Limits to the Methods in Software Cost Estimation

J. Javier DOLADO*
Dept. of Computer Languages and Systems
Univ. of the Basque Country, Spain; dolado@si.ehu.es

Abstract

We present some conclusions related to the use of classical regression, neural networks (NN) and genetic programming (GP) for software cost estimation. Although the estimates of classical regression can be improved by NN and GP, the results are not very impressive. We conclude that either the data points limit the usefulness of the methods, or that better ways have to be found for applying soft-computing techniques for software cost estimation.

1 Introduction

The estimation of cost is one of the more daunting tasks confronted by the software manager at the beginning of the projects. The methods used for making estimates are varied, ranging from expert judgement, classical regression, neural networks, fuzzy logic, case-based reasoning and, lately, genetic programming.

Classical regression or linear regression (LR) is the most common method for adjusting data and making estimates. Although it is an established field, classical regression has many problems related to the assumptions that have to be made about the distribution and normality of the data. Soft-computing methods provide alternatives for solving these and other problems. NN is already a well-known methodology for learning patterns of data and making extrapolations. Neural networks have the capability of approximating any function. Genetic programming [4] is a new technique that has proved to be effective in modelling data and in generating equations for describing processes [5][6].

It is usually believed that the predictions may depend on the method used to derive the estimations. In software cost estimation the need for improving the predictions leads to the use of new methods, while maintaining the variables under study. In the present work we have tried to improve the predictions of the effort-size relationship by using alternative methods for obtaining the estimates.

2 Application of Three Methods

Therefore we have applied these three techniques (LR, NN and GP) to the problems of software cost estimation. We report some other results of the application of these three methods in [2] [3], and we summarize in Table 1 all the results related to the effort-size relationship which have been found on the datasets cited in section 3.

*Work supported by UPV–EHU 141.226 EA 083/98 and CICYT TIC98 1179–E

The criteria for comparing the three methods on the datasets is to compute the values of several predictive parameters. In the case of multiple linear regression an assessment is made by the R^2 (coefficient of multiple determination) and other statistical criteria in order to select the best equation. These variables can be considered *explanation* variables, but not *predictive* ones.

We consider, as other authors do, that the best predictive parameters are the *mean magnitude of the relative error* and the *Prediction at level l* (see [1]). Mean magnitude of relative error (*mmre*), is defined as $mmre = 1/n \sum_1^n |(e_i - \hat{e}_i)/e_i|$, where e_i is the actual value of the effort variable in a project, $\hat{e}_i$ is its estimate and n is the number of projects. Thus if the $mmre$ is small, then we have a good set of predictions. A usual criteria for accepting a model as good is that the model has a $mmre \leq 0.25$. As lower is the value of the $mmre$ as better it is.

Prediction at Level l ($Pred(l)$), where l is a percentage, is defined as the quotient of number of cases in which the estimates are within the l absolute limit of the actual values divided by the total number of cases. For example $Pred(0.1) = 0.9$ means that 90% of the cases have estimates within the 10% range of their actual values. A standard criteria for considering a model as acceptable is $Pred(0.25) \geq 0.75$. This means that at least 75% of the estimates are within the range of the 25% of the actual values. As greater is the value of the Pred(l) as better it is. Next we briefly comment some aspects of the methods of analysis.

2.1 Classical Regression

Linear regression is one of the usual techniques for analyzing and building estimation models. It is assumed that the dependent variable is linearly related with the independent(s) variable(s). In order to have other functions or equations, several transformations can be made in the variables, and usual transformations take the logarithm of the variables. Goodness of fit of the model is usually measured by the coefficient of multiple determination R^2, and by the *mean squared error*. Outlier analysis is also a requirement for asserting the validity of a model.

The main problems of this method lie in its dependence to the distribution and normality of the data points, and the inability to approximate unknown functions or highly nonlinear functions with ease. We have used the statistical package SPSS in our analysis.

2.2 Neural Networks

For the purposes of our work we have used feed-forward NN with two layers and two neurons in the hidden layer. We have tested other configurations without obtaining any significant benefit, and with the drawback that overfitting occurred. This type of networks are able of approximating any function (they are universal approximators). The simulations have been carried out with the Neural Network Toolbox of Matlab.

The appeal for using NN is that if there is an underlying relationship in the data under study, a neural net should be able to find it. The values of the predictions should show how good or bad the approximation is. The drawback of this method lies in the fact that the analyst can't manipulate the parameters of the net once the learning phase has finished. NN in this work act as nonlinear regression models. The hidden layer is composed of two neurons that simulate a logistic-sigmoidal function. The output layer is a linear transfer function.

To apply the learning algorithms correctly, all data is normalized and then denormalized afterwards for evaluation of the predictions. The inclusion of the maximum and minimum of each variable is a necessary condition for good generalization. The learning algorithms used are the Levenberg-Marquardt and the backpropagation methods, the former being faster. The initial weights and biases of the neurons were chosen randomly. Multiple runs were performed to find the correct settings of the parameters.

2.3 Genetic Programming

The genetic programming technique is a type of evolutionary computation technique. It was proposed by Koza [4] after the author tried to automatically evolve lisp-programs. GP has been used in a variety of fields, including the automated synthesis of circuits, nonlinear system identification in chemical process engineering (identifying relevant variables), symbolic regression, etc. Here, genetic programming is used as an automated symbolic regression method to derive the equations.

GP is a nonparametric method since it does not make any assumption about the distribution of the data, and derives the equations according only to fitted values. The use that is made here can be considered an alternative to curve estimation (and Linear Regression), and will also allow us to compare the linear equations with those derived automatically. Since a set of different equations are derived in different runs, only those that give the best results in the evaluation are reported. The equation compared in the present work is the one that gives the best fit in the last generation.

Each run had an initial population of 25 to 50 randomly generated equations. The number of generations in each run varied, but the best results were obtained (surprisingly) with only 3 to 5 generations. The number of equations that remained from one generation to the next represented 10% of the previous one.

The new equations of the new generation were formed by the usual operators (each equation in the population is represented as a tree):

a) *crossover*: two equations exchange parts, preserving the syntax of the mathematical expression. The operation of crossover takes a subtree of two equations and exchanges them to form new members of the populations.
The functional set used is: $+, -, *, /, power, sqrt, square, log, exp$.

b) *mutation*: Mutation randomly modifies a subtree of an equation, i.e., a term of the equation (function, variable or constant) is changed randomly.

Probabilities of mutation and crossover were in the range of $Pm = 0.8$ and $Pc = 0.2$, and multiple combinations were tested. In the implementation used here, for every tree –i.e., symbolic equation–, regression constants in the equations are determined using the Levenberg-Marquardt method of nonlinear least-square optimization. One problem posed by GP is that some limit to the trees has to be set in order to generate simpler expressions. Here the best results were obtained by limiting the number of generations.

The fitness measure in the present work is the mean squared error of the predicted values of the equation. Other works have used the correlation coefficient between the independent and dependent variables. Here, however, the correlation coefficient did not work as expected, and it was useless as fitness measure. The implemented strategy for selecting equations from a generation is the *fitness proportional selection* strategy. In this way an individual is selected from the parent population with a probability proportional to its fitness. The software for GP has been provided by the Newcastle Symbolic Optimisation Group (runs in Matlab 4.2.c) [5][6].

3 The Data sets

We have used several available data sets relating software effort to size, except for the dataset of Matson et al., where we have used an approximation to the original data points (therefore results should be taken with caution). Measures of effort include person-hours and person-months. Measures of software size include lines of code and function points.

Data sets	**No.**	**Effort**	**Size**
Belady and Lehman, 1979	33	person–months	LOC
Boehm, 1981	63	person–months	LOC
Albrecht, Gaffney 1983 plus Kemerer data, 1987	24+15	person–months	Function Points
Dolado, 1997	48	person-hours	LOC
Abran and Robillard, 1996	21	person–days	Function Points
Miyazaki et al., 1994	47	person–months	Thousand LOC
Approximation to the dataset of Matson et al., 1994	104	person–months	Function Points

4 The Predictions

Table 1 presents the main results of applying the three methods to the datasets. In this table we are looking for values higher than 75 in the field of Pred(0.25), and for values lower than 0.25 in the field mmre. We observe that both genetic programming and neural networks have the potential of behaving, at least, as well as the classical methods of linear regression and curve estimation. In some cases they behave significantly better, but in others the results are slightly worse. Only in one case we find acceptable values (Abran and Robillard, 1996), which were found by GP. In the rest of the data sets GP and NN are unable to reaching acceptable levels of prediction, although in some cases they obtain better values than LR. In two cases LR slightly surpasses the two other techniques.

Although it is not reported here, an analysis has been carried out for each dataset by using in each data set a sample with 64% of the data points for model building and the rest for model evaluation. The results have not shown any conclusion worthy of remark different than those obtained with all data points.

5 Discussion

Among the three techniques presented here, the one that has more flexibility from the point of view of approximating the data points is GP, since we can set constraints on the function being approximated. From the point of view of the human estimator GP allows us to explore a vast space of possibilities. The drawbacks of the neural network approach lies in the inability to generate a symbolic equation that could be interpreted by the analyst. But, what we would like to remark at this point is that the "soft-computing" methods haven't obtained impressive results compared to the classical ones. They can improve the predictions in some way, but the improvement is not as dramatic as to completely discard the old methods. The benefits we obtain come from having a different perspective on the estimations, and from the "safety check" that we make on the results.

We obtained similar results, with respect to the application of the methods, in a study about methods for software size estimation. In [3] we obtained similar or slightly better values in the estimates made by GP or NN than in those obtained by LR. Again, although

Data sets		**GP**	**NN**	**LR**
Belady and Lehman, 1979	Pred(0.25)	24.2	18.18	33.33
	mmre	0.848	1.2733	0.6258
Boehm, 1981	Pred(0.25)	15.9	12.70	17.46
	mmre	3.227	5.8576	1.1336
Albrecht, Gaffney 1983	Pred(0.25)	33.3	17.95	7.69
plus Kemerer data, 1987	mmre	0.684	2.123	1.18
Dolado, 1997	Pred(0.25)	43.8	43.75	37.99
	mmre	0.433	0.4211	0.4375
Abran and Robillard, 1996	Pred(0.25)	81	80.95	57.14
	mmre	0.221	0.279	0.2339
Miyazaki et al., 1994	Pred(0.25)	40.4	40.43	42.55
	mmre	0.643	0.5886	0.3999
Approximation to the	Pred(0.25)	30.8	30.77	27.88
dataset of Matson et al. 1994	mmre	0.958	1.1448	0.8485

Table 1: Predictive values of the methods in the data sets

some improvements were achieved, they were not as dramatic as to assert that soft-computing methods are definitely better for making predictions.

We have seen that both genetic programming and neural networks have the capability of predicting the effort-size relationship similarly to curve estimation, at the least. In other cases GP behaves better, since it also provides an equation. The drawback of the NN lies in the fact that although it predicts well, no other hint is provided to the human estimator about possible improvements in the function. Ideally, several methods should be used to make predictions, because each one has its benefits.

As the new methods exploit the numeric data trying only to replace the old methods in the manipulation of the data, the success will be limited. Since the capabilities of the new approaches are huge, as it happens for example in evolutionary computation, the research should be focused on how to integrate other information besides the pure numeric pairs of data points. Soft-computing techniques are equally or more powerful than other classical methods in approximating functions, but in order to finally improve the estimations, simple data such as that used for generating Table 1 may not be enough.

References

[1] Conte, S.D., H.E. Dunsmore, and V.Y. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings (1986).

[2] Dolado, J.J. and L. Fernández, "Genetic Programming, Neural Networks and Linear Regression in Software Project Estimation", *Proc. INSPIRE III, Process Improvement through Training and Education* (Hawkins, P., M. Ross, G. Staples, and J.B. Thompson eds.), British Computer Society, (1998), 157–171.

[3] Dolado, J.J., "A Validation of the Component-Based Method for Software Size Estimation", *IEEE Trans. on Software Engineering*, to appear.

[4] Koza, J.R., *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press (1992).

[5] McKay, B., M.J. Willis, and G.W. Barton, "Steady-State Modelling of Chemical Process Systems using Genetic Programming", *Computers and Chemical Engineering* **21(9)** (1997), 981–996.

[6] Willis, M., M. Hinchliffe, B. McKay and G.W. Barton, "Systems Modelling Using Genetic Programming", *Computers and Chemical Engineering,* **21**, supplement (1997), 1161–1166.