Feature Subset Selection Can Improve Software Cost Estimation Accuracy

Zhihao Chen
Center for Software Engineering
Univ. of Southern California, USA
zhihaoch@cse.usc.edu

Tim Menzies[*] Dan Port[‡]
[*]Computer Science,
[*]Portland State Univ.;
[‡]Computer Science,
[‡]Univ. of Hawaii
tim@menzies.us
dport@hawaii.edu

Barry Boehm
Center for Software Engineering,
Univ. of Southern California
boehm@cse.usc.edu

ABSTRACT

Cost estimation is important in software development for controlling and planning software risks and schedule. Good estimation models, such as COCOMO, can avoid insufficient resources being allocated to a project. In this study, we find that COCOMO's estimates can be improved via WRAPPER- a feature subset selection method developed by the data mining community. Using data sets from the PROMISE repository, we show WRAPPER significantly and dramatically improves COCOMO's predictive power.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Time Estimation; K.6.3 [**Software Management**]: Software Process

General Terms

Algorithms, Measurement, Economics, Experimentation, Theory,Verification

Keywords

COCOMO, feature subset selection, WRAPPER, M5, LSR

1. INTRODUCTION

Good cost estimation models can significantly help the managers of software projects. With such a good model, project stakeholders can make informed decisions about (e.g.) "buy-or-make", how to manage resources, how to control and plan the project, and how to deliver the project on time, on schedule and on budget. However, if managers use inaccurate models, those "informed" decisions may actually be a recipe for disaster.

For these reasons, we study effort estimation models and how to improve them. To the best of our knowledge, this is the first report of applying *feature subset selection* (FSS) to software effort data. Our results, shown in Figure 1, show that FSS can dramatically improve cost estimation. The x-axis of that figure shows five "cut sets", each containing several attributes (this notion of a "cut set" is explained in §4.2). These cut sets were discovered and ordered using FSS. Each x-axis point shows the effects of removing all the attributes in all the cut sets up to that point. "Cut0" shows the baseline results using all available attributes (i.e. no cuts). In all the five data sets studied in Figure 1, discarding at least the first three cut sets *always* lead to a statistically significant improvement. Sometimes, the improvement was quite striking; e.g. "p02"'s performance rises from just over 20% to nearly 100%.

The core technology used in this study is FSS (feature subset selection) and FSS is an efficient heuristic search through subsets of the available attributes. The goal of this search is to find a subset that gives similar, if not superior, performance than using all the attributes. Equation 1 demonstrates how large that space can be. There are 15 parameters except for SIZE (total 16 parameters in COCOMO 81). A naive search through all possible subsets would have to explore the 32768 sets shown in Equation 1. Assuming 60 seconds are needed for each hold-out experiment (training set and test set are separated) on 60 project instances in that domain, the number of total seconds is shown in Equation 2 and 3.74 years are needed to build the model shown

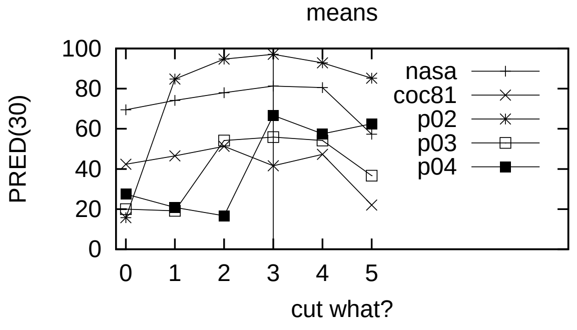

Figure 1: The effect of removing attributes selected by FSS.

in Equation 3.

$$
\begin{aligned}
FS &= \sum_{\alpha=1}^{15} C_{15}^{\alpha} = 32768 & (1)\\
Hold-out &= 60*30*32,768*2 = 117,964,800 & (2)\\
Years &= 117,964,800/60/60/24/3,65 = 3.74 & (3)\\
Hours &= 60*30*6*2/60/60 = 6 & (4)
\end{aligned}
$$

Happily, this study did not take 3.74 years. The FSS methods used in this paper as so efficient that our experiments required only the 6 hours shown in Equation 4.

The rest of this paper explains and expands the Figure 1 result. We begin with some introductory notes on feature subset selection and the COCOMO effort estimation model used in this study. This is followed by some notes on our experimental design and results. T-tests are applied to the results to demonstrate that always in our data sets, removing attributes improves performance *without* increasing the variance in model behavior.

2. COCOMO

In this study, we choose COCOMO, which stands for Constructive Cost Model [1,2]. COCOMO is used for estimating software cost, effort and schedule. COCOMO helps software developers reason about the cost and schedule implications of their software decisions such as software investment decisions; setting project budgets and schedules; negotiating cost, schedule, and performance tradeoffs; making software risk management decisions, and making software improvement decisions.

We use COCOMO since, unlike other models such as Price-S [3] or SLIM [4] or and SEER-SEM [5], it is an *open model* with published data. All detail were published in the text *Software Engineering Economics* [1,2]. There are two version of COCOMO - COCOMO I and COCOMO II. The one we use here, (COCOMO I) was chosen based on the available published data. Our intent was to define a repeatable effort estimation experiment so that others may repeat to refute or improve our results. Hence we base our studies on COCOMO-I data since we know of no large public domain COCOMO-II data sets[1]. In contrast, several COCOMO-I data sets are available in the PROMISE repository [6]:

COCOMO-81: The 63 projects used to define COCOMO-I [1]. This data comes from a variety of domains including engineering, science, financial, etc.

COCOMO NASA: 60 NASA projects from the 1980s and 1990s. Since this data comes from NASA, it is stratified to just aerospace applications.

Project02, 03, 04 : The three largest subsets of COCOMO NASA

Equation 5 shows Boehm's COCOMO I model [1]:

$$months = a * \left(KSLOC^{b}\right) * \left(\prod_{j} EM_j\right) \qquad (5)$$

[1]The COCOMO II data is not published since it was collected on condition of confidentiality with the companies supplying the data. Further research must be conducted in terms of the same conditions.

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
	sced:	schedule constraint
decrease these to decrease 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

Figure 2: COCOMO I effort multipliers.

	very low	low	nominal	high	very high	extra high	productivity range
ACAP	1.46	1.19	1.00	0.86	0.71		2.06
PCAP	1.42.	1.17	1.00	0.86	0.70		2.03
AEXP	1.29	1.13	1.00	0.91	0.82		1.57
MODP	1.24.	1.10	1.00	0.91	0.82		1.51
TOOL	1.24	1.10	1.00	0.91	0.83		1.49
VEXP	1.21	1.10	1.00	0.90			1.34
LEXP	1.14	1.07	1.00	0.95			1.20
SCED	1.23	1.08	1.00	1.04	1.10		
DATA		0.94	1.00	1.08	1.16		-1.23
TURN		0.87	1.00	1.07	1.15		-1.32
VIRT		0.87	1.00	1.15	1.30		-1.49
STOR			1.00	1.06	1.21	1.56	-1.56
TIME			1.00	1.11	1.30	1.66	-1.66
RELY	0.75	0.88	1.00	1.15	1.40		-1.87
CPLX	0.70	0.85	1.00	1.15	1.30	1.65	-2.36

Figure 3: COCOMO I effort multiplier values.

Here, EM_j is one of a set of *effort multipliers* shown in Figure 2. In COCOMO I model, a and b are domain-specific parameters and KSLOC is estimated directly or computed from a function point analysis. In order to use the linear least squares regression, which is the most widely used and the simplest modeling method, we transform COCOMO model into linear model:

$$LN(effort) = b * LN(Size) + LN(EM_1) + LN(EM_2) + \ldots \qquad (6)$$

The values in our data sets come as symbols such as *very low*, *low*, etc. In accordance with Equation 6, these symbols are replaced with the log of the effort multipliers of Figure 3.

COCOMO's performance is often measured in terms of PRED(30). PRED(30) is calculated from the relative error, or RE, which is the relative size of the difference between the actual and estimated value:

$$RE_i = \frac{estimate_i - actual_i}{actual_i}$$

Given a data set of size D, a $Train$ing set of size $(X = |Train|) \leq D$, and a *test* set of size $T = D - |Train|$, then the mean magnitude of the relative error, or MMRE, is the percentage of the absolute values of the relative errors, averaged over the T items in the $Test$ set; i.e.

$$MRE_i = abs(RE_i)$$
$$MMRE_i = \frac{100}{T}\sum_i^T MRE_i$$

PRED(N) reports the average percentage of estimates that were within N% of the actual values:

$$PRED(N) = \frac{100}{T} \sum_{i}^{T} \left\{ \begin{array}{l} 1\, if\ MRE_i \leq \frac{N}{100} \\ 0\ otherwise \end{array} \right.$$

For example, a PRED(30)=50% means that half the estimates are within 30% of the actual. Note that, we report results in terms of PRED(N), not MMRE. This is a pragmatic decision- we have found PRED(N) easier to explain to business users than MMRE. Also, there are more PRED(N) reports in the literature than MMRE. This is perhaps due to the influence of the COCOMO researchers who reported their 1999 landmark study using PRED(N) [7]. Further, we report here PRED(30) results since the major experiments of that 1999 study also used PRED(30).

One of Boehm's original motivation for creating COCOMO was to decrease the number of errors managers make when estimation software projects. At the early software development stage such as investigation and inception phase, the characteristics of software system are unknown; the nature of the processes, team, and personnel experiences are still unclear; the degree of understanding architecture, requirements, and constraints are low. When the software project goes into further development phase, more knowledge of the project is available so predictions better approximate the actual cost.

Strangely, despite the original motivation for COCOMO, very little has been reported on the variance on COCOMO's estimates. Numerous COCOMO calibration studies have been reported [2] and, with only one exception, these studies just report mean results over the training data. The one exception was Chulani et.al.'s report where the min, mean, and max values seen in 15 *holdout studies*[2] [7]. Chulani did not report the standard deviation in their holdout experiments and without that information since they repeated their holdouts only 15 times (a number too small to collect accurate information about standard deviations). Hence, in the sequel, we report variances across a large number (30) of hold-outs studies.

3. FEATURE SUBSET SELECTION

Feature subset selection is the process of identifying the most promising features in a given dataset. Datasets used in practical data mining applications have a large number of features. These data sets often contain several extraneous features which can reduce the efficiency of the learning algorithm. Feature subset selection helps us identify the important attributes and remove redundant ones. If only the most relevant features were to be selected and given to the learning algorithm they can produce smaller theories. This enhances the understanding of the dataset or domain under consideration. Dimensionality reduction also speeds up the learning process. Also, a repeated result in the FSS field (e.g. [8]), is that ignoring features need not degrade the performance of the learned theory.

In this study, we applied the WRAPPER FSS method implemented in WEKA[3] data mining toolkit [9]. When using WRAPPER, a target learner is augmented with a preprocessor that used a heuristic *forward select* search to grow subsets of the available features. At each step in the growth, the target learner is called to determine the performance of the model learned from the current subset. Subset growth is stopped when the growth is *stale*; i.e. after a *MAX_STALE* number of times, adding attributes has not improved the performance.

For example, suppose the set of attributes were {A,B,C, D,E,F,...,Z} and MAX_STALE was 2. WRAPPER starts by selecting one attribute at random (e.g. C) and score its performance.

$$< Selected = \{C\}, Score = 30, Stale = 0 >$$

Next, another randomly selected attribute (e.g. B) is added and scored:

$$< Selected = \{C, B\}, Score = 50, Stale = 0 >$$

Note how the addition of B was not a *stale* addition since it improved the score. However, the addition of the next randomly selected attribute (e.g. E) does *not* improve the score, so *stale* increments:

$$< Selected = \{C, B, E\}, Score = 40, Stale = 1 >$$

Similarly, adding D also fails to improve the score beyond just using {C,B} so scale increments again.

$$< Selected = \{C, B, E, D\}, Score = 42, Stale = 2 >$$

Since MAX_STALE has been reached, WRAPPER would remove from the *selected* set all the attributes implicated in the stale growth ({E,D}). The search would then continue, using other attributes.

One of the major advantages of the WRAPPER approach is that, if some target learner is already implemented, then the WRAPPER is simple to implement. Also, in their comparative evaluation of feature subset selection techniques [8], Hall and Holmes conclude that WRAPPER is the best FSS mechanism, if the data set is not too large. At each step in the heuristic search, WRAPPER makes another call to the target learner. Hence, it many be too slow for large data sets. The data sets used in this study, are small (maximum size: 63 instances) and hence are amenable for WRAPPER.

The Hall and Holmes results were also negative about a widely used technique: principle component analysis (PCA). FSS methods can be grouped according to:

- Whether or not they make special use of the target attribute in the data set such as "development cost";
- Whether or not they use the target learner as part of their FSS analysis.

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

[2]Where $\frac{1}{3}$ of the available data is selected at random for testing, while the remaining data is used for training.

[3]The WEKA data mining toolkit is free, open source, well documented,compatible on many platforms, and easy to install.

COCOMO81			NASA			Project02			Project03			Project04		
instances: 63			instances: 60			instances: 22			instances: 12			instances: 14		
cut	attr.	n	cut	attr.	n	cut	attr.	n	cut	attr.	n	cut	attr.	n
6	loc	10	6	loc	10	6	loc	10	6	loc	10	6	loc	10
5	sced	10	5	acap	10	5	turn	7	5	modp	9	5	modp	9
5	pcap	10	5	time	10	5	lexp	7	4	pcap	1	5	virt	5
5	time	10	5	turn	10	4	time	4	4	rely	1	4	stor	4
5	virt	9	4	stor	8	4	modp	3	4	turn	1	4	turn	4
4	cplx	6	4	vexp	7	3	data	2	3	stor	0	4	cplx	3
4	modp	5	3	data	4	3	tool	2	2	acap	0	3	time	2
4	acap	5	3	aexp	2	3	sced	2	2	vexp	0	3	pcap	2
4	rely	5	2	virt	1	2	rely	1	1	aexp	0	3	rely	2
3	vexp	4	2	pcap	1	2	vexp	1	1	data	0	3	lexp	2
3	tool	3	2	modp	1	2	cplx	1	1	tool	0	2	acap	1
3	data	3	1	cplx	0	1	aexp	0	1	cplx	0	2	data	1
2	aexp	2	1	tool	0	1	pcap	0	1	lexp	0	2	vexp	1
2	stor	2	1	rely	0	1	virt	0	1	sced	0	2	tool	1
1	lexp	1	1	lexp	0	1	acap	0	1	virt	0	2	sced	1
1	turn	0	1	sced	0	1	stor	0	1	time	0	1	aexp	0

Figure 4: Features selected in each data set.

4. EXPERIMENTAL DESIGN

Our experiments were in three phases: *linearization*, followed by *selection* followed by *application*. Each phase was applied to our five data sets.

4.1 Linearization

Linearization was described above: all symbols are replaced with the logarithm of their effort multiplier value.

4.2 Selection

In *selection*, a data set is randomly sub-sampled ten times to generated ten random samples, each containing 90% of the data. WRAPPER is then applied to each of the ten sub-samples and the selected attributes were recorded. "Cut sets" are then formed by clustering the attributes according to how often they were selected.

The *selection* results are shown in Figure 4. In that figure, attributes are sorted but how often they appear (see the results in the *n* column). For example, in COCOMO-81, *loc* (lines of code) was selected in all 10 subsets while *turn* (turnaround time) was never selected.

COCOMO-81's cut sets are shown in the far left-hand-side column (labeled *cut*). *Loc* has a special place in the COCOMO model (recall Equation 6) so it is allocated a cut set all to itself. The other attributes are grouped into five cut sets with the least/most frequently occurring attributes being assigned to cut sets 1/5 (respectively).

4.3 Application

Each selection phase for each data sets resulted in cut sets specialized to that data set. In the *application* phase, the cut sets found for each data set were explored as follows. The attributes in each cut set was removed in the order 1,2,3,4,5 (as defined in Figure 4). After the removal of each attribute set, the remaining data was randomly sampled thirty times to generate a $\frac{2}{3}$rds training set and a $\frac{1}{3}$rds test set. Least squares regression was then applied to the training set to learn a linear model of the form of Equation 6. This model was then applied to the test set. The mean and standard deviation of the model's performance over the 30 subsets was then computed.

Cut What?	cut0	cut1	cut1,2	cut1,2,3	cut1,2,3,4	justLOC
dataset	keep1-6	keep2-6	keep3-6	keep4-6	keep5-6	justLOC
cocomo81	42.4	46.5 *	**51.3** *	41.6	47.3 *	22.1 *
nasa60	69.5	74.2 *	78.0 *	**81.3** *	80.5 *	57.3 *
project04	27.5	20.8 *	16.7 *	**66.7** *	57.5 *	62.5 *
project03	20.0	19.2	54.2 *	**55.8** *	54.2 *	36.7 *
project02	15.7	84.8 *	94.8 *	**97.1** *	92.9 *	85.2 *
MEANs	35	49.1	59.0	**68.5**	66.5	52.8

Figure 5: Mean PRED(30) results

Cut What?	cut0	cut1	cut1,2	cut1,2,3	cut1,2,3,4	justLOC
dataset	keep1-6	keep2-6	keep3-6	keep4-6	keep5-6	justLOC
cocomo81	11.75	9.66	8.17	8.10	8.92	**7.66**
nasa60	8.55	8.21	8.47	7.54	**7.32**	8.58
project04	12.07	12.40	8.78	**6.92**	11.10	10.93
project03	20.13	20.43	18.67	**15.65**	19.79	22.49
project02	23.07	17.47	**15.16**	23.06	23.81	26.06

Figure 6: Pred(30)s standard deviations

5. RESULTS

Figure 5 shows the raw data used to plot the mean PRED(30) results shown in Figure 1. Each column shows the results after removing some combination of the cut sets. The final column in that table shows the effects of removing all attributes except lines of code. In that figure, "*" denotes mean values that are significantly different (at the 99% level) to the *cut*0's mean. In Figure 5, the *maximum mean PRED(30)* values found for any data set are shown underlined and in bold. Note that:

- The *cut*0 means were always lower than means seen in the other cut sets.
- The best (i.e.) largest mean values occurred after removing up to and including cut set 3.

Figure 6 shows the raw data used to plot the standard deviation results shown in Figure 7. As before, each column shows the results after removing some combination of the cut sets and the final column shows the effects of removing all attributes except lines of code. In Figure 6, the *minimum PRED(30) standard deviation* values are shown underlined and in bold. Note that:

- As above, the most best (i.e. smallest) standard de-

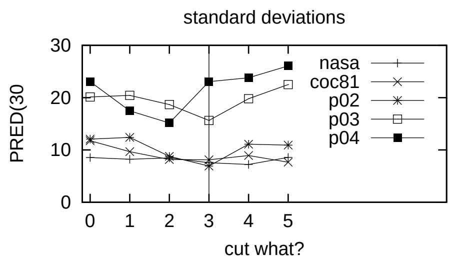

Figure 7: The changes in variance with feature selection.

treatment	Win-Loss	Win	Loss	Tie
cut4	13	16	3	6
cut3	13	16	3	6
cut2	8	14	6	5
cut1	-9	6	15	4
justLOC	-10	6	16	3
cut0	-15	4	19	2

Figure 8: Wins vs Losses

viations occurred after removing up to and including cuts set 3.

The data was studied using t-test comparing the mean PRED(30) results for each data set after removing sets X_0=cut0; X_1=cut1; X_2=cut2; X_3=cut3; X_4=cut4; X_5=cut everything except lines of code. Our experimental rig let us compare results 30 train/test results seen between sets X_i and X_j where $\{i, j\} \in \{0...5\}$ and $j < i$. If the T-tests reject the hypotheses that there is no difference in the difference between the $< X_i, X_j >$ comparison, then that is scored as one "tie". If the comparison does not "tie", then if the means are numerically compared to compute "win"s and "loss"es.

The results, sorted on *win-loss* are shown in Figure 8[4]. Note that culling up to cut set 3 and 4 always generated data sets which, when measured in terms of *win-loss*, score better than other combinations of attributes.

Recall from Figure 5 and Figure 6 that the maximum mean and minimum standard deviations were seen in *cut3*. Hence, even though *cut4* sets scores as well as *cut3* in Figure 8, we recommend cutting up to *cut*3, but not *cut*4.

At *cut*3, Figure 7 shows that the variances were never decreased by removing attributes and sometimes it even decreased. Hence, in summary, the above results are quite positive about feature subset selection for software cost estimation. Not only do the PRED(30) means significantly increase *without* also increasing the standard deviations.

6. DISCUSSION

A curious feature of Figure 4 is that, with the exception of lines of code, different data sets yield different "most important" attributes. Hence, it would be inappropriate to use these results of argue that we should always ignore certain COCOMO attributes. Indeed, our results seem to endorse the opposite view: analysts should collect as many different attributes as possible *before* applying FSS to select the attributes that are most important in a particular domains.

From a practical standpoint, removing cost-drivers variables produces a more efficient cost model turned to an organizations existing project practices. However, if the organization makes a significant change in practices with respect to a removed cost driver variable (for example, going from developing routine business applications to developing real-time, safety-critical systems), there will be a significant risk of estimation inaccuracy. We propose an idea for the extended reduced-parameter model: Extended Effort = Reduced-Parameter Effort * Reduced-But-Current-Significant Parameter. We use the extended reduced-parameter models to answer such a question: what happen if the organization uses the reduced-parameter model without RELY to bid a very-high-Reliability project? Experiments will be conducted to perform to evaluate whether extended models are "better" or not.

7. CONCLUSION AND FUTURE WORK

We have shown that, for the studied data sets, features subset selection always significantly improves mean PRED(30) values *without* increasing variance. Sometimes, that improvement can be quite dramatic; e.g. the project2 results of Figure 1.

In the future, we are planning to do more stratification analysis to better understand the implications of FSS on COCOMO, and validate the idea of the extended reduced-parameter model. Also, we hope to repeat this experiment using other induction methods such as genetic algorithms [10]. Further, we want to experiment on other data sets. Here, we have used COCOMO-I data since we wanted to define a repeatable software cost estimation experiment. While the PROMISE repository contains several COCOMO-I data sets, we know of *no* public domain COCOMO-II data sets.

8. REFERENCES

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

[2] 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.

[3] P. S. L. M. L. NJ, "Your guide to price-s: Estimating cost and schedule of software development and support," 1998.

[4] L. H. Putnam, *Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers, New York.* The Institute of Electrical and Electronics Engineers, Inc., 1980.

[5] D. of USA, "Parametric cost estimating handbook, second edition," 1999.

[6] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases.." School of Information Technology and Engineering, University of Ottawa, Canada, 2005. Available from `http://promise.site.uottawa.ca/SERepository`.

[7] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost

[4]There are 6 feature sets, 5 data sets. Each feature set needs to be compared with other 5 feature sets for each data set, so the total number of comparisons for each feature set is 5x5=25.

models," *IEEE Transactions on Software Engineering*, vol. 25, July/August 1999.

[8] 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.

[9] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[10] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," *IEEE Intelligent Systems*, vol. 13, no. 2, pp. 44–49, 1998.