Finding the Right Data for Software Cost Modeling

Zhihao Chen[†], Tim Menzies[⋆], Dan Port[‡], Barry Boehm[†]
[†]Center for Software Engineering, University. of Southern California
[⋆]Computer Science, Portland State University
[‡]Computer Science, University of Hawaii
zhihaoch@cse.usc.edu;tim@timmenzies.net
dport@hawaii.edu;boehm@cse.usc.edu

Abstract

Strange to say, when building a software cost model, sometimes it is useful to *ignore* much of the available cost data.

Word count: 4000 words+ (5 figures*200)=5000

1 Introduction

Good software cost models can significantly help the managers of software projects. With good models, project stake holders can make informed decisions about (e.g.) how to manage resources, how to control and plan the project, and how to deliver the project on time, on schedule and on budget.

Off-the-shelf "untuned" models have been up to 600% inaccurate in their estimates, e.g. [6]. Hence, the wise manager uses a cost model built from local data. But what data should we use to build a good cost model? Real-world data sets, such as what comes from software engineering projects, often contain noisy or irrelevant or redundant variables.

Prior to getting the data, it is hard to know what parts of the data are most important. However, once a database is available, automatic tools can be used to prune the data back to the most important values. Therefore this paper proposes a change to current practice. Often, cost models are built using all available

		variables					
		1	2	3	...	17	
project	*sub system*	*analyst ability*	*process maturity*	*required reliability*	...	*lines of code*	*development effort(months)*
$proj_1$	$DBapi$	$high$	low	$high$	...	$100,000$	467
$proj_1$	GUI	$high$	low	low	...	$200,000$	847
$proj_2$	$guidance$	$high$	$high$	$nominal$	...	$50,000$	174
...	...	...	...	...	...	...	...

Figure 1: Some project data.

data. Here, we propose that *after* data collection and *before* model building, cost modelers should perform some data pruning experiments. As shown below, such pruning experiments are simple and fast to perform.

Our process starts with a table of historical data divided into columns and rows. Each column is a different *variable* describing some aspect of a software project. Each row shows data from different software sub-systems, so one project can contribute many rows. For example, Figure 1 shows data from 17 variables and rows from three sub-systems from two projects. The data in such a table can be pruned by removing columns or removing rows:

1. In row pruning (also known as *stratification*), rows from related projects are collected together and different cost models are learned from these different subsets.
2. In column pruning (also known as *feature subset selection*), the columns are sorted left-to-right according to their "usefulness"; i.e. how well that column's variable predicts for the target variable (in our case, software development effort). Column pruning then proceeds left to right across the sorted columns, each time removing some less-useful left-hand-side columns. At each step of the pruning, a cost model is learned from the remaining columns.

The benefits of row pruning have been reported previously. For example, Shepperd and Schofeld [14] report experiments with row pruning where estimator performance improved by up to 28% (measured in terms of the "PRED" measure described later in this article), However, and this is the point of this paper, we find that much larger improvements result from pruning *both* rows and columns. For example, the experiments presented here include one data set called $p02$ where estimator performance improved from 15% to 97%.

Further, these large improvements were seen when most of the columns are pruned away. For example, in our experiments, column pruning removed 65% of all columns (on average). Surprisingly, when building a software cost model, it is usually useful to ignore over half of the available cost data.

More importantly, row and column pruning leads to the *largest* improvements in estimator performance in the *smallest* training sets (less than 30 examples). This is a result of tremendous practical significance. Modern software practices change so rapidly that most organizations can't access large databases of relevant project data. Our results suggest that this is not necessarily a problem, provided models are learned via row *and* column pruning.

This rest of this paper describes experiments with our tool for column pruning. This tool is a set of UNIX scripts that use the WRAPPER variable subtraction algorithm from the public-domain WEKA data mining toolkit [15]. The tool is fully automated, runs on a standard LINUX installation, and is available from the authors.

2 Background

This paper applies column pruning methods to cost estimation. These pruning methods were evolved by the data mining community. While these methods have been explored extensively [5, 9], there is very little prior work on column pruning and software cost modeling.

To the best of our knowledge, the only other work on column pruning for cost estimation was a limited experiment by Kirsopp & Shepperd [7]. Like this study, they found that column pruning significantly improves effort estimation. However, their experimental base was much more restrictive than this study (they ran only two data sets while we ran 15). Also, unlike our work, their experiment is not reproducible. The Kirsopp & Shepperd data sets are not public domain while all our COCOMO-I data sets can be downloaded from the PROMISE repository.

An earlier draft of this paper appeared previously in a workshop publication [3]. This paper extends that earlier draft in two ways. Firstly, this paper explores more data than before (double the number of COCOMO-I project analyzed, two new COCOMO-II data sets). Secondly, it includes an expanded discussion on the business implications of column pruning.

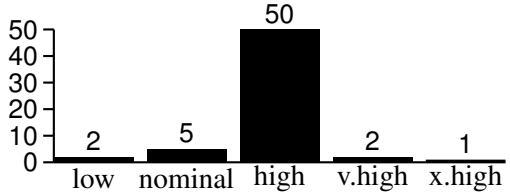

Figure 2: Distribution of software complexity *cplx* within the *na60* data set.

3 Pruning: Why?

The case for pruning rows is quite simple. Software projects are not all the same. For example, real-time safety critical systems are very different to batch financial processors. Given a database of different kinds of software, it is just good sense to divide up the rows into different project types and learn different cost models for each type. Then, in the future, managers can use different cost models depending on what type of software they are developing.

The case for pruning columns is slightly more complex. If a learned cost model uses all the variables in the database then the only way to use that cost model on future sub-systems is to collect information on all those variables. In many business situations, the cost of reaching some goal is a function of how much data you have to collect or monitor along the way. But if the learned model uses only *some* of the variables, then using that model in the future means collecting less data. This would be useful in several scenarios. For example, when monitoring an out-sourced project at a remote site, it is useful to minimize the reporting requirements to just the variables that matter the most. Such a reporting structure reduces the overhead in managing a contract.

From a technical perspective, there are also good reasons to subtract variables:

Under-sampling: The number of possible influences on a project is quite large and, usually, historical data sets on projects for a particular company are quite small. Hence, a variable that is *theoretically* useful may be *practically* useless. For example, Figure 2 shows how in *na60* data set, nearly all those those NASA projects were rated as having a *high* complexity (see Figure 2). Therefore, this data set would not support conclusions about the interaction of, say, extra high complex projects with other variables. A learner would be wise to subtract this variable (and a cost modeling analyst would be wise to suggest to their NASA clients that they refine the local definition of "complexity").

Reducing Variance: Miller offers an extensive survey of column pruning for linear models and regression [12]. That survey includes a very strong argument for column pruning: the variance of a linear model learned by minimizing least squares error decreases as the number of columns in the model. That is, the fewer the columns, the more restrained are the model predictions.

Irrelevancy: Sometimes, modelers are incorrect in their beliefs about what variables effect some outcome. In this case, they might add irrelevant variables to a database. Without column pruning, a cost model learned from that database might contain these irrelevant variables. Anyone trying to use that model in the future would then be forced into excessive data collection.

Noise: Learning a cost estimation model is easier when the learner does not have to struggle with fitting the model to confusing noisy data (i.e. when the data contains spurious signals not associated with variations to projects). Noise can come from many sources such clerical errors or missing data. For example, organizations that only build word processors may have little data on software projects with high reliability requirements.

Correlated variables: If multiple variables are tightly correlated, then using all of them will diminish the likelihood that either variable attains significance. A repeated result in data mining is that pruning away some of the correlated variables increases the effectiveness of the learned model (the reasons for this are subtle and vary according to which particular learner is being used [5]).

4 Pruning: How?

The column pruning method used in this study was is called the "WRAPPER" [8]. The WRAPPER selects combinations of columns and asks some learner to build a cost model using just those columns. The WRAPPER then grows the selected columns and checks if a better model comes from learning over the larger set of columns.

The WRAPPER stops when there are no more columns to select, or 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 speaking, this is a forward select search with a "stale" parameter set to 5.

WRAPPER is thorough but, theoretically, it is quite slow since (in the worst case), it has to explore all subsets of the available columns. However, all the data sets in this study are quite small. Our experiments only required around 20

minutes per data set.

We use the WRAPPER since other experiments by other researchers strongly suggest that it is superior to many other column pruning methods. For example, Hall and Holmes [5] compare the WRAPPER to several other column pruning methods including principle 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 attribute in the data set such as "development cost";
- Whether or not they use the target learner as part of their pruning.

PCA is unique since it *does not* make special use of the target attribute. 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 attribute) while WRAPPER was the best (since it can exploit its special knowledge of the target learner).

5 Cost Modeling with COCOMO

Before pruning data, we first need to understand how cost models might use that data. This study uses COCOMO for our cost modeling. COCOMO stands for Constructive Cost Model [1, 2]. 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. One advantage of COCOMO (and this is why we use it) is that unlike other many other costing models such as SLIM or SEER, COCOMO is an *open model* with numerous published data [1, 2].

There are two versions of COCOMO: COCOMO-I and COCOMO-II. In going from the 1981 COCOMO-I model [1] to the 2000 COCOMO-II model [2], one parameter, "Turnaround Time", was dropped to reflect the almost-universal use of interactive software development. Another parameter, "Modern Programming Practices", was dropped in favor of a more general "Process Maturity" parameter. But several more parameters were added to reflect the subsequently-experienced influences of such factors as "Development for Reuse", "Multisite Development", "Architecture and Risk Resolution", and "Team Cohesion". The COCOMO-II

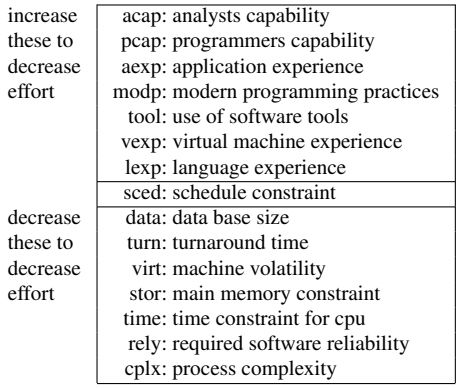

Figure 3: COCOMO I effort multipliers.

book [2] also provides capabilities and guidelines for an organization to add new parameters, reflecting their particular situations.

COCOMO measures effort in calendar months where one month is 152 hours (and includes development and management hours). The core intuition behind COCOMO-based estimation is that as systems grow in size, the effort required to create them grows exponentially. More specifically, Equation 1 shows the COCOMO I model [1]:

$$months = a * \left(KSLOC^b\right) * \left(\prod_j EM_j\right) \quad (1)$$

Here, EM_i is one of 15 *effort multipliers* such as *cplx* (complexity) or *pcap* (programmer capability). The COCOMO-I effort multipliers are shown defined in Figure 3 and their numeric values are shown in Figure 4. In COCOMO-II, the number of effort multipliers changed from 15 to 17.

In the COCOMO-I model, a and b are domain-specific variables and KSLOC (thousands of lines of non-commented source code) is estimated directly or computed from a function point analysis. In COCOMO-II, b was expanded to include *scale factors*:

$$b = 0.91 + \sum_j SF_j$$

where SF_j is one of five *scale factors* that exponentially influence effort. Examples of scale factors include *pmat* (process maturity) or *resl* (attempts to resolve project risks).

	very low	low	nominal	high	very high	extra high
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	1.24.	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		
SCED	1.23	1.08	1.00	1.04	1.10	
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

Figure 4: COCOMO-I effort multiplier values.

A standard method for assessing COCOMO performance is 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:

$$RE = \frac{estimate - actual}{actual}$$

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 sets. PRED(N) reports the average percentage of estimates that were within N% of the actual values. If a data set has R rows, then:

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

For example, a PRED(30)=50% means that half the estimates are within 30% of the actual. 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) [4]. Further, we report here PRED(30) results since the major experiments of that 1999 study also used PRED(30). The results for PRED(25) are similar with those for PRED(30) except that the accuracies for PRED(30) is a little better than those for PRED(25).

In order to use the linear least squares regression, which is the most widely used and the simplest modeling method, it is common to transform COCOMO model into linear model by taking the logs of Equation 1.

$$LN(effort) = b * LN(Size) + LN(EM_1) + LN(EM_2) + \dots \quad (2)$$

Note that, if Equation 2 is used, then *before* computing PRED(N), the estimated effort has to be converted back from a logarithm.

6 Case Study Data

This study uses datasets in both COCOMO-I and COCOMO-II format:

- The *cii0* data set was used to build the COCOMO-II model.
- The *cii4* data set is also in the COCOMO-II format and includes the 72 projects from *cii0* developed after 1990, plus 47 new projects.

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. In contrast, several COCOMO-I data sets are available in the PROMISE repository [13]:

- *Coci* comes from the COCOMO-I text [1] and includes data from a variety of domains including engineering, science, financial, etc.
- *na60* comes from 20 years of NASA projects and is recorded using the COCOMO-I variables.

The *na60* data can be stratified into the following data sets:

- *c01,c02,c03* store data from three different NASA geographical locations;
- *p02,p03,p04* stores data from three different NASA projects;
- *t02,t03* stores data from two tasks such as ground data receiving and flight guidance.

For reasons of confidentiality, the exact details of those centers, tasks, and projects cannot be disclosed. The other centers, projects, and tasks from *na60* were not included in cX, pX, or tX for a variety of pragmatic reasons (e.g. suspicious repeated entries suggesting data entry errors, too few examples for generalization, etc).

Of these data sets, *coci* describes projects from before 1982; *cii4* contain data from the most recent projects; and the NASA data sets (*na60, pX,cX,tX*) described projects newer than *coci* and before *cii4*. Also, the COCOMO-I data sets ($coci, na60, c01, c02, c03, p02, p03, p04, t02, t03, call, pall, tall$) have the 15 columns of Figure 3 COCOMO-II data sets ($cii0, cii4$) have 24 columns.

For experimental purposes, we group the above as follows:

- *call* combines all the center data; i.e. $call = c1 + c2 + c3$;
- *tall* combines all the task data; i.e. $tall = t02 + t03$;
- *pall* combines all the project data; i.e. $pall = p02 + p03$;

7 Experimental Method

Having described COCOMO, the WRAPPER, and our data sets, we can now describe how an analyst can use them all together to find better cost models.

Column pruning using the WRAPPER was discussed above. Recall that the WRAPPER sorts columns are sorted left-to-right according how well that column's variable predicts for the target variable (in our case, software development effort). Column pruning then proceeds left to right across the sorted columns, each time removing some less-useful left-hand-side columns. At each step of the pruning, a cost model is learned from the remaining columns. Column pruning stops when removing more columns does not improve the best result seen so far.

To find the relative value of each column, we ran the WRAPPER ten times (each time using a randomly selected 90% of the rows). The value of a column was then set to the number of times the WRAPPER selected that column.

Once the WRAPPER ordered the columns, we randomized the order of the rows and starting column pruning. To ensure statistical validity, randomization (followed by column pruning) was repeated 30 times.

For each repeat, at each stage of the pruning, the lowest value column was removed. The rows in the remaining columns were then divided into training and test sets (each time using a randomly selected 67% of the rows for the training set). A cost model was learned (using linear regression) from the training set and then assessed, using PRED, on the test set.

Once the 30 repeats were completed, a best model was selected by looking at the mean and standard deviation of model performance at each pruning step. The best model was the one that t-tests confirmed out-performed all the other states of the column pruning. The mean value of that model over the 30 repeats was then reported.

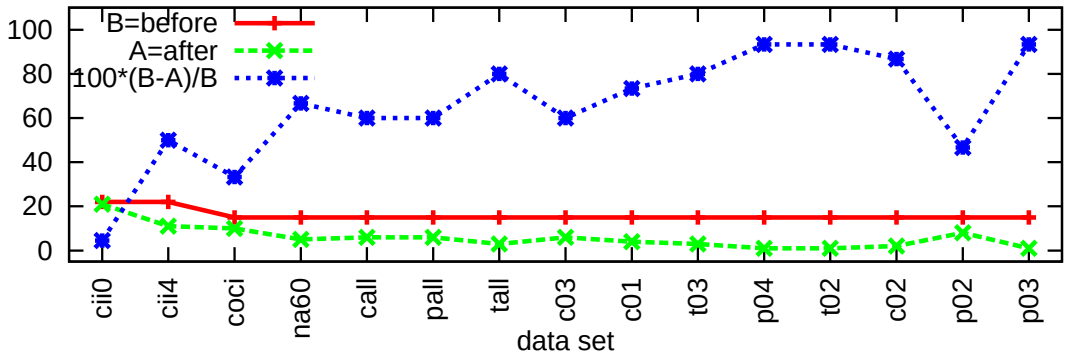

Figure 5: Number of columns.

The above process us fully automated using our own Unix scripts which control a JAVA data mining library called the WEKA [15]. The WEKA comes complete with a linear regression learner and an implementation of the WRAPPER. The whole system is available from the authors.

8 Results

The results of our column pruning are shown in Figure 5. The red and green lines of show the number of columns in our data sets before and after pruning. The data sets are on the x-axis. The $before$ values are 22 for the COCOMO-II data sets ($cii0$ and $cii4$) while the $before$ values for the COCOMO-I data sets are all 15. The blue line shows the percentage of the $before$ columns removed by pruning. For example, very little was pruned from cii while most of the columns were pruned from $p03$. On average, over 65% of the columns were pruned. Sometimes, the pruning was quite heavy with over 80% of the columns pruned away.

A concern with such large scale pruning is that the resulting models would be somehow sub-standard. This proved not to the case. The PRED(30) results associated with the pruned data sets are shown in Figure 6. This figure shows mean values in 30 experiments where the learned model was tested on rows not seen during training. Note that pruning *always* improved PRED.

The red lines on Figure 6 show the mean PRED(30) seen in 30 trials using all the columns. These are the baseline results for learning cost models *before* applying any pruning method. The green lines show the best mean PRED(30)

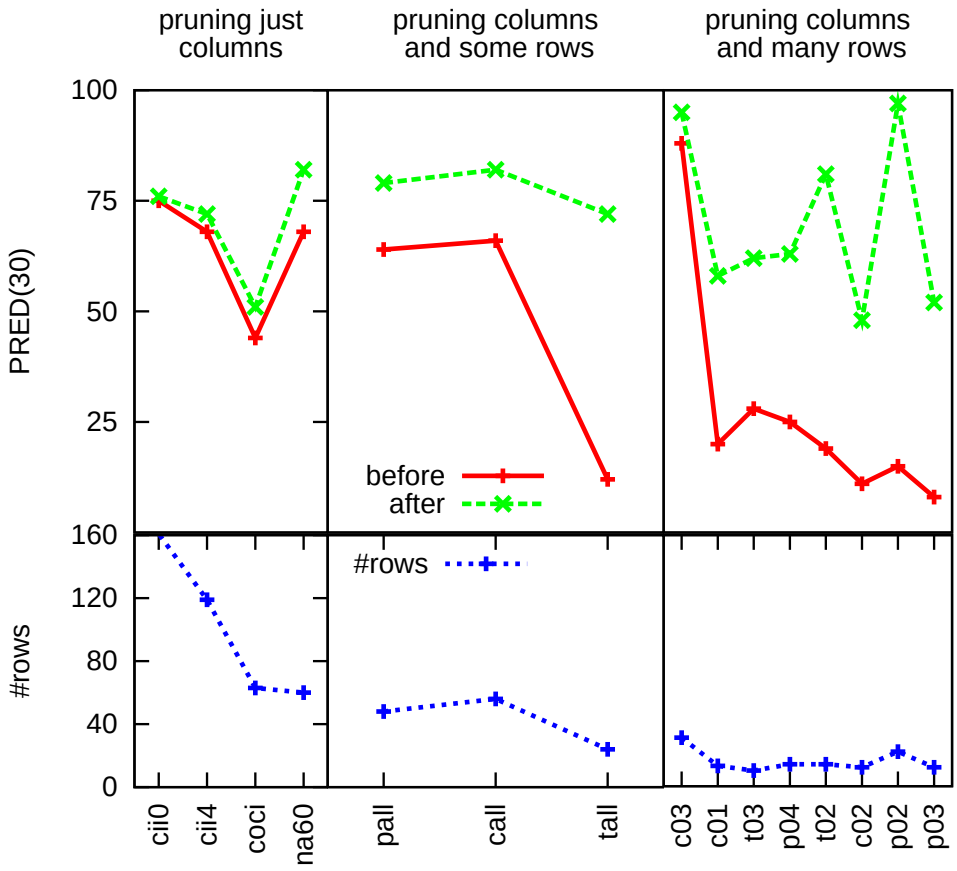

Figure 6: Effects on PRED of different pruning methods. For space reasons, only the PRED(30) results are shown. A longer version of this paper, in preparation, includes PRED(20) and PRED(25) results.

seen *after* automatic column pruning. The difference between the red and the green lines is the improvement produced by pruning.

The data sets are sorted by pruning method into three plots. Within each plot, the data sets are sorted left-to-right in increasing value of $\frac{after-before}{before}$. The x-axis shows the names of the data sets used in these studies. The blue lines show the number of rows in each data set.

The plots of Figure 6 have three labels: "pruning just columns", "pruning

columns and some rows", and "pruning columns and many rows". The data sets are sorted into three labels according to their stratification. The left-hand-side results, labeled "pruning just columns", come from the largest data sets that combing project information form many sources (i.e. $cii0, cii4, coc1, na60$). These data sets are not divided up into data from similar sources. Hence, there is no row pruning used on these data sets.

The right-hand-side results, labeled "pruning columns and many rows", come from eight data sets that have been heavily stratified into just specific NASA centers (i.e. $c01, c02, c03$); or specific NASA projects (i.e. $p01, p02, p03$); or specific NASA software tasks (i.e. $t02, to3$).

The middle results, labeled "pruning columns and some rows", show data sets that have been somewhat stratified. The data sets in this group combine together data from either:

- All the NASA centers (i.e. $call$);
- Or all the NASA projects (i.e. $pall$);
- Or all the NASA tasks (i.e. $tall$).

This middle group samples a point half-way between the unstratified data sets on the left and the heavily stratified data sets on the right.

Three are three features of Figure 6:

1. Pruning *always* improved estimation effectiveness. That is, in all our case studies, it was *always* useful to ignore a portion of the available data.
2. Column pruning by itself can offer some improvements to PRED. However, column pruning combined with row pruning can result in dramatic improvements of effort estimation.
3. With one exception ($c03$), the general trend across the three graphs is clear: as data set size *shrinks*, the improvement *increases*. That is, pruning is most important when dealing with small data sets.

9 When Not to Prune

While column pruning is clearly useful, sometimes it cannot or should not be applied. Firstly, our variable subtraction methods require an historical database of projects. If there is *no* such database, then our column pruning techniques won't work.

Secondly, even if a historical database exists and our techniques suggest pruning variable X, then it may still be important to ignore that advice. If a cost

model *ignores* certain effects that business users believe are important, then those users may *not trust* that model. In that case, even if a variable has no noticeable impact on predictions, it should be left in the model. By leaving such variables in a model we are acknowledging that, in many domains, expert business users hold in their head more knowledge that what may be available in historical databases. Suppose that there is some rarely occurring combination of factors which leads to a major productivity improvement. Even if there is little data on some situation, it still should be included in the model. For example, even though some studies have shown that reduced-parameter function point counting rules are equally good in most situations [2], COCOMO II supports the full International Function Point Users' Group (IFPUG) set of parameters due to their wide usage and acceptance in the IFPUG community.

Thirdly, another reason not to prune variables is that you still might need them. For example, the experiments shown above often subtract over half of the attributes in a COCOMO-I model while (usually) improve effort estimation. However, suppose that a business decision has to be made using some of the pruned variables. The reduced model has no information on those subtracted variables so a business user would have to resort to other information for making their decisions.

Hence, we propose using column pruning with some care. If there is no historical data for learning specialized data sets, then managers should use the general background knowledge within COCOMO. The 1981 regression co-efficients of COCOMO-I or the updated co-efficients of COCOMO-II [2] are the best general-purpose indicators we can currently offer for cost estimation. Management decisions can use that public knowledge to make software process decisions. For example, according to the coefficients on the COCOMO-II *pmat* variable, the increase in cost between a CMM3 and CMM4 project contain N lines of code is $N^{3.13}/N^{1.56}$. With this estimate in hand, a business user could then make their own assessment about the cost of increased software process maturity vs the benefits of that increase.

If historical data from the local site is available, then managers could tune the general COCOMO background knowledge by adjusting the coefficients within the COCOMO equations. COCOMO-I and COCOMO-II contain with several *local calibration* variables that can quickly tune a model to local project data. Our experience has been that 10 to 20 projects are adequate to achieve such tunings [10].

Local calibration is a simple tuning method that is supported by many tools[1].

[1] e.g. `http://sunset.usc.edu/available_tools/index.html`

Currently, our toolkit methods requires more effort (i.e. some UNIX scripting) than local calibration. Figure 6 suggests that the extra effort may well be worthwhile, particularly when building models from a handful of projects. Also, we have found that it is easier to extrapolated costs from old projects to new projects with reduced variable sets [11]. Nevertheless, column pruning is *not* appropriate when there are business reasons to use all available variables (e.g. the three reasons described above).

10 Conclusion

The specific goal of this paper was to encourage more column pruning in cost modeling, particularly when dealing with very small data sets. The improvements seen in Figure 6 seem quite impressive. Row and column and pruning, in combination, are very useful for cost modeling- particularly when dealing with small data sets. Hence, we propose a change to current practice. Cost models *should not* be built using all available data. Rather, *after* data collection and *before* model building, cost modelers perform some data pruning experiments.

The more general goal of our work is to encourage repeatable, refutable, and improvable experiments in software engineering. To that end, as much as possible, we use public domain tools and public domain data sets. Hence, this paper uses a open source cost model (COCOMO) and, as much as possible, publicly available data. All the COCOMO-I data sets used in this study can be downloaded from the PROMISE repository [13]. We urge other researchers to produce more results based on open source models and data sets.

Acknowledgments

The advice of the anonymous reviewers helped to clarify an earlier draft of this paper. Helen Burgess offered invaluable editorial assistance.

References

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

[2] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[3] Zhihoa Chen, Tim Menzies, and Dan Port. Feature subset selection can improves software cost estimation. In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies/pdf/05/fsscocomo.pdf`.

[4] S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineering*, 25(4), July/August 1999.

[5] M.A. 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.

[6] C.F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.

[7] 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.

[8] R. Kohavi, D. Sommerfield, and J. Dougherty. Data minining using mlc++: A machine learning library in c++. In *Tools with AI 1996*, 1996.

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

[10] 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`.

[11] Tim Menzies, Zhihao Chen, Dan Port, and Jairus Hihn. Simple software cost estimation: Safe or unsafe? In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies.us/pdf/05safewhen.pdf`.

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

[13] J. Sayyad Shirabad and T.J. 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`.

[14] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12), November 1997. Available from `http://www.utdallas.edu/~rbanker/SE_XII.pdf`.

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