Software Effort Estimation using Machine Learning Techniques with Robust Confidence Intervals

Petrônio L. Braga and Adriano L. I. Oliveira, *IEEE Senior Member*
Department of Computing Systems, Polytechnic School of Engineering, Pernambuco State University, Recife 50.750-410, Brazil
E-mail: {plb, adriano}@dsc.upe.br

Silvio R. L. Meira
Center of Informatics, Federal University of Pernambuco, Recife 50.732-970, Brazil
E-mail: srlm@cin.ufpe.br

Abstract

The precision and reliability of the estimation of the effort of software projects is very important for the competitiveness of software companies. Good estimates play a very important role in the management of software projects. Most methods proposed for effort estimation, including methods based on machine learning, provide only an estimate of the effort for a novel project. In this paper we introduce a method based on machine learning which gives the estimation of the effort together with a confidence interval for it. In our method, we propose to employ robust confidence intervals, which do not depend on the form of probability distribution of the errors in the training set. We report on a number of experiments using two datasets aimed to compare machine learning techniques for software effort estimation and to show that robust confidence intervals for the effort estimation can be successfully built.

1. Introduction

The use of software grows continuously in the society. Software factories need to produce software of high quality and in time to assure competitiveness. The high competitiveness forces the software industry to conclude software projects as planned, that is, in time and within budget. Therefore, the need for planning and project management has increasingly demanded more attention and control from software project managers. Estimation of the effort is one of the most important tasks in software planning and management.

In a report from the Standish Group's Chaos, 66% of the software projects analyzed were delivered with delay or above the foreseen budget, or worse, they were not finished [11]. One of the major causes of such failures is inaccurate estimates of the effort of the projects [4]. Hence, it is very important to investigate novel methods for improving the accuracy of such estimates. The precision of the effort estimate is very important for software factories because both overestimates and underestimates of the software effort are harmful to software companies.

Several methods have been investigated for software effort estimation, including traditional methods such as the COCOMO [12], and, more recently, machine learning techniques such as radial basis function (RBF) neural networks [1], bagging predictors [9] and support vector regression (SVR) [4]. Machine learning techniques use data from past projects to build a regression model that is subsequently employed to predict the effort of novel projects.

In most methods for software effort estimation, only the estimations of the efforts are given [1][4][6][9][12]. Yet, it would be very important to provide a *confidence interval* for the estimation along with the estimation [10]. This would enable an estimation method to give an interval where the effort would fall.

In many applications, the confidence intervals for predictions are computed by assuming that the errors follow some probability distribution. Often, it assumed that the errors follow a Gaussian distribution. The errors are then used to estimate the parameters and compute the confidence interval. Unfortunately, in many cases the distribution assumed for the errors may not be correct. Robust confidence intervals are more general because they do not make any assumptions about the errors [10]. They were applied recently by Oliveira and Meira to detect novelty in time series [10].

This paper proposes the use of *robust confidence intervals* for defining a confidence interval for software effort estimates. Robust confidence intervals are directly computed from the prediction errors in the training set of the regression algorithm for the datasets under analysis.

This paper is organized as follows. In Section 2 we briefly review some recent related works. Section 3 reviews the regression methods used in this paper. In

1082-3409/07 $25.00 © 2007 IEEE
DOI 10.1109/ICTAI.2007.172

Section 4, we present the method for building robust confidence intervals for predictions. The experiments and results are discussed in Section 5. Finally, Section 6 presents the conclusions.

2. Related Works

This section offers a review of some recent research in the area of estimation of software project effort. This is an active research area, with a number papers published recently. In this section we focus our review on some important papers which have applied machine learning techniques to estimate software project effort.

Shin and Goel [1] noticed that many empirical methods in the area of software engineering use linear regression analysis. Alternatively, they proposed the use of RBF neural networks to provide a flexible way to generalize the linear regression function. Shin and Goel its application of RBF on software effort estimation by using a NASA database of software projects [1].

Burgess and Lefley proposed to investigate the potential of genetic programming in the software effort estimation [8]. They compared their method applied to linear LSR (least square regression), kNN (k - nearest neighbor) and MLP neural networks. In the simulations, they employed the Desharnais dataset. The results showed that genetic programming can obtain satisfactory results but required more time in the experiments.

Oliveira proposed to apply SVR to software effort estimation [4]. He used linear and RBF kernels for the simulations with SVR. The NASA database was used in the experiments. The simulation results showed that SVR outperforms previous results reported in the literature obtained by both RBFNs trained with the SG algorithm [1] and linear regression [4].

Neural networks work as black boxes, with the knowledge distributed in the network, making the interpretation of the results difficult [7]. To overcome this limitation, Huang et al. proposed a neuro-fuzzy Constructive Cost Model for software effort estimation that has good interpretability [12]. This model is based on a neuro-fuzzy approach that works well with imprecise and uncertain inputs and has good generalization capability. Simulations showed that the neuro-fuzzy model outperformed the traditional COCOMO method in software effort estimation [12].

Braga et al. proposed and showed that bagging predictors are able to improve performance of regression methods for estimation of software project effort [9]. Braga et al. also employed a NASA database of software projects in the experiment and showed that bagging with M5P/model trees significantly outperforms linear regression and RBF networks in this task. It was also demonstrated that bagging with M5P/model trees obtains results equivalent to those of SVR [4], with the advantage of producing more interpretable results [9].

In the methods of all the papers reviewed in this section only the estimates of the effort are given for new projects. In contrast, in this paper we propose to generate robust confidence intervals for the estimates, aiming to increase the reliability of the estimates.

3. Regression Methods

The goal of regression methods is to build a function $f(x)$ that adequately maps a set of independent variables $(X_1, X_2,..., X_n)$ into a dependent variable Y. In our case, we aim to build regression models using a training dataset to use subsequently to predict the total effort for the development of software projects in man-months.

3.1. Regression-based Trees

The regression tree is a special type of decision tree developed for regression tasks [3]. Like the regression trees the model trees are also decision trees. However, the main difference between regression trees and model trees is that the leaves of regression trees present numerical values, whereas the leaves of a model tree have linear functions which perform (linear regression).

In this paper we used the M5P algorithm for building regression-based trees [3][6]. M5P is powerful because it implements as much decision trees as linear regression for predicting a continuous variable [3]. Moreover, the M5P implements both regression trees and model trees[3].

The M5P algorithm has three stages [3]: building a tree, pruning the tree and smoothing.

3.2. Multi-Layer Perceptron

The Multi Layer Perceptron (MLP) neural network has been applied successfully to a variety of problems such as regression, time series forecasting and classification [7]. MLPs are feedforward networks that consist of an input layer, one or more hidden layers and an output layer.

In the project of an MLP the main parameters that influence performance are: (1) the number of hidden layers, (2) the number of neurons of each hidden layer, (3) the number of training epochs, and (4) the learning rate and the momentum (parameters of the back-propagation training algorithm). These parameters have to be carefully selected for boosting the performance of the MLP for a given problem [7].

3.3. Support Vector Regression

Support vector machines (SVMs) are a set of machine learning methods used in many areas, such as classification and regression [2][7]. SVMs are based on

structural risk minimization (SRM); instead of attempting to minimize only the empirical error, SVMs simultaneously minimize the empirical error and maximize the geometric margin [7]. This method has outperformed previous ones in many classification and regression tasks.

SVR was designed for regression problems [7]. The investigation of the application of SVR for software project effort estimation was originally carried out by Oliveira [4]. The SVR is defined by the parameters C - the complexity parameter, ε - the extent to which deviations are tolerated, and the kernel parameter (γ in the case of the RBF kernel) [4]. These parameters significantly influence SVR performance. For more details on SVR, see [2][4].

3.4. Bagging Predictors

The bagging is a technique proposed by Breiman [5] that can be used with many classification and regression techniques to reduce the variance associated with the predictions, thereby improving the results.

The idea consists of using multiple versions of a training set; each version is created by randomly selecting samples of the training dataset, with replacement, where *n* is the number of training samples of the original training set [5]. For each dataset created, a regression model is built by applying a previously selected learning algorithm. The learning algorithm must be the same in all iterations. The final prediction is given by the average of the predictions of each regression model created [5]. Bagging has given good results whenever the learning algorithm is unstable [5].

4. Building Robust Confidence Intervals for Software Effort Estimation

In most papers on software effort estimation, especially those using machine learning for regression, only the estimated values are provided [1][4][9][12]. In practice, however, it is very important to provide an indication of the reliability of these predictions as well [10]. This can be achieved by using confidence intervals. In several regression applications these intervals are computed by supposing that the errors follow a normal (Gaussian) distribution or another more general distribution with a number of parameters. Nevertheless, in practice this assumption may not hold and therefore can lead to wrong confidence intervals. Robust confidence intervals were proposed to overcome this limitation, since they do not make assumptions about the distributions of the errors.

4.1. Computing Robust Confidence Intervals

A robust confidence interval for predictions is computed from errors collected from the training set after the regression model is built. The intuition behind robust confidence intervals is that errors of a given size will tend to occur with about the same frequency that they have occurred previously. In other words, the frequency of occurrence of errors of a given size in the training set is assumed to be the same when the machine learning technique is used in the test phase (or in practice, that is, to estimate the effort of novel software projects).

To build robust confidence intervals, the collection of errors obtained in the training phase is firstly sorted in ascending order. Next, the distribution function of the errors, $S_n(e)$, is estimated as follows:

$$S_n(e) = \begin{cases} 0 & \text{if } e < e_1, \\ r/n & \text{if } e_r \le e < e_{r+1}, \\ 1 & \text{if } e_n \le e. \end{cases}$$

The collection of errors used to calculated $S_n(e)$ is representative of the errors that will be obtained when the regression model is used in practice. If the collection of errors is large enough then $S_n(e)$ can be assumed to be close to $F_n(e)$, the true error distribution. In this case the confidence intervals for upcoming predictions are computed simply by keeping as much of the interior of $S_n(e)$ as is desired and by using the limits of this truncated collection to define the confidence interval.

Let *n* be the number of training samples and *p* be the desired confidence level. For very large error collections and moderate values of *p*, $n \times p$ values should be discarded from each extreme in order to build the confidence intervals. For smaller samples, however, the recommended amount to be discarded from each extreme is $n \times p - 1$ [10]. If the result is a real number it should be truncated. *p* is the fraction of probability in each tail of the distribution function.

The process used to build robust confidence intervals can be divided in six stages, described below:

1. Train a regression model using *n* projects;
2. Obtain the collection of *n* errors;
3. Sort the collection of errors in ascending order;
4. Calculate the fraction of probability;
5. Discard $n \times p - 1$ errors from each extreme of the collection of errors;
6. Obtain robust confidence intervals for future predictions.

One practical example that illustrates this procedure used to build robust confidence intervals can be found in Oliveira and Meira [10].

5. Experiments

In this paper, we used two software effort datasets to evaluate the proposed method. The first was the Desharnais dataset [8]. The second was a dataset from NASA [1][4][9].

Table 1. Results obtained for the Desharnais dataset.

Method/parameters	MMRE	PRED(25)	Confidence Intervals
SVR RBF (C = 10, ε = 0.001, γ = 0.01)	0.4736	55.56	$[\hat{x}_p$ - 11349.535, $\hat{x}_p$ + 2696.394]
MLP (L = 0.01, M = 0.3, H = 9, N = 2000)	0.4069	**66.67**	$[\hat{x}_p$ - 7905.373, $\hat{x}_p$ + 3860.582]
M5P(P = true, N = 8) / model trees	0.6135	55.56	$[\hat{x}_p$ - 9120.019, $\hat{x}_p$ + 4570.13]
Bagging(S = 90, C = SVR RBF (C = 10, ε = 0.001, γ = 0.01), I = 10)	0.4636	55.56	$[\hat{x}_p$ - 10987.109, $\hat{x}_p$ + 3310.266]
Bagging(S = 90, C = MLP (L = 0.01, M = 0.3, H = 12, N = 2500), I = 10)	**0.3991**	**66.67**	$[\hat{x}_p$ - 8706.91, $\hat{x}_p$ + 4177.257]
Bagging(S = 70, C = M5P(P = true, N = 8), I = 10) / model trees	0.6054	55.56	$[\hat{x}_p$ - 8658.613, $\hat{x}_p$ + 6010.493]

The Desharnais dataset consists of 81 software projects described by 11 variables, nine independent and two dependent [8]. The first dependent variable is the effort. The second dependent variable (length) was ignored because our aim is to predict the total effort to develop software projects. For our simulations we divided the dataset into two disjoint sets: the training set and the test set. To form the test set we randomly selected 18 projects from the 81 projects [8]. The remaining 63 projects were used to form the training set. This procedure was also used by Burgess and Lefley [8].

The NASA dataset consists of two independent variables, Developed Lines (DL) and Methodology (ME), and one dependent variable, Effort (Y) [4]. For our simulations, we have employed leave-one-out cross-validation (LOOCV) to estimate the generalization error [1][4][9]

In this paper we employ the two measurements that are most commonly used in the literature [1][4][8][9]. They are the MMRE and the PRED(25).

In the M5P simulations, we used the following values for the parameters: unPruned (P) = {false, true}, useUnsmoothed (S) = {false, true} and numInstances (N) = {1, 4, 6, 7, 8, 9, 10}. In the MLP simulations, we used the following parameters: learning rate (L) = {0.01}, momentum (M) = {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8}, number of hidden neurons (H) = {6, 7, 8, 9, 10, 11, 12, 13}, number of training epochs (N) = {1500, 2000, 2500}. In the SVR simulations, we used the following values for the parameters C and ε: C = {10; 100; 1000} and ε = {10^{-3}; 10^{-4}; 10^{-5}}. For RBF kernels we have another important parameter, γ (gamma value). We considered the values γ = {1; 10^{-1}; 10^{-2}}.

In the simulations with bagging, we executed the experiments using the following parameters: bagSizePercent (S) = {100, 90, 80, 70}, regression method (C) = {MLP, M5P/regression tree, SVR - kernel RBF, SVR - kernel linear}, numIterations (I) = {10, 25}.

Table 1 shows the best results obtained by each method for the Desharnais dataset in this paper together with the robust confidence intervals built as described in Section 3.

The results of Table 1 show that bagging improved the performance of the following methods: SVR-RBF (MMRE), MLP (MMRE) and M5P (MMRE) / model tree. We can also observe that it increases the value of the superior limit of the confidence intervals.

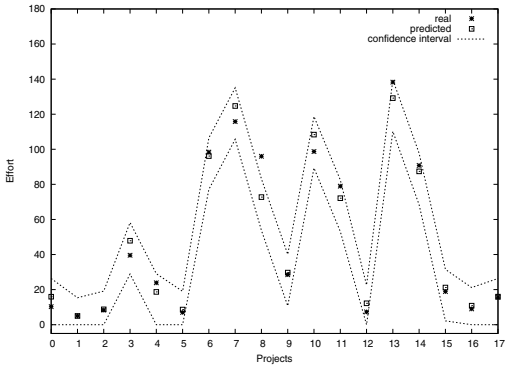

Figure 1. Robust confidence intervals for predictions in the NASA dataset using bagging with M5P.

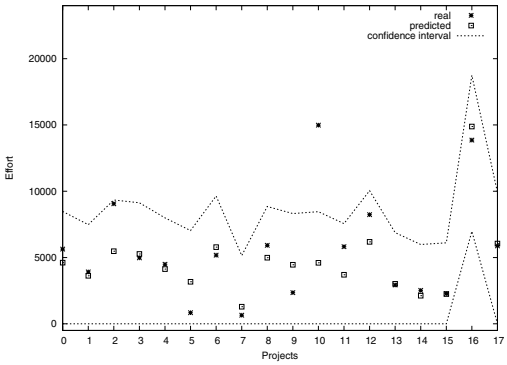

Figure 2. Robust confidence intervals for predictions in the Desharnais's test set using MLP.

We also carried out simulations using the NASA dataset. The best results obtained by each method are shown in Table 2. MLP and bagging with MLP achieved the best results for the NASA dataset in terms of PRED(25). The results obtained by these methods were superior to previous results reported in the literature obtained with RBFNs, SVR and Bagging with M5P/model trees [1][4][9].

Table 2. Results obtained for the NASA dataset with DL-ME-Y data.

Method/parameters	MMRE	PRED(25)	Confidence Intervals
SVR Linear (C = 10, $\varepsilon = 10^{-4}$)	0.1651	88.89	$[\hat{x}_p - 18.589,\ \hat{x}_p + 12.2]$
MLP (L = 0.01, M = 0.6, H = 6, N = 2000)	0.203	**94.44**	$[\hat{x}_p - 16.214,\ \hat{x}_p + 9.553]$
M5P(P = true, N = 7) / model trees	0.1778	83.33	$[\hat{x}_p - 23.077,\ \hat{x}_p + 10.686]$
Bagging(S = 80, C = SVR Linear (C = 10, $\varepsilon = 10^{-4}$), I = 10)	0.1717	88.89	$[\hat{x}_p - 19.973,\ \hat{x}_p + 11.092]$
Bagging(S = 90, C = MLP (L = 0.01, M = 0.6, H = 6, N = 2000), I = 10)	0.1771	**94.44**	$[\hat{x}_p - 14.856,\ \hat{x}_p + 10.801]$
Bagging(S = 100, C = M5P(P = true, N = 6), I = 10) / model trees	**0.1639**	88.89	$[\hat{x}_p - 19.001,\ \hat{x}_p + 10.371]$

Finally, we performed experiments building robust confidence intervals for the predictions with 95% confidence level. Figs. 1 and 2 show real and predicted values along with the robust confidence intervals for the methods that obtained the best results for the NASA dataset and Desharnais set respectively. In each one of these graphics the robust confidence intervals for the predictions are also depicted. These graphics show that most real values from the test set lie within the confidence intervals boundaries for the predictions. Only one of the projects lie outside the confidence intervals.

6. Conclusion

We have proposed the use of machine learning techniques combined with robust confidence intervals to improve the precision of estimates of software project effort. In our simulations, we used two datasets of software projects: Desharnais and NASA. The simulation results showed that bagging was able to improve performance of the following regression methods in the effort estimation task: (1) SVR, (2) MLP and (3) model trees (M5P).

We compared our results to previous ones published in the literature. The comparisons were made considering results reported in [1][4][9]. The results showed that bagging with M5P/model trees achieved the best performance in terms of MMRE. Bagging with MLP outperformed previous methods reported in the literature in terms of PRED(25) [1][4][9].

The simulations have also shown that the proposed method was able to build robust confidence intervals. This is of great importance for the user of a software effort estimation system, since such a system will provide an interval together with the estimation of the effort instead of the single estimation provided by traditional methods based on machine learning.

Acknowledgments

The authors would like to thank CNPq (Brazilian Research Agency) and FACEPE (Pernambuco Reseach Agency) for their financial support.

References

[1] M. Shin, A.L. Goel, “Empirical data modeling in software engineering using radial basis functions”. *IEEE Transactions on Software Engineering*, vol. 26, no. 6, pp. 567-576, June 2000.

[2] A. J. Smola, B. Scholkopf, “A tutorial on support vector regression”. *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.

[3] I. H. Witten, E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed., Morgan Kaufmann, San Francisco, 2005.

[4] A. L. I. Oliveira, “Estimation of software projects effort with support vector regression”. *Neurocomputing*, vol. 69, no. 13-15, pp. 1749-1753, August 2006.

[5] L. Breiman, “Bagging Predictors”. *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.

[6] T. Menzies, Z. Chen, J. Hihn and K. Lum, “Selecting best practices for effort estimation”. *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 883-895, November 2006.

[7] S. Haykin, *Neural Networks: a Comprehensive Foundation*. Prentice Hall, New Jersey, USA, 1999.

[8] C. J. Burgess, M. Lefley. “Can genetic programming improve software effort estimation? A comparative evaluation”. *Information and Software Technology*, vol. 43, pp. 863-873, 2001.

[9] P. L. Braga, A. L. I. Oliveira, G. H. T. Ribeiro and S. R. L. Meira. “Bagging predictors for estimation of software project effort”. In: *IEEE/INNS International Joint Conference on Neural Networks*, IJCNN’ 2007, Orlando-Florida.

[10] A. L. I. Oliveira and S. R. L. Meira. “Detecting novelties in time series through neural networks forecasting with robust confidence intervals”. *Neurocomputing*, vol. 70, no.1-1, pp, 79-92, December 2006.

[11] Software Magazine; “Standish: Project success rates improved over 10 years”; http//www.softwaremag.com/L.cfm?Doc=newsletter/2004-01-15/Standish

[12] X. Huang, D. Ho, J. Ren and L. F. Capretz, "Improving the COCOMO model using a neuro-fuzzy approach". *Applied Soft Computing*, vol. 7, pp, 29-40, January 2007.