Automatic Estimation Techniques are Useful?

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

Abstract

Best practices for software effort estimation can include the use of automatic techniques to summarize past data. There exists a large and growing number of techniques. Which are useful? In this study, 158 techniques were applied to some COCOMO data. $\frac{154}{158} = 97\%$ of the variants explored below add little or nothing to a standard linear model (with simple column and row pruning). For example, learners that assume multiple linear models (such as model trees) or no parametric form at all (such as nearest neighbor estimation) perform relatively poorly. Also, elaborate searches did not add value to effort estimation. Exponential time searching for the best subsets of the columns performed no better than near-linear-time column pruning. It is possible that other techniques, not included in the 158 techniques studied here, might perform better. However, based on current evidence, this study concludes that when technique learners are used for effort estimation, a linear model with simple column and row pruning will suffice (at least, for COCOMO-style data).

Index Terms

COCOMO, effort estimation, data mining, evaluation

I. Introduction

Effort estimation is a complex task requiring careful reflection of by skilled personnel over project details. Automatic learning techniques can usefully *augment* (but not *replace*) that reflection process. Best practices for expert-based and model-based estimation are listed in Figure 1. In expert-based estimation, humans generate estimates using a variety of techniques. In model-based approaches, some technique summarizes the past to build a predictive model for current data. Note that some of the automatic model-based techniques offer much support for several of the expert-based best practices:

- #4: Some of the irrelevant and unreliable information can be identified and ignored using row and column pruning.
- #5: Data from previous development tasks can be converted into predictive models.
- #7: Multiple estimates for parts or all of a project can be automatically and quickly generated.
- #10: The uncertainty of an estimate can be checked by running the learners multiple times on subsets of the past data.
- #11: Estimation accuracy can be assessed by reporting the median estimation accuracies across the subset studies.

Five years ago, we began with the following question: can the new generation of learners (e.g., model trees) offer better support for tasks #4,#5,#7,#10,#11) than traditional techniques (e.g., simple linear regression)? There are many such learners. New open source data mining toolkits, such as the R project,[1] Orange,[2] and the WEKA [1] are appearing with increasing frequency.

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.

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

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

According to Jorgensen [4], expert-based best practices include:

1. Evaluating estimation accuracy, but avoiding high evaluation pressure;
2. Avoiding conflicting estimation goals;
3. Asking the estimators to justify and criticize their estimates;
4. Avoiding irrelevant and unreliable estimation information;
5. Using documented data from previous development tasks;
6. Finding estimation experts with relevant domain background;
7. Estimating top-down and bottom-up, independently of each other;
8. Using estimation checklists;
9. Combining estimates from different experts and estimation strategies.
10. Assessing the uncertainty of the estimate;
11. Providing feedback on estimation accuracy; and
12. Providing estimation training opportunities.

According to Boehm([2], [5]), Chulani([6], [7]), Kemerer [8], Stutzke [9], Shepperd [10], our own work([11], [12], [13]), and a recent tutorial at the 2006 International Conference of the International Society of Parametric Analysts [14], best practices for model-based estimation include at least the following:

13. *Reuse regression parameters* learned from prior projects on new projects;
14. *Log-transforms* on costing data before performing *linear regression* to learn log-linear effort models;
15. *Model-tree* learning to generate models for non-linear relationships;
16. *Stratification*, (i.e., given a database of past projects), and a current project to be estimated just learn models from those records from similar projects;
17. *Local calibration*, (i.e., tune a general model to the local data via a small number of special tuning parameters);
18. *Hold-out* experiments for testing the learned effort model [11];
19. Assessment of effort model uncertainty via the *performance deviations* seen during the hold-out experiments of item #17; and
20. *Variable subset selection* techniques for minimizing the size of the learned effort model([12], [13], [15], [16]).

Fig. 1. Categories of effort estimation best practices: expert-based (at top); model-based (in the middle); techniques that combine expert and model-based (at bottom).

In our experience, recorded in this paper, is that many complex variants of data mining for effort estimation add little value over a very small number of very simple techniques. To justify this conclusion, we report here a large experiment with COCOMO data([2], [3]). After much experimentation, we found that of all the 158 variants we explored, a single linear model (with two simple extensions for column and row pruning) performed as well as or better than anything else. For example, learners that assume multiple linear models (such as model trees [17]) or no parametric form at all (such as nearest neighbor estimation [10]) perform relatively poorly. Also, elaborate searches did not add value to effort estimation. Exponential time searching for the best subsets of the columns performed no better than near-linear-time column pruning.

Our result should be welcomed news to software engineers:

- Engineers have a duty to apply accepted best practices to their tasks. With so many learning techniques readily available, software engineers might be concerned that they are generating effort estimates using less-than-best techniques.

- Our results suggest that they need not explore a very wide range of effort estimation techniques. At least for COCOMO-style data, nearly all the variants explored below add little or nothing to a linear model (with simple column and row pruning).

We focus here on COCOMO-style data for three reasons:

- In our work with United States government contractors, we are aware of numerous organizations that use COCOMO data for justifiying their effort estimates for government work.
- Unlike other effort estimators, such as PRICE-S [18], SEER-SEM [19], and SLIM [20], COCOMO is in the *public domain*, with published data and baseline results [7].
- All of the information we could access was in the COCOMO-I format [2].

Researchers or vendors of commercial effort estimation tools might be less sanguine about our results:

- Researchers should take great care when claiming that one particular technique is "the best." Automatic effort estimates have a large variance in their predictions (see our results, below). Some supposedly better technique might appear better simply due to experimental noise. It is, therefore, vital to survey a very large number of techniques since, at least in this study, only a small minority of techniques ($\frac{154}{158} = 97\%$).
- In this paper, very simple extensions to decades-old techniques out-performed 97% of all other techniques studied here. If those percentages carry over to other effort estimation studies, then there may be little empirical justification for elaborate commercial effort estimation tools such as PRICE-S [18], SEER-SEM [19], or SLIM [20].

In all, this paper makes five contributions:

1) Simplifying the task of a commercial software engineer applying automatic learners to effort estimation: for COCOMO-style data, just use a linear model and perhaps some simple column/row pruning.
2) Documenting a baseline result: many automatic techniques add little or no value.
3) Defining a challenge problem: finding better automatic effort estimation tools.
4) Explaining why prior studies have not discovered demonstrably better techniques for effort estimation: see the discussion, below.
5) Offering a cautionary tale to researchers exploring automatic techniques for effort estimation:
   - Many techniques add little or no value to effort estimation.
   - No single technique is always "best."
   - A small minority of simple techniques may be better than anything else.
   - Hence, in the future, it is insufficient to report results from a study of a handful of techniques. Rather, publications should compare their proposed technique to a large number of alternatives.

The rest of this paper is structured as follows. After some preliminary notes on generality, terminology, and techniques, we describe the data, techniques, and experimental procedure used in this study. This is followed by results and discussion sections, some comments on the external validity of this work, conclusions, and future work.

II. Background

A. Preliminaries

Our experiments are based around COCOMO-I data from Boehm's standard text [2] and a subsequent NASA study. Some of this data is quite old, dating back to the 1980s. If we were proposing a specific relationship between, say, 21^{st}-century analyst capability and development effort, then we could not do so from such old data. However, that is not the goal of this paper. Our task is to assess N techniques using some publically available data, and for that task, our data sets will suffice. Future work should repeat this study on other, more recent, data sets.

Note that we make no claim that these our findings necessarily hold over *all* learning techniques. Indeed, such a "try everything" approach may be fundamentally impossible. Learners 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.

Nor do we claim that our results necessarily hold beyond parametric COCOMO-style effort estimation. Parametric estimation assumes that the effort conforms to some pre-determined analytical form (e.g., a single linear relationship). COCOMO-style estimation requires that domain information be encoded using a fixed set of features such as programmer capability and software process complexity. Other estimation techniques, such as case-based reasoning (CBR) ([21], [22]) allow for arbitrary domain information and do not assume any parametric form. Many estimation techniques such as the expert-based estimation techniques surveyed by Jorgensen [4] (and summarized in Figure 1) do not even use models.

That is, it is theoretically possible that some other estimation technique will refute our result. Such a theoretical possibility is not realized until it is obseved experimentally. Our task here is to record that it is surprisingly hard to find new kinds of superior effort estimation techniques. This result, based on public-domain data, is a repeatable baseline experiment. This paper is a success if other researchers refute our findings and demonstrate the superiority of other techniques not explored here.

B. Terminology

1) Variables, Parameters, Factors, or Features: In this study, data from P projects is stored as rows in a table. The columns of that table offer details on that data. Columns have various names including ”variables,” ”parameters,” ”factors,” or ”features.”

This paper uses the term “features” since that is consistent with the technical term used by the machine learning community for column pruning (i.e., feature subset selection [23]).

C. Methods, Models, or Techniques

To simplify our discussion, we elect to call the 158 variants “techniques.” These 158 variants were implemented using our COSEEKMO toolkit [24], which currently supports 15 learners and 8 row and 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 techniques does not comprise different learning “methods”. Rather it might be more accurate to call them “instances of methods.”

Nevertheless, to any business user of our tools, our menagerie of estimation software contains 158 techniques that may yield different effort estimates. Hence, the merits of each technique much be studied separately. That is, our experiments compare 158 competing techniques that must be assessed and (hopefully) pruned to a more management size.

III. Techniques

The following experiments explore numerous techniques from our COSEEKMO effort estimation workbench [24]. For space reasons, we do not describe them all. Also, such an exposition would be superfluous since, as shown below, most are demonstrably inferior to a small set of four techniques.

Therefore the following notes are selective and include the four techniques that we found to be “best,” the one that performed worst, and other techniques that comment on premises of some prior publications [13]. For more details on the techniques discussed here, see the appendix or some recent survey papers([21], [25]).

Also, before exploring the techniques in detail, we offer some elaboration on the expert/model-based estimation techniques discussed above.

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:	database size
increase	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. 2. The f_i features used in this study. From [2]. Most range from 1 to 6, representing "very low" to "extremely high."

A. Expert- and Model-based Estimation

Figure 1 divided effort estimation into expert- and model- based techniques. This separation is not a strict division since some practices fall into both categories: e.g., #4 and #20 are similar, as are #10 and #19. Also, some research actively tries to combine the two approaches:

- Shepperd's CBR tools [10] explore algorithmic techniques for emulating expert analogical reasoning.
- Chulani and Boehm's *Bayesian tuning technique* [7] for regression models allows an algorithm to carefully combine expert judgment with the available data.
- Elsewhere, we have argued for the use of heuristic *rejection rules* to represent expert intuitions on how to rank different effort models [26].

As mentioned in the introduction, one way to view model-based techniques is that they seek algorithms that offer maximal support for:

- Ignoring irrelevancies (#4).
- Learning from past data (#5).
- Quickly generating multiple estimates (#7).
- Assessing estimate uncertainty (#10).
- Commenting on estimation accuracy (#11).

B. Regression-based Techniques (and COCOMO)

The COCOMO effort estimation model([2], [3]) is widely used by, among others, NASA and American software companies working NASA subcontracts. Unlike other effort estimators, such as PRICE-S [18], SEER-SEM [19], or SLIM [20], COCOMO is in the public domain with published data and baseline results [7]. COCOMO-I, defined in 1981 [2], contains 15 parameters, plus lines of code, and COCOMO-II, defined in 2000 [3] ,contains several more. Our work focuses on COCOMO-I since there exists nearly 100 public-domain examples of prior projects in the COCOMO-I format[3] but none in the COCOMO-II format.[4]

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).

[3]`http://promisedata.org/?s=nasa93`

[4]At the 2005 COCOMO forum, there were discussions about relaxing the access restrictions on the nearly 200 examples used to build the COCOMO-II model. To date, those discussions have not progressed.

		1	2	3	4	5	6
upper (increase these to decrease effort)	ACAP	1.46	1.19	1.00	0.86	0.71	
	PCAP	1.42	1.17	1.00	0.86	0.70	
	AEXP	1.29	1.13	1.00	0.91	0.82	
	MODP	1.2	1.10	1.00	0.91	0.82	
	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 (increase these to increase effort)	DATA		0.94	1.00	1.08	1.16	
	TURN		0.87	1.00	1.07	1.15	
	VIRT		0.87	1.00	1.15	1.30	
	STOR			1.00	1.06	1.21	1.56
	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. 3. The COCOMO-I β_i table [2]. For example, the bottom-right cell is saying that if CPLX = 6, then the nominal effort is multiplied by 1.65.

After much research, Boehm advocated the COCOMO-I features shown in Figure 2. He also argued that effort is exponential on KLOC [2]:

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

where a and b are domain-specific constants and β_i comes from looking up f_i values in Figure 3. When β_i is used in the above equation, it yields 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 of our techniques transform the data in this way so that when collecting performance statistics, the estimates must be unlogged.

Local calibration (LC) is a specialized form of linear regression developed by Boehm [2, p526-529] that assumes effort is modeled via the linear-form Equation 2. Linear regression would try to adjust all of 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 technique for calibrating standard COCOMO data [5].

C. Case-Based-Reasoning

COCOMO's features are both the strongest and weakest part of the technique. 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 (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 CBR approach used by Shepperd [21] and others [22]. CBR accepts data with any set of features. Often, CBR uses a *nearest neighbor* technique 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. This is not true for other techniques:

- Local calibration cannot be applied unless projects are described using the COCOMO ontology (Figure 2).
- Linear regression is best applied to data where most of the values for the numeric factors are known. This is not true for other techniques where most of the values for the numeric factors are known.

The drawback of nearest neighbor algorithms is that, sometimes, they 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.

D. Row and Column Pruning

The genesis of this paper was two observations relating to work of Shepperd and Kadoda [27]. We speculated that more extensive row and column pruning might change the results of that paper.

Shepperd and Kadida compared the effort models learned from a variant of regression, rule induction, CBR, and neural networks. Their results exhibited much conclusion instability. In their study, no single technique was demonstrably "best" since the empirical results differed markedly across different data sets and subsets. However, they found weak evidence that two techniques were generally inferior (rule induction and neural nets) [27, p1020].

Our data sets are expressed in terms of the COCOMO features [2]. These features were selected by Boehm (a widely cited researcher with much industrial experience; e.g., see [28]) 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 Shepperd and Kadoda's experimental results. Estimation techniques can *prune* tables of training data:

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

Note that the two techniques found to be "worst" by Shepperd and Kadoda had no row or column pruning techniques. Pruning data can be useful to remove outliers or "noise" 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.

Pruning can be useful since project data collected in one context may not be relevant in 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. [7] and Shepperd and Schofield [10] 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 eliminates *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 [5]), an analyst applies his or her domain knowledge to select project data that is similar to the new project to be estimated. Every other technique 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 [2] and some more-recent NASA data. Those sources can be divided into various *data sets* representing data from different companies, or different project types. The minimum size of a subset is 20 instances, while a source may contain 93 rows ($NASA93$) or 63 rows ($COC81$).

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

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

Automatic row pruning uses algorithmic techniques to select a subset of the projects (rows). NEAREST and LOCOMO [30] are automatic and use nearest neighbor techniques 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 automatic techniques 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 LC technique.

Column pruners fall into two groups:

- COCOMIN [31] 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.
- WRAPPER and LOCALW are much slower 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; that is, this search takes time $O(2^F)$.

For more information on these pruning methods, see the appendix.

IV. Experiments

A. Data

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

$COC81$: 63 records in the COCOMO-I format. Source: [2, p496-497]. Download from `http://promisedata.org/?s=coc81`.

$NASA93$: 93 NASA records in the COCOMO-I format. Download from `http://promisedata.org/?s=nasa93`.

The data sets represent different subsets of the data (for example, just the ground systems, just the systems that use FORTRAN, etc.) 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 [26].

Different subsets and number of subsets used (in parentheses) 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: either semi-detached, embedded, and organic.

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

Year(2): A $NASA93$ term that selects the development years, grouped into units of five (e.g., 1970, 1971, 1972, 1973, and 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 [26].

As shown in Figure 4 and Figure 5, 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; for example, $NASA93$'s mission planning systems have zero overlap with avionics monitoring.

In the study reported below, it will be shown that for this data, a single linear model (augmented, possibly, with simple row and column pruning) does as well as or better than many other techniques. It might be argued that such a conclusion is an artifact of our data: that is, since COCOMO assumes a linear model, then it is hardly surprising that the "best" techniques found by this study also favor linear models. Such an argument is incorrect. It is true that the collection process that generated our data assumed the COCOMO ontology (i.e., it asked only about *acap, pcap*, etc.). However, that collection process did *not* assume an underlying linear model. When this data was collected, domain experts were queried about aspects of their projects and not about the number relationship between, for example, $acap$ and software development effort.

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 [26].

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 and include items such as lines of code (KLOC), schedule pressure (sced), and analyst capability (acap).

technique = 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 [32] calling M5p [33], $O(2^F)$	model trees
g = LOCALW + LC	✘	✔Chen's WRAPPER [26] calling LC, $O(2^F)$	local calibration
h = LsrW + LSR	✘	✔Kohavi's WRAPPER [32] calling LSR, $O(2^F)$	linear regression
i = NEAREST	✔automatic $O(P^2)$	✘	mean effort of nearest neighbors

Fig. 6. Nine effort estimation techniques explored in this paper. *F* is the number of features (columns), and *P* is the number of projects (rows).

To build an effort model, table rows were divided at random into a $Train$ and $Test$ set (and $|Train| + |Test| = P$). COSEEKMO's techniques were 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 [26], we used 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; $\frac{abs(actual - predicted)}{predicted}$.

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.

Techniques' 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 values 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

Once ranked, 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 100 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 [34]:

- Non-paired tests compare the performance of populations of treatments, while paired tests compare performance deltas of two techniques running on exactly the same train or test data. Since we are using row and column pruning, paired tests are inappropriate: the underlying data distributions in the train test can vary widely when, for example, a technique that does use row 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 techniques. 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 a "win," "loss," or "tie" for all pairs of techniques $L_i, L_j, i \neq j$:

- "Tie": The ranks are statistically the same.
- "Win": Not a tie and the median rank of one technique has a lower error than the other.
- "Loss": Not a tie, and the opposite of a win.

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

In summary, our experimental procedure is 20 repetitions of the following:

- Dividing some subset into $Train$ and $Test$ sets. Note that in the special case of technique e, the "subset" was all the data from one source.
- Learning an effort model from the $Train$ set using COSEEKMO's 158 techniques.
- 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 technique loses, where "loss" is determined by a Mann-Whitney U test (95% confidence).

C. 158 Techniques

COSEEKMO's 158 techniques 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 techniques 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 techniques input project features described using the symbolic range *very low* to *extra high*. Some of the techniques map the symbolic range to numerics 1 through 6. Other techniques map the symbolic range into a set of *effort multipliers* and *scale factors* developed by Boehm (see Figure 3). Previously, we have queried the utility of these effort multipliers and scale factors [26]. COSEEKMO, hence, executes its 79 techniques twice: once using Boehm's values, then again using perturbations of those values. So, in all, COSEEKMO contains $2 * 79 = 158$ techniques.

There is insufficient space in this paper to describe the 158 techniques (for full details, see [30]). Such a complete description would be pointless since, as shown below, most of them are beaten by a very small number of techniques that consistently outperform them. 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 Techniques

This paper focuses on the nine techniques $(a, b, c, d, e, f, g, h, i)$ of Figure 6. Four of these (a, b, c, d) are our best performing techniques, while the other four comment on premises of some prior publications [13]. Each technique may use a column or row pruner or, as with a and e, no pruning at all.

One way to categorize Figure 6 is by the technique's relationship to accepted practice, as defined in the COCOMO texts([2], [5]). Technique a is endorsed as best practice in the COCOMO community. The others are our attempts to do better than current established practice using , for example, intricate learning schemes or intelligent data pre-processors.

Technique e refers to an assumption we have explored previously [35], namely, is there some minimal subset of data required to generate adequate effort estimates? Technique e uses all possible data from one source and ignores any knowledge of the form “this is a ground system, so we should only train our effort estimator using ground system data.”

Technique f is an example of a more intricate learning scheme. Standard linear regression assumes that the data can be fitted to a single model. 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 [33].

In techniques f and h, the notations M5pW and LsrW denote a WRAPPER that uses M5p or LSR as its target learner (respectively).

Technique g refers to a technique that we argued for in a previous publication [26].

Technique i generates estimates by averaging the effort seen in the nearest neighbors to each test instance. Shepperd and Schofield [10] proposed this kind of reasoning for effort estimation from “sparse” data sets. [5] 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.

V. Results

Figures 7, 8, and 9 show results from our experimental procedure. Each mark on these plots shows the number of times a technique loses in seven $COC81$ subsets (left plots) and twelve $NASA93$ subsets (right plots). The x-axis shows results from the techniques $(a, b, c, d, e, f, g, h, i)$ described in Figure 6.

In these plots, techniques that generate *lower* losses are *better*. For example, the top-left plot of Figure 9 shows results for ranking techniques applied to $COC81$ using AR. In that plot, all of techniques (a, d) results from the seven $COCO81$ subsets can be seen at $y = losses \approx 0$ [XXXXX - please clarify]. That is, in that plot, these two techniques *never* lose against the other 158 techniques.

In these results, conclusion instability due to *changing evaluation criteria* can be detected by comparing results across Figures 7, 8, and 9. Also, conclusion instability due to *changing subsets* can be detected by comparing results across different subsets - either across the two sources, $COC81$ and $NASA93$, 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 9 that used different random seeds).

For most techniques, there is much conclusion instability in the results. No single technique was ever “best” across all of Figures 7, 8, and 9. Some techniques exhibited a very wide range of behavior. For example, consider the results for technique h (LsrW+LSR) in the right-hand plot of Figure 7. In one subset, it loses less than ten times out of 158. In other subsets, it losses up to nearly 100 times out of 158.

Nevertheless, while the merits of some techniques differ widely across different data subsets and evaluation criteria, a small minority of the techniques exhibit stable behavior. Specifically, the results from a, b, c, and d fall very close to $y = 0$ losses across all of Figures 7, 8, and 9. The significance of this result is discussed below.

There are some instabilities in our results. For example, the exemplary performance of techniques a and d in the top-left plot of Figure 9 does *not* repeat in other plots. For instance, in the $NASA93$ MRE and MER results shown in Figure 7 and Figure 8, technique b loses much less than methods a and d.

However, in terms of number of losses generated by methods a through i, the following two results hold across all evaluation criteria, and all subsets, and all seeds:

1) One member of techniques a through d always performs better (loses less) than all members of techniques e through h. Also, all members of techniques (e, f, g, h) perform better than i.

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

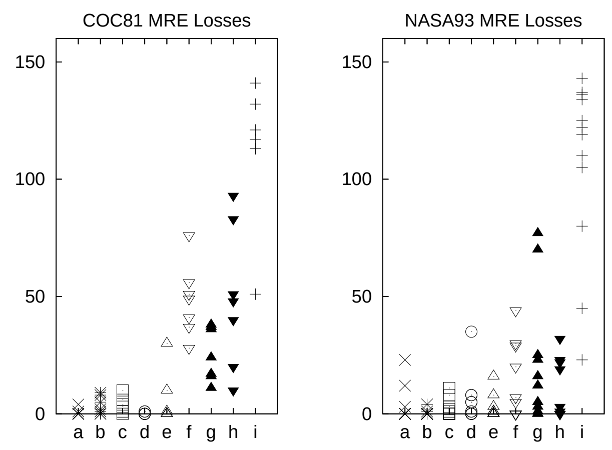

Fig. 7. MRE results. Mann-Whitey (95% confidence). These plots show number of losses of techniques $(a, b, c, d, e, f, g, h, i)$ against 158 techniques as judged by Mann-Whitney (95% confidence). Each vertical set of marks shows results from 7 subsets of $COC81$ or 12 subsets of $NASA93$.

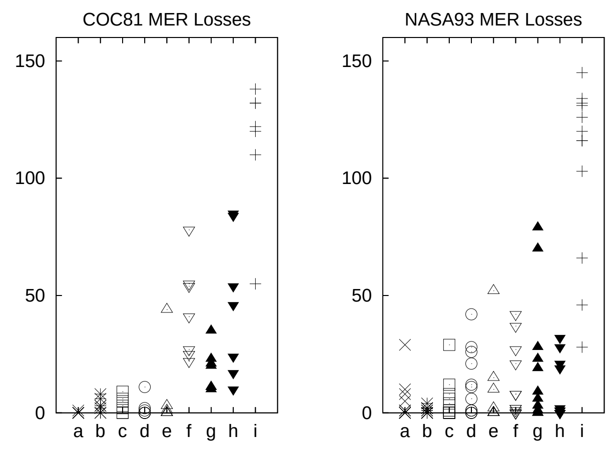

Fig. 8. MER results. Mann-Whitey (95% confidence). Same rig as Figure 7.

2) Compared to 158 techniques, one member of a through d always loses at some rate very close to zero.

In our results, there is no single universal "best" technique. Nevertheless, out of 158 techniques, there are 154 clearly inferior techniques. Hence, we recommend ranking techniques a through d on all the available historical data, then applying the best ranked technique to estimate new projects.

VI. Discussion

The methods recommended above are a strong endorsement of Boehm's 1981 estimation research. All our "best" techniques are based on Boehm's preferred technique for calibrating generic COCOMO models to local data. Technique a is Boehm's LC procedure. 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 ...

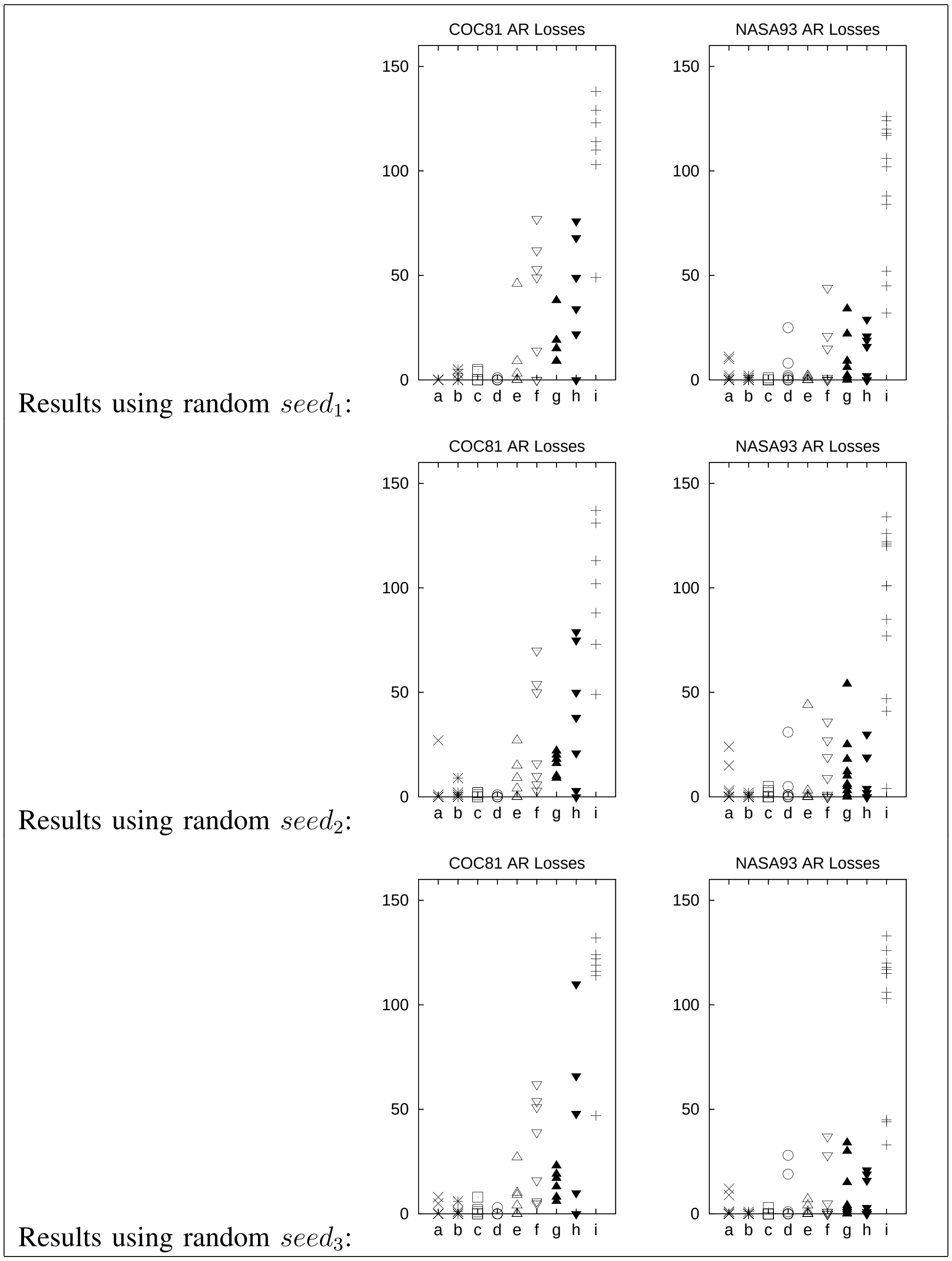

Fig. 9. AR results, repeated three different times with three different random seeds. Same rig as Figure 7.

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 [2].

Our results argue that there is little added value in techniques e through i, at least for our COCOMO-style data sets. This is a useful result because these techniques contain some of our slowest algorithms. For example, the WRAPPER column selection technique used in f, g,amd h is an elaborate heuristic search through, potentially, many combinations.

The failure of model trees in technique f is also interesting. If the model trees of technique f had outperformed a through d, that would have suggested that effort is a multi-parametric phenomenon where, for example, 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 technique e often performed better than other techniques. This is partial evidence that *increasing* the training set size can be as useful as trying smarter algorithms. However, note that technique e was always beaten by one of a through d. Clearly, the value of training sets specialized to particular subsets can be enhanced by row and column pruning.

Of all the techniques in Figure 6, a, b, c, or d perform the best and i performed the worst. The techniques a through d lost less than 20 times in all runs, while the NEAREST technique used in i lost thousands of times. Why?

One distinguishing feature of technique i is the *assumptions* it makes about the domain. The NEAREST neighbor technique i is *assumption-less* since it makes none of the Boehm assumptionsfrom 1981, 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 techniques like NEAREST are a valuable addition to any effort estimation toolkit:

- Nearest neighbors used *in conjunction with other techniques* 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 technique.
- 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 would change. For example, Shepperd and Schofield argue that their CBR techniques, like the NEAREST procedure used in technique i, are better suited to sparse data domains where precise numeric values are *not* available for all factors [10]. None of the data used in the study were sparse.

VII. External Validity

This study has several biases, listed below.

Evaluation bias: We showed conclusion stability across three criteria: absolute residual, magnitude of error relative to estimate, and magnitude of relative error. This does not mean that we have shown stability across *all possible* evaluation biases. Other evaluation biases may rank our estimation techniques differently. Further experimentation is required on this point.

Sampling bias: The automatic techniques described here require data and 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 COCOMO sources: (1) Boehm's 1981 text on Software Engineering [2] and (2) data collected from NASA in the 1980s and 1990s from six different NASA centers, including the Jet Propulsion Laboratory. When we showed our results to researchers in the field, they asked if two sources was enough to draw valid external conclusions. Further, they remark that some of the COCOMO data and all of the NASA93 data comes from one domain (aerospace) and so its conclusions can not be generalized to other domains.

In reply, we comment that our 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 across the United States, using a variety of programming languages. While some of our data is from flight systems (a particular NASA specialty), most come from ground systems, which 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).

Regarding the concern about most of this data's coming from aerospace applications, we comment that the great majority of NASA's software relates to ground systems that store data in standard commercial databases, that visualize data using standard commercial packages, or that generate reports in formats acceptable to standard commercial reporting tools. NASA software is often developed by contractors who 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. [36], 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. In addition, our data source is as large as the proprietary COCOMO data sets used in prior IEEE publications [7].

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 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 and column pruning analysis described here.

VIII. Conclusion

This paper concludes five years of research that began with the following question: can the new generation of learners offer better support for the following five model-based effort estimation tasks?

- Ignoring irrelevancies, e.g. using row and column pruning;
- Learning from past data; e.g. using data mining estimations;
- Quickly generating multiple estimates; e.g. using 10-way cross validation
- Assessing estimate uncertainty; e.g. by recording the variance in the estimates in the 10-ways;
- Commenting on estimation accuracy; e.g. by recoding the median in the estimates in the 10-ways.

We report that, at least for the techniques studied here, many complex variants of data mining for effort estimation add little value over a very small number of very simple techniques. In other work [31], we detected no improvement using bagging [37] and boosting [38] techniques for COCOMO-style data sets. In this work, we have found that one of four techniques is always better than another 154 techniques:

- A single linear model is adequate for the purposes of effort estimation. All of the techniques 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 of the $O(2^F)$ column pruners do worse than near-linear-time column pruning.
- The more intricate techniques, such as model trees, do no better than other techniques.

Our conclusions come from NASA COCOMO data, and we reject arguments that this data (a) is too narrow a source ("just" NASA), (b) is too old to be relevant to current research, or (c) had a simple linear model "hard-wired" into its internal structure:

a) NASA is really an umbrella organization that loosely co-ordinates a very large group of subcontractors who work not only for NASA, but also for other, non-NASA, clients.
b) The goal of this study was to explore some public-domain data source with numerous estimation techniques. Future work should repeat this study on other, more recent, data sets. In the meanwhile, the collection date of this data does not stop conclusions of the form "techniques A,B,C... were applied to data X,Y,Z..., and the following conclusions were observed."

c) When this data was collected, domain experts were queried about aspects of their projects and not about the number relationship between, for example, analyst capability and software development effort. That is, while the data collection process was biased by the COCOMO ontology, it was *not* biased by underlying assumptions of the relationship between (say) lines of code and development effort.

While the specifics of our experiment related to NASA COCOMO data, our general point is that it is possible to find stable rankings for effort estimation techniques. These rankings were stable across:

- Different data sets.
- Many different subsets of that data selected at random.
- Even different evaluation criteria.

Shepperd and Kadoda did not find the conclusion stability reported here. We speculate that making stable conclusions in the field of effort estimation requires careful management of estimation variance. Such variance is clearly present in our results: no single technique was ever "best" in the results of Figures 7 through 9. Indeed, some techniques exhibited a wide range of behavior. For example, compared to 158 other techniques, technique h (LswR+LSR) loses between zero and 75 times, depending on the evaluation criteria and choice of data. However, while some techniques have very unstable performance, others are remarkably stable.

We attribute the stability of our results to:

- An experimental procedure that explored a large number of techniques. Surveying a large number of techniques is important since, at least in this study, only a small minority of techniques are stable.
- The use of of ranked statistical tests (Mann-Whitney) that are not distracted by large outliers.
- The use of row and column pruning to cull irrelevant data that might confuse the modeling process.

Therefore, we hope that this paper inspires a large number of similar studies where researchers try to repeat our results on their data using a large number of techniques, ranked statistical analysis, and row and column pruning.

As to what the practical implications of this study are, we remark that simply applying one or two techniques in a new domain is not enough. In the study reported in this paper, one technique out of a set of four was always the best but *that best technique 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.

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 techniques implemented in our COSEEKMO toolkit [26] added little or nothing to Boehm's 1981 regression procedure [2]. Hence, during a selection study, such techniques could be ignored. Such a selection study need only try the following techniques and the one that does best on historic data (assessed using the Mann-Whitney U test) used to predict new projects:

- Adopt Boehm's three assumptions from 1981 and use LC-based techniques.
- Try LC with none of LOCOMO's row pruning or COCOMIN's column pruning. While some row and column pruning can be useful, elaborate column pruning (requiring an $O(2^F)$ search) is not.

IX. Future Work

We make no claim that this study explores the entire space of possible effort estimation techniques. Indeed, when we review the space of known techniques (see Figure 1 in [39]), it is clear that COSEEKMO covers only a small part of that total space. Readers may know of other effort estimation techniques they believe we should try. Alternatively, readers 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 technique 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 techniques. This paper offers such a demonstration and hopes to encourage much future work in this direction.

More specfically, 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 was it 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 techniques.

APPENDIX

A. *Pre-processing with Row Pruning*

The LOCOMO tool [30] in COSEEKMO is a row pruner that combines a nearest neighbor technique with LC. LOCOMO prunes away all projects except those k nearest 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.[6]

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.

B. *Pre-processing with Column Pruning*

Kirsopp and Schofeld [16] and Chen, Menzies, Port, and Boehm [13] report that column pruning improves effort estimation. Miller's research [15] explains why. Column pruning (a.k.a. feature subset selection [23] or variable subset selection [15]) reduces the deviation of a linear model learned by minimizing least squares error [15]. To see this, consider a linear model with constants β_i that inputs features f_i to predict for y:

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

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 techniques 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

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

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 [32] is a standard best-first search through the space of possible features. At worst, the WRAPPER must search a 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 of 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([13], [26]), 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 a powerful tool and can be extensively and usefully customized (e.g., using a hash-table cache to hold the frequently seen combinations; alternative search techniques to best-first search; etc). We refer the interested reader to the thorough treatment of the subject found in Miller [15] and Kohavi and Johns [32].

Theoretically, WRAPPER's (and LOCALW's) exponential time search is more thorough, and more useful, than simpler techniques 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 *ordered* 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] I. H. Witten and E. Frank, *Data mining. 2nd edition.* Los Altos, US: Morgan Kaufmann, 2005.
[2] B. Boehm, *Software Engineering Economics.* Prentice Hall, 1981.

[3] ——, “Safe and simple software cost analysis,” *IEEE Software*, pp. 14–17, September/October 2000, available from http://www.computer.org/certification/beta/Boehm_Safe.pdf.

[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] 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.

[6] B. B. S. Chulani, B. Clark and B. Steece, “Calibration approach and results of the cocomo ii post-architecture model,” in *Proceceedings ISPA,98*, 1998.

[7] 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.

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

[9] R. Strutzke, *Estimating Software-Intensive Systems: Products, Projects and Processes*. Addison Wesley, 2005.

[10] 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.

[11] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes, “Validation methods for calibrating software effort models,” in *Proceedings, ICSE*, 2005, available from http://menzies.us/pdf/04coconut.pdf.

[12] Z. Chen, T. Menzies, and D. Port, “Feature subset selection can improve software cost estimation,” 2005, available from http://menzies.us/pdf/05/fsscocomo.pdf.

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

[14] “Certified parametric practioner tutorial,” 2006.

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

[16] 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.

[17] R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992, iSBN: 1558602380.

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

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

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

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

[22] 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.

[23] 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.

[24] T. Menzies and J. Richardson, “Making sense of requirements, sooner,” *IEEE Computer*, October 2006, available from http://menzies.us/pdf/06qrre.pdf.

[25] 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.

[26] 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.

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

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

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

[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, available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443.

[32] 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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] L. Brieman, “Bagging predictors,” *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[38] 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.

[39] 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.