+F)$	local calibration
e = Manual Stratification + LC	✔manual	✘	local calibration
f = M5pW + M5p	✘	✔Kohavi's WRAPPER [25] calling M5p [26], $O(2^F)$	model trees
g = LOCALW + LC	✘	✔Chen's WRAPPER [13] calling LC, $O(2^F)$	local calibration
h = LsrW + LSR	✘	✔Kohavi's WRAPPER [25] calling LSR, $O(2^F)$	linear regression
i = NEAREST	✔automatic $O(P^2)$	✘	mean effort of nearest neighbors

Fig. 4. Eight effort estimation methods explored in this paper. F is the number of features (columns) and P is the number of projects (rows).

the $Train$ and $Test$ sets. Recall that our data came from two different sources (Boehm's 1981 work and NASA in the 1990s). Hence, according to conclusion instability, there should be instability in the conclusions reached from the data from different sources or across the different randomly selected $Train$ and $Test$ sets.

C. 158 Methods

COSEEKMO's 158 methods combine:

- Some *learners* such as standard linear regression, local calibration, and model trees.
- Various *pre-processors* that may prune rows or columns.
- Various *nearest neighbor* algorithms that can be used either as learners or as pre-processors to other learners.

Note that only some of the learners use pre-processors. In all, COSEEKMO's methods combine 15 learners without a pre-processor and 8 learners with 8 pre-processors; i.e. $15 + 8 * 8 = 79$ combinations in total.

COSEEKMO's methods input project features described using the symbolic range *very low* to *extra high*. Some of the methods map the symbolic range to numerics 1..6. Other methods map the symbolic range into a set of *effort multipliers* and *scale factors* developed by Boehm and are shown in the appendix (Figure 9). Previously, we have queried the utility of these effort multipliers and scale factors [13]. COSEEKMO hence executes its 79 methods twice: once using Boehm's values, then once again using perturbations of those values. Hence, in all, COSEEKMO contains $2 * 79 = 158$ methods.

There is insufficient space in this paper to describe the 158 methods (for full details, see [27]). Such a complete description would be pointless since, as shown below, most of them are beaten by a very small number of preferred methods. For example, our previous concerns regarding the effort multipliers and scale factors proved unfounded (and so at least half the runtime of COSEEKMO is wasted).

D. Brief Notes on 8 Methods

This paper focuses on the eight methods $(a, b, c, d, ef, g, h, i)$ of Figure 4. Four of these, (a, b, c, d), are our preferred methods while the other four comment on premises of some prior publications [28].

One way to categorize Figure 4 is by their relationship to accepted practice (as defined in the COCOMO texts [1], [24]). Methods (a, e) are endorsed as best practice in the COCOMO community. The others are our attempts to do better than current established practice using e.g. intricate learning schemes or intelligent data pre-processors.

Method f is an example of a more intricate learning schemes. Standard linear regression assumes that the data can be fitted to a single model. On the other hand, the model trees used in f [26] permit the generation of multiple models (as well as a decision tree for selecting the appropriate model).

As to intelligent data pre-processors, COSEEKMO's pre-processors prune *irrelevant* projects and features. After pruning, the learner executes on a new table $P' * F'$ where $P' \subseteq P$ and $F' \subseteq F$. Pruning is useful since project data collected in one context may not be relevant to another. Kitchenham et.al. [29] take great care to document this effect. In a systematic review comparing estimates generated using historical data *within* the same company or *imported* from another, Kitchenham et.al. found no case where it was better to use data from other sites. Indeed, sometimes, importing such data yielded significantly worse estimates. Similar projects have less variation and so can be easier to calibrate: Chulani et.al. [23] & Shepperd and Schofield [30] report that row pruning improves estimation accuracy.

Row pruning can be *manual* or *automatic*. In *manual row pruning* (also called "stratification" in the COCOMO literature [24]), an analyst applies their domain knowledge to select project data that is similar to the new project to be estimated. Unlike other methods, the manual stratification used here uses different subsets to create $Train$ sets.

- In every case except for the manual stratification, $Train$ and $Test$ sets are created from the same subsets of $NASA93$ or $COC81$.
- In manual stratification, the $Test$ set is created in the same manner from the subsets. However, the $Train$ set is created from the projects drawn from the $NASA93$ or $COC81$ and not their subsets.

Automatic row pruning uses algorithmic techniques to select a subset of the projects (rows). NEAREST and LOCOMO [27] are automatic and use nearest neighbor methods on the $Train$ set to find the k most relevant projects to generate predictions for the projects in the $Test$ set. The core of both automatic algorithms is a distance measure that must compare all pairs of projects. Hence, these automatics methods take time $O(P^2)$. Both NEAREST and LOCOMO learn an appropriate k from the $Train$ set and the k with the lowest error is used when processing the $Test$ set. NEAREST averages the effort associated with the k nearest neighbors while LOCOMO passes the k nearest neighbors to Boehm's local calibration (LC) method.

Column pruners fall into two groups:

- WRAPPER and LOCALW are very thorough search algorithms that explore subsets of the features, in no set order. This search takes time $O(2^F)$.
- COCOMIN [31] is far less thorough. COCOMIN is a near linear-time pre-processor that selects the features on some heuristic criteria and does not explore all subsets of the features. It runs in $O(F{\cdot}log(F))$ for the sort and $O(F)$ time for the exploration of selected features.

Each method may use a column or row pruner or, as with (a, i), no pruning at all. In methods (f, h), the notation M5pW and LsrW denotes a WRAPPER that uses M5p or LSR as its target learner (respectively).

For more details on these eight methods, see the appendix.

IV. Results

Figures 5, 6, and 7 show results from 20 repeats of:

- Dividing some subset into $Train$ and $Test$ sets;
- Learning an effort model from the $Train$ set using COSEEKMO's 158 methods;
- Applying that model to the $Test$ set;
- Collecting performance statistics from the $Test$ set using AR, MER, or MRE;
- Ranking the performance results from different methods using Mann-Whitney U test.

In these results, conclusion instability due to *changing evaluation criteria* can be detected by comparing results across Figure 5, Figure 6, and Figure 7. Also, conclusion instability due to *changing subsets* can be detected by comparing results across different subsets generated by changing the random seed controlling $Train$ and $Test$ set generation (i.e. the three runs of Figure 5 that used different random seeds).

A single glance shows our main result: the plots are very similar. Specifically, the (a, b, c, d) results fall very close to $y = 0$ losses. The significance of this result is discussed below.

Each mark on these plots shows the number of times a method loses in seven $COC81$ subsets (left plots) and twelve $NASA93$ subsets (right plots). The x-axis shows results from the methods $(a, b, c, d, e, f, g, h, i)$ described in Figure 4.

In these plots, methods that generate *lower* losses are *better*. For example, the top-left plot of Figure 5 shows results for ranking methods applied to $COC81$ using AR. In that plot, all of methods (a, d) results from the seven $COCO81$ subsets can be seen at $y = losses \approx 0$. That is, in that plot, these two methods *never* lose against the other 158 methods.

In a result consistent with the KFM findings, there are some instabilities in our results. For example, the exemplary performance of methods (a, d) in the top-left plot of Figure 5 does *not* repeat in other plots. For example in the $NASA93$ MRE and MER results shown in Figure 6 and Figure 7, method b loses much less than methods (a, d).

However, in terms of number of losses generated by methods (a, b, c, d, e, f, g, h), the following two results holds across all evaluation criteria and all subsets:

1) One member of method (a, b, c, d) always performs better (loses least) than all members of methods (e, f, g, h). Also, all members of methods (e, f, g, h) perform better than i.

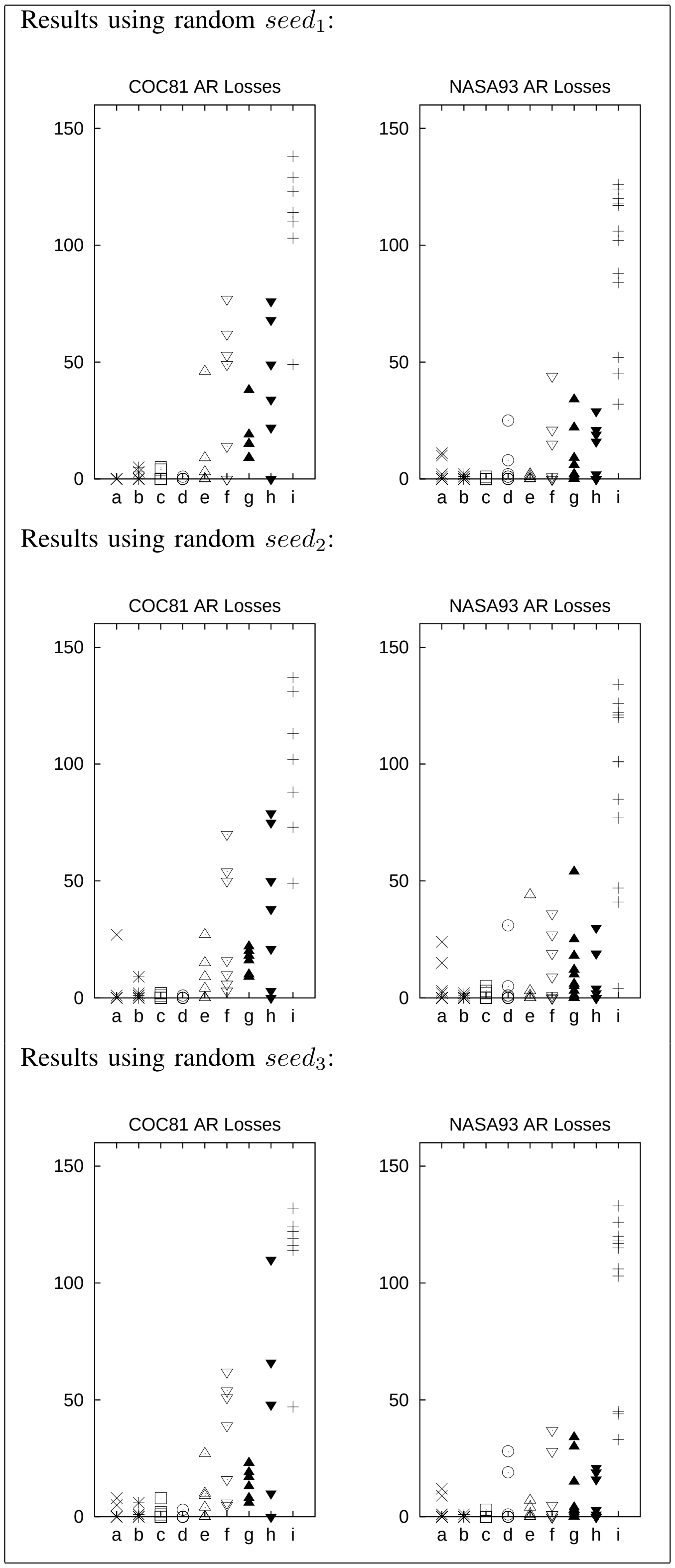

Fig. 5. U tests using AR and repeated three times with different random seeds.

2) Compared to 158 methods, one member of (a, b, c, d) always loses at some rate very close to zero.

As observed by KFM, there is no single universal *best* method. Nevertheless, out of 158 methods, there are 154

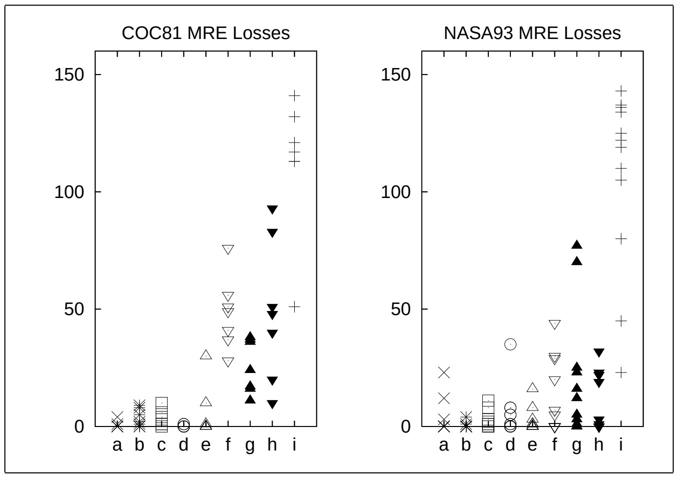

Fig. 6. U tests using MRE.

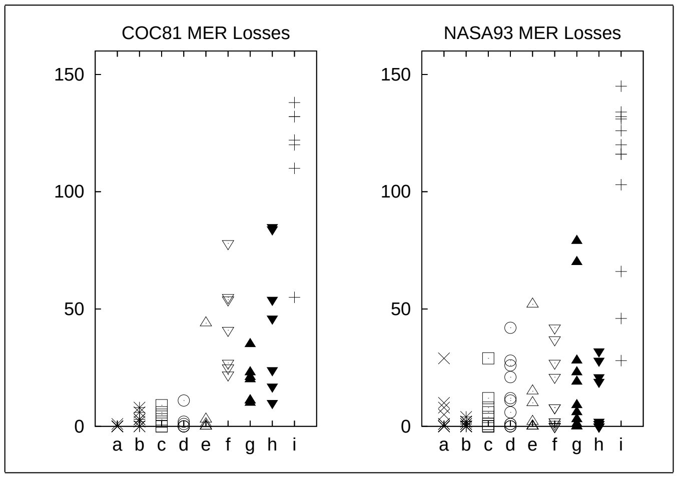

Fig. 7. U tests using MER.

clearly inferior methods. Hence, we recommend ranking methods (a, b, c, d) on all the available historical data, then applying the best ranked method to estimate new projects.

The superiority of (a, b, c, d) is a strong endorsement of Boehm's 1981 estimation research. These four methods are based around Boehm's preferred method for calibrating generic COCOMO models to local data. Method a is Boehm's *local calibration* (or LC) procedure (defined in the appendix). Methods b and d augment LC with pre-processors performing simple column or row pruning (and method c combines both b and d). Methods (a, b, c, d) endorse three of Boehm's 1981 assumptions about effort estimation:

Boehm'81 assumption 1:
Effort can be modeled as a single function that is exponential on lines of code ...

Boehm'81 assumption 2:
...and linearly proportional to the product of a set of effort multipliers;

Boehm'81 assumption 3:
The effort multipliers influence the effort by a set of pre-defined constants that can be taken from Boehm's textbook [1].

Our results endorse some of Boehm's estimation modeling work, but not all of it. Method e is manual stratification, a commonly recommended method in the COCOMO literature. This method performs surprisingly well and often out-performs many intricate automatic methods. However, as shown above, method e is always inferior to more than one of (a, b, c, d). Hence, contrary to the COCOMO literature, we recommend replacing manual stratification with automatic methods.

Our results argue that there is little added value in methods (f, g, h). This is a useful result since these methods contain some of our slowest algorithms. For example, the WRAPPER column selection method used in (f, g, h) is an elaborate heuristic search through, potentially, all combinations of the columns.

The failure of model trees in method f is also interesting. If the model trees of method f had out-performed (a, b, c, d), that would have suggested that effort is a multi-parametric phenomenon where, e.g. over some critical size of software, different effects emerge. This proved not to be the case, endorsing Boehm's assumption that effort can be modeled as a single parametric log-linear equation.

Of all the methods in Figure 4, (a, b, c, d) perform the best and i performs the worst. One distinguishing feature of method i is the *assumptions* it makes about the domain. The NEAREST neighbor method i is *assumption-less* since it makes none of the *Boehm'81* assumptions listed above. But, while assumption-less, NEAREST is not *assumption-free*. NEAREST uses a simple n-dimensional Euclidean distance to find similar projects. Wilson & Martinez caution that this measure is inappropriate for sparse data sets [32]. Such sparse data sets arise when many of the values of project features are unavailable. Shepperd & Schofield argue that their case-based reasoning methods, like NEAREST procedure used in method i, are better suited to sparse data domains where precise numeric values are *not* available on all factors [30]. All our data sets are non-sparse. Hence, it is not surprising that method i performs poorly on our data.

V. External Validity

The case was made above that our conclusions are valid across different evaluation criteria and samplings. However, no empirical evaluation is bias free. Some biases remain including additional evaluation bias, sampling bias, a paradigm bias, a modeling bias, and a bias in our selection of methods.

Additional evaluation bias: We have shown stability across three evaluation criteria: AR, MER, and MRE. This does not mean that we have shown stability across *all possible* evaluation biases. It is certainly possible that biases other than those explored here will offer different rankings to our estimation methods. For example, this study does not explore PRED(30)[9] since Shepperd (personal communication) depreciates it and neither Foss et.al. [10] or Myrtveit et.al. [11] advocate its use. However, at the very least, we have shown that the problem of ranking estimation methods may not be as difficult as suggested by KFM (at least, for non-sparse data in the COCOMO format).

[9]PRED(N) is the percent of the MRE less than N%.

Sampling bias: Our model-based estimation methods use data and so are only useful in organizations that maintain historical data on their projects. Such data collection is rare in organizations with low process maturity. However, it is common elsewhere; e.g. amongst government contractors whose contract descriptions include process auditing requirements. For example, it is common practice at NASA and the United States Department of Defense to require a model-based estimate at each project milestone. Such models are used to generate estimates or to double-check an expert-based estimate.

Another source of sampling bias was already mentioned above; our data sets are non-sparse and sparse data sets may be more suitable for nearest neighbor tools.

Yet another source of bias is that some of the data used here comes from NASA and NASA works in a particularly unique market niche. Nevertheless, we argue that results from NASA are relevant to the general software engineering industry. NASA makes extensive use of contractors. These contractors service many other industries. These contractors are contractually obliged (ISO-9001) to demonstrate their understanding and usage of current industrial best practices. For these reasons, other noted researchers such as Basili, Zelbowitz, et.al. [33] have argued that conclusions from NASA data are relevant to the general software engineering industry.

Biases in the paradigm: The paper explores model-based methods (e.g. COCOMIN, LOCOMO, LC) and not expert-based methods. Model-based methods use some algorithm to summarize old data and make predictions about new projects. Expert-based methods use human expertise (possibly augmented with process guidelines or checklists) to generate predictions. Jorgensen [4] argues that most industrial effort estimation is expert-based and lists 12 *best practices* for such effort-based estimation. The comparative evaluation of model-based vs. expert-based methods must be left for future work. Before we can compare any effort estimation methods (be they model-based or expert-based) we must first demonstrate that any two methods can be comparatively assessed. For more on expert-based methods, see [4], [23], [30], [34].

Biases in the model: This study uses COCOMO data sets since these were the only public domain data we could access. Nevertheless, the techniques described here can easily be generalized to other models. For example, here we use COSEEKMO to select best parametric methods in the COCOMO format [1], [24] but it could just as easily be used to assess other model-based tools like PRICE-S [20], SEER-SEM [22], or SLIM [21]. However, it should be noted that in the above study, 154 out of 158 methods were demonstrably inferior. If those percentages carry over to a study of SEER-SEM vs. PRICE-S vs. SLIM, then we would predict that it will yield similar performances.

Biases in the selection of methods: Another source of bias in this study is the set of methods explored by this study. We can make no claim that Figure 4 or COSEEKMO's other 150 methods represents the space of possible effort estimation methods. Indeed, when we review the space of known methods (see Figure 1 in [11]), it is clear that COSEEKMO covers only a small part of that total space.

Instead of claiming that (a, b, c, d) are "best", we really should say that (a, b, c, d) are the best we have seen so far after four years of trying many alternatives. The reader may know of other effort estimation methods they believe we should try. Alternatively, the reader may have a design or an implementation of a new kind of effort estimator. In either case, before it can be shown that an existing or new method is better than those shown in Figure 4, we first need a demonstration that there exists statistical tests that distinguish between methods. This paper offers such a demonstration.

VI. Conclusion

Our goal was the rejection of sub-optimum effort estimation methods. If this goal is not possible, then an effort estimation workbench can grow to unmanageable proportions. For example, the 158 methods of COSEEKMO take nearly a day to run. Much of that execution is wasted since, as shown above, 154 of those methods are superfluous.

Previous studies have doubted the practicality of selecting the "best" estimation method. For example, Myrtveit. et.al. concluded that

> *…the conclusions on "which model is best" to a large extent will depend on the (evaluation criteria) chosen. This is a serious problem because, at present, we have no theoretical foundation to prefer, say, (mean) MRE to (mean) MER or (mean) AR …* [11, p390]

Our alternate conclusion is that the means of any measure is counter-indicated by the presence of large outliers. The effect of large outliers can be mitigated by the use of non-parametric ranked statistics that compare medians (the U test). We have shown above that such non-parametric methods do not suffer from KFM's conclusion instability. Also, our results suggest that there is no need to decide between (e.g.) MRE, MER, or AR since they can all report that the same set of four methods is "best".

Further, we have shown above that Myrtveit et.al. are quite correct when they report

> *…for most of the (evaluation criteria), the results are not sufficiently reliable across the samples for the same accuracy indicator …*
> *This implies that the conclusions on "which model is best" to some extent depend on the particular sample at hand, even for samples drawn from the same population.* [11, p390]

For example, while we advocate four methods, none of them are always best in all sub-samples of the data. However, our results are far more optimistic that KFM: we have seen above that one of these four methods is always better than the other 154 methods:

- A single linear model is adequate for the purposes of effort estimation. All the methods that assume multiple linear models, such as model trees (f), or no parametric form at all, such as nearest neighbor (i), perform relatively poorly.

- Elaborate searches do not add value to effort estimation. All the $O(2^F)$ column pruners do worse than near-linear-time column pruning.
- The more intricate methods such as model trees do no better than other methods.

Finally, we comment on the practical implications of this study. *There is no best estimation method.* However, there exists a very small number of most useful estimation methods. We advise that the following methods should be tried and the one that does best on historic data (assessed using Mann-Whitney U test) should be used to predict new projects:

- Adopt the three Boehm'81 assumptions and use LC-based methods.
- While some row and column pruning can be useful, elaborate column pruning (requiring an $O(2^F)$ search) is not. Hence, try LC with zero or more of LOCOMO's row pruning or COCOMIN's column pruning.
- If the training data is sparse, then try averaging the efforts seen in nearest neighbors (for more details, see [30]).

Acknowledgments

Martin Shepperd was kind enough to make suggestions about different evaluation biases and the design of the NEAREST and LOCOMO methods.

Appendix

A. Data Used in This Study

In this study, effort estimators were built using all or some *part* of data from two sources:

$COC81$: 63 records in the COCOMO-I format. Source: [1, p496-497]. Download from `http://unbox.org/wisp/trunk/cocomo/data/coc81modeTypeLangType.csv`.

$NASA93$: 93 NASA records in the COCOMO-I format. Download from `http://unbox.org/wisp/trunk/cocomo/data/nasa93.csv`.

Taken together, these two sets are the largest COCOMO-style data source in the public domain (for reasons of corporate confidentiality, access to Boehm's COCOMO-II data set is highly restricted). $NASA93$ was originally collected to create a NASA-tuned version of COCOMO, funded by the Space Station Freedom Program and contains data from six NASA centers including the Jet Propulsion Laboratory. For more details on this dataset, see [13].

Different subsets and number of subsets used (in parenthesis) are:

All(2): selects all records from a particular source.

Category(2): $NASA93$ designation selecting the type of project; e.g. avionics.

Center(2): $NASA93$ designation selecting records relating to where the software was built.

Fg(1): $NASA93$ designation selecting either "f" (flight) or "g" (ground) software.

Kind(2): $COC81$ designation selecting records relating to the development platform; e.g. max is mainframe.

Lang(2): $COC81$ designation selecting records about different development languages; e.g ftn is FORTRAN.

Mode(4): designation selecting records relating to the COCOMO-I development mode: one of semi-detached, embedded, and organic.

Project(2): $NASA93$ designation selecting records relating to the name of the project.

Year(2): is a $NASA93$ term that selects the development years, grouped into units of five; e.g. 1970, 1971, 1972, 1973, 1974 are labeled "1970".

There are more than 19 subsets overall. Some have fewer than 20 projects and hence were not used. The justification for using 20 projects or more is offered in [13].

B. Learners Used in This Study

1) Learning with Linear Regression: Linear regression assumes that the data can be approximated by one linear model that includes lines of code (KLOC) and other features f seen in a software development project:

$$effort = \beta_0 + \sum_i \beta_i \cdot f_i$$

Linear regression adjusts β_i to minimize the *prediction error* (the difference between predicted and actual values for the project).

Boehm argues that effort is exponential on KLOC [1]:

$$effort = a \cdot KLOC^b \cdot \prod_i \beta_i$$

(where a and b are domain-specific constants). Such exponential functions can be learned via linear regression after they are converted to the following linear form:

$$log(effort) = log(a) + b{\cdot}log(KLOC) + \sum_i log(\beta_i)$$

All our methods transform the data in this way. Hence, when collecting performance statistics, it is necessary to unlog the estimates.

2) Learning with Model Trees: Model trees are a generalization of linear regression. Instead of fitting the data to *one linear model*, model trees learn *multiple linear models*, and a decision tree that decides which linear model to use. Model trees are useful when the projects form regions and different models are appropriate for different regions. COSEEKMO includes the M5p model tree learner defined by Quinlan [26].

3) Learning with Local Calibration: Local calibration (LC) is a specialized form of linear regression developed by Boehm [1, p526-529]. LC assumes project effort is exponential on KLOC; i.e.

$$effort = a \cdot KLOC^b \cdot \prod_i \beta_i$$

Figure 9 shows the β_i values recommended by Boehm (the names on the left hand side are defined in Figure 8). When β_i is used in the above equation, they yield estimates in months where one month is 152 hours (and includes development and

upper:	acap:	analysts capability
increase	pcap:	programmers capability
these to	aexp:	application experience
decrease	modp:	modern programming practices
effort	tool:	use of software tools
	vexp:	virtual machine experience
	lexp:	language experience
middle	sced:	schedule constraint
lower:	data:	data base size
decrease	turn:	turnaround time
these to	virt:	machine volatility
increase	stor:	main memory constraint
effort	time:	time constraint for CPU
	rely:	required software reliability
	cplx:	process complexity

Fig. 8. Features used in this study. From [1]. Most range from 1 to 6 representing "very low" to "extremely high".

		1	2	3	4	5	6
upper	ACAP	1.46	1.19	1.00	0.86	0.71	
(increase	PCAP	1.42	1.17	1.00	0.86	0.70	
these to	AEXP	1.29	1.13	1.00	0.91	0.82	
decrease	MODP	1.2	1.10	1.00	0.91	0.82	
effort)	TOOL	1.24	1.10	1.00	0.91	0.83	
	VEXP	1.21	1.10	1.00	0.90		
	LEXP	1.14	1.07	1.00	0.95		
middle	SCED	1.23	1.08	1.00	1.04	1.10	
lower	DATA		0.94	1.00	1.08	1.16	
(increase	TURN		0.87	1.00	1.07	1.15	
these to	VIRT		0.87	1.00	1.15	1.30	
increase	STOR			1.00	1.06	1.21	1.56
effort)	TIME			1.00	1.11	1.30	1.66
	RELY	0.75	0.88	1.00	1.15	1.40	
	CPLX	0.70	0.85	1.00	1.15	1.30	1.65

Fig. 9. The COCOMO-I β_i table [1]. For example, the bottom right cell is saying that if CPLX=6, then the nominal effort is multiplied by 1.65.

management hours). To operate, LC linearizes the exponential equation to generate

$$log(effort) = log(a) + b \cdot log(KLOC) + \sum_i log(\beta_i)$$

Linear regression would try to adjust all the β_i values. This is not practical when training on a very small number of projects. Hence, LC fixes the β_i values while adjusting the $< a, b >$ values to minimize the prediction error. We shall refer to LC as "standard practice" since, in the COCOMO community at least, it is the preferred method for calibrating standard COCOMO data [24].

4) Learning with Nearest Neighbor: Nearest neighbor makes predictions using past data that is similar to a new situation. Some distance measure is used to find the k nearest older projects to each project in the $Test$ set. An effort estimate can be generated from the mean effort of the k nearest neighbors.

The benefit of nearest neighbor algorithms is that they make the fewest domain assumptions. That is, they can process a broader range of the data available within projects. For example:

- LC cannot be applied unless projects are described using the COCOMO ontology (Figure 8).
- Linear regression and model trees are best applied to data where most of the values for the numeric factors are known.

The drawback of nearest neighbor algorithms is that, sometimes, the domain assumptions they ignore are important to that domain. For example, if effort is really exponential on KLOC, a standard nearest neighbor algorithm has no way to exploit that.

C. Pre-Processors Used in This Study

1) Pre-processing with Row Pruning: The LOCOMO tool [27] in COSEEKMO is a row pruner that combines a nearest neighbor method with LC. LOCOMO prunes away all projects except those k "nearest" to the $Test$ set data.

To learn an appropriate value for k, LOCOMO uses the $Train$ set as follows:

- For each project $p_0 \in Train$, LOCOMO sorts the remaining $Train - p_0$ examples by their Euclidean distance from p_0.
- LOCOMO then passes the k_0 examples closest to p_0 to LC. The returned $< a, b >$ values are used to estimate effort for p_0.
- After trying all possible k_0 values, $2 \leq k_0 \leq |Train|$, k is then set to the k_0 value that yielded the smallest mean MRE[10].

This calculated value k is used to estimate the effort for projects in the $Test$ set. For all $p_1 \in Test$, the k nearest neighbors from $Train$ are passed to LC. The returned $< a, b >$ values are then used to estimate the effort for p_1.

2) Pre-Processing with Column Pruning: Kirsopp & Schofeld [35] and Chen & Menzies & Port & Boehm [28] report that column pruning improves effort estimation. Miller's research [36] explains why. Column pruning (a.k.a. feature subset selection [37] or variable subset selection [36]) reduces the deviation of a linear model learned by minimizing least squares error [36]. To see this, consider a linear model with constants β_i that inputs features f_i to predict for y:

$$y = \beta_0 + \beta_1 \cdot f_1 + \beta_2 \cdot f_2 + \beta_3 \cdot f_3 ...$$

The variance of y is some function of the variances in f_1, f_2, etc. If the set F contains "noise" (spurious signals unconnected to the target variable y) then random variations in f_i can increase the uncertainty of y. Column pruning methods decrease the number of features f_i, thus increasing the stability of the y predictions. That is, the fewer the features (columns), the more restrained are the model predictions.

Taken to an extreme, column pruning can reduce y's variance to zero (e.g. by pruning the above equation back to $y = \beta_0$) but increases model error (the equation $y = \beta_0$ will ignore all project data when generating estimates). Hence, intelligent column pruners experiment with some proposed subsets $F' \subseteq F$ before changing that set. COSEEKMO currently contains three intelligent column pruners: WRAPPER, LOCALW, and COCOMIN.

WRAPPER [25] is a standard best-first search through the space of possible features. At worst, the WRAPPER must search an space exponential on the number of features F; i.e. 2^F. However, a simple best-first heuristic makes WRAPPER practical for effort estimation. At each step of the search, all

[10]A justifications for using the mean measure within LOCOMO is offered at the end of the appendix.

the current subsets are scored by passing them to a *target leaner*. If a set of features does not score better than a smaller subset, then it gets one "mark" against it. If a set has more than $STALE = 5$ number of marks, it is deleted. Otherwise, a feature is added to each current set and the algorithm continues.

In general, a WRAPPER can use any target learner. Chen's LOCALW is a WRAPPER specialized for LC. Previously [13], [28], we have explored LOCALW for effort estimation.

Theoretically, WRAPPER (and LOCALW)'s exponential time search is more thorough, hence more useful, than simpler methods that try fewer options. To test that theory, we will compare WRAPPER and LOCALW to a linear-time column pruner called COCOMIN [31].

COCOMIN is defined by the following operators:

$$\{sorter, order, learner, scorer\}$$

The algorithm runs in linear time over a *sorted* set of features, F. This search can be *order*ed in one of two ways:

- A "backward elimination" process starts with all features F and throws some away, one at a time.
- A "forward selection" process starts with one feature and adds in the rest, one at a time.

Regardless of the search order, at some point the current set of features $F' \subseteq F$ is passed to a *learner* to generate a performance *score* by applying the model learned on the current features to the $Train$ set. COCOMIN returns the features associated with the highest score.

COCOMIN pre-sorts the features on some heuristic criteria. Some of these criteria, such as standard deviation or entropy, are gathered without evaluation of the target learner. Others are gathered by evaluating the performance of the learner using only the feature in question plus any required features, such as KLOC for COCOMO, to calibrate the model. After the features are ordered, each feature is considered for backward elimination, or forward selection if chosen, in a single linear pass through the feature space, F. The decision to keep or discard the feature is based on an evaluation measure generated by calibrating and evaluating the model with the training data.

Based on [31], the version of COCOMIN used in this study:

- sorted the features by the highest median MRE;
- used a backward elimination search strategy;
- learned using LC;
- scored using mean MRE.

Note that mean MRE is used internally to COCOMIN (and LOCOMO, see above) since it is fast and simple to compute. Once the search terminates, this paper strongly recommends the more thorough (and hence more intricate and slower) median non-parametric measures to assess the learned effort estimation model.

REFERENCES

[1] B. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.

[2] C. Kemerer, "An empirical validation of software cost estimation models," *Communications of the ACM*, vol. 30, no. 5, pp. 416–429, May 1987.

[3] Spareref.com, "Nasa to shut down checkout & launch control system," August 26, 2002, http://www.spaceref.com/news/viewnews.html?id=475.

[4] M. Jorgensen, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, vol. 70, no. 1-2, pp. 37–60, 2004.

[5] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," January 2007, available from http://www.simula.no/departments/engineering/publications/Jorgensen.200%5.12.

[6] L. Briand, T. Langley, and I. Wieczorek, "A replicated assessment and comparison of common software cost modeling techniques," in *Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland*, 2000, pp. 377–386.

[7] K. Lum, J. Powell, and J. Hihn, "Validation of spacecraft cost estimation models for flight and ground systems," in *ISPA Conference Proceedings, Software Modeling Track*, May 2002.

[8] D. Ferens and D. Christensen, "Calibrating software cost models to Department of Defense Database: A review of ten studies," *Journal of Parametrics*, vol. 18, no. 1, pp. 55–74, November 1998.

[9] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd, "What accuracy statistics really measure," *Software, IEE Proceedings*, vol. 148, no. 3, pp. 81–85, 2001.

[10] T. Fos, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985 – 995, November 2003.

[11] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineerining*, vol. 31, no. 5, pp. 380–391, May 2005.

[12] I. H. Witten and E. Frank, *Data mining. 2nd edition*. Los Altos, US: Morgan Kaufmann, 2005.

[13] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting best practices for effort estimation," *IEEE Transactions on Software Engineering*, November 2006, available from http://menzies.us/pdf/06coseekmo.pdf.

[14] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 1947, available on-line at http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&hand%le=euclid.aoms/1177730491.

[15] J. Demsar, "Statistical comparisons of clasifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006, avaliable from http://jmlr.csail.mit.edu/papers/v7/demsar06a.html.

[16] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, pp. 80–83, 1945.

[17] T. Menzies, Z. Chen, D. Port, and J. Hihn, "Simple software cost estimation: Safe or unsafe?" in *Proceedings, PROMISE workshop, ICSE 2005*, 2005, available from http://menzies.us/pdf/05safewhen.pdf.

[18] M. Shepperd, "Software project economics: A roadmap," in *International Conference on Software Engineering 2007: Future of Software Engineering*, 2007.

[19] J. Li and G. Ruhe, "Decision support analysis for software effort estimation by analogy," in *Proceedings, PROMISE'07 workshop on Repeatable Experiments in Software Engineering*.

[20] R. Park, "The central equations of the price software cost model," in *4th COCOMO Users Group Meeting*, November 1988.

[21] L. Putnam and W. Myers, *Measures for Excellence*. Yourdon Press Computing Series, 1992.

[22] R. Jensen, "An improved macrolevel software development resource estimation model," in *5th ISPA Conference*, April 1983, pp. 88–92.

[23] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE Transaction on Software Engineerining*, vol. 25, no. 4, July/August 1999.

[24] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[25] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997. [Online]. Available: citeseer.nj.nec.com/kohavi96wrappers.html

[26] J. R. Quinlan, "Learning with Continuous Classes," in *5th Australian Joint Conference on Artificial Intelligence*, 1992, pp. 343–348, available from http://citeseer.nj.nec.com/quinlan92learning.html.

[27] O. Jalali, "Evaluation bias in effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007.

[28] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Finding the right data for software cost modeling," *IEEE Software*, Nov 2005.

[29] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross- vs. within-company cost estimation studies: A systematic review," *IEEE Transactions on Software Engineering*, pp. 316–329, May 2007.

[30] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 12, November 1997, available from http://www.utdallas.edu/~rbanker/SE_XII.pdf.

[31] D. Baker, "A hybrid approach to expert and model-based effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007.

[32] D. Wilson and T. Martinez, "Improved heterogeneous distance functions," *Journal of Artificial Intelligence Research*, vol. 6, pp. 1–34, 1997.

[33] V. Basili, F. McGarry, R. Pajerski, and M. Zelkowitz, "Lessons learned from 25 years of process improvement: The rise and fall of the NASA software engineering laboratory," in *Proceedings of the 24th International Conference on Software Engineering (ICSE) 2002, Orlando, Florida*, 2002, available from http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/83.88.pdf.

[34] M. Jorgensen and K. Molokeen-Ostvoid, "Reasons for software effort estimation error: Impact of respondent error, information collection approach, and data analysis method," *IEEE Transactions on Software Engineering*, vol. 30, no. 12, December 2004.

[35] C. Kirsopp and M. Shepperd, "Case and feature subset selection in case-based software project effort prediction," in *Proc. of 22nd SGAI International Conference on Knowledge-Based Systems and Applied Artificial Intelligence, Cambridge, UK*, 2002.

[36] A. Miller, *Subset Selection in Regression (second edition)*. Chapman & Hall, 2002.

[37] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions On Knowledge And Data Engineering*, vol. 15, no. 6, pp. 1437– 1447, 2003, available from http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf.