Software Metrics Data Analysis – Exploring the Relative Performance of Some Commonly Used Modeling Techniques

Andrew R. Gray and Stephen G. MacDonell
Software Metrics Research Lab
Department of Information Science
University of Otago, PO Box 56, Dunedin, New Zealand
+64 3 4798135 (ph.) +64 3 4798311 (fax), stevemac@infoscience.otago.ac.nz

Abstract

Whilst some software measurement research has been unquestionably successful, other research has struggled to enable expected advances in project and process management. Contributing to this lack of advancement has been the incidence of inappropriate or non-optimal application of various model-building procedures. This obviously raises questions over the validity and reliability of any results obtained as well as the conclusions that may have been drawn regarding the appropriateness of the techniques in question. In this paper we investigate the influence of various data set characteristics and the purpose of analysis on the effectiveness of four model-building techniques – three statistical methods and one neural network method. In order to illustrate the impact of data set characteristics, three separate data sets, drawn from the literature, are used in this analysis. In terms of predictive accuracy, it is shown that no one modeling method is best in every case. Some consideration of the characteristics of data sets should therefore occur before analysis begins, so that the most appropriate modeling method is then used. Moreover, issues other than predictive accuracy may have a significant influence on the selection of model-building methods. These issues are also addressed here and a series of guidelines for selecting among and implementing these and other modeling techniques is discussed.

Keywords: Software metrics, analysis, statistical methods, connectionist methods

1. INTRODUCTION

The management of software development projects has a relatively poor reputation in terms of avoiding cost and schedule overruns. In an effort to improve this track record, many mature organizations (in a software development sense) have invested heavily in the development and use of software metrics – measures derived from and applied to software products or processes. Such metrics have been collected with the understanding that they may lead to the development of models that will enable greater control to be exercised over the increasingly complex software development process. One of the most widely investigated metric-based models is the intuitively plausible relationship between software product size and complexity on the one hand and associated development effort on the other. In this way, larger and more complex systems are assumed to take longer to develop. An extensive array of predictive models has been generated largely as a result of empirical analyses, COCOMO (Boehm, 1981), SLIM (Putnam et al., 1984) and Function Point Analysis (FPA) (Albrecht and Gaffney, 1983) being among the most popular approaches. In essence these and similar models incorporate one or more product size measures, normally program size in estimated lines of code or specification size in numbers of screens, files and so on, along with some measure of complexity, calibrated under linear regression in a model predicting development effort and/or duration. Although some standard models are available, the effectiveness of the predictions is generally improved when organization- or domain-specific data is used in calibration, especially since many other factors can influence development effort, such as tools and methodologies used, and these are usually more consistent within organizations than between organizations.

There is little doubt that such approaches have the potential to substantially improve the accuracy of effort estimation and consequently the management of development schedules. This depends on a number of contingent factors, including having effective measures of size and complexity, the existence of relevant data, and the use of appropriate analysis methods. It is this third area of research that we are currently investigating. Because of their relative accessibility (through widely available software packages, including SPSS, SAS, MINITAB, MATLAB, and S-Plus) standard statistical analyses have been most commonly adopted in the determination of predictive models (for example, see recent papers by Ebrahimi (1999), Stensrud and Myrtveit

(1998), Heiat and Heiat (1997) and Dolado (1997)). While empirical analysis is certainly to be encouraged, particularly over the guesswork that tended to be prevalent before the promotion of software metrics, its use must be tempered by an awareness of the relevance, generalisability, and limitations of particular analysis and modeling techniques.

It is the inherent nature of software metrics data that contributes to the demand for a greater appreciation of the applicability of various modeling methods. Software engineering data sets are often skewed to the right (that is to say, they exhibit a distribution with most values at the lower end and with a few very large observations creating a long tail extending to the right), and may contain a number of outlier observations. Moreover, since we are as yet to establish (and it would seem likely that we will never establish) an underlying *theoretical* model of software development, a full awareness of the variables that contribute to development effort and how they interact may be unattainable. Since many standard analysis methods assume data sets that conform to a normal distribution and involve relatively simple, if any, interactions, these 'standard' modeling methods may not be particularly appropriate. This is not to say that such analysis methods should be discarded entirely, but rather that some exploratory examination of the data should always occur first and analysis methods chosen as a result of this analysis, with the purpose of the model – e.g. predictive accuracy, process understanding – also always in mind.

For example, a simple model involving only one predictor variable will generally be best served by using a regression model rather than a feedforward neural network. This is due to the greater speed of the regression procedure compared to training the neural network, and also because of the greater analysis and information obtainable from the regression model. When dealing with many variables and with complex interactions assumed, a feedforward neural network could appear to be a better choice since this simplifies the model structure selection process. If data is likely to be extremely contaminated then outlier resistant techniques, such as robust regression, may be preferred. In these ways the data suggest that certain modeling techniques may be more effective than others. These techniques are explained below, as is their use in software metric modelling.

2. MODEL-BUILDING METHODS

In the past, software development effort models have been built almost exclusively using least-squares linear regression techniques. More recently, this approach has been complemented by the application of machine learning methods, especially neural networks, in reaction to continuing problems with regression-based solutions (Lee et al., 1998; Finnie et al. 1997; Hakkarainen et al., 1993). Both approaches are considered below. One other method is also examined - that of robust regression (in two forms).

2.1. Regression Methods

Given a particular set of data points or observations, least-squares (LS) regression attempts to determine the function that minimizes the sum of squared errors in the relationship between predicted and actual values, with the predictions being based on the weighted contribution of one or more variables (Neter et al., 1996). The resulting function is normally expressed as an equation such as that shown in Equation (1) where development effort is related to the number of screens (NumScreens) and the number of reports (NumReports) in a system. A constant term is included which may represent administrative overhead, but could also be used to allow a linear function to approximate a non-linear one over a particular range.

$$\begin{aligned}\text{Effort(hours)} = 1000 + 50\text{NumScreens} + \\ 75\text{NumReports}\end{aligned} \qquad (1)$$

The development of a regression function is normally preceded by the examination of scatter plots and correlation analyses, along with cluster analysis to identify distinct groups of systems where appropriate. Interaction terms may need to be formed to enable the predictive model to cope with the effect of combinations of variables, and transformations can be used to partially compensate for non-linearities. The general form of a least-squares regression graph is shown in Figure 1. In this graph the number of man-months (MM) required for development are expressed as a function of the number of screens (SCRN) with a constant term added. The individual data points used to develop the model (the line) are shown as squares on the graph. This constant may (as explained above) reflect the administrative overhead of a project or compensate for a nonlinear relationship over a small range of SCRN (as in this case where the constant is negative and the data exhibit an exponential pattern). The Rsq value given to the right of the graph indicates the proportion of the variation in MM that is explained by SCRN, in this case some 78%.

There are several advantages associated with the use of the least-squares regression method. It is a method that has sound theoretical foundations from its, comparatively speaking, long history. Least-squares is easily accessible through most statistical analysis packages and is widely examined in standard statistical texts (for example, Neter et al., 1996) – thus the software metrics practitioner is likely to have ready access to the technique and to the means of gaining expertise in its use. A further significant advantage of least-squares regression (and of regression in general) is the visibility of the resultant models. Inherent in the model produced is the expression of those variables of statistical influence (in terms of predicting the dependent variable) along with the weightings expressed in raw and standardized forms. This enables the modeler to immediately consider the validity of the model's structure in light of her/his own knowledge and experience.

This visible structure can, however, have equally important disadvantages, in that regression may lead to the provision of equations that are difficult to interpret in an *operational* sense, particularly when many variables, transformations, and interaction terms are included. More significantly, least-squares models may be extensively confounded by outlier data points (common in software engineering data with the relatively common 'rogue' projects) and they may not adequately cope with complex variable interrelationships, particularly given the availability of only small data sets. Least-squares equations, by attempting to minimize the sum of the squared residuals from the model, can be dramatically changed by altering one single point to be suitably *unusual* (in fact, the regression line can be changed to any equation desired by this editing of a single observation). With small data sets there is even greater risk that unusual data points will exert undue influence over the smaller number of representative points.

The degree to which data can be arbitrarily contaminated before the regression line can be changed to any desired equation is called its breakdown point. Least squares has a breakdown point of 0%. Other, robust, regression techniques are available with higher breakdown points, as discussed below (Rousseeuw and Leroy, 1987).

Finally, the nature of the model must be formed by the developer, in terms of variables considered, transformations needed and interaction terms required. As with any technique, attempting to address these questions without sufficient expertise is likely to lead to anomalous or spurious models. While the software metrician is not required to be an expert in statistics as well as project management, they do require the knowledge of what they can do themselves and what should be left for a more qualified statistician.

Least-median-squares (LMS) and least-trimmed-squares (LTS) regression (Rousseeuw and Leroy, 1987) belong to the family of robust regression methods, so-called because they produce predictive models that are generally more effective for making predictions for the main body of observations in data sets containing outlier observations. Although a number of such models exist, we have adopted the LMS and LTS techniques as they are easily compared to the LS approach. Rather than using the sum of squared residuals as the basis for error minimization, LMS regression minimizes the median of the squared residuals and LTS minimizes a trimmed sum of the residuals. Thus both techniques are unaffected by a proportion of outlier values in a data set (called resistance to contamination, or breakdown point, this being 50% in the case of LMS, and a user-definable value for LTS). This is of benefit for skewed data sets since in these cases any outlier observations can be systematically identified and then dealt with accordingly (e.g. included, reweighted, or discarded). The result is a more 'characteristic' model in terms of the data set underlying its development. (See Figure 2 for an illustration of the influence of outlier points on regression models.)

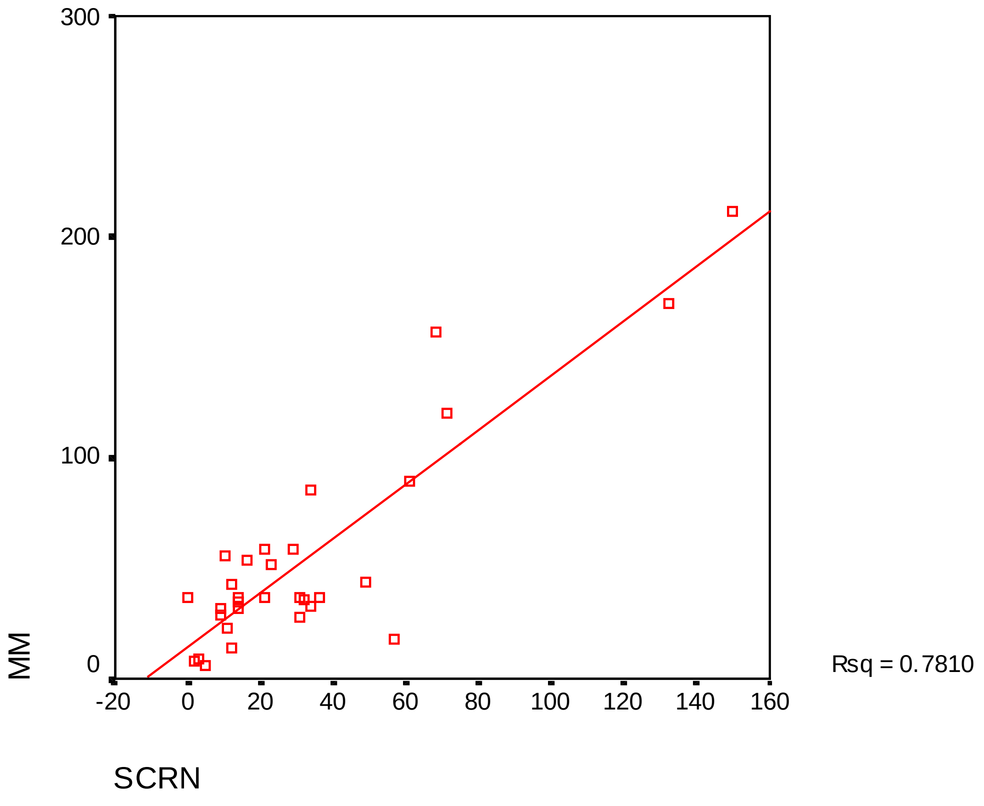

Figure 1. General form of a least-squares regression graph for predicting development effort

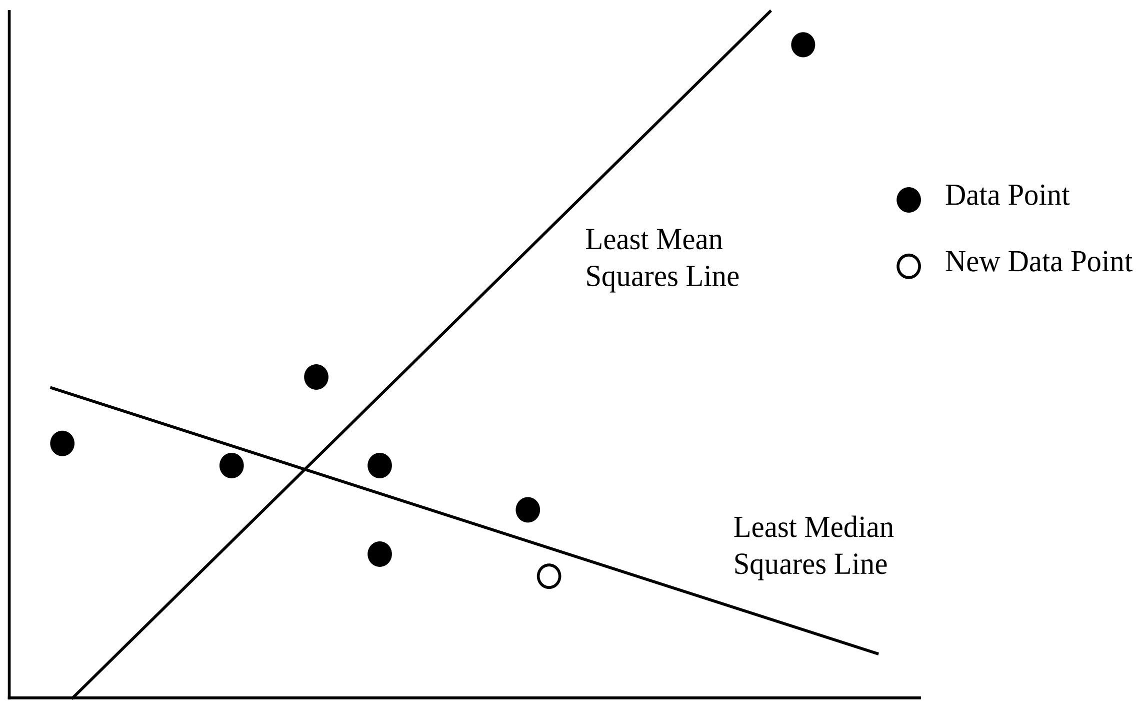

Figure 2. Outlier influence on regression models developed using least-squares and least-median of squares

In terms of disadvantages associated with these methods, they are generally less powerful than the least-squares approach when data set characteristics are not problematic, and they are not as widely accessible in terms of software availability. As with the least-squares method, the analyst must still specify the form of the models.

2.2. Neural Networks

A neural network can be defined, in a simplistic manner, as a number of interconnected units, with weighted connections passing signals from unit to unit. In general some of these units are model inputs and some model outputs. There are a vast number of neural network architectures, along with learning algorithms for determining the parameters (weights) of models expressed using these architectures. In this paper, we limit ourselves to the most common and widely used architecture/learning algorithm combination for prediction purposes. This is the method of back-propagation applied to a feed-forward neural network architecture (Hertz et al., 1991).

A neural network of this type can be seen as a non-linear modeling technique, that consists of inputs (the independent variables), processing units (the hidden nodes), and outputs (the dependent variables). The form of the model is not selected in advance as with regression models; instead the network is theoretically capable of approximating any given function subject to certain restrictions (Hornik et al., 1989). (It is inappropriate to discuss these technical restrictions here; in general terms, given a sufficient number of hidden nodes, any reasonable function can be approximated.)

The use of this type of model requires the same steps as with any statistical technique. These are to select the model parameters, develop the model using data, test the model on withheld data to assess its performance, and implement the outcomes of the model in the development process.

The process of selecting the inputs and outputs is much the same as with any other technique: select those inputs that provide some predictive power with respect to the outputs. However, in the case of a neural network the training stage will determine the form of the model so concerns about transformations and linear relationships can usually be omitted. Similarly, providing composite measures is generally unnecessary, especially of the simple types often found in software metrics. Certainly for the purposes of software metrics data, providing any input variable that relates to the output will in general be satisfactory. Avoiding unnecessary input variables is the only real restriction, either through a lack of relationship or extreme correlation (possibly non-linear) with another independent variable since this will slow training, increase data collection costs, and, most seriously, potentially introduce spurious relationships into the network's behavior.

The most important decision to be made by the metrician in a neural network model is the number of hidden nodes. It is this choice that determines the accuracy with which the network can fit the data. The greater the number of nodes, the more precisely the network can learn a given function. While this may imply, and many researchers take it to be so, that more is better, this only applies where the goal is to learn some deterministic function, where an exact mapping from inputs to outputs exists. In such a case identical projects would have identical outputs (such as effort or error density). This is obviously not the case with software development. For such stochastic data, the goal is instead to develop a model that generalizes to new data. Too few hidden nodes and the network will be unable to learn the relationship well; too many and the network will *overlearn* and start to fit the training data too closely. In addition, using more hidden nodes and thus more weights generally increases the chance that the network will find a solution that is not the best possible (called a local minimum). In general the best advice is to start with a small number of hidden nodes, perhaps even with one or two, and after developing the model add more nodes gradually until performance on validation data stops improving. (Much more sophisticated methods exist, but they are beyond the scope of this paper.)

This leads to the training process itself. Generally this proceeds by first splitting the data file into three subsets; these are the training file that is used to adapt the weights, the validation file that is used to stop training and possibly to select between architectures, and a testing file that is only used at the very end to assess the network's performance. Four data files could be used here (training, validation, selection, and testing), but given the small data sets used for software metrics this would appear to be extravagant. The ratios for these data sets depend on the amounts of data available, the distributions and relationships in the data, the number of variables, and the desired accuracy of the assessment of the model's true generalisability. In general a ½:¼:¼ split for the training, validation, and testing data respectively is recommended, although more data may be withheld for testing as is done in the case studies here. Each data set should be as representative of the entire set as possible, and stratification (breaking the data into groups of similar observations and sampling equally from each group) can be used to enhance this. Generally, however, a random splitting process is used.

Once the data splitting has been performed and a network has been created with the customary small random weights (often between –0.1 and +0.1), training can start. By presenting the network with the training data the weights are iteratively adjusted by small amounts so as to reduce the prediction error in terms of the mean of the squared errors. Periodically, the network should be used to recall (make predictions) on the validation data set. When these errors have been minimized (this usually requires saving the weights and going back once further improvement has ceased) the network can be said to be trained. The use of this validation data to stop training prevents the network from overlearning on the training data, since the network cannot explicitly adjust to fit data that it is not presented with (the validation set). Figure 3 illustrates the behavior of errors over time. Note that while the testing error is shown, this should not be calculated during training if this information will have any effect on the model selected (network architecture, training time, and parameter selection). This is provided here to show that optimizing the validation error approximately optimizes the testing error.

Finally, as with any software metrics model the performance needs to be assessed. While some analysts will use the validation data for this purpose, the error rate on this data is still biased since it was its minimization that stopped training. This is where the testing data set should be used to make predictions and calculate the error, using the desired error measure(s). It is this error, assuming that the data was divided randomly or representatively, that enables us to make an estimate of the degree of confidence we can place in the network. This is an important aspect of the implementation of the model, as how we manage our projects will depend on the certainty we can attach to our estimates.

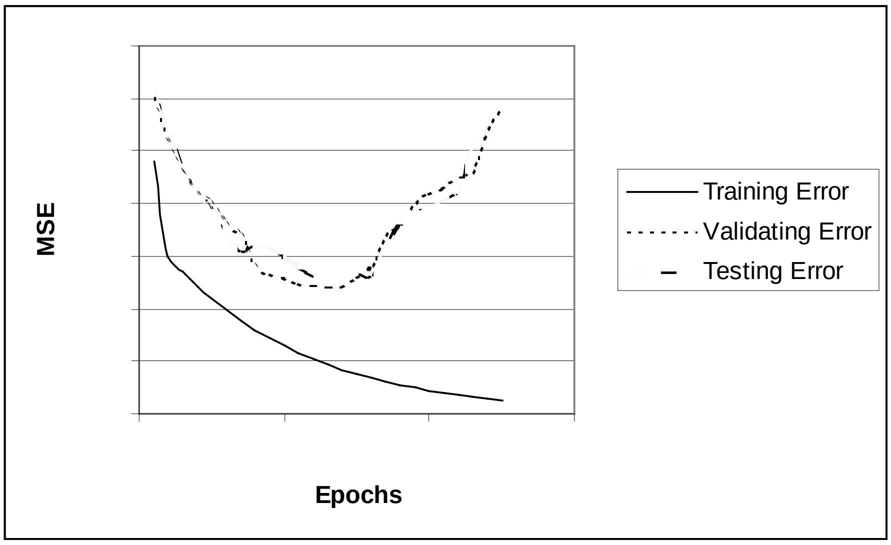

Figure 3. Typical behavior of errors over time during neural network training

3. EMPIRICAL ANALYSIS

In this paper three published data sets are examined from the perspective of the empirical results obtained using the various model-building techniques. The data from Miyazaki et al. (1994) is a set of 48 observations, detailing the development effort and various functional measures of the systems (Table 1 in that paper). This data set has already been analyzed by Miyazaki et al., providing us with a baseline against which to assess the techniques employed in this study in terms of several error measures. The testing set used here consists of 16 randomly selected observations. The second data set is the QUES data from Li and Henry (1993). This data provides a number of object-oriented measures of systems, and the subsequent maintenance changes made to those systems. A total of 71 observations are provided, with 24 of these making up the testing set. While Li and Henry provide some analysis of models predicting changes from the system measures, their choice of error measures is somewhat limited for comparative purposes. The final data set examined here is that of Dolado (1997). This data set consists of 24 observations, giving function point measures and the development effort expended. Eight of these were withheld for testing. Dolado provides some analysis of the data, but little that can be used for comparison with the results here, as no predictive models were actually constructed by Dolado.

Each of the three data sets was evaluated under regression and neural network prediction models with development effort or maintenance changes used as the dependent variable and all available independent variables used initially, as illustrated in the following sections. Influential independent variables were then selected using stepwise regression techniques. The performance of the models on the various data sets (development and testing for regression models, training, validation, and testing sets for neural network models) is given. In addition some more detailed information about the performance of the alternative regression techniques is provided.

3.1. Regression Predictions

In all cases the model development data set (roughly two-thirds of the entire set) was analyzed using correlation analyses, scatter plots and stepwise regression, in order to identify influential variables with a view to sensible and pragmatic model creation. Models were developed based on the outcomes of this exploratory analysis under the least-squares, least-median-squares and least-trimmed-squares methods. The resulting models were then applied to the testing subset (the remaining third of observations) and were evaluated using the following error measures.

The magnitude of relative error (MRE) is a normalized measure of the discrepancy between the actual data values (V_A) and the fitted values (V_F):

$$\mathrm{MRE} = \frac{|V_A - V_F|}{V_A} \qquad (2)$$

The mean MRE (MMRE) is therefore the mean value for this indicator over all observations in the sample. A lower value for MMRE generally indicates a more accurate model from the perspective of a project manager.

The Pred(l) measure provides an indication of overall fit for a set of data points, based on the MRE values for each data point:

$$\mathrm{Pred(l)} = \frac{i}{n} \qquad (3)$$

In equation (3) l is the selected threshold value for MRE (from equation (2)), i is the number of data points with MRE less than or equal to l, and n is the total number of data points.

The Balanced MMRE measure (BMMRE) is also used here, as defined in Miyazaki et al. (1994). The relative error (R) is taken as the lower of the MRE with the actual as the denominator and the MRE with the predicted as the denominator. The BMMRE is then the mean value of the absolute values of R. The AR(l) measure refers to the proportion of BMMRE values less than l%, similar to the Pred(l) measure above. An absolute error measure (Average Absolute error – AAR) is also included here since relative errors may not be the only concern of project managers, although it would appear reasonable that relative errors are more informative.

3.1.1 Miyazaki, Terakado, Ozaki, and Nozaki data set

Results presented in Table 1 illustrate the effectiveness of the best-performing regression model for this data set, that built under least-squares and incorporating two independent variables. Least-squares was selected as it had the best MMRE by far and was superior, or equivalent to, all but one of the Pred(l) measures. One very simple way in which such an evaluation can be performed in a quantitative manner is to assign rankings to the performance of each model according to each error measure and then determine the average ranking for each approach – this has been used in this investigation for illustrative purposes. For instance, in Table 2 the performance of the models in terms of minimizing MMRE would see the LS approach ranked 1, the LMS model ranked 2, and the LTS model ranked 3. Where a tie is evident, the sum of the rankings can be divided by the number of tied models and each assigned the same value. For example, in the Pred(10) column of Table 2 the LMS model is best (maximized) so it receives a ranking value of 1, but the remaining two models are tied. The ranks to be awarded are 2 and 3 so we can simply total these two values (5), divide this sum by the number of tied models (2) and award each tied model this value (2.5). Finally, we can total the ranking values across the error criteria and then divide it by the number of error measures (in this case 5) to get an average ranking value. The model with the lowest average ranking value can then be selected as the optimal model for this data set. (Note: this is a very simple approach – for one thing it assumes that all the criteria are equally important and that the magnitude of differences between error values is not important. These may not be realistic assumptions in practice – however this is simply an illustration, and the specific approach chosen will vary from organization to organization depending on their measurement goals.) Applying such an approach to this data set, the LS approach receives an average ranking value of 1.6, the LMS approach 2.1, and the LTS method 2.3, suggesting that the LS method is optimal in this case.

Under the LS model the relative error measures indicate adequate but unspectacular performance, but the average absolute error suggests some significant problems are still evident in the model. Interestingly, we were unable to replicate the superior performance of the robust regression methods achieved by the original authors (Table 2 shows our results using robust regression) – this may have been due to the selection of different robust methods in this case, or an artifact of the data splitting procedure.

Table 1. Selected model and results for Miyazaki, Terakado, Ozaki, and Nozaki data set - LS regression (Actual model: Effort = -18.754 + 1.753FORM + 1.457FILE)

	Development Data	Testing Data	All Data
MMRE	1.01	0.76	0.92
Pred(10)	0.16	0.13	0.15
Pred(25)	0.34	0.25	0.31
Pred(35)	0.41	0.38	0.40
Pred(50)	0.50	0.44	0.48
BMMRE	0.47	0.46	0.46
AR(10)	0.16	0.13	0.15
AR(25)	0.38	0.31	0.35
AR(35)	0.44	0.44	0.44
AR(50)	0.59	0.56	0.58
AAR	29.2	78.3	45.5

Table 2. Performance of regression models on testing data for Miyazaki et al.

	MMRE	Pred (10)	Pred (25)	Pred (35)	Pred (50)
Least Squares	0.76	0.13	0.25	0.38	0.44
Least Median Squares	1.03	0.19	0.19	0.25	0.44
Least Trimmed Squares	1.24	0.13	0.19	0.38	0.44

3.1.2 Li and Henry data set

Of particular note in relation to the effectiveness of the optimal regression model chosen here (the LMS method) are the Pred(*l*) thresholds and average absolute error (Table 3). The LMS method was chosen on the basis of the average ranking value comparison described above, in this case resulting in values of 1.9 for LS, 1.8 for LMS and 2.3 for LTS. Given the closeness of the results it is clear that LMS is not unambiguously the best model, but it seems sensible to treat it as best on the basis of the average ranking value comparison. Whilst the MMRE indicates some modeling problems over the data set as a whole, the Pred(*l*) and AR indicators suggest that performance for specific observations is reasonably sound. The gains from robust regression are fairly small, but nonetheless robust regression could be seen as slightly better performing (Table 4) in terms of accuracy.

3.1.3 Dolado data set

Analysis of the Dolado data resulted in the selection of the least trimmed squares model, based on average ranking values of 2.3 for LS, 2.5 for LMS and 1.2 for LTS. Both the MMRE and Pred(*l*) indicators suggest that the LTS model fits the data reasonably well, and it is unambiguously superior in terms of accuracy to the two other models presented here. A need for caution is borne out, however, by the high value for average absolute error, suggesting significant problems with the model. The use of the robust LTS regression seems to assist with some of the problems in this data set (Table 6).

3.2. Neural Network Predictions

All of the neural network models were developed in the same manner, by following these steps:

1. The data set for model development was split into the

training and validation sets (roughly two-thirds and one-third respectively). The data sets are exactly the same as those used in the regression cases above, with the regression development set equal to the training and validation sets here. The test data sets are identical in all cases and this enables comparisons between models in terms of using available information.

2. A simulation was prepared that created networks with a variety of architectures (the number of hidden units was varied from one to ten in each case) and different training parameters.

Table 3. Selected model and results for Li and Henry data set – LMS regression (Actual model: Changes = 8.075 + 0.259SIZE1 - 1.560SIZE2)

	Development Data	Testing Data	All Data
MMRE	0.31	0.37	0.33
Pred(10)	0.26	0.21	0.24
Pred(25)	0.49	0.25	0.41
Pred(35)	0.64	0.46	0.58
Pred(50)	0.85	0.67	0.79
BMMRE	0.25	0.34	0.28
AR(10)	0.26	0.25	0.25
AR(25)	0.53	0.29	0.45
AR(35)	0.70	0.46	0.62
AR(50)	0.94	0.75	0.87
AAR	15.7	29.9	20.5

Table 4. Performance of regression models on testing data for Li and Henry

	MMRE	Pred (10)	Pred (25)	Pred (35)	Pred (50)
Least Squares	0.42	0.21	0.38	0.42	0.79
Least Median Squares	0.37	0.21	0.25	0.46	0.67
Least Trimmed Squares	0.37	0.13	0.38	0.42	0.63

Table 5. Selected model and results for Dolado data set – LTS regression (Actual model: Effort = 149.4 + 1.39INDE)

	Development Data	Testing Data	All Data
MMRE	0.36	0.31	0.34
Pred(10)	0.31	0.25	0.29
Pred(25)	0.50	0.63	0.54
Pred(35)	0.56	0.63	0.58
Pred(50)	0.81	0.75	0.79
BMMRE	0.25	0.30	0.27
AR(10)	0.38	0.25	0.33
AR(25)	0.50	0.63	0.54

	Development Data	Testing Data	All Data
AR(35)	0.63	0.63	0.63
AR(50)	0.94	0.75	0.88
AAR	71.5	158.2	100.4

Table 6. Performance of regression models on testing data for Dolado

	MMRE	Pred (10)	Pred (25)	Pred (35)	Pred (50)
Least Squares	0.34	0.13	0.50	0.50	0.75
Least Median Squares	0.36	0.13	0.50	0.63	0.63
Least Trimmed Squares	0.31	0.25	0.63	0.63	0.75

3. The networks were then trained using the training data set, until the lowest error on the validation data set was achieved. It should be noted that because a number of simulations were performed (as mentioned in step 2), the chance of the validation data set fitting better than the training is greater than normal.

4. The best network, in terms of having the lowest validation data set error, was used to recall (make predictions) for the testing data.

5. A number of error measures were then calculated on the network's performance for the three splits of each data set.

3.2.1 Miyazaki, Terakado, Ozaki, and Nozaki data set

With this data set the best architecture was a one-hidden node network. All inputs were used, and the training continued for 9540 epochs (in steps of 10 epochs). The network achieved a low average absolute error, although the relative errors are significantly higher than would be desired (Table 7). This is reflected by poor predictions for the very small systems, where small absolute errors lead to large relative ones.

These results compare favorably to those presented in Miyazaki et al. (1994) where the BMMREs were 0.663 and 2.032 for robust and least-squares regression respectively. It should be noted that these figures from Miyazaki et al. were based on the entire data set, and that the neural network model performs significantly better than their regression models, even on new, unseen, data.

3.2.2 Li and Henry data set

The best network found for this data set used five hidden nodes, and was trained for 260 epochs (in steps of 10 epochs) (Table 8). There is some evidence that the network found was over-fitted to the validation data (recall that a large number of simulations were performed), and perhaps performs less effectively on the training and testing data than would be desirable. This illustrates the dangers of running a large number of simulations, where even selecting on the basis of withheld data performance does not always ensure generalisability. As was mentioned above, a selection data set could have been used here to attempt to overcome this problem but with such small data sets this is impractical.

3.2.3 Dolado data set

The best performing network for this small data set used one hidden node, trained for 990 epochs (in steps of 10 epochs) (Table 9) and using four main inputs. This network shows definite signs of overfitting to the validation data, partly due to the small size of the data set.

3.3. Comparing the Predictions

The above results suggest that no single modeling technique outperforms the others on all data sets, at least in regard to measures of predictive error. The results are summarized in Table 10, showing the optimum MMRE and Pred(*l*) error measures achieved under the 'best' regression method and neural network architecture. For the Miyazaki et al. (1994) data the neural network performs better on the testing data in terms of the prediction threshold error measure, while for the Li and Henry (1993) data the LMS regression model performs slightly better on this criteria. On the Dolado (1997) data set the regression technique, LTS this time, is superior based on the prediction threshold error measure. In all cases the regression models perform better according to the MMRE measure. This may suggest that where MMRE is more important than Pred(*l*) performance, regression should be used over neural network models and vice versa.

4. SELECTING THE OPTIMAL TECHNIQUE

In this paper a number of different data analysis and modeling techniques have been described and tested on case study data sets. It has been suggested that use of the most appropriate technique can lead to more *accurate* models; however, this should not be the only criterion used for selecting a technique. This section discusses a subset of the issues that need to be considered when making such a decision. These include the quantities of data available, the nature of the relationships in that data, the accuracy required from the model, the available expertise, and the necessity to communicate and learn from the models produced. Many of these are qualitative and difficult to measure, but must still be kept in mind by the metrician.

Table 7. Results for Miyazaki, Terakado, Ozaki, and Nozaki data set neural network performance

	Training Data	Validating Data	Testing Data	All Data
MMRE	0.75	1.10	1.32	1.02
Pred(10)	0.24	0.55	0.25	0.31
Pred(25)	0.33	0.55	0.50	0.44
Pred(35)	0.52	0.55	0.50	0.52
Pred(50)	0.62	0.73	0.56	0.63
BMMRE	0.36	0.21	0.36	0.33
AR(10)	0.24	0.55	0.31	0.33
AR(25)	0.43	0.55	0.50	0.48
AR(35)	0.57	0.64	0.50	0.56
AR(50)	0.76	0.82	0.69	0.75
AAR	22.4	16.4	27.0	22.5

Table 8. Results for Li and Henry data set neural network performance

	Training Data	Validating Data	Testing Data	All Data
MMRE	0.49	0.39	0.51	0.47
Pred(10)	0.19	0.19	0.08	0.15
Pred(25)	0.45	0.44	0.29	0.39
Pred(35)	0.55	0.56	0.42	0.51
Pred(50)	0.84	0.69	0.58	0.72
BMMRE	0.29	0.27	0.37	0.31
AR(10)	0.19	0.19	0.08	0.15
AR(25)	0.45	0.50	0.38	0.44
AR(35)	0.65	0.69	0.46	0.59
AR(50)	0.87	0.94	0.63	0.80
AAR	21.0	19.4	32.9	24.7

Table 9. Results for Dolado data set neural network performance

	Training Data	Validating Data	Testing Data	All Data
MMRE	0.46	0.21	0.35	0.36
Pred(10)	0.10	0.17	0.25	0.17
Pred(25)	0.50	0.67	0.38	0.50
Pred(35)	0.70	0.83	0.50	0.67
Pred(50)	0.80	1.00	0.63	0.79
BMMRE	0.29	0.21	0.32	0.28
AR(10)	0.10	0.17	0.25	0.17
AR(25)	0.50	0.67	0.50	0.54
AR(35)	0.70	0.83	0.63	0.71
AR(50)	0.80	1.00	0.63	0.79
AAR	82.0	64.7	163.8	104.9

Table 10. Summary of results on testing sets (accuracy only)

	Miyazaki et al. (1994)		Li and Henry (1993)		Dolado (1997)	
	Regression – LS	NN – 1 hidden	Regression – LMS	NN – 5 hidden	Regression – LTS	NN – 1 hidden
MMRE	0.76	1.32	0.37	0.51	0.31	0.35
Pred(10)	0.13	0.25	0.21	0.08	0.25	0.25
Pred(25)	0.25	0.50	0.25	0.29	0.63	0.38
Pred(35)	0.38	0.50	0.46	0.42	0.63	0.50
Pred(50)	0.44	0.56	0.67	0.58	0.75	0.63

4.1. Quantities of Data

It is well known that building more complex statistical models requires greater amounts of data, because more parameters need to be determined. This is often a motivation for software metricians to develop simple models for predicting the development process. The same principle applies to neural network modeling. Since there are many more parameters (a network with 6 inputs, four hidden units, and one output generally contains some 33 adjustable weights), it is important that reasonable amounts of data are available (although techniques such as bootstrapping can be used to overcome some deficiencies in data quantity). In relation to the empirical evaluations shown above, for example, use of the neural network method with the Dolado data set was ineffective, possibly partially because of the small number of observations available.

4.2. Nature of Relationships

The choice of technique also depends on what type of relationship (if any) the metrician anticipates finding. For many systems a simple linear model may largely reflect the underlying physical phenomena (for example, the effort required to check test cases where each case takes the same length of time). In such situations, the use of a non-linear model such as a neural network would represent unnecessary complexity. In a similar manner, the use of robust regression techniques can really only be justified if the data may contain some outliers. While software metricians may well be faced with such contaminated data sets, there are situations where least-squares regression will perform more effectively. In our empirical evaluation above, for instance, this was the case for the Miyazaki et al. data.

4.3. Accuracy Required from the Model

As a general rule the accuracy of a model increases as the complexity of the modeling technique rises (although as stated previously overfitting can be a problem). It is vitally important to distinguish between accuracy in learning the relationships contained in the training/development data, and in learning the relationships inherent in the physical realization of the process.

The case studies above illustrate that more complex techniques such as neural networks are potentially capable of fitting more accurate models to the training data by virtue of their non-linearities and interactions. In the same way, using transformations, more independent variables, and interaction terms in a regression model can have the same effect. However this opens up the risk of over-fitting, even with all due care taken, when using small data sets such as are common for software metrics research. It should also be recalled that it is only *in potential* that such increases in accuracy exist. They will not necessarily be realized in all cases.

In some cases, however, the accuracy requirements for a model may be low when compared to other goals, and these other motivations may be overriding, especially where the improvements in accuracy are small. Gaining an *understanding* of what it is that makes a system more error-prone, for instance, may be much more important to a manager than a 5% increase in predictive accuracy that may be gained from a much more complex and costly model. It is also important to specify pre-analysis what is meant by accuracy for a particular modeling task. In some cases absolute errors are of concern, while in others relative errors are more important. Also, a choice must be made between the relative importance of threshold-based errors (like Pred(l)) and average errors. These issues are certainly not trivial ones, to be faced with the same solution for all projects.

4.4. Available Expertise

The availability of sufficient modeling expertise may be seen as perhaps one of the most restrictive requirement for the technique selection process. While there has been enthusiastic adoption of neural network models in many fields, including software metrics, there is often an under-appreciation of the expertise required for their use. This is not assisted by the all-too-common perception of such models as automatic, universal approximators that overcome all of the problems with traditional statistical methods. In fact, the effective use of neural network models requires at least as much expertise as non-linear regression, and in some ways may require even more as their flexibility opens up new pitfalls.

4.5. Communicating and Learning from Models

Finally, there is the issue of interpretability of the models in their final form. Regression equations (both least-squares and robust) have the advantage of being presented in a manner that most managers can understand. It is relatively easy to see the influence of a single independent variable on the model's output. However, by using a black box technique such as a neural network, the opportunities to view the form of the model are limited. While the weights are available, it is difficult to then interpret these in a meaningful manner.

Some options for extracting rules from neural networks do exist (Kasabov, 1996; Wang and Mendel, 1992). However, these techniques then require even more expertise and the software is much less accessible, still being an area of research rather than widespread practice. A simpler approach is sensitivity analysis where the network's outputs for a variety of inputs are plotted, holding all but one input variable (or perhaps two variables) constant. This can be useful for confirming relationships in the model (as with examining the slopes in regression models).

4.6. The Actual Selection of the Technique

The factors that have been briefly mentioned above all need to be considered in conjunction. For a particular project, the relative importance of each should be assessed, along with the advantages/disadvantages of each technique when compared to the others. In many cases the best solution is to not select any one technique, but rather to develop models using two or more methods. This may have the consequent advantage of discouraging absolute reliance on a single number that is produced as model output – it is only an *estimate*, after all. If more than one technique is used, a *range* of estimates will result.

More information on these techniques from a software metrics perspective, and other techniques such as fuzzy logic and case-based reasoning, is available in Gray and MacDonell (1997) and MacDonell and Gray (1997). It is only by using these techniques on a wide range of data sets than any useful guidelines to their general use can be formulated. The dangers of relying on another's (or even your own) success with a technique on a single data set are obvious.

5. CONCLUSIONS

In this paper we have investigated a number of issues that need to be considered as part of the model building process for software metrics. While the most overriding consideration is generally the accuracy of the model, in particular for predictions on new data, other issues such as data characteristics, expertise, and interpretability should also be taken into account. While many of these issues are difficult to quantify, some consideration must still be made. In order to make these decisions the software metrician needs to be aware of the techniques, both in terms of strengths/weaknesses and in terms of correct usage.

We have demonstrated that empirically the techniques can perform quite differently on various data sets, thus leading to the conclusion that no one technique can be used as a panacea for software metrics' analysis problems. Our intention is to continue with this area of research and formulate a more rigorous examination of the data set characteristics of software metrics, and the effects of these on the modeling process.

REFERENCES

Albrecht, A.J. and Gaffney, J.E., Jr. 1983. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE Transactions on Software Engineering* 9(6): 639-648

Boehm, B.W. 1981. *Software Engineering Economics*. Englewood Cliffs NJ: Prentice-Hall.

Dolado, J.J. 1997. A study of the relationships among Albrecht and Mark II function points, lines of code 4GL and effort. *Journal of Systems and Software* 37: 161-173.

Ebrahimi, N.B. 1999. How to improve the calibration of cost models. *IEEE Transactions on Software Engineering* 25(1): 136-140.

Finnie, G.R., Wittig, G.E. and Desharnais, J.-M. 1997. A comparision of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software* 39: 281-289.

Gray, A.R., and MacDonell, S.G. 1997. A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology* 39: 425-437.

Hakkarainen, J., Laamanen, P. and Rask, R. 1993. Neural networks in specification level software size estimation. In *Proceedings of the 26th Hawaii International Conference on System Sciences*. Hawaii, USA, IEEE Computer Society Press, 626-634.

Heiat, A. and Heiat, N. 1997. A model for estimating efforts required for developing small-scale business applications. *Journal of Systems and Software* 39: 7-14.

Hertz, J., Krogh, A., and Palmer, R.G. 1991. *Introduction to the Theory of Neural Computation*. Redwood City CA: Addison-Wesley.

Hornik, K., Stinchcombe, M., and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2: 359-366.

Kasabov, N.K. 1996. *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering*. Cambridge MA: MIT Press.

Lanubile, F. and Visaggio, G. 1997. Evaluating predictive quality models derived from software measures: lessons learned. *Journal of Systems and Software* 38: 225-234.

Lee, A., Cheng, C.H. and Balakrishnan, J. 1998. Software development cost estimation: integrating neural network with cluster analysis. *Information & Management* 34: 1-9.

Li, W., and Henry, S. 1993. Object-oriented metrics that predict maintainability. *Journal of Systems and Software* 23: 111-122.

MacDonell, S.G., and Gray, A.R. A comparison of modeling techniques for software development effort prediction. *Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems*, Dunedin, New Zealand, 869-872.

Miyazaki, Y., Terakado, M., Ozaki, K., and Nozaki, H. 1994. Robust regression for developing software estimation models. *Journal of Systems and Software* 27: 3-16.

Neter, J., Kutner, M.H., Nachtsheim, C.J., and Wasserman, W. 1996. *Applied Linear Statistical Models*. Chicago: Irwin.

Putnam, L.H., Putnam, D.T. and Thayer, L.P. 1984. A tool for planning software projects. *Journal of Systems and Software* 5: 147-154

Rousseeuw, P.J., and Leroy, A.M. 1987. *Robust Regression and Outlier Detection.* New York NY, USA: John Wiley & Sons.

Stensrud, E. and Myrtveit, I. 1998. Human performance estimating with analogy and regression models: an empirical validation. In *Proceedings of the Fifth International Software Metrics Symposium (Metrics'98).* Los Alamitos, California, IEEE Computer Society Press, 205-213.

Wang, L.-X., and Mendel, J.M. 1992. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics* 22: 1414-1427.