

When Will It Be Done? Machine Learners Answer the 300- Billion-Dollar Question

Gary D. Boetticher, *University of Houston-Clear Lake*

When will it be done?" Senior managers will ask their software project managers this question over 250,000 times this year. Corporations, which collectively commit over 300 billion dollars annually toward new software project initiatives,¹ will want to know the answer. However, when you consider Barry Boehm's claim that early

software life-cycle estimates vary by a factor of four (25 to 400 percent),² providing an accurate, reliable project estimate presents a challenge indeed.

To answer the question, a project manager might resort to one of three estimation approaches: human-based, algorithmic, or machine learner-based. Managers will use a human-based approach—which includes expert judgement, analogy, and rule of thumb—87 percent of the time, algorithmic or machine learners only 13 percent of the time.³ Considering the popularity of algorithmic models (such as Function Point Analysis) for estimating effort, machine learner use is probably much less than half of 13 percent—perhaps three or four percent.

This is surprising considering recent successes in machine learner estimation. In three instances, machine learners produced project effort estimates within 25 percent accuracy at least 75 percent of the time. The high-water mark of these efforts produces estimates with 25 percent accuracy 83 percent of the time.⁴ All three cases greatly exceed Boehm's factor-of-four expectation. I contend that a greater application of AI, through machine learners, in software estimation would greatly improve accuracy, reliability, and repeatability in software development.

Case studies

To standardize results, I define two benchmarks for the case studies:

- *Prediction accuracy* in project estimation refers to how often a machine learner can achieve a result

within a specified range. The informal standard in effort estimation is 25 percent accuracy, denoted as $\text{pred}(0.25)$. So, a result of $\text{pred}(0.25) = 75$ percent means the machine learner model produces results with 25 percent accuracy 75 percent of the time.

- *Mean magnitude of relative error* is the average difference between actual and calculated results in absolute terms. Many consider an MMRE below 25 percent a good model and below 50 percent a usable model.

All three cases assume the accuracy rate of $\text{pred}(0.25)$ and build their machine learner-based models using data extracted from industrial settings, as opposed to deriving models from simulations. This reinforces these results' real-world applicability.

Case 1: Wittig and Finnie

Gerhard Wittig and Gavin Finnie estimated development effort by applying a back-propagation neural network model to the Australian Software Metrics Association data set.⁵ You can obtain the ASMA data set from the International Software Benchmarking Standards Group (www.isbsg.org) for a nominal cost. The current version, ISBSG data suite 8.0, contains data (42 metrics) from 2,000 software projects.

Wittig and Finnie used ISBSG data suite 5.0 in their experiments. They started with a data set of 136 samples composed of seven inputs and one output. After eliminating extreme outliers, they partitioned

*The international
\$300-billion software
development industry
needs better predictors
for software
development costs.
Data miners can learn
such predictors with
impressive accuracy.*

the remaining 115 samples into 105 training and 10 test vectors. They replicated their experiment three times by randomly selecting the independent training and test vectors. The ASMA experiments produced an MMRE of 17 percent with an accuracy of 75 percent pred(0.25).

Wittig and Finnie observed that they consistently obtained optimal results using a learning rate of 0.3 and a momentum coefficient of 0.6 on a network with one hidden layer. They also found that the Gaussian and Sigmoid transfer functions performed the best, with the Sigmoid resulting in the lower prediction errors.

Case 2: Chulani and colleagues

Sunita Chulani and her colleagues synthesized machine learning (Bayesian analysis) with expert estimation and an algorithmic model.⁶ They initially applied a multiple-regression approach to a Cocomo II (*Cost Constructive Model*) data set and identified counterintuitive values for some of the effort multipliers (for example, the reuse effort multiplier). Cocomo II is an effort estimation formula defined as

$$Person\ Months = a(Size)^b * P(EM_i),$$

where *a* refers to the project development mode, *Size* is based on either source lines of code or function points, *b* refers to one of five size factors, and *EM_i* represents one of 17 effort multipliers (assuming a postarchitecture model).

To remedy the counterintuitive results, Chulani and her colleagues assembled a group of project estimation experts, also referred to as a Delphi group, to assess each of the 17 Cocomo effort multipliers. Each member of the Delphi group offered a set of ratings; the group tabulated the results and then peer reviewed them. Several iterations of assessment reduced the variance in ratings.

These ratings served as a priori data for tuning the Cocomo II model. Chulani and her colleagues produced a new Cocomo II version that included a 10 percent weighted average to adjust the a priori expert-determined model parameters. Applying the new Cocomo II version to a 1998 Cocomo II data set that included 161 samples, produced estimates of 55 percent and 63 percent accuracy and pred(0.25)—before and after stratification, respectively.

Chulani and her colleagues extended this

new version by combining sample Cocomo II data with the a priori Delphi ratings to produce a posteriori Bayesian updates (combined updates). The Bayesian theorem combines prior information (Delphi expert ratings) with sample information (data model) and derives the posterior information (final estimates). They obtained the multiplier information by examining variances from the prior and sample information. If the variance of an a priori (Delphi expert) probability distribution for a certain multiplier is smaller than the corresponding sample data variance, the posterior distribution will be nudged closer to the a priori distribution. This implies that the sample information is noisy and you must lean more

The whole neural network modeling process completed in less than an hour. Considering this approach's potential payoff, this is a relatively trivial time investment.

heavily on the prior (expert) information.

Applying the a posteriori Bayesian updates to the 1998 Cocomo II data set, with 161 samples, yielded estimates of 68 percent and 76 percent accuracy and pred(0.25)—before and after stratification. So, extending Cocomo II with the Bayesian learner improved accuracy results by 13 percent.

This hybrid approach is appealing because it shows that you can integrate machine learner and traditional approaches (as opposed to a replacement).

Case 3: Boetticher

When blending machine learner techniques with algorithmic models, you must address the algorithmic models' subjectivity. I have eliminated this subjectivity issue by extracting metrics from a GUI specifications document and by constructing a supervised neural network effort estimator.⁴

The inputs for the supervised neural network experiments consist of various GUI-based widget counts (for example, number

of buttons, list boxes, or combo boxes) extracted from a business-to-business petrochemical application. I collected a total of 12 different widget counts.

The output value is the effort associated with each form (corresponding screen). Most organizations track effort at the project level rather than at the product level. Consequently, you can't assess effort per form. For these experiments, I kept fine-granularity effort information (for example, hours per form) in each form's coding.

The B2B application comprises four subsystems distributed among 109 different forms. Each form is an instance in the data set. I conducted a 10-way cross validation on the 109 samples using a variant of the backpropagation neural network called Quickprop. Quickprop exploits second-order derivatives to quickly reach a solution.⁷

All experiments used a learning rate of 1, a momentum of 1, a neural network architecture of 12 inputs, five hidden nodes, another layer of 11 hidden nodes, a third hidden layer of five hidden nodes, and an output layer of one node. I let the neural networks run for 1,000 epochs. However, most models converged within 250 epochs. As a precaution, I let a couple of models run for 10,000 epochs. The extra time did not improve accuracy.

I grouped estimates from the individual forms by their corresponding subsystem. Using the aggregate values generates results averaging 83.3 percent accuracy, pred(0.25), and an MMRE averaging 14.7 for client-side components.

Interestingly, the whole neural network modeling process completed in less than an hour. Considering this approach's potential payoff, this is a relatively trivial time investment.

In an ideal setting, a project's requirements document would serve as the basis for producing an accurate, reliable estimate. Unfortunately, a requirements document provides sparse data for building such a model. However, a GUI specifications document is rich in GUI metrics. This work demonstrates that you can formulate an estimate very early in the software life cycle. Using GUI-based metrics eliminates the uncertainty found in algorithmic models. Counting widgets is a simple, objective, and repeatable process that requires minimal technical training. So, this methodology is easy to incorporate.

The Author



Gary D. Boetticher is an assistant professor in the School of Science and Computer Engineering at the University of Houston, Clear Lake. His research interests include data mining, machine learning, bioinformatics, and software metrics. He received his PhD in computer science from West Virginia University. He is a member of the ACM and IEEE. Contact him at the Dept. of Software Eng., Univ. of Houston, Clear Lake, 2700 Bay Area Blvd., Houston, TX 77058; boetticher@cl.uh.edu.

This article has cited a few of AI's high-water marks in software effort estimation. Hopefully, the case studies' results will inspire project managers to seriously consider applying AI to estimate effort. An AI process might not necessarily replace current techniques within an organization but certainly ought to be considered as additional resource. ■

References

1. *Chaos Chronicles v.3.0*, The Standish Group, Jan. 2003; www.standishgroup.com/chaos/toc.php.
2. B. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.
3. M. Jørgensen, "A Review of Studies on Expert Estimation of Software Development Effort," to be published in *J. Science and Software*, 2004.
4. G. Boetticher, "Applying Machine Learners to GUI Specifications in Formulating Early Life Cycle Project Estimations," *Software Engineering with Computational Intelligence*, T.M. Khoshgoftar, ed., Kluwer, 2003.
5. G. Wittig and G. Finnie, "Estimating Software Development Effort with Connectionist Models," *Information and Software Technology*, vol. 39, no.7, 1997, pp. 469–476.
6. S. Chulani, B. Boehm, and B. Steece, "Bayesian Analysis of Empirical Software Engineering Cost Models," *IEEE Trans. Software Eng.*, vol. 25, no. 4, July/Aug. 1999, pp. 573–583.
7. S.E. Fahlman, *An Empirical Study of Learning Speed in Back-Propagation Networks*, tech. report CMU-CS-88-162, Dept. of Computer Science, Carnegie Mellon Univ., 1988.

For more information on this or any other computing topic, please visit our digital library at <http://computer.org/publications/dlib>.