The Deviance Problem in Effort Estimation

Tim Menzies
Lane Department of Computer Science,
West Virginia University
tim@menzies.us

Karen Lum, Jairus Hihn
Jet Propulsion Laboratory, Pasadena, California
karen.t.lum@jpl.nasa.gov
jairus.m.hihn@jpl.nasa.gov *

Abstract

Selecting the best data miner for effort estimation is complicated by the large deviations *in estimation model performance. Selection must therefore use other criteria than just mean performance.*

1. Introduction

According to our reading of the literature (e.g. [1, 2]), "best practices" in model-based effort estimation include:

- Local calibration (or LC); i.e. using local data to set two special tuning parameters;
- Stratification; i.e. given a database of past projects, and a current project to be estimated, restrict local calibration to just those records from similar projects.

These two approaches can easily be shown to fall short of what should be expected from a "best practice". If stratification improved performance, then subsets of the data should usually generate better models with lower error rates than models learned from all the data. As shown in Figure 1, this is not necessarily the case. The horizontal lines of Figure 1 show the error rates of models learned from all data of three different sources. The crosses of Figure 1 show error rates seen in models learned from subsets of the data. Ten subsets fall below the lines; i.e. those subsets generate models with *lower* and *better* error rates; But an equal number fall above the lines; i.e. they generate *higher* and *worse* error rates.

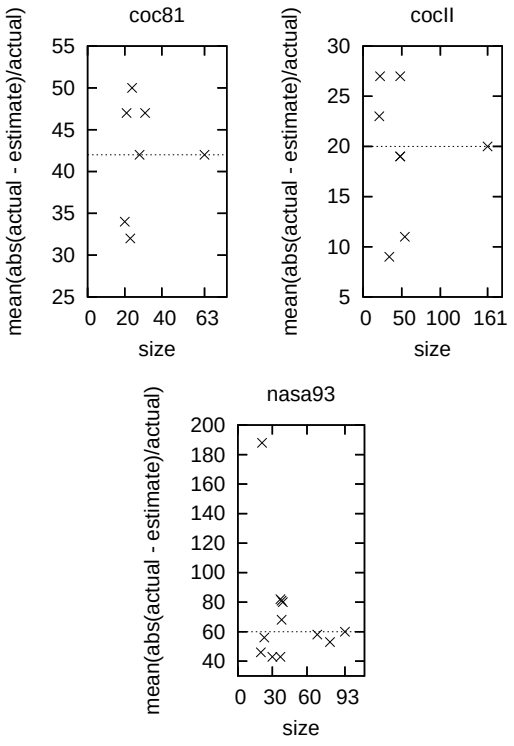

Figure 1. Models were learned from various subsets of three sources: one confidential and two from the PROMISE repository (nasa93 and coc81). The crosses show the $mean\ abs(predicted - actual)/actual$ **found after 30 repeats of:**
- randomizing the order;
- removing 10 instances for testing;
- training on the remaining instances.
Training was conducted using Boehm's COCOMO local calibration method [1, p526-529]. The horizontal lines shows the mean performance using all instances.

*Submitted to the 2006 Promise Workshop.
Download from `http://menzies.us/pdf/06deviations.pdf`.

The research described in this paper was carried out at the University of West Virginia and the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the 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 United States Government.

Similarly, while LC sometimes produces the best models, often it does not. Figure 2 compares standard LC with the multiple data mining algorithms in our COSEEKMO effort estimation workbench. COSEEKMO is a general workbench containing multiple data mining tools, all of which are unleashed on a data set and a best method is selected on a dataset-by-dataset basis. Note the large differences between the performance of models derived using standard methods (the black lines marked with squares) and those derived using data mining (the dashed lines). While standard methods sometimes do well, data mining algorithms yielded models with overall smaller mean error and much smaller standard deviation.

The remainder of this paper describes data mining methods for effort estimation which, as shown in Figure 2 often out-perform standard methods. The most prominent results from this study is that comparing the performance of learners to select a *best* learner is problematic. The variance in the performance seen over these small data sets is very large. For example, in Figure 2, the standard deviations are often much larger than the mean. Such large deviations complicates comparisons across different data sets. Hence, we must compare model performance using some heuristic *rejection rules* that compare more than just mean performance.

Note that this paper is about finding better practices for automatic model-based effort estimation. For a discussion on best practices in manual expert-based estimation, see [5].

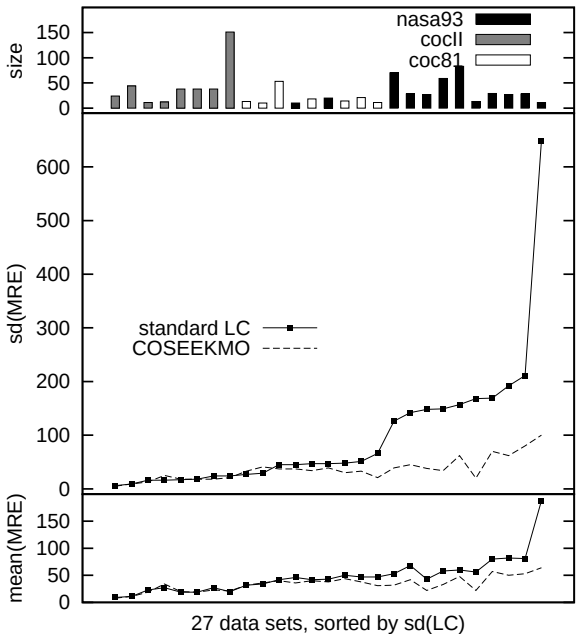

Figure 2. More results from the Figure 1 experiment. The top plot shows the number of projects in 27 subsets of our three data sources. The middle and bottom plots show the standard deviation and mean in performance error. Data subsets are sorted by the error's standard deviation. Effort models were learned via either standard LC or COSEEKMO.

2. Experiment

2.1. Data

The data for this study comes in Boehm's COCOMO format. COCOMO (the COnstructive COst MOdel) was developed in 1981 [1] and extensively revised in 2000 [3]. COCOMO-based estimation assumes effort grows exponentially on size:

$$effort(personmonths) = a * \left(KLOC^b\right) * \left(\prod_j EM_j\right) \quad (1)$$

Here, $KLOC$ is thousands of delivered source instructions. $KLOC$ can be estimated directly or via a *function point estimation*. Also, EM_j is one of the COCOMO *effort multipliers*, such as *cplx* (complexity) or *pcap* (programmer capability). Lastly, a and b are two *local calibration* parameters that can be used to tune the model to local data. The Figure 1 results and the solid line in Figure 2 were generated using a simple regression procedure that fixed EM_j while seeking a and b that minimized model error [1, p526-529]. In a later variant to COCOMO, Boehm extended b to include a set of $scalefactors$ that exponentially affected effort (e.g. reuse) [2].

This study uses the standard *linearization* pre-processor; i.e. Equation 1 is converted to

$$ln(effort) = ln(a) + b * ln(KLOC) + ln(EM_1) + \ldots \quad (2)$$

In practice, this means taking the logarithm of all the numerics in a COCOMO data set.

2.2. Learners

This study applied Boehm's local calibration procedure and a range of learners from the WEKA tool [9] to linearized COCOMO data:

- LSR: least squares regression
- M5P: Quinlan's model-tree learner [8].

Further, the WEKA's WRAPPER feature subset selector [6] was used with LC as the target learner. Starting with the empty set, the WRAPPER adds some combination of

```
01 for Datum ∈ {nasa93,coc81,cocII}
02   for Part ∈ Datum
03     if Data.Part.size ≥ 20   # ignore very small data sets
04     then
05       30 times do
06         Test1 ← Data.Part.any(10)    # note: random selection
07         Train1 ← Data.Part - Test1
08         Subset ← WRAPPER(Train1)
09         for Subset ∈ Subsets
10           Test2 ← Test1.variables(Subset)
11           Train2 ← Train1.variables(Subset)
12           for Learn ∈ {LC, LSR, M5P}
13             print Test(Learn(Train2),Test2)
```

Figure 3. This experiment.

Treatment		Results				
		mean	mre			
$\|Subset\|$	Learn	PRED(30)	mean	sd	(sd/mean)%	correlation
17	e	46	40	34	85	0.93
17	LC	48	38	34	88	0.86
17	LC	50	39	34	87	0.86
16	LC	50	39	34	87	0.85
16	LC	47	43	38	89	0.81
15	LC	47	43	38	88	0.85
15	LC	41	45	42	93	0.88
15	LSR	26	367	3177	863	0.64

Figure 4. Survivors of pair-wise regression using t-tests on MMRE for $Datum$=$coc81$ and $Part$=$embeddedsystems$.

columns (of variables) and asks some learner (in our case, the LC method discussed below) to build an effort model using just those columns. The WRAPPER then grows the set of selected variables and checks if a better model comes from learning over the larger set of variables. The WRAPPER stops when there are no more variables to select or when there has been no significant improvement in the learned model for the last five additions (in which case, those last five additions are deleted). Technically, this is a forward select search with a "stale" parameter set to 5.

We use the WRAPPER since experiments by other researchers strongly suggest that it is superior to many other variable pruning methods. For example, Hall and Holmes [4] compare the WRAPPER to several other variable pruning methods including principal component analysis (PCA- a widely used technique). Column pruning methods can be grouped according to:

- Whether or not they make special use of the target variable in the data set such as "development cost";
- Whether or not pruning uses the target learner.

PCA is unique since it *does not* make special use of the target variable. WRAPPER is also unique, but for different reasons: unlike other pruning methods, it *does* use the target learner as part of its analysis. Hall and Holmes found that PCA was one of the worst performing methods (perhaps because it ignored the target variable) while WRAPPER was the best (since it can exploit its special knowledge of the target learner).

These learners were applied using the Figure 1 procedure, which is detailed in Figure 3. In the following sections, a *treatment* is some combination of $\{Subset, Learn\}$ (selected at line 8 and 12).

2.3. Performance Measures

The results for each treatment were compared using each treatment's MMRE, PRED(30), and correlation. MMRE comes from the magnitude of the relative error, or MRE, the absolute value of the relative error:

$$MRE = |predicted - actual|/actual$$

The mean magnitude of the relative error, or MMRE, is the average percentage of the absolute values of the relative errors over an entire data set. Given T tests, the MMRE is:

$$MMRE = \frac{100}{T}\sum_{i}^{T}\frac{|predicted_i - actual_i|}{actual_i}$$

PRED(N) reports the average percentage of estimates that were within N% of the actual values. Given T tests, then:

$$PRED(N) = \frac{100}{T}\sum_{i}^{T}\begin{cases} 1 \; if \; MRE_i \leq \frac{N}{100} \\ 0 \; otherwise \end{cases}$$

For example, a PRED(30)=50% means that half the estimates are within 30% of the actual.

Another performance measure of a model predicting numeric values is the correlation between predicted and actual values. Correlation ranges from +1 to -1 and a correlation of +1 means that there is a perfect positive linear relationship between variables.

2.4. Cyclic Pair-wise Rejection

The treatments were examined in pairs and if either seemed to perform worse, that one was rejected. This process repeated until no treatment could be shown to be worse than any other. The remaining treatments were called the *survivors* and were printed.

3. Results

Initially, the results were hardly convincing. The deviations in model performance over the 30 repeats of Figure 3

```
function worse(x,y)
  if    statisticallyDifferent(x,y)
  then
       if  error(x) < error(y)                then return y fi  # rule1
       if  error(y) < error(x)                then return x fi  # rule1
  else
       if  correlation(x) <  correlation(y)    then return x fi  # rule2
       if  correlation(y) <  correlation(x)    then return y fi  # rule2

       if  sd(x)/mean(x)<  sd(y)/mean(y)      then return y fi  # rule3
       if  sd(y)/mean(y)<  sd(x)/mean(x)      then return x fi  # rule3

       if  pred(x)         <  pred(y)         then return x fi  # rule4
       if  pred(y)         <  pred(x)         then return y fi  # rule4

       if  |Subset(x)|     <  |Subset(y)|     then return y fi  # rule5
       if  |Subset(y)|     <  |Subset(x)|     then return x fi  # rule5
  fi
  return 0  # if no reason to return true
```

Figure 5. Rejection rules. *Error* **is MMRE.** *Worse*'**s** *statisticallyDifferent* **test compares two MMREs** x **and** y **using a two-tailed t-test at the 95% confidence interval; i.e.** $\frac{|mean(x)-mean(y)|}{\sqrt{(sd(x)^2/(n(x)-1))+sd(y)^2/(n(y)-1)}} > 1.96$

were alarmingly large. Figure 4 shows some results of pair-wise rejection using standard statistical tests on MMRE. Note the last row: a surviving treatment had a standard deviation of 863% (!!) bigger than the mean. Clearly, standard statistical tests are insufficient in this domain to select preferred modeling methods.

In order to generate convincing results, we augmented a pair-wise rejection procedure with the rejection rules of Figure 5. These rules perform well-founded statistical tests and, if those fail, they resort to more heuristic comparisons. For example, rule#5 is a modeling heuristic from Miller [7] who advises that regression models with fewer variables have smaller deviations.

A trace facility was added so analysts could watch and tune the rules. Figure 6 shows such a trace of the rejection rules $rule1 \wedge rule3$ executing on the embedded systems of $coc81$. The last two columns show a justification for rejecting a treatment. For example, the first row was rejected since some other row (39) has a 124% lower MMRE. Row 38 contains the troubling example, with a disturbingly large standard deviation. Observe how the rules rejected this row in favor of the last row, which has a (much) lower coefficient of variation.

The ordering of tests in Figure 5's worse function imposes a rule priority (lower rules can fire only if higher rules fail). Well-founded statistical tests are given higher priority than heuristic tests. Hence, *rule1* is listed first and $rule4$ and $rule5$ are last. $Rule2$ was made higher priority than $rule3$, since that prioritization could still reproduce Boehm's 1981 result for embedded and organic systems.

This prioritization influences how frequently the different rules are applied. As shown in Figure 7, lower priority rules e.g. $rule5$ fire far less frequently than higher priority rules (e.g. $rule1$), since the lower priority rules are only tested when all the higher priority rules have failed.

Treatment			Results					Notes	
treatment number	$\|Subset\|$	Learn	mean PRED(30)	mre mean	mre sd	mre (sd/mean)%	correlation	rejected by treatment	since...
1	13	LC	41	53	49	92	0.72	39	mmre / 124%
2	2	LC	29	72	92	127	0.65	39	mmre / 144%
...	...	...	...	...	...	...	...	....	...
23	17	M5P	33	81	104	127	0.52	39	mmre / 150%
24	17	LC	50	39	34	87	0.86	39	sd/mmre / 102%
25	16	LSR	19	172	356	206	0.3	39	mmre / 176%
...	...	...	...	...	...	...	...	....	...
37	14	LC	42	46	41	89	0.75	39	mmre / 114%
38	15	LSR	26	367	3177	863	0.64	39	sd/mmre / 190%
39	17	e	46	40	34	85	0.93	survivor	

Figure 6. Partial trace of *Rejection Rule 1* **and** *Rule 2* **running on** *coc81*'**s embedded systems. Deleted lines marked with "...".**

	checks for..	COCOMO 81 (nasa93, coc81)	COCOMO II (cocii)
rule1	mmre	59.2%	63.6%
rule2	correlation	34.5%	28.7%
rule3	sd/mmre	5.5%	3.6%
rule4	pred	0.1%	3.0%
rule5	subset size	0.7%	1.0%

Figure 7. Percent frequency of rule firings.

4. Discussion

Effort models should be learned from *similar examples* and these are often in short supply. Generalizing from small samples is inherently under-constrained. Minor changes in that sample can lead to *very large deviations* in the performance of the learned model.

Unless the large deviance problem is addressed, effort estimates will be unreliable (due to large deviances), and rival effort-estimation methods cannot be comparatively assessed. For example, a standard t-test on MMRE accepted all the results of Figure 4, including one with a extremely large deviation. Hence, we propose an extension to standard tests; i.e. cyclic pair-wise rejection using the rules of Figure 5.

One side-effect of the rejection rules is that they can dramatically reduce the deviation in model performance. In the results of Figure 2, $nasa93$'s normalized deviations ($\frac{sd}{mean}\%$) from Boehm's local calibration method had a median of 254% and, in $\frac{10}{12}$ cases, was over 200%. In the results selected using Figure 5 $nasa93$'s normalized deviations have a median value of 122%, and in no case was it over 200%.

The reason for deviance reduction is simple to explain. In 14 of our 28 experiments, the WRAPPER discarded five to six variables on average, and sometimes many more (e.g. in five cases, the surviving models ignored half the variables). Miller noted that models with fewer variables have smaller deviations. Hence, we argue that COSEEKMO reduces deviations by recognizing and removing noisy variables.

Since these rules are the core of our system, they were written with great care. One requirement that we had was that the rules could reproduce at least one historical expert effort estimation study. We recommend the current rule set since this one selects the same models proposed by Boehm in his 1981 study [1] (while an earlier version of the rules could not).

References

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

[2] B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000. Available from `http://www.computer.org/certification/beta/Boehm_Safe.pdf`.

[3] 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.

[4] M. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering*, 15(6):1437– 1447, 2003.

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

[6] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

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

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

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