Stable Rankings for Different Effort Models

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

Abstract

There exists a large and growing number of proposed estimation methods but little conclusive evidence ranking one method over another. Prior effort estimation studies suffered from "conclusion instability" where the rankings offered to different methods were not stable across (a) different evaluation criteria; (b) different data sources; or (c) different random selections of that data. This paper reports a study of 158 effort estimation methods on data sets based on COCOMO features. Four "best" methods were detected that were consistency better than the "rest" of the other 154 methods. These rankings of "best" and "rest" methods were stable across (a) three different evaluation criteria applied to (b) multiple data sets from two different sources that were (c) divided into hundreds of randomly selected subsets using four different random seeds. Hence, while there exists no single universal "best" effort estimation method, there appears to exist a small number (four) of most useful methods. This result both complicates and simplifies effort estimation research. The complication is that any future effort estimation analysis should be preceded by a "selection study" that finds the best local estimator. However, the simplification is that such a study need not be labor intensive, at least for COCOMO style data sets.

Index Terms

COCOMO, effort estimation, data mining, evaluation

I. INTRODUCTION

Software effort estimates are often wrong 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 cancelation, 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 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.

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 [9]. All the learners in all these toolkits can be *stacked* by *meta-learning* schemes where the conclusions of one data miner influences the next. There exists one stacking for every ordering of N learners; so five learners can be stacked $5! = 120$ ways and ten learners can be stacked in millions of different ways.

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.eps.

Manuscript received Nov 31, 2009; revised XXX, XXXX.

[1] http://www.r-project.org/

[2] http://www.ailab.si/orange/

Prior attempts to rank and prune different methods have been inconclusive. Shepperd and Kadoda [10] compared the effort models learned from a variant of regression, rule induction, case-based reasoning (CBR), and neural networks. Their results exhibited much *conclusion instability* where the performance results:

- Differed markedly across different data sets;
- Differed markedly when they repeated their runs using different random seeds.

Overall, while no single best method was "best" they found weak evidence that two methods were generally inferior (rule induction and neural nets) [10, p1020].

The genesis of this paper were two observations suggesting that it might be worthwhile revisiting the Shepperd & Kadoda results. Firstly, our data sets are expressed in terms of the COCOMO features [1]. These features were selected by by Boehm (a widely-cited and experienced researcher with much industrial experience; e.g. see [11]) and subsequently tested by a large research and industrial community (since 1985, the annual COCOMO forum has met to debate and review the value of those features). Perhaps, we speculated, conclusion instability might be tamed by the use of better features.

Also, there is an interesting quirk in their experimental results. Estimation methods can *prune* tables of training data:

- CBR prunes away irrelevant *rows*;
- Stepwise regression prunes away *columns* that add little information to the regression equation;
- The Shepperd & Kadoda's rule induction and neural net instantiations have no row/column pruning.

Note that the two methods found to be "worst" by Shepperd & Kadoda had no row or column pruning methods. Pruning data can be useful to remove outliers or "noisy" information (spurious signals unconnected to the target variable). One symptom of outliers and noise is conclusion instability across different data sets and different random samplings. Hence, we wondered if conclusion instability could be tamed via the application of more elaborate row *and* column pruners.

The above observations lead to the experiment reported in this paper. Previously [12], we have built the COSEEKMO effort estimation workbench that supports 158 estimation methods[3]. The methods are defined over COCOMO features, and make extensive use of combinations of different row and column pruners. When we ran this toolkit over our data sets, we found only minor conclusion instability. In fact, a very clear pattern in the results was apparent:

1) Many estimation methods offer very little added value;
2) A small number (four) of estimation methods consistently out-perform the rest;
3) Within the set of four demonstrably useful methods, there is no consistently best estimator.

The methods in COSEEKMO were selected to cover a range of standard and novel theories of effort estimation. Based on these above results, we are now concerned that there is an excess of theoretical elaboration and not enough empirical ranking of supposedly useful methods. For our own work, we are planning to prune away many of the methods in COSEEKMO and we advise other researchers to consider doing the same. For example, in this paper, very simple extensions to decades-old techniques outperformed 97% of all the methods studied here. If those percentages carry over to other effort estimation paradigms and data sets then the implication is that:

- Certain commercial estimation models such as PRICE-S [13], SEER-SEM [14], or SLIM [15] have too many variables and model elaborations;
- Certain seemingly sophisticated estimation methods proposed in the academic literature do not add significant value for the task of effort estimation.

[3]To be precise, COSEEKMO currently supports 15 learners and 8 row/column pre-processors which can be applied to two different sets of internal tuning parameters. In one view, this combination of (15+8*8)*2=158 different estimation generation methods are not really different "methods"; rather it might be more accurate to call them "instances of methods". This paper does not adopt that terminology for the following reason. To any user of our tools, our menagerie of estimation software contains 158 *oracles* that may yield different effort estimates. Hence, in the view adopted by this paper, they are 158 competing methods that must be assessed and (hopefully) pruned to a more management size.

The rest of this paper describes our experiments and discusses their implications on prior and future work.

II. Related Work

There are many methods for effort estimation. The rest of this paper offers brief notes on some of them (with supporting details in the appendix). For a more extensive survey of methods, see [5], [16].

In order to introduce the reader to a range of estimation, in this related work section we will just focus on two methods: regression-based COCOMO and case-based reasoning. This will be followed by a brief tutorial on row and column pruning.

A. Regression-Based COCOMO

Two factors make us prefer COCOMO-based methods:

- *Public domain:* Unlike other effort estimators such as PRICE-S [13], SEER-SEM [14], or SLIM [15], COCOMO is a public domain with published data and baseline results [17].
- *Data availability:* All the information we could access was in the COCOMO-I format [1].

COCOMO is based on linear regression which 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* (predicted minus actual values for the project).

After much research, Boehm advocated the COCOMO-I features shown in Figure 1. He also argued that effort is exponential on KLOC [1]:

$$effort = a \cdot KLOC^b \cdot \prod_i \beta_i \tag{1}$$

where a and b are domain-specific constants and β_i comes from looking up f_i values in Figure 2. When β_i is used in the above equation, they yield estimates in months where one month is 152 hours (and includes development and management hours).

Exponential functions like Equation 1 can be learned via linear regression after a conversion to a linear form:

$$log(effort) = log(a) + b{\cdot}log(KLOC) + \sum_i log(\beta_i) \tag{2}$$

Most our methods transform the data in this way so when collecting performance statistics, the estimates must be unlogged.

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

Fig. 1. The f_i 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. 2. 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.

Local calibration (LC) is a specialized form of linear regression developed by Boehm [1, p526-529] that assumes effort is modeled via the linear form Equation 2. 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 [18].

In 2000, Boehm et al. updated the COCOMO-I model [18]. After the update, numerous features remained the same:

- Effort is assumed to be exponential on model size.
- Boehm et al. still recommends local calibration for tuning generic COCOMO to a local situation.

At the 2005 COCOMO forum, there were discussions about relaxing the access restrictions on the COCOMO-II data. 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. Case-Based-Reasoning

COCOMO's features are both the strongest and weakest part of that method. On the one hand, they have been selected and tested by a large community of academic and industrial researchers led by Boehm. This group meets annually at the COCOMO forums (these are large meetings that have been held annually since 1985). On the other hand, these features may not be available in the databases of a local site. Hence, regardless of the potential value added of using a well-researched feature set, those features may not be available.

An alternative to COCOMO is the case-based reasoning (CBR) approach used by Shepperd [16] and others [19]. CBR accepts data with any set of features. Often, CBR uses a *nearest neighbor* method to make 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 (for details on finding k, see below).

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:

- Local calibration cannot be applied unless projects are described using the COCOMO ontology (Figure 1).
- Linear regression is best applied to data where most of the values for the numeric factors are known.

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

	category-missionplanning	center-2	year-1975	mode-embedded	center-5	project-gro	fg-g	project-X	all	mode-semidetached	category-avionicsmonitoring
year-1980	15 / 43	13 / 62	0 / 75	9 / 50	14 / 63	9 / 52	31 / 87	13 / 63	38 / 93	27 / 80	5 / 63
category-missionplanning		1 / 56	3 / 54	2 / 39	7 / 52	1 / 42	20 / 80	7 / 51	20 / 93	18 / 71	0 / 50
center-2			10 / 64	5 / 53	0 / 76	23 / 37	32 / 85	0 / 75	37 / 93	32 / 74	13 / 54
year-1975				12 / 46	23 / 53	0 / 60	31 / 86	23 / 52	37 / 93	25 / 81	20 / 47
mode-embedded					13 / 47	3 / 41	8 / 93	12 / 47	21 / 93	0 / 90	3 / 48
center-5						0 / 62	33 / 86	38 / 39	39 / 93	23 / 85	17 / 52
project-gro							20 / 83	0 / 61	23 / 93	20 / 72	4 / 49
fg-g								33 / 85	80 / 93	69 / 80	30 / 80
project-X									38 / 93	23 / 84	17 / 51
all										69 / 93	30 / 93
mode-semidetached											24 / 75
category-avionicsmonitoring											

Fig. 3. $NASA93$: intersection / union of examples in different data sets.

III. A Brief Tutorial on Row and Column Pruning

Pruning can be useful since project data collected in one context may not be relevant to another. Kitchenham et al. [20] 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. [17] & Shepperd and Schofield [21] report that row pruning improves estimation accuracy.

Given a table $P * F$ containing one row for P projects described using F features, row and column pruning prune *irrelevant* projects and features. After pruning, the learner executes on a new table $P' * F'$ where $P' \subseteq P$ and $F' \subseteq F$.

Row pruning can be *manual* or *automatic*. In *manual row pruning* (also called "stratification" in the COCOMO literature [18]), an analyst applies their domain knowledge to select project data that is similar to the new project to be estimated. Every other method explored in this study is fully automatic. Such automation enables an exploration of a broad range of options.

Instead, in the sequel, we compare the results from using source subsets to using all the data from one source. Recall that our data sets come from two $sources$: Boehm's COCOMO text [1] and some more recent NASA data. Those sources divide into various *data sets* representing data from different companies, or different project types (see the appendix for details). Minimum size of a subset is 20 instances while a source may contain 93 rows ($NASA93$) or 63 rows ($COC81$).

Automatic row pruning uses algorithmic techniques to select a subset of the projects (rows). NEAREST and LOCOMO [22] 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:

- COCOMIN [23] 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.

	mode-e	lang-ftn	kind-min	lang-mol	kind-max	mode-org
all	28 / 63	24 / 63	21 / 63	20 / 63	31 / 63	23 / 63
mode-e		7 / 45	16 / 33	13 / 35	10 / 49	0 / 51
lang-ftn			6 / 39	0 / 44	16 / 39	12 / 35
kind-min				14 / 27	0 / 52	4 / 40
lang-mol					2 / 49	4 / 39
kind-max						15 / 39

Fig. 4. $COC81$: intersection / union of examples in different data sets.

- WRAPPER and LOCALW are very thorough search algorithms that explore subsets of the features, in no set order. In the worst case, this search is an exhaustive examination of all combinations; i.e this search takes time $O(2^F)$.

IV. EXPERIMENTS

A. Data

This paper is based on 19 data sets from two sources:

- $COC81$[4] comes from Boehm's 1981 text on effort estimation.
- $NASA93$[5] comes from a study funded by the Space Station Freedom Program. $NASA93$ contains data from six different NASA centers including the Jet Propulsion Laboratory.

The data sets represent different subsets of the data: e.g. just the ground systems; just the systems that use FORTRAN; etc. As shown in Figure 3 and Figure 4, there is some overlap between these subsets:

- Occasionally this overlap is quite large; e.g. the 80 records shared by $NASA93$ "all" and the ground systems labeled "fg-g".
- However, in the usual case, the overlap is less than a third (for $COC81$) and a quarter (for $NASA93$) of the number of examples found in the union of both subsets.
- Also, sometimes it is zero; e.g. $NASA93$'s mission planning systems have zero overlap with avionics monitoring.

For more details on this data, see the appendix.

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 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 a prior study [12]. 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, table rows were divided at random into a $Train$ and $Test$ set (and $|Train| + |Test| = P$). COSEEKMO's methods are then applied to the $Train$ set to generate a model which was then used on the $Test$ set. In order to compare this study with our work [12], 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; $\frac{abs(predicted-actual)}{actual}$;

[4] http://promisedata.org/repository/data/coc81/.
[5] http://promisedata.org/repository/data/nasa93/.

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

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

Methods' performance scores were assessed using performance ranks rather than exact values. To illustrate the process of replacing exact values with ranks, consider the following 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

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. This is very important for experiments with effort estimation. In our experiments, we can build thousands to tens of thousands of estimators that exhibit infrequent, but very large outliers. For example, the relative error of an estimate is $RE = \frac{predicted-actual}{actual}$. In our work we have seen data sets generate RE% below 100 then suddenly spike in one instance to over 8000%.

After ranking the performance scores, we applied the Mann-Whitney U test [24]:

- Non-paired tests compare the performance of populations of treatments while paired tests compare performance deltas of two methods running on exactly the same train/test data. Since we are using row/column pruning, paired tests are inappropriate since the underlying data distributions in the train test can vary widely when (e.g.) a method that does use row and/or column pruning is compared to one that does not.
- Mann-Whitney supports very succinct summaries of the results without intricate post-processing. This is a very important requirement for our work since we are comparing 158 methods. Mann-Whitney 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.

Mann-Whitney can be used to report "win", "loss", or "tie" for all pairs of methods $L_i, L_j, i \neq j$:

- "Tie" ; i.e. the ranks are statistically the same;
- "Win" ; i.e. not a tie and the median rank of one method has a lower error than the other;
- "Loss" ; i.e. not a tie and the opposite to a win.

Given L learning methods, the sum of $tie + win + loss$ for any one method is $L - 1$. When discussing discarding a method, an insightful metric is the number of losses. If this is non-zero, then there is a case for discarding that method.

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

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 = ALL + LC	✘	✘	local calibration on all the data from one source
f = M5pW + M5p	✘	✔Kohavi's WRAPPER [25] calling M5p [26], $O(2^F)$	model trees
g = LOCALW + LC	✘	✔Chen's WRAPPER [12] 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. 5. Nine effort estimation methods explored in this paper. F is the number of features (columns) and P is the number of projects (rows).

range into a set of *effort multipliers* and *scale factors* developed by Boehm and are shown in the appendix (Figure 2). Previously, we have queried the utility of these effort multipliers and scale factors [12]. 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 [22]). 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 Nine Methods

This paper focuses on the nine methods $(a, b, c, d, e, f, g, h, i)$ of Figure 5. Four of these, (a, b, c, d), are our preferred methods while the other four comment on premises of some prior publications [27]. Each method may use a column or row pruner or, as with (a, e), no pruning at all.

One way to categorize Figure 5 is by their relationship to accepted practice (as defined in the COCOMO texts [1], [18]). Method a is 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 e refers to an assumption we have explored previously [28], namely: is there some minimal subset of data required to generate adequate effort estimates? Method e uses all possible data from one source and ignores any knowledge of the form "this is a ground systems so we should only train our effort estimator using ground system data".

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

In methods (f, h), the notation M5pW and LsrW denotes a WRAPPER that uses M5p or LSR as its target learner (respectively).

Method g referes to the technique that we argued for in a previous publication [12].

Method i generates estimates by averaging the effort seen in the nearest neighbors to each test instance. Shepperd and Schofield [21] proposed this kind of reasoning for effort estimation from "sparse" data sets [6]. Note that this kind of reasoning does not use Boehm's assumptions about the parametric nature of the relationship between COCOMO attributes and the effort estimate.

For more details on these methods, see the appendix.

V. Results

Figures 6, 7, and 8 show results from 20 repeats of:

- Dividing some subset into $Train$ and $Test$ sets. Note that in the special case of method e, the "subset" was all the data from one source.

[6] A table of data is "sparse" if many of its cells are empty. All our COCOMO data is non-sparse.

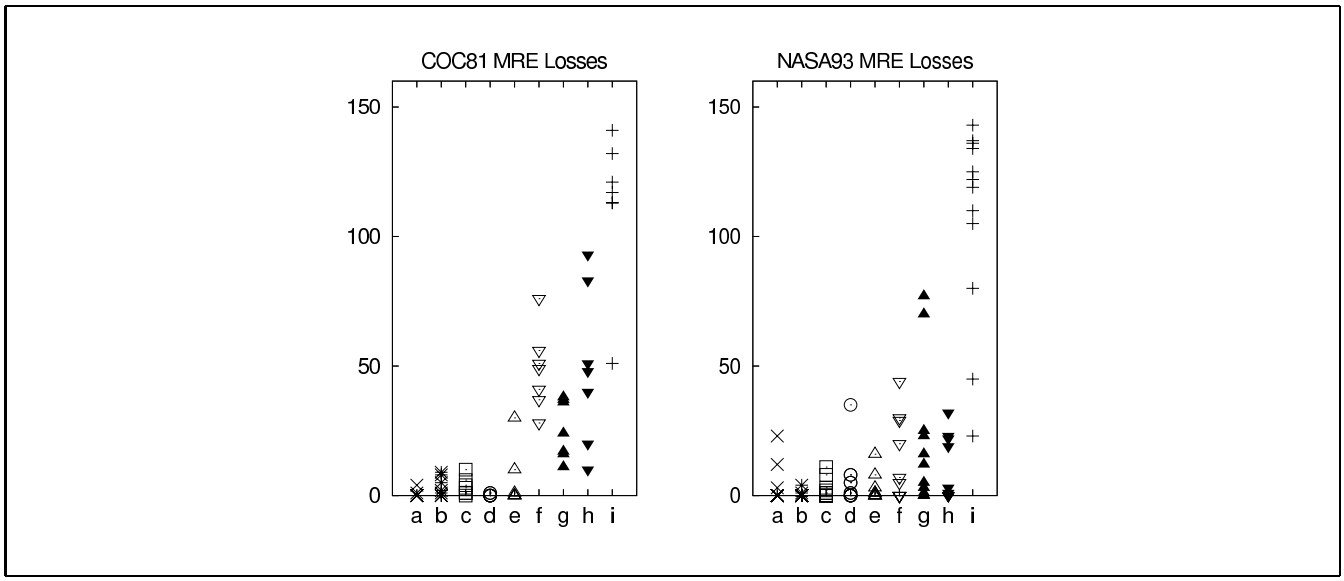

Fig. 6. MRE results. Mann-Whitey (95% confidence). These plots show number of losses of methods $(a, b, c, d, e, f, g, h, i)$ against 158 methods as judged by Mann-Whitney (95% confidence). Each vertical set of marks shows results from 7 subsets of $COC81$ or 12 subsets of $NASA93$.

- 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;
- Reporting the number of times a method losses, where "loss" is determined by a Mann-Whitney U test (95% confidence);

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 5.

In these plots, methods that generate *lower* losses are *better*. For example, the top-left plot of Figure 8 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 these results, conclusion instability due to *changing evaluation criteria* can be detected by comparing results across Figures 6, 7, and 8. Also, conclusion instability due to *changing subsets* can be detected by comparing results across different subsets; either across the two sources, $COC81$ and $NASA9$, or across their subsets selected during cross validation. In order to make this test more thorough, we also conducted the study using different random seeds controlling $Train$ and $Test$ set generation (i.e. the three runs of Figure 8 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.

There are some instabilities in our results. For example, the exemplary performance of methods (a, d) in the top-left plot of Figure 8 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 $(a, b, c, d, e, f, g, h, i)$, the following two results holds across all evaluation criteria and all subsets and all seeds:

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

In our results, there is no single universal "best" method. Nevertheless, out of 158 methods, there are 154 clearly inferior methods. Hence, we recommend ranking methods (a, b, c, d) on all the available

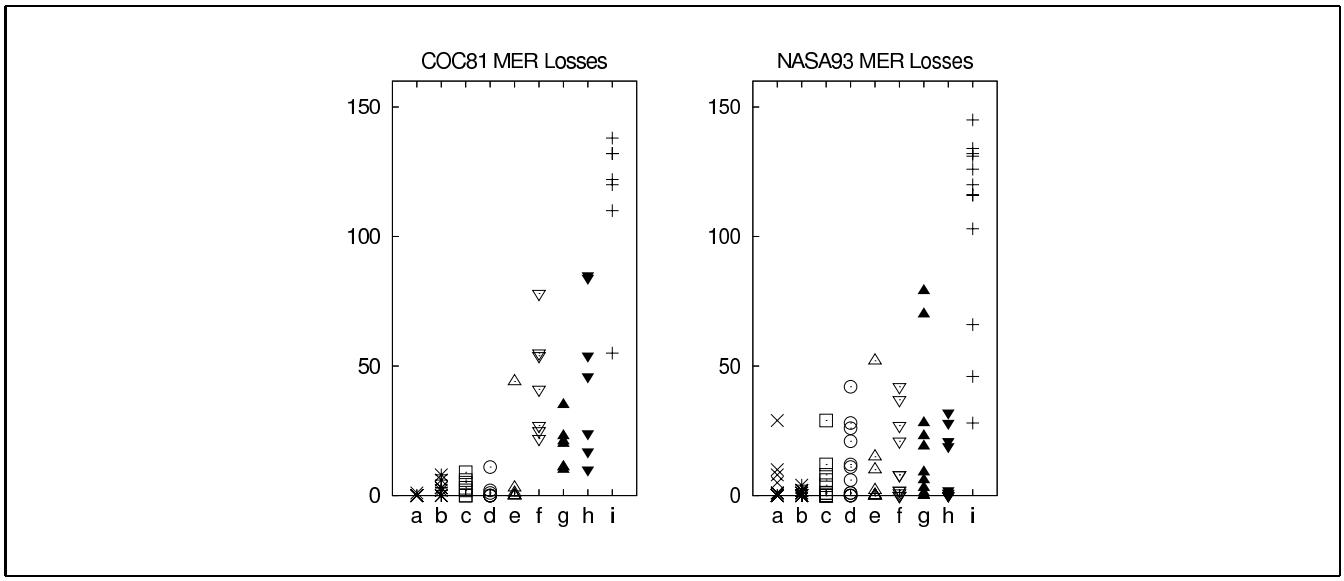

Fig. 7. MER results. Mann-Whitey (95% confidence). Same rig as Figure 6.

historical data, then applying the best ranked method to estimate new projects.

VI. Discussion

With the exception of the final notes on NEAREST, the following discussion notes should not be generalized beyond COCOMO-style data sets.

The methods recommended above are strong endorsement of Boehm's 1981 estimation research. All our "best" 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). This result endorses 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 argue that there is little added value in methods (e, f, g, h, i), at least for our COCOMO-style data sets. 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, many combinations.

The failure of model trees in method f is also interesting. If the model trees of method f had outperformed (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.

Note that method e often performs better than other methods. This is partial evidence that *increasing* the training set size can be as useful as trying smarter algorithms. However, note that method e was always beaten by one of (a, b, c, d). Clearly, the value of training sets specialized to particular subsets can be enhanced by row and column pruning.

Of all the methods in Figure 5, (a, b, c, d) perform the best and i performed the worst. The methods (a, b, c, d) lost less than 20 times in all runs while the NEAREST method used in i lost thousands of times. Why?

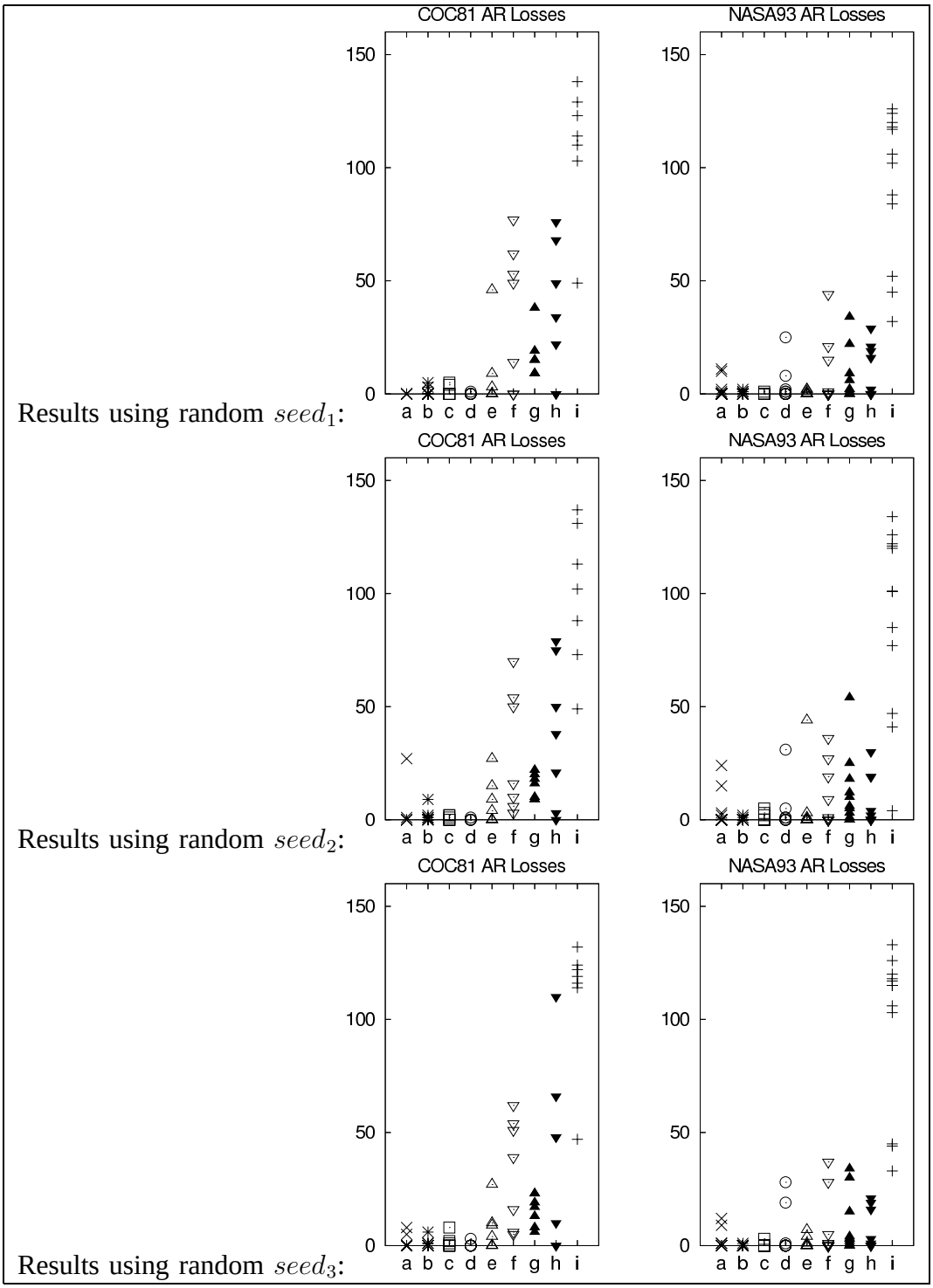

Fig. 8. AR results, repeated three different times with three different random seeds. Same rig as Figure 6.

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. We hypothesize that Boehm's assumptions are useful when processing COCOMO data.

To the above comment, we hasten to add that while NEAREST performed relatively worse *in isolation*, we still believe that nearest neighbor methods like NEAREST are a valuable addition to any effort estimation toolkit:

- Nearest neighbors used *in conjunction with other methods* can be quite powerful. For example, in

the above results, nearest neighbor row pruning proved to be a powerful addition to Boehm's local calibration method.

- As mentioned above, not all domains are described in terms that can be used by the parametric forms like COCOMO. If the available domain data is in another format favorable to some of the other learners, then it is possible that our ranking orders would change. For example, 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 [21]. None of the data used in the study were sparse.

VII. External Validity

This study has several biases, listed below.

Biases in the paradigm: The paper explores model-based methods (e.g. COCOMO) 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. The evaluation of expert-based methods will be a human-intensive study and human subjects may rebel at participating in the lengthy experimental procedure described below. Before comparing any effort estimation methods (be they model-based or expert-based) we must prune the space of model-based methods. For more on expert-based methods, see [4], [17], [21], [29].

Evaluation bias: We showed conclusion stability across three criteria; absolute residual; magnitude of error relative to estimate; or magnitude of relative error. This does not mean that we have shown stability across *all possible* evaluation biases. Other evaluation biases may offer different rankings to our estimation methods.

Sampling bias: Model-based estimation methods use data and so are only useful in organizations that maintain historical data. 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, United States government contracts often 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 is that the data sets used in this study come from two sources: (1) Boehm's 1981 text on Software Engineering [1] and (2) data collected from NASA in the 1980s and 1990s from six different NASA centers including the Jet Propulsion Laboratory (for details on this data, see the appendix). When we show our results to researchers in the field, they ask if two sources is enough to draw valid external conclusions. In reply, we comment that these two sources were repositories that accepted data from a wide range of projects. For example, our NASA data comes from different teams working at geographical locations spread throughout the United States using a variety of programming languages. While some of our data is from flight systems (a particular NASA specialty), most are ground systems and share many of the properties of other terrestrial software (same operating systems, development languages, development practices). Much of NASA's software is written by contractors who service a wide range of clients (not just NASA). These contractors are contractually obliged (ISO-9001) to demonstrate their understanding and usage of current industrial best practices. For this reason, prior research has argued that conclusions from NASA data are relevant to the general software engineering industry. Basili, Zelkowitz, et al. [30], for example, published extensively for decades their conclusions taken from NASA data.

Yet another source of sampling bias is that our conclusions are based only on the data sets studied here. The data used in this study is the largest public domain set of COCOMO-style data available. Also, our data source is as large as the proprietary COCOMO data sets used in prior TSE publications [17].

Biases in the model: This study adopts the COCOMO model for all its work. This decision was forced on us: the COCOMO-style data sets, described in the appendix, are the only public domain data we

could access. Also, all our previous work was based on COCOMO data since our funding body (NASA) makes extensive use of COCOMO. The implications of our work on other estimation frameworks is an open and (as mentioned in the introduction) pressing issue. We strongly urge researchers with access to non-COCOMO data to repeat the kind of row/column pruning analysis described here.

Note that we make no claim that this study explores the entire space of possible effort estimation methods. Indeed, when we review the space of known methods (see Figure 1 in [31]), it is clear that COSEEKMO covers only a small part of that total space. 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 the four we advocate here, we first need a demonstration that it is possible to make stable conclusions regarding the relative merits of different estimation methods. This paper offers such a demonstration.

VIII. CONCLUSION

This paper concludes five years of research that began with the following question; can the new generation of data miners offer better effort estimates than traditional methods?

In other work [23] we detected no improvement using bagging [32] and boosting [33] methods for COCOMO-style data sets. In this work, we have found that one of four methods is always better than another 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.

Unlike Shepperd & Kadoda's results, we were able to find stable conclusions across different data sets, different random number seeds, and even different evaluation criteria. Consequently, we argue for both a *complication* and *simplification* of effort estimation research:

- *Complication #1:* One reason for *not* using COCOMO is that the available data has to be expressed in terms of the COCOMO features, which is problematic if companies have not collected their historical data using the COCOMO ontology. Nevertheless, our results suggest that it may be worth the effort to use COCOMO-style data collection, if only to reduce the instability in the conclusions.
- *Complication #2:* Simply applying one or two methods in a new domain is not enough. In the study reported in this paper, one method out of a set of four was always the best but *that best method was data set-specific*. Therefore, prior to researchers drawing conclusions about aspects of effort estimation properties in a particular context, there should be a *selection study* to rank and prune the available estimators according to the details of a local domain.
- *Simplification:* Fortunately, our results also suggest that such *selection studies* need not be very elaborate. At least for COCOMO-style data, we report that $\frac{154}{158} = 97\%$ of the methods implemented in our COSEEKMO toolkit [12] added little or nothing to Boehm's 1981 regression procedure [1].

Such a selection study could proceed as follows. For COCOMO-style data sets, 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.

For future work, we recommend an investigation of an ambiguity in our results:

- Prior experiments found conclusion *instability* after limited application of row and column pruning to non-COCOMO features.

- Here, we found conclusion *stability* after extensive row and column pruning to COCOMO-style features.

It is hence unclear what removed the conclusion instability. Was it pruning? Or the use of the COCOMO features? To test this, we require a new kind of data set. Given examples expressed in whatever local features are available, those examples should be augmented with COCOMO features. Then, this study should be repeated:

- With and without the local features;
- With and without the COCOMO features;
- With and without pruning;

We would be interested in contacting any industrial group with access to this new kind of data set.

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 data set, see [12].

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 data sets overall. Some have fewer than 20 projects and hence were not used. The justification for using 20 projects or more is offered in [12].

B. Learners Used in This Study

1) 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].

2) Other Learning Methods: See the *Related work* section for notes on learning with linear regression; local calibration; and nearest neighbor methods.

C. Pre-Processors Used in This Study

1) Pre-processing with Row Pruning: The LOCOMO tool [22] 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[7].

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 [34] and Chen & Menzies & Port & Boehm [27] report that column pruning improves effort estimation. Miller's research [35] explains why. Column pruning (a.k.a. feature subset selection [36] or variable subset selection [35]) reduces the deviation of a linear model learned by minimizing least squares error [35]. 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 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 the current subsets are scored by passing them to a *target learner*. 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 [12], [27], we have explored LOCALW for effort estimation.

The above description of WRAPPER should be read as a brief introduction to all the techniques associated with this kind of column pruner. A WRAPPER is powerful tool and can be extensively and usefully customized (e.g. using a hash-table cache to hold the frequently seen combinations; alternative search methods to best-first search; etc). We refer the interested reader to the thorough treatment of the subject found in Miller [35] and Kohavi & Johns [25].

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

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 [23].

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

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

[11] B. Boehm, "A spiral model of software development and enhancement," *Software Engineering Notes*, vol. 11, no. 4, p. 22, 1986.

[12] 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.

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

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

[15] L. Putnam and W. Myers, *Measures for Excellence*. Yourdon Press Computing Series, 1992.
[16] M. Shepperd, "Software project economics: A roadmap," in *International Conference on Software Engineering 2007: Future of Software Engineering*, 2007.
[17] 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.
[18] 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.
[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*, 2007.
[20] 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.
[21] 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.
[22] O. Jalali, "Evaluation bias in effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007.
[23] D. Baker, "A hybrid approach to expert and model-based effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007, available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443.
[24] 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.
[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] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Finding the right data for software cost modeling," *IEEE Software*, Nov 2005.
[28] 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.
[29] M. Jorgensen and K. Molokken-Ostvold, "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.
[30] 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.
[31] 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.
[32] L. Brieman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.
[33] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *JCSS: Journal of Computer and System Sciences*, vol. 55, 1997.
[34] 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.
[35] A. Miller, *Subset Selection in Regression (second edition)*. Chapman & Hall, 2002.
[36] 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.