# Feature Subset Selection Methods for COCOMO Based Software Effort Estimation

Daniel Baker,Tim Menzies *Member, IEEE*

***Abstract*— This paper demonstrates the results of feature subset selection methods for the COCOMO model. Two algorithms are tested, COCOMIN and COCOMOST. COCOMIN was designed as a 'strawman' approach to feature subset selection. COCOMOST was designed to efficiently evaluate every subset in the attribute space instead of using heuristics because the domain of software cost estimation is sufficiently data starved to warrant exhaustive methods.**

## I. Introduction

CURRENTLY cost estimation is accomplished using models such as COCOMO [**?**]. These models predict the development cost for a new software project based on past project data. For an accurate prediction the training data needs quantity, quality, and relevance to the new project. Unfortunately this is difficult in practice and estimates are often made using inadequate training data. Consequently, these models are plagued with problems including highly inaccurate predictions and the variance problem. It has been shown that the variance can be reduced by feature subset selection methods that discard irrelevant, redundant, noisy, and unreliable attributes. This paper explores some of these attribute pruning techniques.

## II. Background

### A. COCOMO

The case study material for this paper uses COCOMO-format data. COCOMO (the COnstructive COst MOdel) was originally developed by Barry Boehm in 1981 [?] and was extensively revised in 2000 [**?**]. The core intuition behind COCOMO-based estimation is that as a program grows in size, the development effort grows exponentially. More specifically:

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

Here, $KLOC$ is thousands of delivered source instructions. $KLOC$ can be estimated directly or via a *function point estimation*. Function points are a product of five defined data components (inputs, outputs, inquiries, files, external interfaces) and 14 weighted environment characteristics (data comm, performance, reusability, etc.) [?], [**?**]. A 1,000 line Cobol program would typically implement about 14 function points, while a 1,000-line C program would implement about seven[1].

In Equation **??**, $EM_j$ is one of *effort multipliers* such as *cplx* (complexity) or *pcap* (programmer capability). In order to model the effects of $EM_j$ on development effort, Boehm proposed reusing numeric values which he generated via regression on historical data for each value of $EM_i$.

In practice, effort data forms exponential distributions. Appendix B describes methods for using such distributions in effort modeling.

Note that in COCOMO 81, Boehm identified three common types of software: *embedded, semi-detached*, and *organic*. Each has their own characteristic "$a$" and "$b$" (see Figure **??**). COCOMO-II ignores these distinctions. This study used data sets in both the COCOMO 81 and COCOMO-II format. For more on the differences between COCOMO 81 and COCOMO-II, see Appendix A.

### B. Data

The software project data we used in this study came from two sources (see Figure **??**). $Coc81$ is the original COCOMO data used by Boehm to calibrate COCOMO 81. $Nasa93$ comes from a NASA-wide database recorded in the COCOMO 81 format. This data has been in the public domain for several years but few have been aware of it. It can now be found on-line in several places including the PROMISE (Predictor Models in Software Engineering) web site[2]. $Nasa93$ was originally collected to create a NASA-tuned version of COCOMO, funded by the Space Station Freedom Program. $Nasa93$ contains data from six NASA centers including the Jet Propulsion Laboratory. Hence, it covers a very wide range of software domains, development processes, languages, and complexity as well as fundamental differences in culture and business practices between each center. All of these factors contribute to the large variances observed in this data set.

When the $nasa93$ data was collected, it was required that there be multiple interviewers with one person leading the interview and one or two others recording and checking documentation. Each data point was cross-checked with either official records or via independent subjective inputs from other project personnel who fulfilled various roles on the project. After the data was translated into the COCOMO 81 format, the data was reviewed with those who originally provided the data. Once sufficient data existed the data was analyzed

Dan Baker is with the Lane Department of Computer Science, West Virginia University and can be reached at danielryanbaker@gmail.com

Dr. Menzies is with the Lane Department of Computer Science, West Virginia University and can be reached at tim@menzies.us

This research was conducted with funds from the NASA Software Assurance Research Program led by the NASA IV&V Facility.

Manuscript received January 1, 2006; revised XXX YY, 200Z.

[1] http://www.qsm.com/FPGearing.html

[2] http://promise.site.uottawa.ca/SERepository/ and http://unbox.org/wisp/trunk/cocomo/data.

| Mode | a | b | notes |
|---|---|---|---|
| Organic | 3.2 | 1.05 | projects from relatively small software teams develop software in a highly familiar, in-house environment. |
| Embedded | 2.8 | 1.2 | projects operating within (is embedded in) a strongly coupled complex of hardware, software, regulations, and operational procedures. |
| Semi-Detached | 3.0 | 1.12 | An intermediary mode between organic and embedded. |

Fig. 1. Standard COCOMO 81 development modes.

to identify outliers and the data values were re-verified with the development teams once again if deemed necessary. This typically required from two to four trips to each NASA center. All of the supporting information was placed in binders, which we still on occasion reference even today.

Using Boehm's COCOMO-I "local calibration" the $nasa93$ data has been shown to contain large deviations due to the wide variety of projects in that data set, and *not* poor data collection. Our belief is that $nasa93$ was collected using methods equal to, or better, than standard industrial practice. If so, then industrial data would suffer from deviations equal to or lager than those seen in the $nasa93$ data.

### C. Performance Measures

The performance of models generating continuous output can be assessed in many ways, including PRED(30), MMRE, correlation, etc. PRED(30) is a measure calculated from the relative error, or RE, which is the relative size of the difference between the actual and estimated value. One way to view these measures is to say that training data contains records with variables $1, 2, 3, .., N$ and performance measures add additional new variables $N + 1, N + 2, ....$

The magnitude of the relative error, or MRE, is the absolute value of that 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. MMRE results were shown in Figure **??** in the *mean% average test error* column. Given $T$ tests, MMRE is calculated as follows:

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

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

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

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

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. Appendix C shows how to calculate correlation.

All these performance measures (correlation, MMRE and PRED) address subtly different issues. Overall, PRED measures how *well* an effort model performs while MMRE measures *poor* performance. A single large mistake can skew the MMREs and not effect the PREDs. Shepperd and Schofield comment that:

> MMRE is fairly conservative with a bias against overestimates while PRED(30) will identify those prediction systems that are generally accurate but occasionally wildly inaccurate [**?**, p736].

Since they measure different aspects of model performance, COSEEKMO uses combinations of PRED, MMRE, and correlation (using the methods described later in this paper).

## III. COCOMIN

FOLLOWING the slow but steady success of COSEEKMO, it becomes necessary to find a similarly accurate algorithm which is fast enough to be used by business users in the real world. We decided to begin the search for such an algorithm using a minimal approach, that is, a feature subset selection algorithm that used a greedy search to minimize evaluations of the attribute space. This algorithm was named COCOMIN because it was a minimal approach to feature subset selection on the COCOMO model. Pseudocode for COCOMIN is in Figure **??**. The code can be found at `http://unbox.org/wisp/trunk/cocomost/cocomin`. We decided to rank the attributes using correlation and to use the ranked results as the order to grow the attribute set in the case of a forward select greedy search, or prune the set in the case of a backward elimination greedy search. At each step the attribute set is evaluated with the MMRE, Pred(30), deviation, and correlation after using COCOMO regression with the attribute set in question. If the change is considered an improvement then it is kept and the search continues, otherwise it stops. We introduced a stale variable that could be set to allow for the search to continue even without improvement.

Early results showed that this algorithm wasn't very accurate. Also, by noticing that the results tended to improve with higher stale values, we reasoned that the attribute space needed more exploration. Finally, with a dataset as small as the COC81 and NASA93 datasets, the importance of heuristics to prune the state space explosion were examined. Next we decided to build an efficient algorithm that evaluated all 32,768 attribute combinations.

## IV. COCOMOST

THE COCOMOST algorithm, as outlined in Figure **??**, uses feature subset selection to prune irrelevant, redundant, noisy, and unreliable attributes from the COCOMO model. The code begins at `http://unbox.org/wisp/trunk/cocomost/cocomost`. It executes a complete search of the attribute space, evaluating attribute sets using local calibration. Thus, it is a "wrapper" attribute selection technique instead of a "filter" because it evaluates using the

| | |
|---|---|
| Data sources | *Coc81:*63 records in the COCOMO 81 format. Source: [?, p496-497].<br>Download from `http://unbox.org/wisp/trunk/cocomo/data/coc81modeTypeLangType.csv`.<br>*Nasa93:* 93 NASA records in the COCOMO 81 format.<br>Download from `http://unbox.org/wisp/trunk/cocomo/data/nasa93.csv`.<br>*CocII:* 161 records in the COCOMO II format from the COCOMO consortium (co-ordinated by USC). This data is not in the public domain. |
| Data subsets | *All:* selects all records from a particular source; e.g. ”coc81_all”.<br>*Category:* is a NASA-specific designation selecting the type of project; e.g. avionics, data capture, etc.<br>*Dev:* indicates the development methodology; e.g. div.waterfall.<br>*DevEnd:* shows the last year of the software project.<br>*Fg:* selects either “$f$” (flight) or “$g$” (ground) software.<br>*Kind:* selects records relating to the development platform; max= mainframe and mic= microprocessor.<br>*Lang:* selects records about different development languages.<br>*Project* and *center*: $nasa93$ designations selecting records relating to where the software was built and the name of the project.<br>*Mode=e:*<br>selects records relating to the *embedded* COCOMO 81 development mode. The different COCOMO 81 development models were described in Figure **??**.<br>*Mode=o:*<br>selects COCOMO 81 *organic* mode records.<br>*Mode=sd:*<br>selects COCOMO 81 *semi-detached* mode records.<br>*Org:* is a $cocII$ designation showing what organization provided the data.<br>*Size:* is a $cocII$ specific designation grouping the records into (e.g.) those around 100KLOC.<br>*Type:* selects different $coc81$ designations and include “bus” (for business application) or “sys” (for system software).<br>*Year:* is a $nasa93$ term that selects the development years, grouped into units of five; e.g. 1970,1971,1972,1973,1974 are labeled “1970”. |

Fig. 2. Data sets (top) and parts (bottom) of the data used in this study.

target learner [**?**]. However, it shares several of the advantages of a filter. Unlike most wrappers, COCOMOST is fast enough to search the entire attribute space instead of using heuristics to limit the state space explosion. This introduces the vulnerability that every attribute beyond the 15 used in this study will double COCOMOST's execution time.

```
for attribute in rankedList
  newSet = bestSet + attribute
  if (search==backward)
    tmpSet=inverse(newSet)
  else
    tmpSet = newSet
  oldScore = newScore
  results = LC(train, train, tmpSet)
  newScore = results.eval
  if (newScore better than oldScore)
    bestSet = newSet
    staleCount = stale
  else
    newScore = oldScore
    staleCount--
  if (staleCount < 1)
    exit for
next attribute
if (search==backward)
  bestSet=inverse(bestSet)
return bestSet
```

Fig. 3. The cocomin algorithm which builds a subset of attributes using a greedy search guided by an attribute ranking.

```
attributes = null
bestMMRE = LC(train, attributes)
bestAttributes = null
for attributes in 2^15
  newMMRE = LC(train, attributes)
  if newMMRE > bestMMRE
      bestMMRE = newMMRE
      bestAttributes = attributes
return bestAttributes
```

Fig. 4. Cocomost performs a complete search over the attribute space and evaluates the attribute sets using the target learner: COCOMO-based local calibration.

## V. Experimental Design

Considering the small amount of data available, we decided to use n-fold cross validation to compare the COCOMIN and COCOMOST learners against standard COCOMO-based local calibration as shown in Figure **??**. The code for this experiment can be found at `http://unbox.org/wisp/trunk/cocomost/compete`.

## VI. Quartile Charts of Performance Deltas

One method we used to assess the experimental results was to create quartile charts of the performance deltas of

```
for data  in dataSets
 num = countRecords(data)
  for i  in 1  to num
   test       =  data.i
   train      =  data - test
   attr       =  (all COCOMO 81 attributes)
   results    =  LC(test, train, attr)
   print variables and results
   attr       =  cocomost(train)
   results    =  LC(test, train, attr)
   print variables and results
   for attr in COCOMO_81_attributes
    rank correlation using LC(train, train, attr)
    order += sorted list of ranked attrs
   for search in "forward backward"
    for stale in "0 1 2 4 8 16"
     for eval in "mmre sd_mre pred30 corr"
      attr=cocomin(test,train,search,stale,eval,order)
      results = LC(test, train, attrs)
      print variables and results
```

Fig. 5. This experiment benchmarks standard COCOMO-based local calibration against local calibration that uses cocomost to perform feature subset selection, or local calibration that uses each customization of cocomin for feature subset selection.

the learners. We prefer quartile charts of performance deltas to other summarization methods for M*N studies. Firstly, they offer a very succinct summary of a large number of experiments. Secondly, they are a *non-parametric* display; i.e. they make no assumptions about the underling distribution. Standard practice in data mining is to compare the mean performance of different methods using t-test [**?**]. T-tests are a *parametric method* that assume that the underling population distribution is a Gaussian. Recent results suggest that there are many statistical issues left to explore regarding how to best to apply those t-tests for summarizing M*N-way studies [**?**].

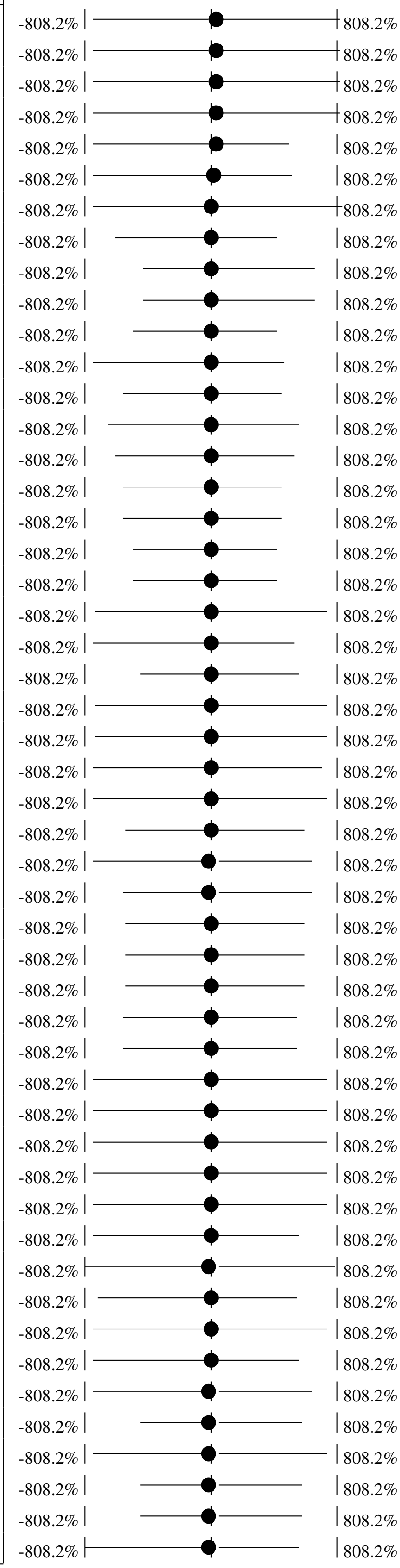
The performance deltas were computed using simple subtraction, defined as follows. A *positive performance delta* for method X means that method X has out-performed some other method in *one* comparison. Using performance deltas, we say that the best method is the one that generates the largest performance deltas over *all* comparisons.

The performance deltas for each method were sorted and displayed as *quartile charts*. To generate these charts, the performance deltas for some $method$ were sorted to find the lowest and highest quartile as well as the median value; e.g.

$$\overbrace{\underbrace{-59}_{min}, -19, -19}^{lowest\ quartile}, -16, -14, -10, \underbrace{-10}_{median}, 5, 14, 39, \overbrace{42, 62, \underbrace{69}_{max}}^{highest\ quartile}$$

In a quartile chart, the upper and lower quartiles are marked with black lines; the median is marked with a black dot; and vertical bars are added to mark (i) the zero point and (ii) the minimum possible value and (iii) the maximum possible value (in our case, -100% and 100%). The above numbers would therefore be drawn as follows:


| method | median |
|---|---|
| cocomin-f-0-sd-mre | 4.9 |
| cocomin-f-0-pred30 | 4.9 |
| cocomin-f-0-mmre | 4.9 |
| cocomin-f-0-corr | 4.9 |
| cocomin-f-1-pred30 | 4.1 |
| cocomin-f-1-sd-mre | 2.2 |
| cocomin-f-1-mmre | 1.7 |
| cocomin-f-2-pred30 | 1.5 |
| cocomin-b-16-sd-mre | 0.0 |
| cocomin-b-8-sd-mre | 0.0 |
| cocomin-f-4-pred30 | 0.0 |
| cocomin-b-4-sd-mre | 0.0 |
| cocomin-f-4-sd-mre | 0.0 |
| cocomin-f-1-corr | 0.3 |
| cocomin-f-2-sd-mre | 0.0 |
| cocomin-f-16-sd-mre | 0.0 |
| cocomin-f-8-sd-mre | 0.0 |
| cocomin-f-8-pred30 | 0.0 |
| cocomin-f-16-pred30 | 0.0 |
| cocomin-b-16-pred30 | 0.0 |
| cocomin-b-2-sd-mre | 0.0 |
| cocomin-f-2-mmre | 0.0 |
| cocomin-b-8-pred30 | 0.0 |
| cocomin-b-4-pred30 | 0.0 |
| cocomin-b-1-sd-mre | 0.0 |
| cocomin-b-1-pred30 | 0.0 |
| cocomin-f-2-corr | 0.0 |
| cocomin-b-8-mmre | -0.3 |
| cocomin-b-16-mmre | -0.5 |
| cocomin-f-8-corr | 0.0 |
| cocomin-f-4-corr | 0.0 |
| cocomin-f-16-corr | 0.0 |
| cocomin-b-8-corr | 0.0 |
| cocomin-b-16-corr | 0.0 |
| cocomin-b-0-sd-mre | 0.0 |
| cocomin-b-0-pred30 | 0.0 |
| cocomin-b-0-mmre | 0.0 |
| cocomin-b-0-corr | 0.0 |
| cocomin-b-2-pred30 | 0.0 |
| cocomin-b-2-corr | 0.0 |
| LC | -1.3 |
| cocomin-b-4-corr | 0.0 |
| cocomin-b-1-mmre | 0.0 |
| cocomin-b-1-corr | 0.0 |
| cocomin-b-4-mmre | -0.2 |
| cocomin-f-4-mmre | -0.7 |
| cocomin-b-2-mmre | -0.6 |
| cocomin-f-8-mmre | -0.7 |
| cocomin-f-16-mmre | -0.7 |
| cocomost | -2.0 |




## VII. Results

Figure **??** shows the performance deltas of LC, COCOMOST, and each variation of COCOMIN. COCOMOST performs the best but by a slight margin. In addition, some variations of COCOMIN outperformed standard COCOMO-based local calibration. Another depiction of these slight yet clear differences can be seen in Figure **??**.

Over all 19 datasets, COCOMIN works best when a forward select is used, the Stale factor is set to 16, correlation is used to rank the attributes, and MMRE is used to evaluate improvement. For the rest of this paper we will consider COCOMIN as this variation only. In Figure **??**, one can also observe the tendency of higher stale values to outperform lower stale values, except with backward select searches. A backward select search with a low stale value would tend to keep most of the attributes and thus be similar to LC which explains this phenomenon.

```
Average # of Attributes Used: 13.6

Average # of Dropped Attributes: 1.4

Percentage of Times Each Attribute is Dropped:
rely : 2.4%
data : 11.0%
cplx : 9.0%
time : 0.5%
stor : 12.9%
virt : 1.4%
turn : 64.3%
acap : 1.0%
aexp : 0.0%
pcap : 0.0%
vexp : 0.0%
lexp : 1.9%
modp : 19.0%
tool : 17.1%
sced : 0.5%
```

Fig. 6. Attributes Dropped by COCOMOST for the COC81 data.

```
Average # of Attributes Used: 8.1

Average # of Dropped Attributes: 6.9

Percentage of Times Each Attribute is Dropped:
rely : 34.1%
data : 5.3%
cplx : 30.9%
time : 9.7%
stor : 67.0%
virt : 30.9%
turn : 15.0%
acap : 25.0%
aexp : 32.6%
pcap : 68.0%
vexp : 61.5%
lexp : 72.2%
modp : 75.6%
tool : 82.9%
sced : 79.4%
```

Fig. 7. Attributes Dropped by COCOMOST for the NASA93 data.

It is clear from Figure **??** and Figure **??** that the effectiveness of feature subset selection, at least as seen from the COCOMOST and COCOMIN algorithms, is highly dependent on the underlying data.

Figure **??** shows that both COCOMOST and COCOMIN were able to reduce the deviation of the mean relative error. Interestingly, Figure **??** and Figure **??** both show that COCOMOST and COCOMIN tend to help in the worse cases, as the deviation and mmre rise, but don't make improvements on the better predictions. Note that this is consistent with the observation that the effectiveness of feature subset selection is highly dependent on the underlying data, i.e. the learners had a positive impact on the Nasa93 data but not on the Coc81 data. In addition, the effectiveness of the feature subset selectors on the worse estimates but not the better estimates explains the higher Pred30 values for LC seen in Figure **??**.

Information about how many and which attributes were dropped by COCOMOST can be found in Figure **??** and Figure **??**, and Figure **??** and Figure **??** for COCOMIN. Note that the attributes from the NASA93 dataset were discarded far more often than those from the COC81 dataset.

Finally, in this experiment COCOMOST took about 2 seconds for each execution while COCOMIN took about 27. This is because the core of COCOMOST was written in C++ and optimized while COCOMIN uses lots of bash and gawk calls. Local Calibration took less than a second for each execution.

## VIII. Conclusion

The results show that feature subset selection can improve upon the error and deviation of COCOMO estimates, but this effectiveness is highly dependent on the underlying data. If the effort multipliers aren't properly correlated to the output then it is more likely that a feature subset selector will find improvement by discarding them. In addition, this improvement was observed on the datasets that gave the worst errors and deviations but not the datasets with better error

```
Average # of Attributes Used: 13.6

Average # of Dropped Attributes: 1.4

Percentage of Times Each Attribute is Dropped:
rely : 1.0%
data : 14.3%
cplx : 13.8%
time : 10.0%
stor : 2.4%
virt : 1.9%
turn : 36.2%
acap : 11.4%
aexp : 0.5%
pcap : 2.4%
vexp : 7.1%
lexp : 7.1%
modp : 12.4%
tool : 13.8%
sced : 01.0%
```

Fig. 8. Attributes Dropped by COCOMIN for the COC81 data.

```
Average # of Attributes Used: 7.7

Average # of Dropped Attributes: 7.3

Percentage of Times Each Attribute is Dropped:
rely : 36.8%
data : 45.5%
cplx : 41.0%
time : 6.1%
stor : 52.2%
virt : 21.0%
turn : 16.6%
acap : 22.5%
aexp : 35.2%
pcap : 60.2%
vexp : 58.7%
lexp : 65.3%
modp : 93.9%
tool : 88.0%
sced : 82.5%
```

Fig. 9. Attributes Dropped by COCOMIN for the NASA93 data.

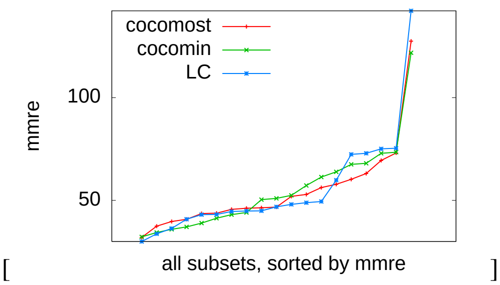



Fig. 10. Mean Magnitude of Relative Error

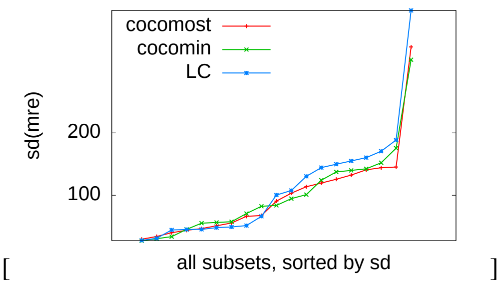



Fig. 11. Standard Deviation of Magnitude of Relative Error

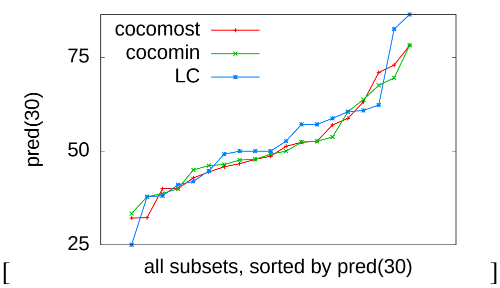



Fig. 12. PRED(30)

rates and deviations, standard COCOMO showed better Pred30 values due to this effect. In addition, more complete searches of the attribute space were shown to be more effective than limited searches. COCOMOST was found to have the best MMRE of the learners used in the experiment, and was also shown to be effective at lowering the standard devation of the errors which will make it easier to distinguish rival methods. Finallly, while COCOMOST ran quickly in this set up it clearly won't scale well to growing numbers of attributes in the dataset.

## IX. Future Work

THERE is a lot of work to be done in improving software cost estimation models. Some possibilities suggested by this research include:

- Feature Subset Selection using other evaluation methods and learners.
- Expanding upon COCOMOST with more data mining techniques such as bagging.
- More comparisons of learners using COCOMOST as a "strawman" FSS model.
- Due to COCOMOST's fairly quick runtime it could be used by WRAPPER algorithms such as COSEEKMO.

## References

[1] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.
[2] T. Jones, *Estimating Software Costs*. McGraw-Hill, 1998.
[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] I. H. Witten and E. Frank, *Data mining. 2nd edition*. Los Altos, US: Morgan Kaufmann, 2005.
[5] 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.
[6] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Feature subset selection can improve software cost estimation accuracy." in *Proceedings of the 2005 Workshop on Predictor Models in Software Engineering*, ACM Press, New York, NY, May 2005.
[7] T. Menzies, D. Port, Zhihao Chen, and J. Hihn, "Validation methods for calibrating software effort models," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on* vol., no.pp. 587- 595, 15–21 May 2005.
[8] T. Menzies and J. Hihn, "Evidence-Based Cost Estimation for Better-Quality Software," in *IEEE Software* , vol.23, no.4, pp. 64–66, July-Aug. 2006.
[9] G. V. Lashkia and L. Anthony, "Relevant, irredundant feature selection and noisy example elimination," in *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol.34, no.2, pp. 888–897, 2004.
[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] K. Strike, K. E. Emam and N. H. Madhavji, "Software Cost Estimation with Incomplete Data" in *IEEE Transactions on Software Engineering*, vol.27, no.10, pp. 890–908, 2001.
[12] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.
[13] R. Bouckaert, "Choosing between two learning algorithms based on calibrated tests," in *ICML'03*, 2003, available from http://www.cs.pdx.edu/~timm/dm/10x10way.

**Tim Menzies** is an associate professor at the University of West Virginia and works with NASA on software quality. His recent research concerns modeling and learning, with a particular focus on lightweight modeling methods. He received his PhD in AI and knowledge engineering from the University of New South Wales. Contact him at tim@menzies.us.

**Dan Baker** is a graduate student at West Virginia University. He received his Bachelor's degree in Computer Science from West Virginia University in May of 2006. For his senior project he developed the database used in the West Virginia Crime Reduction and Information Management Effort. He is currently working on his Master's degree in Computer Science at West Virginia University while working as a research assistant for Dr Menzies. His research focuses on data mining in data starved domains such as software cost estimation. He is the president of the student chapter of Upsilon Pi Epsilon at West Virginia University. UPE is the first and only international honor society for the Computing Sciences.

## APPENDIX

**APPENDIX A - COCOMO-I vs COCOMO-II:** In COCOMO II, the exponential COCOMO 81 term $b$ was expanded into the following expression:

$$b + 0.01 * \sum_j SF_j \tag{2}$$

where b is 0.91 in COCOMO II 2000, and $SF_j$ is one of five *scale factors* that exponentially influence effort. Other changes in COCOMO II included dropping the development modes of Figure **??** as well as some modifications to the list of effort multipliers and their associated numeric constants (see appendix E).

**APPENDIX B - Calculating Correlation:** Given a test set of size $T$, correlation is calculated as follows:

$$\begin{array}{lllllll} \bar{p} & = & \frac{\sum_I^T predicted_i}{T} & & \bar{a} & = & \frac{\sum_I^T actual_i}{T} \\ S_p & = & \frac{\sum_i^T (predicted_i - \bar{p})^2}{T-1} & & S_a & = & \frac{\sum_i^T (actual_i - \bar{a})^2}{T-1} \\ S_{pa} & = & \frac{\sum_i^T (predicted_i - \bar{p})(actual_i - \bar{a})}{T-1} & & & & \\ corr & = & S_{pa}/\sqrt{S_p * S_a} & & & & \end{array}$$

**APPENDIX C - Local Calibration:** This approach assumes that a matrix $D_{i,j}$ holds:

- The natural log of the $KLOC$ estimates;
- The natural log of the actual efforts for projects $i \leq$j$\leq t$;
- The natural logarithm of the cost drivers (the scale factors and effort multipliers) at locations $1 \leq i \leq 15$ (for COCOMO 81) or $1 \leq i \leq 22$ (for COCOMO-II).

With those assumptions, Boehm [?] shows that for COCOMO 81, the following calculation yields estimates for "$a$" and "$b$" that minimizes the sum of the squares of residual errors:

$$\left.\begin{array}{rcl} EAF_i & = & \sum_j^N D_{i,j} \\ a_0 & = & t \\ a_1 & = & \sum_i^t KLOC_i \\ a_2 & = & \sum_i^t (KLOC_i)^2 \\ d_0 & = & \sum_i^t (actual_i - EAF_i) \\ d_1 & = & \sum_i^t ((actual_i - EAF_i) * KLOC_i) \\ b & = & (a_0 d_1 - a_1 * d_0)/(a_0 a_2 - a_1^2) \\ a_3 & = & (a_2 d_0 - a_1 d_1)/(a_0 a_2 - a_1^2) \\ a & = & e^{a_3} \end{array}\right\} \tag{3}$$

**APPENDIX D - COCOMO Numerics:** Figure **??** shows the COCOMO 81 $EM_j$ (effort multipliers). The effects of that multiplier on the effort are shown in Figure **??**. Increasing the *upper* and *lower* groups of variables will *decrease* or *increase* the effort estimate, respectively.

Figure **??** shows the COCOMO 81 effort multipliers of Figure **??**, rounded and simplified to two significant figures.

Figure **??**, Figure **??** and Figure **??** show the COCOMO-II values analogies to Figure **??**, Figure **??** and Figure **??** (respectively).

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

Fig. 13. COCOMO 81 effort multipliers.

| | | very low | low | nominal | high | very high | extra high |
|---|---|---|---|---|---|---|---|
| upper | ACAP | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | |
| (increase | PCAP | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | |
| these to | AEXP | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | |
| decrease | MODP | 1.2 | 1.10 | 1.00 | 0.91 | 0.82 | |
| effort) | TOOL | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | |
| | VEXP | 1.21 | 1.10 | 1.00 | 0.90 | | |
| | LEXP | 1.14 | 1.07 | 1.00 | 0.95 | | |
| middle | SCED | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | |
| lower | DATA | | 0.94 | 1.00 | 1.08 | 1.16 | |
| (increase | TURN | | 0.87 | 1.00 | 1.07 | 1.15 | |
| these to | VIRT | | 0.87 | 1.00 | 1.15 | 1.30 | |
| increase | STOR | | | 1.00 | 1.06 | 1.21 | 1.56 |
| effort) | TIME | | | 1.00 | 1.11 | 1.30 | 1.66 |
| | RELY | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | |
| | CPLX | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |

Fig. 14. The precise COCOMO 81 effort multiplier values.

| | | very low | low | nominal | high | very high | extra high |
|---|---|---|---|---|---|---|---|
| upper | ACAP | 1.2 | 1.1 | 1.00 | 0.9 | 0.8 | |
| (increase | PCAP | 1.2 | 1.1 | 1.00 | 0.9 | 0.8 | |
| these to | AEXP | 1.2 | 1.1 | 1.00 | 0.9 | 0.8 | |
| decrease | MODP | 1.2 | 1.1 | 1.00 | 0.9 | 0.8 | |
| effort) | TOOL | 1.2 | 1.1 | 1.00 | 0.9 | 0.8 | |
| | VEXP | 1.2 | 1.1 | 1.00 | 0.9 | | |
| | LEXP | 1.2 | 1.1 | 1.00 | 0.9 | | |
| middle | SCED | 1.2 | 1.1 | 1.00 | 1.1 | 1.2 | |
| lower | DATA | | 0.9 | 1.00 | 1.1 | 1.2 | |
| (increase | TURN | | 0.9 | 1.00 | 1.1 | 1.2 | |
| these to | VIRT | | 0.9 | 1.00 | 1.1 | 1.2 | |
| increase | STOR | | | 1.00 | 1.1 | 1.2 | 1.3 |
| effort) | TIME | | | 1.00 | 1.1 | 1.2 | 1.3 |
| | RELY | 0.8 | 0.9 | 1.00 | 1.1 | 1.2 | |
| | CPLX | 0.8 | 0.9 | 1.00 | 1.1 | 1.2 | 1.3 |

Fig. 15. Rounded COCOMO 81 effort multiplier values.

| | |
|---|---|
| scale factors (exponentially decrease effort) | prec: have we done this before? |
| | flex: development flexibility |
| | resl: any risk resolution activities? |
| | team: team cohesion |
| | pmat: process maturity |
| upper (linearly decrease effort) | acap: analyst capability |
| | pcap: programmer capability |
| | pcon: programmer continuity |
| | aexp: analyst experience |
| | pexp: programmer experience |
| | ltex: language and tool experience |
| | tool: tool use |
| | site: multiple site development |
| | sced: length of schedule |
| lower (linearly increase effort) | rely: required reliability |
| | data: secondary memory storage requirements |
| | cplx: program complexity |
| | ruse: software reuse |
| | docu: documentation requirements |
| | time: runtime pressure |
| | stor: main memory requirements |
| | pvol: platform volatility |

Fig. 16. The COCOMO II scale factors and effort multipliers.

| | | extra low | very low | low | nominal | high | very high | extra high |
|---|---|---|---|---|---|---|---|---|
| scale | prec | | 6.20 | 4.96 | 3.72 | 2.48 | 1.24 | 0.00 |
| factors | flex | | 5.07 | 4.05 | 3.04 | 2.03 | 1.01 | 0.00 |
| (exponentially | resl | | 7.07 | 5.65 | 4.24 | 2.83 | 1.41 | 0.00 |
| decreases | team | | 5.48 | 4.38 | 3.29 | 2.19 | 1.10 | 0.00 |
| effort) | pmat | | 7.80 | 6.24 | 4.68 | 3.12 | 1.56 | 0.00 |
| upper | acap | | 1.42 | 1.19 | 1.00 | 0.85 | 0.71 | |
| (linearly | pcap | | 1.34 | 1.15 | 1.00 | 0.88 | 0.76 | |
| decreases | pcon | | 1.29 | 1.12 | 1.00 | 0.90 | 0.81 | |
| effort) | aexp | | 1.22 | 1.10 | 1.00 | 0.88 | 0.81 | |
| | pexp | | 1.19 | 1.09 | 1.00 | 0.91 | 0.85 | |
| | ltex | | 1.20 | 1.09 | 1.00 | 0.91 | 0.84 | |
| | tool | | 1.17 | 1.09 | 1.00 | 0.90 | 0.78 | |
| | site | | 1.22 | 1.09 | 1.00 | 0.93 | 0.86 | 0.80 |
| | sced | | 1.43 | 1.14 | 1.00 | 1.00 | 1.00 | |
| lower | rely | | 0.82 | 0.92 | 1.00 | 1.10 | 1.26 | |
| (linearly | data | | | 0.90 | 1.00 | 1.14 | 1.28 | |
| increases | cplx | | 0.73 | 0.87 | 1.00 | 1.17 | 1.34 | 1.74 |
| effort) | ruse | | | 0.95 | 1.00 | 1.07 | 1.15 | 1.24 |
| | docu | | 0.81 | 0.91 | 1.00 | 1.11 | 1.23 | |
| | time | | | | 1.00 | 1.11 | 1.29 | 1.63 |
| | stor | | | | 1.00 | 1.05 | 1.17 | 1.46 |
| | pvol | | | 0.87 | 1.00 | 1.15 | 1.30 | |

Fig. 17. The precise COCOMO II numerics.

| | | extra low | very low | low | nominal | high | very high | extra high |
|---|---|---|---|---|---|---|---|---|
| Scale | PREC | | 6.3 | 5.1 | 3.8 | 2.5 | 1.3 | 0 |
| Factors | FLEX | | 6.3 | 5.1 | 3.8 | 2.5 | 1.3 | 0 |
| | RESL | | 6.3 | 5.1 | 3.8 | 2.5 | 1.3 | 0 |
| | TEAM | | 6.3 | 5.1 | 3.8 | 2.5 | 1.3 | 0 |
| | PMAT | | 6.3 | 5.1 | 3.8 | 2.5 | 1.3 | 0 |
| upper | ACAP | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | |
| | PCAP | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | |
| | PCON | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | |
| | AEXP | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | |
| | PEXP | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | |
| | LTEX | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | |
| | TOOL | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | |
| | SITE | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | 0.8 |
| | SCED | | 1.3 | 1.1 | 1.0 | 0.9 | 0.8 | |
| lower | RELY | | 0.8 | 0.9 | 1.0 | 1.1 | 1.3 | |
| | DATA | | | 0.9 | 1.0 | 1.1 | 1.3 | |
| | CPLX | | 0.8 | 0.9 | 1.0 | 1.1 | 1.3 | 1.5 |
| | RUSE | | | 0.9 | 1.0 | 1.1 | 1.3 | 1.5 |
| | DOCU | | 0.8 | 0.9 | 1.0 | 1.1 | 1.3 | |
| | TIME | | | | 1.0 | 1.1 | 1.3 | 1.5 |
| | STOR | | | | 1.0 | 1.1 | 1.3 | 1.5 |
| | PVOL | | | 0.9 | 1.0 | 1.1 | 1.3 | |

Fig. 18. The rounded COCOMO II numerics.