Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 81 (2008) 356–367
The Journal of Systems and Software
www.elsevier.com/locate/jss

An investigation of artificial neural networks based prediction systems in software project management

Iris Fabiana de Barcelos Tronto *, José Demísio Simões da Silva, Nilson Sant'Anna

Laboratory for Computing and Applied Mathematics – LAC, Brazilian National Institute for Space Research – INPE, Av. Astronautas, 1758, Jardim da Granja, Zip 13081-970, São José dos Campos, SP, Brazil

Available online 2 June 2007

Abstract

A critical issue in software project management is the accurate estimation of size, effort, resources, cost, and time spent in the development process. Underestimates may lead to time pressures that may compromise full functional development and the software testing process. Likewise, overestimates can result in noncompetitive budgets. In this paper, artificial neural network and stepwise regression based predictive models are investigated, aiming at offering alternative methods for those who do not believe in estimation models. The results presented in this paper compare the performance of both methods and indicate that these techniques are competitive with the APF, SLIM, and COCOMO methods.

Keywords: Software effort estimation; Predictive accuracy; Artificial neural networks; Linear regression; Data mining

1. Introduction

Software developing companies have faced a tremendous increase in the demand for new software products and services resulting from the continuous hardware and software development, and the world economic interaction phenomenon. The increase in demand has contributed to the increase of the competition among software producing and delivering companies to produce low cost and high quality software in short time.

A quality level and international productivity can be achieved through the use of effective software process management, by focusing on people, product, process, and project. The project requires planning and accompaniment supported by a group of activities, among which the estimates (of effort, resources, time, etc.) are fundamental to guide the other activities.

Effective software project estimation is one of the most challenging and important activities in software development. Proper project planning and control may be achieved with sound and reliable estimates. However, the software industry requires focusing on the efforts in order to improve project estimates.

Under-estimating a project may lead to under-staffing it (that may result in staff burnout), under-scoping the quality assurance effort (that may increase the risk of low quality deliveries), and setting short schedule (that may result in loss of credibility as deadlines are missed). For those who figure out a way to avoid this situation by generously padding the estimate, overestimating a project can be just as bad for the organization. If you give a project more resources than it really needs without sufficient scope controls it may use them. The project is then likely to cost more than it should (a negative impact on the bottom line), take longer to deliver than necessary (resulting in lost opportunities), and delay the use of the resources in the next project.

The four basic steps in software project estimation are: (1) estimate the size of the development product (this generally ends up in either Lines of Code or Function Points, but there are other possible units of measure); (2) estimate the effort in person-months or person-hours; (3) estimate

* Corresponding author. Tel.: +55 12 39456543; fax: +55 12 39456375.
E-mail addresses: iris_barcelos@lac.inpe.br (I.F. de Barcelos Tronto), demisio@lac.inpe.br (J.D.S. da Silva), nilson@lac.inpe.br (N. Sant'Anna).

0164-1212/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.05.011

the schedule in calendar-months; (4) estimate the project cost in dollars (or local currency). The predictive process involves the set of procedures presented in Fig. 1 (Agarval, 2001).

An accurate size estimate of the software to be built is the first step towards effectively determining software project efforts (Jones, 1986; Lai and Huang, 2003; Hasting and Sajeev, 2001; Briand and Wieczorek, 2002). However, according to a research report published by the Brazilian Ministry of Science and Technology – MCT in 2001, only 29% of the companies accomplished size estimates and 45.7% accomplished software effort estimate in Brazil (MCT, 2001). There is not a specific study that may identify the possible causes for low effort estimate indices. Possible causes may be the reliability and usability level of the models. The statistics presented by MCT Brazil, emphasizes the importance of using an effort estimate alternative approach, through which one may achieve reliable estimates with simple execution models.

Predicting software development effort with high precision is still a great challenge for project managers. Consequently, there is an ongoing, high level activity in this research field in order to build, to evaluate, and to recommend prediction techniques (for examples, see Boehm, 1981; Albrecht and Gaffney, 1983; Shepperd and Schofield, 1997; Zhong et al., 2004; Sentas et al., 2005; Bisio and Malabocchia, 1995; Myrtveit et al., 2005; Samson et al., 1997).

The prediction techniques are classified into three general categories:

(1) *Expert judgment* – This technique has been used widely. However, the estimates are accomplished through a way which is not explicit and, consequently it is not possible to repeat them. According to Gray et al. (1999), although expert judgment is always difficult to quantify, it can be an effective estimate tool when used as an adjustment factor for algorithmic models.
(2) *Algorithmic models* – These are the most popular models in the literature. They need to be calibrated or adjusted to local circumstances in order to try to establish the relationship between effort and one or more characteristics. Usually, the principal effort driver used in such models is software size (for instance, the amount of function points, source lines of code, use case points, pages, etc.). Some examples of algorithmic models are: the COCOMO model (Boehm, 1981); Function Points Analysis (Albrecht and Gaffney, 1983); and the SLIM model (Putnam, 1978).
(3) *Machine learning* – In the last decade, the machine learning approaches have been used as a complement or an alternative for the two previous categories. Examples of machine learning approaches include fuzzy logic (Kumar et al., 1994), regression trees (Selby and Porter, 1998), artificial neural networks (Srinivazan and Fisher, 1995), and case based reasoning (Shepeerd et al., 1996; Gray and MacDonell, 1997).

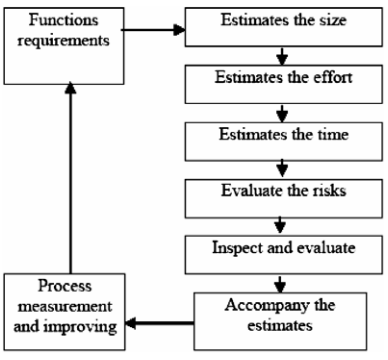

Fig. 1. The predictive process.

In the last 10 years these techniques have been applied in the context of software engineering estimation models. Given the availability of a variety of different predictive models (estimation models and predictive models are considered synonyms), recent research works have attempted at determining the best approach based mostly on one or more accuracy measures (Gray and MacDonell, 1997; Briand et al., 2000; Jeffery et al., 2001; Shepeerd et al., 1996; Angelis and Stamelos, 2000; Finnie et al., 1997). Since there are many factors that can vary from one study to another, for example the dataset can present different characteristics (outliers, co-linearity, number of variables, number of observations, etc.), it is not surprising that the outcomes of theses studies have not converged to similar answers, and thus there is still a doubt to advice practitioners as to what prediction models they should follow.

There are a number of factors that should be considered in the selection of a prediction technique, and it is likely that trade-offs will need to be made in the process. Technique choice should be driven by both organizational needs and capabilities. In terms of needs, the most common aim is to maximize the accuracy in prediction; however, other issues may also need to be considered. For instance, a technique that produces slightly less accurate predictions but generally more robust models might be preferred, especially in cases where the organizations do not have access to locally calibrated, well-behaved data sets. Whilst it is very positive that more sophisticated (and potentially more useful) techniques have been employed to build predictive models, genuine benefits will be achieved if such techniques are appropriately used.

Traditionally, research works have conducted comparative predictive techniques studies on the COCOMO dataset (Kitchenham, 1998; Sentas et al., 2005). For example, Barcelos Tronto et al. (2006a) report the results obtained by conducting experiments with regression analysis and artificial neural networks techniques to predict the software effort expressed as a function of size. The authors showed that the neural network model yielded more accurate estimate results than regression analysis by applying the

measurement of the mean magnitude of the relative error (MMRE), in Eq. (1)

$$\mathrm{MMRE} = \frac{\left(\sum_{i=1}^{n}\left|\frac{M_{\mathrm{est}}-M_{\mathrm{act}}}{M_{\mathrm{act}}}\right| * 100\right)}{n} \quad (1)$$

where n is the amount of projects; M_{act} is the actual effort; and M_{est} is the predicted effort.

This paper, however, focus on the investigation of the behavior of these techniques when other predicting variables with different characteristics (for instance, categorical variables) are used. The independent variables can be nominal, ordinal or in an absolute scale. Thus, a case study was performed to examine the accuracy of the predictions of the models using two approaches: (1) a multilayer perceptron neural network; (2) a stepwise multiple regression analysis based procedure. The COCOMO database (Boehm, 1981) is used as the dataset to learn and to test the models.

The paper is organized as follows: Section 2 provides some background on the different prediction techniques used as the basis for the conducted study. It is followed in Section 3 by a description of the regression analysis and artificial neural networks techniques. A case study is presented in Section 4 with the models obtained and the corresponding accuracy analysis. Finally, Section 5 concludes with a discussion on the significance of the results and ideas to the continuity of the research.

2. The related work

Accurate and consistent prediction of resource requirements is a crucial component in the effective management of software projects. Despite the amount of researches over the last 20 years, the software community is still significantly challenged when it comes to effective resource prediction. As a main stream, research efforts have focused on the development of quantitatively based techniques, in an effort to remove or reduce subjectivity in the estimation process. Examples of this work include the original parametric and regression-based models: Function Points Analysis (Albrecht, 1979), COCOMO Models (Boehm, 1981; Boehm et al., 2000), and the Ordinal Regression Model (Sentas et al., 2005).

However, other techniques for exploratory data analysis, such as clustering, case-based reasoning and ANN have been effective as means of predicting software project effort. Zhong et al. (2004) describe the use of clustering to predict software quality. A case-based approach called ESTOR was developed for software effort estimation (Vicinanza et al., 1990). They have shown that ESTOR is comparable to a specialist and performs significantly better than COCOMO and Function Points on restricted samples of problems. Some research works have used artificial neural networks to produce more accurate resource estimates Gray and MacDonell, 1997; Witting and Finnie, 1997).

In Karunanitthi et al. (1992), neural networks are used to predict software reliability. They conducted experiments with both feed-forward and Jordan networks and the cascade correlation learning algorithm. Witting and Finnie (1994) describe their use of the back propagation learning algorithm on a multilayer perceptron to predict development effort. The work of Samson et al. (1997) uses an Albus multiplayer perceptron in order to predict software effort on the Boehm's COCOMO dataset and hey compare a linear regression with a neural networks approach.

Srinivazan and Fisher (1995) also report the use of a neural network with a back propagation learning algorithm. They found that the neural network outperformed other techniques and led to results of MMRE = 70%. However, it is not clear how the dataset was divided for the training and the validation purposes.

Khoshgoftaar et al. (2000) presented a case study considering real time software to predict the testability of each module from source code static measures. They consider ANNs as promising techniques to build predictive models, because they are capable of modeling nonlinear relationships.

Finally, in the last years, the interest on the use of ANNs has grown. ANNs have been successfully applied to several problem domains, in areas such as medicine, engineering, geology, and physics, generally to design solutions for estimate, classification, control problems, etc. They can be used as predictive models because they are modeling techniques capable of modeling complex functions. Machine learning algorithms such as Artificial Neural Networks offer a means of addressing the problem of many factors. They are effective when you have a relatively large number of data points as well as a large number of factors.

This paper presents a case study using Artificial Neural Networks (ANN), and the regression analysis procedure to estimate software effort from the project size (given by the amount of lines of code source) and other cost and effort drivers. Next, we compare the results obtained with the two approaches.

3. The prediction techniques

3.1. Artificial neural network

ANNs are massively parallel systems inspired by the architecture of biological neural networks, comprising simple interconnected units (artificial neurons). The neuron computes a weighted sum of its inputs and generates an output if the sum exceeds a certain threshold. This output then becomes an excitatory (positive) or inhibitory (negative) input to other neurons in the network. The process continues until one or more outputs are generated.

The neural network results from the arrangement of such units in layers, which are interconnected one to another. The resulting architectures solve problems by learning the characteristics of the available data related

to the problem. There exist many different learning algorithms.

According to Gray and MacDonell (1997), neural networks are the most common software estimation model-building technique used as an alternative to mean least squares regression. They are estimation models that can be "trained" using historical data to produce ever better results by automatically adjusting their algorithmic parameter values to reduce the delta between known actual and model predictions. The most common form of a neural network used in the context of software estimation is a feed-forward multilayer perceptron network which is trained by the error back propagation algorithm that requires a differentiable activation function. The development of such a neural model first requires an appropriate layout of neurons, or connections between network nodes. This includes defining the number of layers of neurons, the number of neurons within each layer, and the manner in which they are all linked.

The training consists of searching for the best weight values among the nodes once the network has been built. This is accomplished by providing it with a set of historical project data inputs and the corresponding known actual values for project effort/cost. The ANN then iterates on its training algorithm, automatically adjusting the parameters until the least square mean error between the estimate and the actual values are within some pre-specified satisfactory range. The specification of this delta value is important. Without it, an ANN could theoretically become overtrained to the known historical data thus weakening the applicability of the algorithm to a broader set of more general data.

The ANN is initialized with random weights and gradually learns the implicit relationships among the training data by adjusting its weights when presented to these data.

In general the works concerned with the use of ANNs to predict software development effort have focused mostly on the accuracy comparison of algorithmic models rather than on the suitability of the approach for building software effort prediction systems. An example is the work of Witting and Finnie (1997). They explore the use of a multilayer (perceptron) neural network on the Desharnais and Australian Software Metrics Association (ASMA) data sets. For the Desharnais data set they randomly split the projects three times between 10 projects in the testing set and 71 projects in the training (a procedure we largely follow in our analysis). The results from three validations sets are aggregated and yield a high level of accuracy (Desharnais MMRE = 27% and ASMA MMRE = 17%) with some outlier values excluded. However, other factors such as exploratory value and configurability are equally important and also need to be investigated.

3.2. Linear regression

Linear regression attempts at finding linear relationship between one or more predictor parameters and a dependent variable, minimizing the mean square of the error across the range of observations in the data set. The philosophy is essentially one of solving local prediction problems before attempting at constructing universal models. The resulting prediction systems take the form

$$Y_{\text{est}} = \beta_0 + \beta_1 X_1, \ldots, \beta_n X_n \quad (2)$$

where Y_{est} is the estimated value and $X_1, \ldots, X_n$ are independent variables, such as project size (in source code lines) which the estimator found to significantly contribute to predicting effort.

A disadvantage with this technique is its vulnerability to extreme outlier values although robust regression techniques, that are less sensitivity to such problems, have been successfully used (Briand and Wieczorek, 2002). Another potential problem is the impact of co-linearity – the tendency of independent variables to be strongly correlated with one another – upon the stability of a regression type prediction system.

4. The case studies

This section describes the proposed neural network and multiple stepwise regression based analysis predictive methods, both of which were tailored and calibrated with the historical data. Thus, we consider the current process as a model-centered estimation process. The aim of the case study was to compare the performance of both methods by using the MMRE metric to allows us to assess the accuracy of the estimates. In this section, we explain the approach followed to build the models, describe the data preparation activities, and discuss the results.

4.1. The dataset

The dataset used to develop this case study is the COCOMO database (Boehm, 1981), which has been used in previous works in the software engineering literature (Sentas et al., 2005; Srinivazan and Fisher, 1995; Samson et al., 1997). It is a public dataset available in Boehm's book in which he uses it to describe and test one of the most important cost/effort estimative methods: the COCOMO model (Boehm, 1981; Boehm et al., 2000).

This dataset is comprised of 63 instances of software projects written using the programming languages COBOL, PLI, HMI, and FORTRAN, for mainly the business, scientific, and system software areas. The effort driver variables considered in this work are presented in Table 1. The total effort, given by the variable MMACT (the amount of man-hour for the software development), was considered the dependent variable. The independent variables are: code size and the cost drivers.

The code size is expressed in thousands of source lines of code (ADJKDSI). ADJKDSI is a continuous numeric variable while the cost drivers are categorical, variables (each category is represented by a continuous numeric value). Cost drivers are used to capture characteristics of the soft-

Table 1
Software development effort multipliers

Variable	Description
RELY	Reliability
DATA	Database size
CPLX	Application complexity
TIME	Restriction of time
STOR	Restriction of main store
VIRT	Volatility of virtual machine
TURN	Time of machine performance
ACAP	Analyst capability
AEXP	Experience with application
PCAP	Programmer capability
VEXP	Experience with virtual machine
LEXP	Experience with programming language
MODP	Use programming modern practice
TOOL	Use software tools
SCHED	Schedule for development
RVOL	Requirements volatility
ADJKDSI	Software size

ware development that effect the effort to complete the project. In COCOMO database all cost drivers have qualitative rating levels that express the impact of the drivers on development effort. These rating can range from Extra Low to Extra High. Each rating level of every cost driver has a value, called an effort multiplier (EM) associated with it. This scheme translates a cost driver's qualitative rating into a quantitative one for use in the model. The EM value assigned to a cost driver's nominal rating is 1.00. If a multiplicative cost driver's rating level causes more software development effort, then its corresponding EM is above 1.00. Conversely, if the rating level reduces the effort then the corresponding EM is less than 1.00 (Boehm et al., 2000).

The dataset has been preprocessed to attain a better discernibility power. This has been accomplished by performing numerical adjustments for each factor, in each project, rather than using specific factor levels as in the original COCOMO model. Thus, adaptations were conducted by replacing the numerical adjustments with an equivalent factor level. Table 2 shows the adjustment procedure used to convert from adjustment factors to levels for the TIME constraint. Appendix A presents the levels assigned to other adjustment factors.

It is important to observe that in the statistical literature the term, variables (independent variables), and factors are often used interchangeably.

Table 2
Levels for the TIME factor

Adjustment factor	Level assigned
1.0	1
1.06, 1.07, 1.08	2
1.11, 1.15	3
1.27, 1.30, 1.35	4
1.46, 1.66	5

Barcelos Tronto et al. (2006b) have applied neural networks and linear regression on the raw dataset to predict software effort. In this work the authors apply the same techniques on the COCOMO dataset adjusting the levels of the predictor factors by applying the analysis of variances (ANOVA) approach.

Analysis of variance is the usual method of deciding whether different levels of a factor affect a response variable (e.g., effort). It allows us to test whether there are significant differences between the factor levels' means. The purpose of this work, restricts to the average of the levels of each factor (or predicted variable).

4.2. Preparation of the variables

All of the 63 completed projects were used in the present analysis. Most of the variables are categorical (factors). Since the building of a reliable model requires the existence of enough observations, in every interaction of the values between dependent and independent variables, fewer categories were chosen to adjust the data. That is why a preliminary study was conducted in order to recode the categories of the predicting variables into homogeneous groups for each factor. This approach was based on the work of Angelis et al. (2001). Each one of these categorical variables were submitted to a one-way ANOVA process in order to check the impact of every factor on the original-dependent variable and to identify the various homogeneous categories that have to be concatenated in every factor. Through these tests it is possible to compare each category of a factor with all the other categories in the same factor and designate the significant difference. The categories that are not significantly different can be concatenated. The new categorical variables are presented in Table 3.

The only numerical independent variable in our analysis is size identified as ADJKDSI. This variable was then preprocessed by a logarithmic transformation to yield better fitting and prediction results.

4.3. Model predictive accuracy

The aim of this work is to evaluate and compare the predictive accuracy of the artificial neural networks and the regression models. There are several criteria by which estimation models can be judged. For example, it is usual to perform a crossvalidation exercise based on removing one data point at a time form the dataset, recalculating the model parameters and predicting the value of the omitted data point. However, this technique is not well-suited to a human-intensive model construction process. Another approach is to omit a random set of data points (called a test dataset), develop a model on the remaining data points (the learning dataset), and assess the predictive power of the model on the test dataset. An alternative is to use a mixed approach, by creating several different learning and test dataset pairs.

Table 3
New categorical variables

Factor	Original level	Concatenated level	Level name
MODP	1	1	Very high
	2 and 3	2	High
	4	3	Nominal
	5	4	Low
	6	5	Very low
PCAP	1	1	Very high
	2 and 3	2	High
	4	3	Nominal
	5, 6, 7	4	Below average
RVOL	1	1	Highly stable
	2 and 3	2	Stable
	4	3	Unstable
	5 and 6	4	Highly unstable
TOOL	1	1	Very high
	2 and 3	2	High
	4	3	Nominal
	5	4	Low
	6	5	Very low
RELY	1	1	Very high
	2 and 3	2	High
	4	3	Nominal
	5	4	Low
	6	5	Very low
ACAP	1	1	Very high
	2 and 3	2	High
	4	3	Nominal
	5 and 6	4	Low
	7	5	Very low

In order to reach this objective the COCOMO dataset was split into samples to train the learning system and samples to test the precision of the classifier in predicting software development effort. Thus, the assessment of the accuracy of the models follows a similar procedure as the one used in Kitchenham (1998), using a separate validation dataset. The training dataset was used to build the predictive model and the testing dataset was used to verify the efficacy of the predictive model.

Following this Kitchenham's procedure, six different pairs of learning and testing datasets were created by removing every sixth project starting (the first time) from the first project. For example, the first learning dataset was constructed by removing projects: 1, 7, 13, 19, 25, 31, 37, 43, 49, 55, and 61; the second learning dataset was constructed by removing projects: 2, 8, 14, 20, 26, 32, 38, 44, 50, 56, 62, and so forth. The removed projects comprised the respective testing datasets. Since the COCOMO dataset has an odd number (63) of projects, three learning datasets were formed with 52 projects and the other three with 53 projects.

Data sets could be intercalated with other projects (for example, 1, 4, 7, 10, etc.), but was followed a similar procedure as one used by Kitchenham (1998), which permitted obtain samples with reasonable length for the learning and testing datasets. Although may make a random choice of the data and use only one learning and testing dataset, this combination of six pairs of data sets can reduce the bias and prevent a bias recurrence of a possible software project data sequences.

The predictive accuracy of the models were evaluated by the mean magnitude of the relative error (MMRE), given by Eq. (1).

4.4. Generation of the ANN predictive model

The use of ANNs to predict software development effort does not require a priori knowledge and analysis of the data distribution. ANN based models maybe achieved by manipulating the data acquired through experimentation, provided the data are mapped into a n-dimensional unitary space $[0,1]^n$ where n is the number of input variables. This process is preceded by a normalization step whose objective is to minimize the bias of the input variables that present high values when compared to the other variables. The normalization is performed by establishing possible maximum values for each variable in a realistic way. The normalization process may also be performed by mapping the data to a hyper-space within the interval $[-1,1]^n$.

Visual analysis of the COCOMO dataset revealed two variables presenting very high values when compared to the others. Thus, they both were preprocessed by a logarithmic transformation in order to make their ranges compatible with the other variables.

The basic neural network architecture developed in this work consisted of one hidden layer with 23 neurons, with logistic sigmoidal activation function, and one output neuron with a linear activation function. Table 1 shows the input variables submitted to training process of the neural network to estimate the effort given by Ln(MMACT). The training phase was repeated five times, in an attempt to reach the best network to solve the problem. Besides, different neural network architectures were tried. But, the results presented in this paper (Table 4) correspond to the neural network with the best generalization performance on the testing dataset.

The predicted values using ANN, after the reclassification process of the variables are closer to actual values when compared with those obtained before the reclassification process, reported in a previous work (Barcelos Tronto et al., 2006b).

4.5. Generation of the regression model

This section illustrates how the analysis based on the regression technique works on the COCOMO dataset for effort prediction. Several experiments were conducted using multiple linear regression models to predict software effort on the six training datasets derived from the COCOMO dataset. There are, of course, many other types of models and the choice of a regression model is motivated principally by the need for simple models to support a preliminary attempt to understand software effort estimate. This

Table 4
Artificial neural network estimates

Conjunto1		Conjunto2		Conjunto3		Conjunto4		Conjunto5		Conjunto6	
Obs	Est	Obs	Est	Obs	Est	Obs	Est	Obs	Est	Obs	Est
2040	1801.4	1600	2120.8	243	175.2	240	221.0	33	30.1	43	18.2
8	9.9	1075	610.3	423	571.2	321	192.8	218	333.0	201	218.0
79	77.27	73	20.1	61	93.0	40	27.9	9	6.8	11400	3427.2
6600	6854.5	6400	4169.4	2455	685.7	724	947.2	539	601.6	453	157.8
523	871.7	387	222.0	88	139.8	98	77.4	7.3	5.8	5.9	17.1
1063	1156.1	702	888.8	605	347.0	230	137.9	82	82.7	55	94.2
47	64.5	12	22.8	8	9.0	8	11.2	6	8.1	45	41.4
83	72.8	87	130.8	106	86.6	126	114.9	36	36.7	1272	848.1
156	146.9	176	266.5	122	168.4	41	52.4	14	36.7	20	19.1
18	8.6	958	739.5	237	198.7	130	124.0	70	34.9	57	104.2
50	51.9	38	21.0	15	19.9						

section illustrates how the analysis is performed on the second learning dataset.

Since linear regression is a parametric technique, it requires data with an approximate normal distribution. Thus, the first step is to verify whether effort (given by the MMACT variable) is normally distributed. Fig. 2 shows the effect of producing a normal probability plot for the effort values obtained from the second learning dataset.

The scatterplot is clearly nonlinear; so it is reasonable to assume that the data is not normal distributed. In order to improve normality, it is necessary to transform the raw data. A natural transformation (i.e., log to base e) is often effective particularly if one wants to stabilize the variance. Fig. 3 shows the normal probability plot after a logarithmic transformation. The scatterplot is now much more linear, thus the analysis procedure is continued using the transformed data.

The effort variable was transformed to improve the normality. The regression models were developed by applying stepwise regression with forward elimination. An alpha-value of 0.05 was used to enter variables. Each one of the

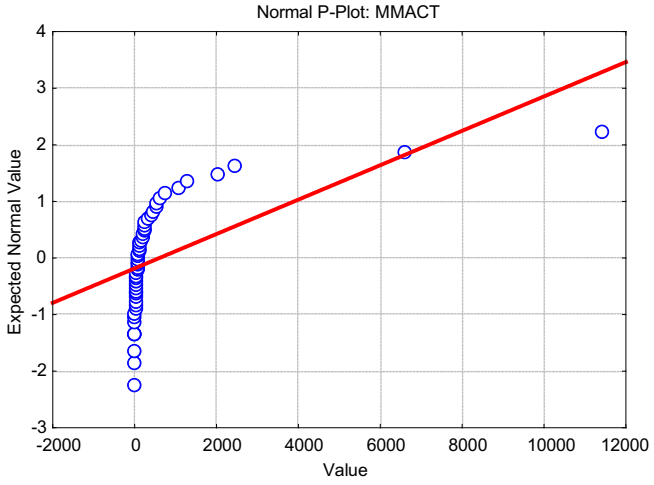

Fig. 2. Normal probability plot for effort.

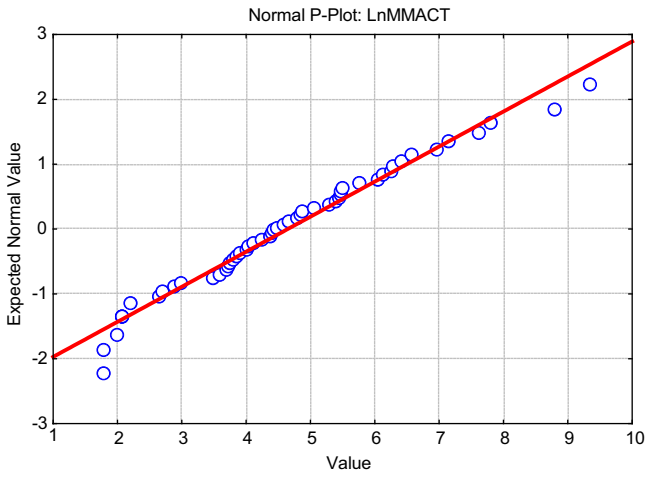

Fig. 3. Normal probability plot for transformed data.

models was calibrated on its respective learning dataset, by considering Ln(MMACT) the dependent variable. The independent variables are all the categorical variables presented in Table 1 and the project size variable given by Ln(ADJKDSI).

Statistical analysis was performed using General Regression Models (GRM) module of the Statistics software version 6.0 - to meet the 10 best subset-models, and the best model among them. The square of the multiple correlation coefficients (R^2) indicates the fit of the model. Where there are few observations and many variables, as in the present study, it is important to apply the R^2-adjusted coefficient instead of the R^2. The R^2-adjusted indicates the proportional reduction in the mean square deviation between actual and estimated effort, rather than the reduction in the sum of the squares.

The R^2-adjusted coefficient allowed direct comparison of the models found and supported the choice of the "best" model from them. For each learning dataset the model (among the 10 generated models) that led to the largest value of R^2-adjusted was chosen. Six models resulted from the six training datasets. It was observed that the model

parameters are not very stable for each of subset. The only variables present in all models are the Ln(ADJKDSI) and the RVOL.

Some variables of the best-models found present a non-significant beta-value where the considered level of significance is 0.05. Once the objective is to obtain a better predictive accuracy, a stepwise forward regression was applied taking as independent variables only those that present a significant beta-value on the model. The stepwise forward regression builds a prediction model where the variable which presents the largest partial correlation with the dependent variable is added to the model (at each step) by considering all the present variables in the model. We want to find a group of independent variables that maximize the F statistics, which evaluates if the independent variables are significantly associated with the dependent variable, when they are all jointly analyzed. An independent variable is added to the model if it increases the F-value for the regression, for some amount k. When an independent variable reduces the F-value, for some amount w, it is removed from the model. Table 5 shows the final regression models constructed for each learning subset.

The test datasets were used to assess the predictive accuracy of the models showed in Table 5. Since the models are based on transformed data, the resulting values need to be transformed back to the raw data scale to avoid unrealistic values. Table 6 shows the predict values for each testing dataset.

4.6. Comparison of the techniques

The main propose of this work is to compare the estimation accuracy of the neural network models with the accuracy of the multiple regression models. Two measures for estimating accuracy that are most popular in the cost estimation community are the mean magnitude relative error (MMRE) and the Pred(25) statistics, both of which were suggested by Conte et al. (1986). The mean magnitude of the relative error is calculated by Eq. (1). Pred(25) is the proportion of project estimates within 25% of the actual. For example, if Pre(25) is 0.60, then 60% of the estimates are within 25% of the actual. These statistics give an overall summary of estimate accuracy.

In this paper, the MMRE (Mean Magnitude of Relative Error) is adopted as an indicator of the predictive accuracy of the estimates produced by the models. Table 7 summarizes the MMRE values obtained through the estimates accomplished with the six testing datasets using ANN and regression analysis models. Pred(25) was not used for this case.

Other researchers have been using R^2-adjusted (or determination coefficient) to indicate the variation percentage in

Table 5
The final model for each learning dataset

Model	Regression equation
Dataset 1	LnMMACT = −0.71831 + 0.208257 * RELY + 0.599015 * RVOL + 1.08353 * LnADJKDSI
Dataset 2	LnMMACT = −2.2070 + 0.224217 * RELY + 0.081843 * DATA + 0.218017 * TIME + 0.085651 * STOR + 0.390645 * VIRT − 0.23117 * PLATFORM + 0.204759 * ACAP + 0.277066 * PCAP + 0.301366 * RVOL + 1.03700 * LnADJKDSI
Dataset 3	LnMMACT = −1.6174 + 0.286916 * RELY + 0.332576 * ACAP + 0.524855 * RVOL + 1.08734 * LnADJKDSI
Dataset 4	LnMMACT = −2.9387 + 0.149604 * RELY + 0.282402 * TIME + 0.524036 * ACAP + 0.311149 * LEXP + 0.188171 * MODP + 0.318291 * RVOL + 1.14161 * LnADJKDSI
Dataset 5	LnMMACT = −1.3577 + 0.137437 * ACAP + 0.616360 * RVOL + 0.541879 * MODE + 1.00858 * LnADJKDSI
Dataset 6	LnMMACT = −2.4885 + 0.148765 * RELY + 0.282343 * TIME + 0.320646 * ACAP + 0.260250 * MODP + 0.305558 * RVOL + 0.258748 * MODE + 1.07799 * LnADJKDSI

Table 6
The predict values for each test dataset

Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
748.08	1571.67	702.32	476.82	37.55	24.44
16.13	739.48	315.81	274.20	415.58	347.09
143.24	21.10	45.77	43.46	6.09	10284.05
7850.12	10560.55	1418.70	1072.68	459.47	263.82
780.92	342.20	124.70	141.95	5.26	3.11
390.74	926.76	340.37	227.81	166.71	52.56
61.44463	34.45	18.11	7.75	12.32	27.80
137.4543	148.46	118.19	209.11	41.13	2277.90
897.3218	204.26	132.93	117.28	16.09	14.07
18.00275	602.13	92.04	133.45	26.67	55.18
137.4543	23.25	30.45			

Table 7
The accuracy assessment for each test dataset

Test dataset	Artificial neural network		Regression analysis	
	MMRE	R^2	MMRE	R^2
1	36.47	0.99	102.21	0.92
2	46.52	0.92	51.126	0.98
3	37.95	0.64	61.64	0.83
4	25.19	0.91	47.36	0.91
5	39.87	0.96	47.92	0.80
6	63.17	0.98	36.93	0.98

the dependent variable that is explained in terms of the independent variables. In this work a linear regression analysis was accomplished to evaluate the predictions by considering M_{est} (estimated values) as the independent variable and M_{act} (real values) as the dependent variable.

Table 7 shows the R^2-adjusted values resulting from a linear regression of M_{act} (actual effort) values and M_{est} (estimated values) obtained from prediction using the same models as for the MMRE. The value of R^2-adjusted indicates the amount of variation in the real values of effort due to a linear relationship with the estimated values. When the values are close to 1.0, it suggests that there is a strong linear relationship and when they are close to 0.0, it suggests there is a nonlinear relationship.

The ANN results shown in Table 7 represent average values of 5 different experiments for each of the six datasets considered. The development of five experiments for each dataset attempted at minimizing biases that might have favored the performance of some datasets.

Table 8 shows the results obtained by Kemerer (1987) with the COCOMO-basic models, Function Points and SLIM.

These results indicate that predictions from linear regression and ANN show a strong linear relationship with the actual development effort values for all test projects. On the R^2 dimension, the performance of the ANN model is better than the SLIM's performance (0.89), the Function Point Analysis and the COCOMO models (0.58 and 0.70, respectively) in Kemerer's experiments. In terms of MMRE, the ANN performs strikingly well compared to the other approaches, and it is better than the regression based model.

For each of the experiments the predicted and observed efforts were compared. The results achieved show that neural network based approach performed better than regression analysis for five datasets in terms of MMRE. But it did not present a good performance for the sixth dataset. It is to be noticed that the neural network approach will perform better if the training datasets represent most of the possible software development scenarios. Thus, it is necessary to gather more data to make the model learn (be trained on) all the possible intrinsic characteristics of the data. Although the neural network fits showed to be better than the linear regression, these estimates maybe subjected to error and should be interpreted with caution.

Table 8
Other results on COCOMO dataset

	MMRE	R^2
Function Points Analysis	103	0.58
SLIM	772	0.89
COCOMO Basic	610	0.70

5. Conclusion and future works

This paper has compared a neural network based method with the linear regression approach to accomplish software effort estimation. Artificial neural network and multiple regressions were applied to Boehm's COCOMO dataset in order to predict effort from the cost drivers and software size variables. The ANN estimate results presented better accuracy indices than those obtained with multiple regressions, and simple linear regression to estimate effort from software size reported in previous work (Barcelos Tronto et al., 2006a).

It is known that the software size variable (ADJKDSI) presents the larger significance for effort estimate. Considering the COCOMO dataset, there are two projects with very large efforts that are out of all proportions to their sizes, as well as one with a small effort for its size, and a linear function of size will not be very successful at predicting these. On the other hand, an attempt to solve these outliers could influence negatively in the accuracy regression in accomplishing prediction for other observations.

Thus ANN should be chosen to model the complexities involved in the software effort estimate domain. Is has shown competitive results when compared to multiple regression, SLIM, COCOMO, and Function Points. A primary advantage of an ANN approach is that it is adaptable and nonparametric; thus predictive models can be tailored to the data at a particular site. Once the ANN is not limited to a linear function, it can deal more successfully with observations that lie far from the best straight line. The neural network based models are able to capture the parameters that influence in the development effort. A more homogeneous dataset with no outliers would show the regression method with a better advantage. The ANN approach would possibly perform better on such a dataset, but it would all depend upon the training phase.

The results obtained in this work are better than those obtained in a previous work (Barcelos Tronto et al., 2006b) in which the COCOMO dataset was not transformed nor it was pre-processed (using analysis of variance to establish the categories). Thus, we may infer the data preprocessing has impacted the accuracy of the software effort estimate model for neural networks and for the regression approaches when compared to existing effort prediction methods in the literature.

Although the ANN approach has demonstrated significant advantages for certain circumstances, it does not replace regression and should be regarded as another powerful tool to be used in the calibration of software effort models. It is observed that there is a strong relationship

between the success of a particular technique and the size of the learning dataset, the function cost nature, and dataset characteristics (outliers, co-linearity, number of variables, etc). Thus the best technique may not always be the right approach to be adopted. This premise was investigated in this work, where various experiments revealed there may be a considerable variation in the performance of these systems when the nature of the historical data changes. For instance, for one of the datasets the linear regression performed better than the neural network model which leads to the continuity of the study.

Consequently, new experiments will be conducted to combine the neural network and multiple regression techniques to calibrate and to test prediction models on other datasets, such as, the ISBSG database (*International Software Benchmarking Standard Group*). It contains information on software projects developed with modern software development techniques. We aim at improving the performance of the neural network models obtained in this work and to obtain a model that can safely be used in software development.

Acknowledgements

The authors would like to thank CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) for the financial support and the SBES (Software Engineering Brazilian Symposium) reviewers for their useful comments.

Appendix A. Data description

Variable	Adjustment factor	Level assigned
Type	(BUS)	1
	(CTL)	2
	(HMI)	3
	(SCI)	4
	(SUP)	5
	(SYS)	6
LANG	COB	1
	PLI	2
	FTN	3
	MOL	4
	JOV	5
	HOL	6
	PSC	7
RELY	Very low = 0.75	1
	Low = 0.88; 0.94	2
	Nominal = 1	3
	High = 1.15	4
	Very high = 1.4	5
DATA	Low = 0.94	1
	Nominal = 1	2
	High = 1.04	3
	Very high = 1.08	4
	Extra high = 1.16	5
CPLX	Very low = 0.7	1
	Low = 0.85	2
	Nominal = 1	3
	High = 1.07	4
	Very high = 1.15	5
	Extra high = 1.30; 165	6
STOR	Nominal = 1	1
	High = 1.06	2
	Very high = 1.14	3
	Extra high = 1.21	4
	Extra-extra high = 1.56	5
VIRT	Low = 0.87	1
	Nominal = 1	2
	High = 1.15	3
	Very high = 1.30	4
TURN	Very low = 0.87	1
	Low = 0.94	2
	Nominal = 1	3
	High = 1.07	4
	Very high = 1.15	5
PLATF	MAX	1
	MID	2
	MIN	3
	MIC	4
ACAP	Extra high = 0.71	1
	Very high = 0.78; 0.86	2
	High = 1	3
	Nominal = 1.10; 1.19	4
	Low = 1.46	5
AEXP	Very high = 0.82	1
	High = 0.91	2
	Nominal = 1	3
	Low = 1.13	4
	Very low = 1.29	5
PCAP	Very high = 0.7	1
	High = 0.86 e0.93	2
	Nominal = 1	3
	Below average = 0.8; 1.17; 1.42	4
VEXP	High = 0.9	1
	Nominal = 1	2
	Low = 1.1	3
	Very low = 1.21	4

(*continued on next page*)

Appendix A (*continued*)

Variable	Adjustment factor	Level assigned
LEXP	High = 0.95	1
	Nominal = 1	2
	Low = 1.07	3
	Very low =	4
MODP	Very high = 0.82	1
	High = 0.91; 0.95	2
	Nominal = 1	3
	Low = 1.1	4
	Very low = 1.24	5
TOOL	Very high = 0.83	1
	High = 0.91; 0.95	2
	Nominal = 1	3
	Low = 1.1	4
	Very low = 1.24	5
SCHED	Nominal = 1	1
	High = 1.04	2
	Very high = 1.08	3
	Extra high = 1.23	4
RVOL	Low = 0.91	1
	Nominal = 1; 1.09	2
	High = 1.19	3
	Very high = 1.38; 1.62	4
MODE	Organic	1
	Semidetashed	2
	Embedded	3
CONT	LO	1
	NOM	2
	HI	3

References

Agarwal, R., 2001. Estimating software projects. Software Engineering Notes 26, 60–67.

Albrecht, A.J., 1979. Measuring application development productivity. In: Proc. IBM Application Development Symposium, pp. 83–92.

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

Angelis, L., Stamelos, I., 2000. A simulation tool for efficient analogy based cost estimation. Empirical Software Engineering 5, 35–68.

Angelis, L., Stamelos, I., Morisio, M., 2001. Building a software cost estimation model based on categorical data. In: Proc. Seventh IEEE Int. Software Metrics Symposium, pp. 4–15.

Barcelos Tronto, I.F., Silva, J.D.S., Sant'Anna, N., 2006a. Comparison of artificial neural network and regression models in software effort estimation. In: IEEE International Joint Conference on Neural Networks, Orlando, USA, August 2007.

Barcelos Tronto, I.F., Silva, J.D.S., Sant'Anna, N., 2006b. Uma Investigação de Modelos de Estimativas de Esforço em Gerenciamento de Projeto de Software. In: Simpósio Brasileiro de Engenharia de Software – SBES, Florianópolis, Brasil, October 16–October 20.

Bisio, R., Malabocchia, F., 1995. Cost estimation of software projects through case base reasoning. 1st Intl. Conf. on Case-Based Reasoning Research & Development. Springer-Verlag, pp. 11–22.

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

Boehm, W., Horowitz, E., Madachy, R., Reifer, D., Clark, B.K., Steece, B., Brown, A.D., Abts, C., 2000. Software Cost Estimation with COCOMOII. Prentice-Hall.

Briand, L.C., Langley, T., Wieczorek, I., 2000. A replicated assessment and comparison of common software cost modeling techniques. In: Pro. ICSE 2000, Limerick, Ireland, pp. 377–386.

Briand, L.C., Wieczorek, I., 2002. Software resource estimation. Encyclopedia of Software Engineering (2), 1160–1196.

Conte, S.D., Dunsmore, H.E., Shen, V.Y., 1986. Software Engineering Metrics and Models. Benjamin/Cummings, California.

Finnie, G.R., Witting, G.E., Desharnais, J.-M., 1997. A comparison 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., MacDonell, S.G., 1997. A comparison of model building techniques to develop predictive equations for software metrics. Information and Software Technology 39, 425–437.

Gray, A.R., MacDonell, S.G., Shepperd, M., 1999. Factors systematically associated with errors in subjective estimates of software development effort: the stability of expert judgment. In: Proc. IEEE 6th Metrics Symposium.

Hasting, T.E., Sajeev, A.S.M., 2001. A vector based approach to software size measurement and effort estimation. IEEE Transactions on Software Engineering 27 (4).

Jeffery, R., Ruhe, M., Wieczorek, I., 2001. Using public domain metrics to estimate software development effort. In: Proc. IEEE 7th Metrics Symposium, London, UK, pp. 16–27.

Jones, C., 1986. Estimating Software Costs. McGraw-Hill.

Karunanitthi, N., Whitley, D., Malaiya, Y.K., 1992. Using neural networks in reliability prediction. IEEE Software 9 (4), 53–59.

Kemerer, C.F., 1987. An empirical validation of software cost estimation models. Comunication of ACM 30, 416–429.

Khoshgoftaar, T.M., Allen, E.B., Xu, Z., 2000. Predicting testability of program modules using a neural network. In: Proc. 3rd IEEE Symposium on Application-Specific Systems and Sof. Eng. Technology, pp. 57–62.

Kitchenham, B., 1998. A procedure for analyzing unbalanced datasets. IEEE Transactions on Software Engineering 24, 278–301.

Kumar, S., Krishna, B.A., Satsangi, P.S., 1994. Fuzzy systems and neural networks in software engineering project management. Journal of Applied Intelligence 4, 31–52.

Lai, R., Huang, S., 2003. A model for estimating the size of a formal communication protocol specification and its implementation. IEEE Transaction on Software Engineering 29 (1), 46–62.

MCT – Ministerio da Ciência e Tecnologia, 2001. Qualidade e Produtividade no setor de software. In: http://www.mct.gov.br/Temas/info/Dsi/Quali2001/2001Tab40.htm, Tabela 40 – Práticas de Engenharia de Software no Desenvolvimento e Manutenção de Software.

Myrtveit, I., Stensrud, E., Shepperd, M., 2005. Reliability and validity in comparative studies of software prediction models. IEEE Transaction on Software Engineering 31 (5), 46–62.

Putnam, L.H., 1978. A general empirical solution to the macro sizing and estimating problem. IEEE Transaction on Software Engineering 4, 345–361.

Samson, B., Ellison, D., Dugard, P., 1997. Software cost estimation using albus perceptron (CMAC). Information and Software Technology 39, 55–60.

Selby, R.W., Porter, A.A., 1998. Learning from examples: generation and evaluation of decision trees for software resource analysis. IEEE Trans on Software Engineering 14, 1743–1757.

Sentas, P., Angelis, L., Stamelos, I., Bleris, G., 2005. Software productivity and effort prediction with ordinal regression. Journal Information and Software Technology (47), 17–29.

Shepeerd, M.J., Shofield, C., Kitchenham, B., 1996. Effort estimation using analogy. In: Proc. ICSE-18, Berlin.

Shepperd, M., Schofield, C., 1997. Estimating software project effort using analogies. IEEE Transactions on Software Engineering 23 (12), 736–743.

Srinivazan, K., Fisher, D., 1995. Machine learning approaches to estimating software development effort. IEEE Transactions on Software Engineering 21 (2), 126–137.

Vicinanza, S., Prietula, M.J., Mukhopadhyay, T., 1990. Case-based reasoning in software effort estimation. In: Proc. 11th Int. Conf. Info. Syst, pp. 149–158.

Witting, G., Finnie, G., 1994. Using artificial neural networks and function points to estimate 4GL software development effort. Journal Information and Systems 1 (2), 87–94.

Witting, G., Finnie, G., 1997. Estimating software development effort with connectionist models. Information and Software Technology 39, 369–476.

Zhong, S., Khoshgoftaar, T.M., Seliya, N., 2004. Analysing software measurement data with clustering techniques. IEEE Intelligent Systems, 20–27.

Iris Fabiana de Barcelos Tronto is Computer Science bachelor by Universidade Federal de Uberlândia – UFU. She received her Master's degree in computer science from Universidade de São Paulo – ICMC/USP. São Paulo, Brazil. She was an assistant teacher and researcher at the Universidade Estadual de Minas Gerais State. She is currently a doctorate student at Instituto Nacional de Pesquisas Espaciais – INPE/ Laboratório Associado de Computação e Matemática Aplicada – LAC. Her primary research interests are in exploring applications of data analysis techniques to software engineering.

José Demísio Simões da Silva is an Electrical Engineer by the Universidade Federal da Paraiba – UFPB, where he received his Master's degree in Electrical Engineering. He is a Ph.D. in Applied Computation from Instituto Nacional de Pesquisas Espaciais – INPE. Currently he is a researcher for the Laboratório Associado de Computação e Matemática Aplicada – LAC, at INPE. He is also a professor at Universidade Braz Cubas – UBC. His primary research interests are in artificial intelligence, neural networks, data mining, and data analysis.

Nilson Sant' Anna is an Electrical Engineer by the Escola Federal de Engenharia de Itajubá – EFEI. He is a Ph.D. in Applied Compuation from Instituto Nacional de Pesquisas Espaciais – INPE, where he also received his Master's degree in Applied Computing. Currently he is a researcher for the Laboratório Associado de Computação e Matemática Aplicada – LAC, at INPE. He has worked both in industry and academia and has been involved with consulting in software quality control. His primary research interests are in software environment, software quality control, software project management.