Adopting Curvilinear Component Analysis to Improve Software Cost Estimation Accuracy

Model, Application Strategy, and an Experimental Verification

Salvatore A. Sarcia'[1], Giovanni Cantone[1] and Victor R. Basili[2,3]

[1]DISP, Università di Roma Tor Vergata, via del Politecnico 1, 00133 Rome, Italy
sarcia@disp.uniroma2.it, cantone@uniroma2.it
[2]Dept. of Computer Science, University of Maryland, A.V. Williams Bldg. 115, College Park 20742, MD, USA
[3]Fraunhofer Center for Experimental Software Engineering Maryland, College Park, Maryland, 20742
{basili, sarcia}@cs.umd.edu

Cost estimation is a critical issue for software organizations. Good estimates can help us make more informed decisions (controlling and planning software risks), if they are reliable (correct) and valid (stable). In this study, we apply a variable reduction technique (based on auto-associative feed--forward neural networks – called Curvilinear component analysis) to log-linear regression functions calibrated with ordinary least squares. Based on a COCOMO 81 data set, we show that Curvilinear component analysis can improve the estimation model accuracy by turning the initial input variables into an equivalent and more compact representation. We show that, the models obtained by applying Curvilinear component analysis are more parsimonious, correct, and reliable.

Prediction models, Curvilinear Component Analysis, Software Cost Estimation, Software Economics, Feature Reduction, Neural Networks.

1 INTRODUCTION

Cost estimation is a critical issue for software organizations. The need is to get the best estimate when planning a new project. Improving the prediction capability of software organizations is a way of improving their competitive advantage. Prediction is a major task when trying to better manage resources, mitigate the project risk, and deliver products on time, on budget and with the required features and functions. This is our motivation for proposing estimation improvement techniques for software organizations that need to increase their competitive advantage.

Estimates can help us make decisions that are more informed if, and only if, we can rely on the results to be accurate. We call a result accurate if it is *reliable* (correct) and *valid* (stable). Better estimates can be obtained by improving the estimation model. An estimation model is composed of some input variables (explanatory or independent variables), one output variable (the estimate or the dependent variable), and a function that calculates the outputs from the inputs. There are many ways of improving the estimates. For instance, we can choose: (1) a better function (e.g., the one that describes more appropriately the relationship between inputs and output), and/or (2) more explanatory input variables. In the former case, we can choose the type of function, e.g., linear or logarithmic that fits best. In the latter, the real problem is that, once we have selected the complete input variable set, we need to remove the redundancy that negatively affects the performance of the estimation model (e.g., irrelevant variables). In fact, a more parsimonious model (with fewer parameters) is preferable to one with more parameters because the former is able to provide better estimates with the same number of observations [13]. This task can be performed in many ways, e.g., shrinking the input set into an equivalent pattern or removing irrelevant variables (e.g., stepwise methods [9]). In this work, we use *Curvilinear Component Analysis* (CCA) as a input shrinkage technique, which produces (shrunken) data sets where we apply ordinary least squares (OLS) [14]. We also define an application strategy to figure out whether CCA is worth to use or not. This core of this strategy is based on auto-associative artificial neural networks [3, pp. 314-319] that, to the best of our knowledge, have never been applied in the context of software estimation [11] even though its applicability is well known in the image-processing field. In particular, we apply CCA to a COCOMO 81 data set [12] and OLS functions. Even though we apply the CCA to the COCOMO 81 data set, the proposed methodology can be applied to any estimation model that uses past observations for prediction such as machine learning, neural networks, and ordinary least squares functions, and to any quantity of interest (e.g., effort, fault proneness).

This paper is based on the research hypothesis that CCA can improve software cost estimation accuracy. We argue that a methodology improves accuracy if improves the correctness and the variability does not worsen (i.e., variability is the same or better). It may happen that, if the correctness (bias) improves, the variability (spread) gets worse. So, we cannot argue that the accuracy is improved. For this reason, we investigate both the variability and the correctness of the estimation model. In order to investigate the research hypothesis we:

(1) utilized two summary statistics as measures of bias and spread, calculated on the *Relative Error* (RE), where RE = (Actual Effort – Estimated Effort)/(Actual Effort), letting RE^i be an accuracy measure on the i-th

project and T be the number of the projects being estimated (Test set), and using Mean(RE^i) and STD(RE^i), for i = 1 to T, to measure the estimation model correctness (bias) and variability (spread), respectively,

(2) elaborated an application strategy, and

(3) tested it by using a number of randomly selected models.

In this research, we show that the used technology (CCA) provides a significant accuracy improvement, that is, it is more correct with a similar variability than the estimate produced without applying any improvement methodology. However, it is possible that, applying CCA to different data sets and models, would lead to no improvement. This may happen when the data has no alignment into the space (this concept of "alignment" is explained in Section 3). So, we should apply different techniques for improving the accuracy as there is no best technique for all situations. Improvement techniques are substantially divided into two not mutually exclusive sets, stepwise input variable removal [9, 14], where the input variables are recursively taken out, and input variable reduction [7], where the input variables are transformed into a shrunken representation. CCA refers to the latter and it is able to overcome some drawbacks of principal component analysis (PCA); this point is explained in Section 3. Note that, one may decide to apply first CCA and subsequently stepwise technique, only CCA, only stepwise, or no technique. The decision about whether and which technique to apply should be made based on the ratioin between cost and effectiveness [14, 3]. Unfortunately, the only approach for achieving the best result in terms of input variable reduction and accuracy is the exhaustive one. It consists of taking into account all of the possible configurations that can be obtained by combining the input variables. For instance, if we have N variables we can obtain 2^N different configurations. The problem is that, this procedure cannot be generally applied because 2^N is usually a number too big to be handled. For this reason, we study affordable techniques such as CCA to improve the estimation accuracy without applying the exhaustive procedure.

The rest of this paper explains the results of applying CCA compared to the results without improvement. We start with some introductory remarks on COCOMO and curvilinear component analysis. We continue with a discussion of our experimental design and results. We then apply statistical tests to the results to show that using CCA improves the accuracy of the log-linear estimation models in terms of correctness avoiding worsening the variability.

2 COCOMO

The estimation model that we considered in this study is COCOMO-I (COstructive COst MOdel) [1, 2]. We use this model since it is an open model with published data [12]. Currently, COCOMO-I has evolved to COCOMO-II [2,4], but for the latter there is no published data. Our aim is to show the reliability and stability of using CCA so that others may repeat our experiment based on that available data. Therefore, we are not proposing the use of the older COCOMO-I model as an estimation model, we only use it to show that CCA is able to improve the accuracy of log-linear OLS functions. COCOMO-I is based on equation (1):

$$\text{months} = a * (\text{KSLOC}^b) * \left(\prod_i \text{EM}_i\right) . \quad (1)$$

Where, the effort (*months*) is measured by calendar months of 152 hours, including development and management hours, *a* and *b* are two parameters related to the domain (e.g., *a* is a value between 2.80 and 3.20 and *b* between 1.05 to 1.2), and *KSLOC* is estimated or calculated from a function point analysis. EM_i is a set of 15 multipliers [1, 2, 12], which aim at weighting Eqn. (1) to provide more suitable results. For instance, there are seven multipliers (ACAP, PCAP, AEXP, MODP, TOOL, VEXP, and LEXP), which affect the effort more strongly as they increase, e.g., ACAP = "Analysts' CAPability" is a value ranging from 1.46 (= very low) to very high (= 0.71). There are seven multipliers (DATA, TURN, VIRT, STOR, TIME, RELY, and CPLX), which affect the effort less strongly as they increase, e.g., CPLX = "process ComPLeXity" is a value ranging from 0.70 (= very low) to extra high (= 1.65). There is another multiplier (SCED = SChEDule constraint), which affects the effort more strongly either as it increases or decreases, e.g., giving analysts either too much or too little time can increase the effort). It takes values ranging from 1.23 (= very low) to 1.10 (= very high), but the central value (nominal) is 1.00.

COCOMO-I can also be calibrated to local data to find a better fitting model. Since the COCOMO model is based on the assumption that the effort increases geometrically, Eqn. (1), we need to transform the model into another one where we can apply OLS (linearization). For this reason, we use a logarithmic transformation in taking the logarithm of Eqn. (1). Then, the resulting model is the following:

$$\text{Ln}(\text{months}) = \beta_0 + \beta_1 \text{Ln}(\text{KSLOC}) + \beta_2 \text{Ln}(\text{EM}_1) + ,...,+ \beta_{16} \text{Ln}(\text{EM}_{15}) . \quad (2)$$

That is,

$$Z = \beta_0 + \beta_1 H_1 + ,...,+ \beta_{16} H_{16} . \quad (3)$$

Where, Z = Ln(months), H_1 = Ln(KSLOC), and H_{i+1} = Ln(EM_i) with i = 1 to 15. Then, β_0 is the intercept, $\beta_{1..16}$ are the model coefficients, $H_{1..16}$ are the independent variables, and Z is the dependent variable (effort). In practice, before

applying OLS to Eqn. (3), one has to calculate the natural logarithm (Ln) of each value in the data set (note that, in order to calculate β_0 a vector composed of only 1s has to be inserted into the data set).

Any prediction model is evaluated by taking into account the error between the estimated and actual values. An absolute error (Actual – Estimated) makes no sense in software cost estimation, because the error should be relative to the size of the project (e.g., a larger project may have a greater error). For this reason, Boehm [1] defined the COCOMO performance in terms of *Relative Error* (RE), as formula (4) shows, where RE_i is the relative error of project *i* in the test set.

$$\mathrm{RE_i} = \frac{\mathrm{Actual_i} - \mathrm{Estimated_i}}{\mathrm{Actual_i}} \ . \tag{4}$$

Figure 1 reports on the accuracy procedure calculation that we considered in this paper. In particular, given a *data set* (DS) of size S_{DS} and a training set of size S_{TrS} with $S_{TrS} < S_{DS}$, and a test set of size $S_{TsS} = (S_{DS} - S_{TrS})$, the accuracy is calculated by Eqn. (5).

$$\mathrm{MnRE} = \mathrm{Mean}(\mathrm{RE_i}) = \frac{1}{S_{TsS}} \sum_{i=1}^{S_{TsS}} \mathrm{RE_i} \ . \tag{5}$$

Since the best accuracy is zero, MnRE is the bias of the estimation model, and the standard deviation of RE_i is a measure of spread of the estimation model.

In this work, we mainly focus on the bias (correctness) of the estimation model and its stability (validity). It is very important to note that, sometimes COCOMO is evaluated in considering the *Magnitude of Relative Error* (MRE) [5, 12], where $MRE_i = abs(RE_i)$, instead of RE_i. Then Eqn. (5) becomes the following:

$$\mathrm{MMRE} = \frac{1}{S_{TsS}} \sum_{i=1}^{S_{TsS}} \mathrm{MRE_i} \ . \tag{6}$$

Another way of evaluating COCOMO is to use PRED (N).

$$\mathrm{PRED(N)} = \frac{100}{S_{TsS}} \sum_{i=1}^{S_{TsS}} \begin{cases} 1 \rightarrow \mathrm{MRE_i} \le \dfrac{\mathrm{N}}{100} \\ 0 \rightarrow \mathrm{otherwise} \end{cases} \ . \tag{7}$$

A PRED (25) = 80% means that 80% of the estimates are within 25% of the actual error [5, 12], i.e, 80% of the estimates in the test set has an MRE value not greater than 0.25. It is possible to prove that, formulas (6) and (7) are not accuracy indicators of the estimation model [8].

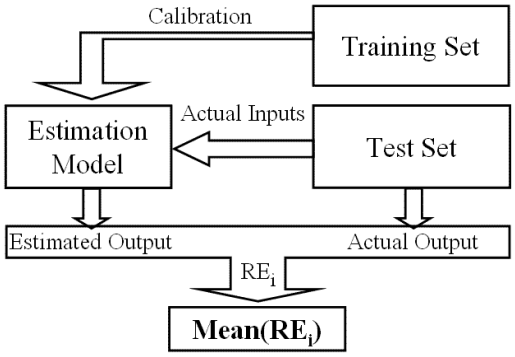

Fig. 1. COCOMO evaluation.

This means that, it is incorrect to measure COCOMO (or similar parametric models) in terms of equations (6) and (7). In particular, Kitchenham et al. show that MMRE and PRED(N) measure the spread of the kurtosis of the random variable Z = Estimated/Actual. This is the reason why, in this paper, we evaluate COCOMO through Eqn. (5). Note that, MMRE may be a useful measure when evaluating the goodness-of-fit of a model.

3 CURVILINEAR COMPONENT ANALYSIS

CCA is a procedure for feature reduction based on auto-associative multi-layer feed-forward neural networks [3, pp. 314-319]. Applying CCA does not require being an expert in neural network (NN) computation. A CCA

implementation can be found in any mathematics application that is able to deal with NNs and/or matrixes. Implementing CCA requires just a few lines of code. Even if one does not have the opportunity to use a mathematics suite, the CCA algorithm can be easily implemented with the same programming language used for calculating OLS. For space limitation, we focus on CCA reporting just some principal notes on NN(s) [3, 6].

3.1 Multi-layer Feed-forward Neural Networks

A neuron is a parameterized and bounded function (Figure 2-a), which can be linear or nonlinear.

A neuron (also called Unit) calculates an output (y) such that $y = f(\sum_{i=1}^{N} w_i \cdot X_i)$, where w_i are the weights (or parameters), X_i are the inputs, and *f* is called activation function. If *f* is a nonlinear function (e.g., a sigmoidal function such as logistic function, hyperbolic tangent function), then the neuron is called Nonlinear; if *f* is a linear function (*f* is the identity function), then the neuron is called Linear. A feed-forward NN is generally a nonlinear function, which is composed of some neurons (Figure 2-b) and layers. In the feed-forward networks, the data can just flow from the inputs to the outputs. In recurrent networks, the data flow can be circular.

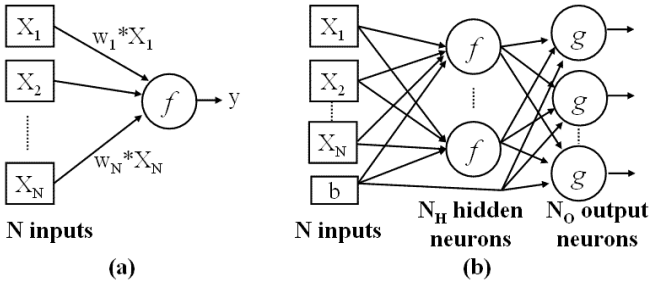

Fig. 2. A neuron (a) and a multi-layer feed-forward neural network (b).

In Figure 2, *g* may be different from *f*. For instance, in regression problems, *g* is nonlinear and *f* is linear. In discrimination problems, both *g* and *f* are nonlinear functions. The input labeled "b" is called bias. It is an input that provides a constant value of 1. The bias plays the same role as the intercept in a polynomial function. In Figure 2(b), *f* units are called hidden because they are intermediate. Hidden layers express the complexity of a model. In particular, the number of hidden units corresponds to the degree of a polynomial. For instance, an NN having two hidden units is more complex than an NN with just one hidden unit just as a second order polynomial is more complex than a first-order polynomial. Based on observations (both inputs and outputs), the problem is then to calculate the model weights (w_i) such that the input values are mapped to output values. The weight calculation is also called model training. In order to get this mapping, a cost function has to be minimized [3, pp.194-201]. The most common cost function is the Euclidian distance. When using polynomials, it is possible to apply OLS to calculate the model parameters, but it is not applicable with NNs. Usually, the best training technique is Backpropagation [10]. This is an iterative method based on calculating gradients. In particular, the gradient of the cost function is calculated. It happens for each step and the gradient is used to update the parameters found in the previous step. The algorithm stops when satisfactory conditions have been met [3]. It is very important to note that, the hidden neurons play a principal role here. In fact, their output can be considered as a representation of the input in mapping the output [10]. This property will be used for implementing auto-associative NNs.

3.2 Auto-Associative Multi-layer Neural Networks

An auto-associative neural network (AANN) is a particular kind of multi-layer feed-forward NN. Figure 3 shows an example of AANN topology. The aim of this kind of neural network is to perform nonlinear dimensionality reduction. The strategy is to map N input variables into N output variables. The observed outputs used to train the network (targets) are just the observed inputs themselves (for this reason this network is called auto-associative). The auto-associative network in Figure 3 tries to map each observed input into itself [3, p. 314]. This strategy is worth for dimensionality reduction when the number M of the neurons in the second hidden layer (Figure 3) is less than N. To get a correct dimensionality reduction, the output units must be linear (Lin = Linear) as well as the M units in the second hidden layer (Lin). The first and the third hidden layer must be nonlinear (Sig = Sigmoidal function). The training of this kind of network is based on minimizing an Euclidian distance similar to the one mentioned above [3, p. 314]. Note that, AANN in Figure 3 can be considered as composed of two different networks. The first network (F_1, dashed rectangle) projects the initial N-dimensional data onto an M-dimensional space (M<N). This space is composed of the output of F_1 when feeding it with the original observations. The curvilinear components of this space are encapsulated in F_1. This means that, once F_1 has been calibrated, it can be used for transforming any input into an equivalent representation with respect to the original one with fewer dimensions (from N to M dimensions). The second network (F_2) maps the output of F_1 having M dimensions back into the initial N-dimensional space.

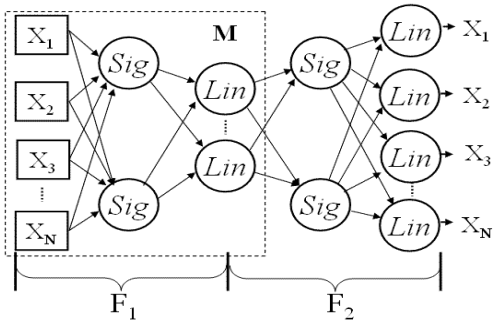

Fig. 3. An auto-associative multi-layer neural network for nonlinear dimensionality reduction.

The result is that, the output of F_1 is a nonlinear representation (projection) of the original N-dimensional space onto a shrunken space composed of M components. This network actually performs a curvilinear component analysis also called Nonlinear Principal Component Analysis (see [7] for the definition of Principal Component Analysis). This important result is made possible because of the presence of nonlinear activation functions in the first and third hidden layer (Sig). Note that, this kind of technology is able to perform both linear and curvilinear component analysis.

3.3 Using CCA together with OLS (The Strategy)

The ten steps below explain the strategy that we propose for dimensionality reduction with CCA together with OLS. Note that, this strategy would be the same even if we considered different parametric models (e.g., machine learning, neural networks). The aim of this strategy is to figure out whether the available data can be shrunk by a curvilinear transformation because of the redundancy of some input variables unknown. So, a model that is more parsimonious (i.e., the one having less parameters because built upon fewer input variables) should provide better results in terms of accuracy. The strategy is the following:

1. Split up the available data (i.e., past observations) into two subsets as explained in Section 2 and Figure 1 (e.g., 2/3 – 1/3 or 80% – 20%), so obtain both training set, TrS (e.g. 2/3), and test set, TsS (e.g. 1/3)
2. Let N be the number of input variables (N = 16 for the COCOMO model). Based upon the split made in Step 1, use TrS to train N-1 models applying CCA as many times, where each time the data set is reduced by 1 component (i.e., in the first CCA application, TrS turns into N-1 dimensions, in the second one, it turns into N-2 dimensions, and so on up to 1)
3. Calculate the Mean(RE) and STD(RE) for the obtained N-1 models feeding each model with TsS
4. Among the N-1 models obtained in Step 2, select the model having the best score calculated in Step 3, i.e., the Mean(RE) closest to zero
5. Use TrS to train a log-linear function applying OLS without CCA
6. Calculate the Mean(RE) and STD(RE) feeding the model with TsS
7. Repeat Steps 1 through 6 for a statistically sufficient number of times (e.g. 30) changing the composition of TrS and TsS and get two distributions for each considered summary statistic, i.e., $MnRE_{CCA} \equiv \{Mean_{CCA}(RE^1) \dots Mean_{CCA}(RE^{30})\}$, $STDRE_{CCA} \equiv \{STD_{CCA}(RE^1) \dots STD_{CCA}(RE^{30})\}$, and $MnRE_{NO\text{-}CCA} \equiv \{Mean_{NO\text{-}CCA}(RE^1) \dots Mean_{NO\text{-}CCA}(RE^{30})\}$, $STDRE_{NO\text{-}CCA} \equiv \{STD_{NO\text{-}CCA}(RE^1) \dots STD_{NO\text{-}CCA}(RE^{30})\}$, respectively
8. Based upon suitable statistical tests (i.e., parametric or non-parametric), evaluate the hypotheses whether (1) the distribution $MnRE_{CCA}$ is significantly better than $MnRE_{NO\text{-}CCA}$ and (2) the distribution $STDRE_{CCA}$ is insignificantly different from $STDRE_{NO\text{-}CCA}$. If the statistical tests significantly confirm hypotheses (1) and (2), then execute Steps 9 and 10, otherwise stop this procedure because CCA cannot significantly improve the accuracy. In the latter case, other feature selection techniques should be considered (e.g., stepwise [9])
9. Select the model corresponding to the best value in $MnRE_{CCA} \equiv \{Mean_{CCA}(RE^1) \dots Mean_{CCA}(RE^{30})\}$. If two models have the same score choose the one having the smallest spread
10. Use this model to make predictions (on new projects).

4 EXPERIMENT DESIGN

Our experimental setting is based on the strategy reported in Section 3.3. The aim of this experiment is to show that applying the procedure in Section 3.3 can lead to improving the estimation accuracy of a log-linear function trained by OLS (Section 2). To this end, we organized the available data as reported in Figure 4.

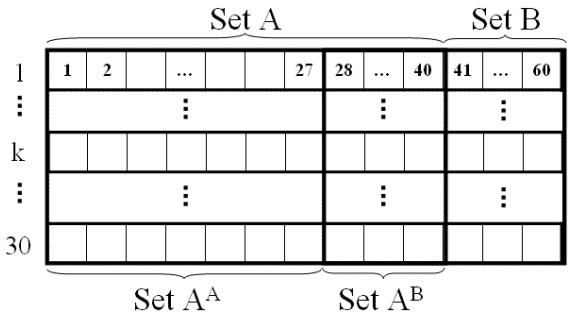

Fig. 4. Experimental setting

We used 60 projects of the COCOMO data set [12] for building randomly 30 different data sets. The first row of the 30x60 matrix in Figure 4 includes the experiment-projects' identifiers, which we assigned randomly from 1 to 60; each of the remaining rows is composed of a different randomly selected circular permutation of the first row. Then, we split up this matrix into two subsets of columns (A and B) and considered set A as a set of 30 different training sets and set B as a set of 30 different test sets. The split proportion was 2/3 – 1/3, thus each item of set A included 40 project instances, and each item of set B included 20 instances. Our experimental setting simulated the situation where there is a data set of past observations (Set A) and a set of projects being estimated (set B), unknown at estimation time. The insight is that, if actually CCA was able to improve the accuracy of a log-linear OLS function, we should observe for it more accurate results (i.e., in terms of bias and spread) than the ones obtained without applying any improvement technique and it should happen for a significant number of times (at least 30 times with randomization).

We started with calculating the log-linear OLS functions with CCA. To this end, we considered set A as a set of past observations and used set B for calculating bias and spread of each of the 30 obtained functions by applying CCA, i.e., we calculated $MnRE_{CCA(k)}$ and $STDRE_{CCA(k)}$, for k = 1 through 30. We used $MnRE_{CCA}$ and $STDRE_{CCA}$ for denoting the two distributions (Appendix 1). Applying the proposed strategy to set A meant dividing this set into two further subsets, A^A and A^B (Figure 4) with the proportion 2/3 and 1/3, as explained in Section 3.3. Set A^A was used for training and set A^B for selecting the best function (Section 3.3., Step 4). Note that, each element of sets A^A, A^B and B was made of different projects with respect to each other element of the same set.

With respect to the log-linear OLS functions without applying CCA, first we considered the 30 elements of set A^A for training as many log-linear OLS functions and then we used set B for calculating $MnRE_{NO\text{-}CCA(k)}$ and $STDRE_{NO\text{-}CCA(k)}$, with k = 1 through 30, thus we got $MnRE_{NO\text{-}CCA}$ and $STDRE_{NO\text{-}CCA}$ (Appendix 1). Then, we compared the obtained distributions in order to figure out whether $MnRE_{CCA}$ was better than $MnRE_{NO\text{-}CCA}$ and whether, at the same time, $STDRE_{CCA}$ was insignificantly different from $STDRE_{NO\text{-}CCA}$.

In a real case, any hold out method (i.e., the ones that split up the observations into two subsets, one for training, and one for test) may lead to loosing information because of the hold out strategy. In fact, projects in set A^B cannot be used for training the log-linear OLS function with CCA. For this reason, we wondered whether the accuracy of the functions with CCA was better than the accuracy of functions trained with the complete set A. To this end, we retrained 30 log-linear OLS functions by considering each element of set A as a training set and used the corresponding elements of set B for calculating the two bias and spread distributions, thus we got $MnRE^A_{NO\text{-}CCA}$ and $STDRE^A_{NO\text{-}CCA}$ (Appendix 1), where the apex A refers to functions trained using set A. Similarly to the previous case, we tested the hypothesis {$MnRE_{CCA}$ is significantly better than $MnRE^A_{NO\text{-}CCA}$} and {$STDRE_{CCA}$ is insignificantly different from $STDRE^A_{NO\text{-}CCA}$}.

5 RESULTS AND DATA ANALYSIS

First, for each considered distribution, we performed some statistical tests for normality (Chi-Square goodness-of-fit statistic, Shapiro-Wilks, Z score for skewness, and Z score for kurtosis).

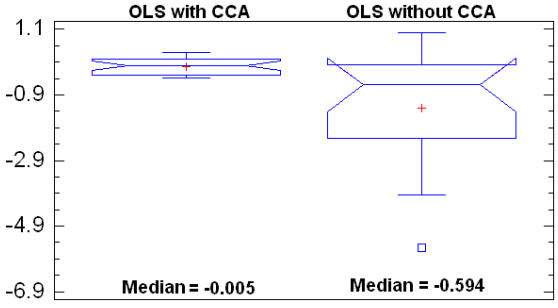

Fig. 5. Bias Analysis ($MnRE_{CCA}$ vs $MnRE_{NO\text{-}CCA}$)

All these tests rejected the idea that the distributions came from a normal population with 99% of confidence. Then, we had to apply non-parametric tests.

Based on the experimental design in Section 4, we started with comparing $MnRE_{CCA}$ to $MnRE_{NO\text{-}CCA}$ (Figure 5 and Appendix 1). We applied both Mann-Whitney test (p-value = 0.015) to compare the medians of the two samples and Signed rank test (p-value = 0.045). Even though the two obtained samples were independent and the experiment was not paired, we applied the Signed rank test to increase the power of our non-parametric inference [8]. However, both tests rejected the null hypothesis with 95% of confidence.

Then, we tested the hypothesis whether CCA was able to provide estimates as stable as the ones provided without applying CCA. To this end, we compared $STDRE_{CCA}$ to $STDRE_{NO\text{-}CCA}$ (Figure 6 and Appendix 1).

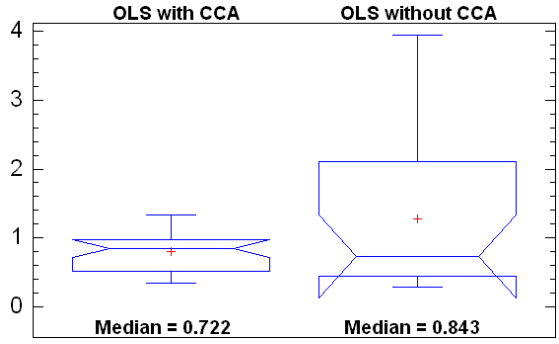

Fig. 6. Spread Analysis ($STDRE_{CCA}$ vs $STDRE_{NO\text{-}CCA}$)

We applied both Mann-Whitney test (p-value = 0.437) and Signed rank test (p-value = 0.813) for the same reasons explained above. Both tests did not reject the null hypothesis with 95% of confidence, i.e., they pointed out that the two considered distributions were not statistically different. Note that, the distributions in Figure 6 were obtained from the same randomization as the one applied for the bias distributions (Figure 5 and Appendix 1).

Based on these results, we concluded that CCA was able to improve the accuracy of the log-linear OLS functions without making worse their variability. Then we tested the second hypothesis, i.e., we compared the distribution $MnRE_{CCA}$ to $MnRE^{A}_{NO\text{-}CCA}$ (Figure 7 and Appendix 1).

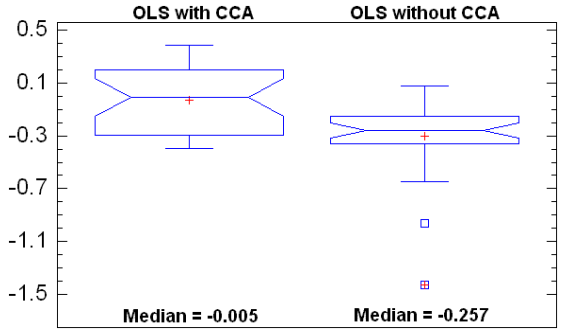

Fig. 7. Bias Analysis ($MnRE_{CCA}$ vs $MnRE^{A}_{NO\text{-}CCA}$)

Based on the strategy applied for the previous analyses, we performed both Mann-Whitney test (p-value = 0.002) and Signed rank test (p-value = 0.003), which rejected the null hypothesis with 95% of confidence. Then, we tested the hypothesis whether CCA was able to provide estimates as stable as the ones provided without applying CCA. To this end, we compared $STDRE_{CCA}$ to $STDRE^{A}_{NO\text{-}CCA}$ (Figure 8 and Appendix 1).

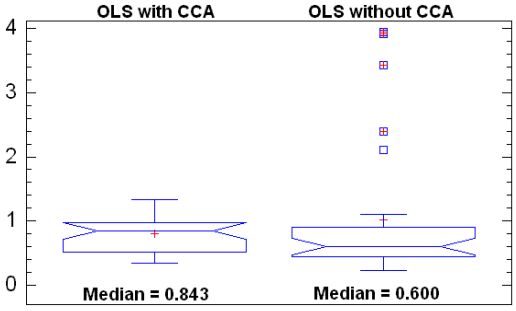

Fig. 8. Spread Analysis ($STDRE_{CCA}$ vs $STDRE^{A}_{NO\text{-}CCA}$)

We applied again both Mann-Whitney test (p-value = 0.181) and Signed rank test (p-value = 0.813). Since tests did not reject the null hypothesis with 95% of confidence, we concluded that the two considered distributions were

non-statistically different. Overall, we concluded that applying CCA to log-linear OLS functions improved the accuracy for the COCOMO data set without making worse the variability.

6 DISCUSSION AND CONCLUSION

Let us consider the implications of the results reported in Section 5. Based on the COCOMO data set, we have shown that applying CCA to log-linear OLS functions produces estimates that are more accurate than the ones provided by the same kind of functions without applying CCA.

A valuable result is that the proposed technology increases the correctness of the estimates without worsening the variability. In order to evaluate the reliability of our experiment, we also compared the variability (spread) of the obtained distributions. With respect to Figures 5 and 6, the spread of the CCA distributions is less than the ones without applying CCA. Note that, the spread is expressed by the length of the box from lower tail to upper tail. This happens both for the bias and spread distributions. This means that, CCA is able to provide more stable estimates with respect to the same kind of functions trained without applying CCA.

With respect to Figure 7, we can see that the functions trained with a greater number of data points and without applying CCA (i.e., $MnRE^{\hat{A}}_{NO\text{-}CCA}$) provide a distribution slightly sharper than the one obtained by applying CCA. However, this is the effect of using more data points. In fact, this spread improvement is not confirmed in Figure 8, where the standard deviation obtained by applying CCA is better than the one obtained without CCA.

Although, running a CCA procedure requires non-negligible effort, this loss can be compensated by obtaining estimates that are more accurate. If CCA cannot provide better estimates, it can improve the estimation reliability. In fact, we have shown that CCA is able to provide distributions having fewer outliers (Figures 5, 6, 7, and 8). Then, practitioners and organizations dealing with software estimates may apply CCA for reducing the number of outliers even though the accuracy would not be improved. Since reducing the number of outliers expresses reliability, CCA can be a useful tool not only for improving estimation accuracy, but also for improving its reliability.

Another advantage of applying CCA is that, it can be used with any kind of data and prediction model (e.g., software cost, fault proneness, defect slippage). However, the effectiveness of applying CCA to different data sets and contexts has to be evaluated empirically through replicated experiments that we wish researchers would try out.

From a practical point of view, another advantage of applying CCA is that we do not need to know the relevance of each attribute being removed with respect to the considered context. This is an advantage because, for instance, stepwise feature selection requires knowing that [9]. Moreover, CCA does not suffer from multicollinearity (i.e., having two or more input variables whose effect cannot be separated on the output), which can affect stepwise methods. CCA overcomes this problem by considering the simultaneous effect of every input variable through a nonlinear auto-associative neural network, (i.e., CCA does not separate the effect of each variable on the output, but it finds a nonlinear, equivalent, and more compact representation keeping the effect of each variable along with the others. In fact, CCA reduces multicollinearity by finding out the redundant variables (i.e., variables that can be expressed by a linear or nonlinear transformation of other variables). A further advantage is that, CCA can be implemented as an automatic procedure for estimation model improvement. Note that, once we have implemented it for the first time, we can reuse it thereafter without changes. We believe that, results of the proposed work can be used by practitioners, academics, and organizations as a baseline for further empirical investigations aiming at figuring out whether CCA can be effectively applied to other data sets, as well.

CCA has some drawbacks. For instance, the procedure that we proposed in Section 3.3 is based on the assumption that we have enough data to split up the data set into two subsets (TrS and TsS). Conversely, CCA would not be applicable. CCA is based on NNs, which require knowing some optimization techniques to reduce the training time. Not applying any optimization technique, may increase the effort and reduce the gain of applying it. The successful application of the CCA to the COCOMO data set that we have shown in this work should be considered as a first step towards such an emerging approach, which may eventually integrate canonical statistics that the scientific community has effectively undertaken so far.

In the future, we plan to compare the proposed approach with other feature selection techniques based on stepwise methods [9] as well as explore the possibility of combining together CCA and stepwise to get benefits from both techniques.

7 REFERENCES

1. Boehm, B.: Software Engineering Economics. Prentice Hall (1981).
2. Boehm, B., Horowitz, E., Madachy, R., Reifer, D., Clark, B.K., Steece, B., Brown, A.W., Chulani, S., and Abts, C.: Software Cost Estimation with COCOMO II. Prentice Hall (2000).
3. Bishop C.: Neural Network for Pattern Recognition, Oxford University Press (1995).
4. The COCOMO II Suite: http://sunset.usc.edu/research/cocomosuite/index.html (2004).
5. Conte, S.D., Dunsmore, H.E., Shen, V.Y: Software Engineering Metrics and Models, The Benjamin/Cummings Publishing Company, inc., Menlo Park (1986)
6. Dreyfus G.: Neural Networks Methodology and Applications, Springer (2005).
7. Jollife, I.T.: Principal Component Analysis, Springer (1986).
8. Kitchenham, B., MacDonell, S., Pickard, L., and Shepperd, M.: What accuracy statistics really measure, IEE Proceedings - Software Engineering, 48, pp81-85 (2001).
9. Kohavi, R., John, G.H.: Wrappers for feature subset selection, Artificial Intelligence, Vol. 97, no.1-2, pp.273-324 (1997).
10. Rumelhart, D.E., Hilton, G.E., Williams, R.J.: Learning Internal Representations by Error Propagation, Parallel Distributing Computing: Explorations in the Microstructure of Cognition, pp. 318-362, MIT press (1986).
11. Sarcia, S.A., Basili, V.R., and Cantone, G.: An Approach to Improving Estimation Accuracy Over Time Based on Past Experience, TR001-ESEG-UMD-2008 (2008).
12. Shirabad, J.S. and Menzies, T.: The PROMISE Repository of Software Engineering Databases, School of Information Technology and Engineering, University of Ottawa, Canada. http://promise.site.uottawa.ca/SERepository (2005).
13. Vapnik, V.N.: The Nature of Statistical Learning Theory, Springer (1995).
14. Weisberg, S.: Applied Linear Regression, 2nd Ed., John Wiley and Sons (1985).

Appendix A

Table 1

Experimental Results

OBJ	$MnRE_{CCA}$	$STDRE_{CCA}$	$MnRE_{NO-CCA}$	$STDRE_{NO-CCA}$	$MnRE^{A}_{NO-CCA}$	$STDRE^{A}_{NO-CCA}$
1	0.14371208	-0.054929955	1.0562069	0.53698988	-5.80E+09	0.53699
2	0.21722044	-0.35892088	0.43654432	0.89398832	-22726.606	0.893988
3	-0.16971426	-0.96359809	0.9497333	3.4363606	-0.8234748	3.436361
4	0.27820802	-0.57894423	0.40653336	2.1020914	-0.7888032	2.102091
5	-0.29615245	-1.4307502	0.83354207	3.9235776	-0.9651689	2.401476
6	-0.30193239	-0.32886636	0.9661099	0.77139013	-2.9967061	3.923578
7	-0.2996719	-0.25792587	0.8046919	0.78477421	-2703.6933	0.77139
8	-0.36172906	-0.28649363	0.84352196	0.77192256	0.005064	0.784774
9	-0.099800576	-0.18388187	1.1393905	0.80648144	-0.0846081	0.771923
10	0.12056456	0.078546418	0.42813771	0.35174426	0.01248237	0.806481
11	-0.05480207	0.027372728	0.94871949	0.36093028	-2.2124166	0.351744
12	-0.08093991	0.054227376	1.2444559	0.28582551	-0.2821303	0.36093
13	-0.005545097	0.05052095	0.8794604	0.29834485	-0.339955	0.285826
14	-0.34571939	-0.072447687	1.0164966	0.22644037	-0.1514146	0.298345
15	-0.22464008	-0.086719192	0.394165	0.23561617	-7158.3085	0.22644
16	-0.39813303	-0.16536665	0.8162177	0.38368032	-4.45E+35	0.235616
17	-0.21172515	-0.23266126	0.97092145	0.45347181	-280.54155	0.38368
18	0.004817188	-0.22589982	1.0815658	0.47739908	0.99145512	0.445251
19	0.2684083	-0.32992664	0.40555368	0.52906098	0.13757051	0.453472
20	0.04383431	-0.33450622	0.51076848	0.59924966	0.98993829	0.477399
21	-6.4798323	-0.32688714	13.513739	0.58941184	-3932705.1	0.529061
22	0.08410293	-0.36335451	0.52954395	0.76358409	-1.41E+18	0.59925
23	0.03498114	-0.55314384	1.1002098	1.1088234	-5.5540794	0.589412
24	0.3896364	-0.36060205	0.59261704	0.68180914	0.15092503	0.763584
25	0.23389932	-0.38411264	0.8175946	1.0134382	-195.4707	1.108823
26	0.29336838	-0.16646417	0.97438834	0.53712992	-39.399898	0.681809
27	0.34308829	-0.15403468	0.50449756	0.45911063	-2.31E+08	1.013438
28	0.20007009	-0.28893062	0.34205892	3.9460798	-342.03693	0.53713
29	-0.387412	-0.22527862	1.3345788	0.44525143	-1.01E+27	0.459111
30	-0.33046474	-0.65085382	0.8553278	2.4014763	-6.8954419	3.94608