Software Effort Prediction using Regression Rule Extraction from Neural Networks

Rudy Setiono *, Karel Dejaeger †, Wouter Verbeke †, David Martens † ‡, and Bart Baesens †
*National University of Singapore
3 Science Drive 2, Singapore 117543
rudys@comp.nus.edu.sg
†Catholic University of Leuven
Department of Decision Sciences and Information Management
{firstname.lastname}@econ.kuleuven.be
‡University College Ghent, University Ghent
Department of Business Administration and Public Management

***Abstract*—Neural networks are often selected as tool for software effort prediction because of their capability to approximate any continuous function with arbitrary accuracy. A major drawback of neural networks is the complex mapping between inputs and output, which is not easily understood by a user. This paper describes a rule extraction technique that derives a set of comprehensible IF-THEN rules from a trained neural network applied to the domain of software effort prediction. The suitability of this technique is tested on the ISBSG R11 data set by a comparison with linear regression, radial basis function networks, and CART. It is found that the most accurate results are obtained by CART, though the large number of rules limits comprehensibility. Considering comprehensible models only, the concise set of extracted rules outperform the pruned CART tree, making neural network rule extraction the most suitable technique for software effort prediction when comprehensibility is important.**

***Index Terms*—Data mining, Software effort prediction, Rule extraction**

I. Introduction

Resource planning is considered a key issue in business environments. In the context of a software developing company, the different resources are amongst others computers, workspace, and personnel. In recent years, computing power has become a less important resource for software developing companies as computing power doubles nearly every 18 months. The personnel costs are often a considerable expense in the budget of software developing companies. Hence, proper planning of personnel effort is a key aspect for companies. There is a growing interest in the literature involving software effort prediction [1]. In this field of research, the effort needed to develop a new project is estimated based on historical data from previous projects. This information can be used by the management to improve the planning of personnel, to make more accurate tendering bids, and to evaluate risk factors.

The first attempts to estimate software development effort date back to the late 60's [2]. In these cases, expert judgement in which a domain expert applies his/her prior experience to estimate the effort was utilized. Since then, a myriad of estimation techniques have been applied to software effort prediction. Typically, a distinction between formal models and data mining approaches is made. Formal models rely upon a preset formula, often relating only project size to development effort. A limited set of other attributes are used as (less important) markup factors. For example, the COnstructive COst MOdel (COCOMO) I intermediate model relates project size to development effort, and 15 markup factors, taking the following form:

$$\text{Effort} = a \times \text{size}^b \times \left(\prod_{j=1}^{15} EM_j^{a_j} \right) \tag{1}$$

where $EM_1, \ldots, EM_{15}$ are effort multipliers which are used to take specific properties of a project into account. Size is expressed in SLOC (Source Lines Of Code) or an equivalent measure, and a, b, and a_j are calibration coefficients, estimated using statistical techniques such as regression or taken from the literature [3]. Other well known formal models include the Software LIfe cycle Model (SLIM) [4] and Function Point Analysis (FPA) [5].

While the formulaic underpinning of these models allows for an (somewhat limited) analysis of the estimates, the use of formal models has a number of drawbacks, including the limited set of allowed inputs and the somewhat subjective nature of these models. More recent, formal models have been superseded by a number of techniques originating from the data mining literature [1]. Common data mining approaches include techniques like regression, CART, neural networks, radial basis function networks, and others [6]. Some of these techniques, such as neural networks, can be considered to be black boxes as the relation between inputs and output is non linear, thus limiting the comprehensibility to the end user [7, 8]. Due to this lack of comprehensibility, the applicability of neural networks in a business setting is limited.

Data sets in the field of software engineering tend to be difficult to collect due to the nature of the data [9]. A number Table I provides a non exhaustive overview of data sets frequently used in the software effort prediction literature. For most data mining techniques, data sets with a sufficient number of observations are preferred to allow for better generalization

TABLE I
OVERVIEW OF SOFTWARE EFFORT PREDICTION DATA SETS USED IN PREVIOUS RESEARCH

Data set	Number of projects	Number of features	Public [1]
Kemerer [16]	15 projects	3 features	✓
DPS database	24 projects	5 features	
Rao et al. [17]	36 projects	14 features	
Cocomo81 [3]	63 projects	16 features	✓
Desharnais [18]	81 projects	11 features	✓
ISBSG R11	5052 projects	38 features	

on previously unseen samples. From this overview, it can be concluded that the ISBSG R11 data set is one of the largest data sets available for software effort prediction.

The contribution of this paper lies in the application of a novel technique for regression rule extraction from neural networks (NNs) in the domain of software effort prediction. The application of rule extraction to NNs results in a set of comprehensible IF-THEN rules that relate inputs to output (e.g. predicted effort). This allows to combine the strong elements of NNs such as the ability to capture non linearities in the data with the comprehensibility of a set of IF-THEN rules. The NN rule extraction algorithm is compared to Ordinary Least Squares (OLS) regression, Radial Basis Function Networks (RBFN), and Classification And Regression Trees (CART) to assess applicability and performance of this technique in the field of software effort prediction.

NNs have been extensively investigated for software effort prediction (e.g. see [10, 11, 12, 13, 14]), while the application of rule extraction is less extensively investigated in this context. Currently, we are only aware of a study by Idri *et al.*, in which a fuzzy rule extraction algorithm was used resulting in a difficult to understand rule set [15].

The remainder of this paper is structured as follows. Section II provides an overview of previous studies that applied NNs. Section III explains the different techniques used in this study. In Section IV, background information concerning the data set and the methodology of our study is given. Section V discusses the results, and the paper is concluded in Section VI.

II. RELATED RESEARCH

Neural networks (NNs) are mathematical representations inspired by the functioning of the human brain [19], and have previously been applied in various contexts, including software effort prediction, as they enjoy some beneficial properties.

- NNs have previously been applied with success on data sets with complex relationships between inputs and output, and where the input data is characterized by high noise levels [20].
- NNs with one hidden layers have been proven to be universal approximators which can approximate any continuous function to a desired degree of accuracy [21].

Feedforward neural networks are most commonly used; i.e. networks containing no recursive loops. A number of previous studies assessed the applicability of NNs in software effort prediction. To learn the underlying relationships within the data, NNs need sufficient observations. Table I provides the number of observations of the data sets discussed in the following paragraphs. For instance, Srinivasan *et al.* compared CART and NNs using a combination of the COCOMO81 and the Kemerer data set. It was found that the data mining techniques performed similarly to formal model approaches [10].

Finnie *et al.*, while comparing NNs, regression, and case based reasoning on the ASMA data set [2], found that NNs were capable of effectively predicting software effort, but at the expense of reduced comprehensibility [11].

This was confirmed by Wittig *et al.* who assessed NNs using the Desharnais data set and an artificially generated data set. NNs were found to outperform regression and case based reasoning thus NNs were considered to be a promising technique in a software effort prediction context [12].

Heiat assessed two different types of NNs, multilayered perceptron and radial basis function networks, and compared it to linear regression using the DPS database and the Kemerer data set. It was concluded that the NN approach was competitive to the regression approach [13].

More recently, a study by Shukla *et al.* also confirmed these results analyzing the Rao data set, stating that NNs can be used for software effort prediction [14].

It can be concluded from previous research that NNs can be applied with success to software effort prediction. However, as the mapping from inputs to output in a neural network is unclear to the end user, the resulting model can be considered to be a black box. However, comprehensibility is of key importance as the predictive model needs to be validated before it is put in use in a business context [22, 23, 24]. Well documented, formal models like COCOMO and FPA are often the preferred choice in businesses for this reason. Also techniques from which a rule set can be derived, such as CART, are often used.

A number of rule extraction techniques exists that derive a comprehensible rule set from a NN. The majority of these rule induction techniques are however only applicable in the context of classification problems (i.e. with a discrete class variable) [25]. In case of a continuous target attribute such as development effort, one possibility is to discretize the target using equal width, equal binning or clustering, and successively apply a rule extraction technique. This idea is used for instance in the RECLA system [26]. In this paper, an alternative approach is proposed by generating linear regression rules from a trained neural network for software effort prediction, thus not requiring discretisation.

III. TECHNIQUES

In this section, the algorithm for regression rule extraction from NNs is described. First however, an introduction to CART, OLS, RBFN, and NNs is provided.

[1]Public data sets can be found at the Promise Repository: http://promisedata.org

[2]The ASMA data set is now know as the ISBSG data set which is also used in this study

The following notation is used throughout the remainder of the paper. A scalar $x \in \mathbb{R}$ is denoted in normal script while a vector $\mathbf{x} \in \mathbb{R}^n$ is in boldface script. A vector is always a column vector unless indicated otherwise. A row vector is indicated as the transposed of the associated column vector, $\mathbf{x}'$. A matrix $\mathbf{X} \in \mathbb{R}^{N \times n}$ is in bold capital notation. $x_i(j)$ is an element of matrix $\mathbf{X}$ representing the value of the j^{th} variable on the i^{th} observation. N is used as the number of observations in a data set while n represents the number of variables.

In the ISBSG data set, the target variable used is effort in man-hours. The actual effort of the i^{th} software project is indicated as e_i while the predicted effort is indicated as $\hat{e}_i$.

A. OLS regression

Ordinary Least Squares (OLS) regression is arguably one of the oldest and most widely applied techniques for software effort prediction. In case of this well documented technique, the goal is to fit a linear regression function to a data set containing a dependent, e_i, and multiple explanatory variables, $x_i(1)$ to $x_i(n)$. OLS regression assumes the following linear model of the data:

$$e_i = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i \tag{2}$$

where $\mathbf{x}_i'$ represents the row vector containing the values of the i^{th} observation, $x_i(1)$ to $x_i(n)$, and ϵ_i the error associated with each observation. $\boldsymbol{\beta}$ is the column vector containing the regression parameters that are to be estimated by minimizing the squared error, thus obtaining the following estimate for $\boldsymbol{\beta}$:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}(\mathbf{X}'\mathbf{e}) \tag{3}$$

with $\mathbf{e}$ representing the column vector containing the effort and $\mathbf{X}$ a $N \times n$ matrix with the associated explanatory variables.

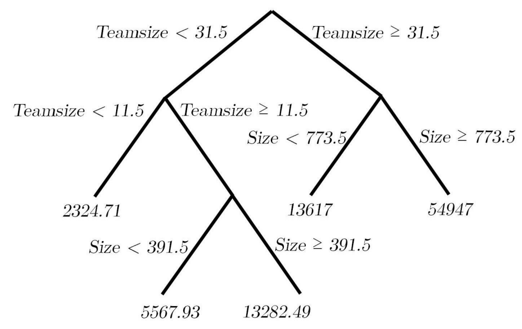
B. CART

CART (Classification And Regression Tree) [27] is an algorithm that takes the well known idea of decision trees for classification, and adjusts it to a regression context by recursively splitting the data using e.g. a least squared deviation criterion:

$$\min \left[\sum_{i \in L} (e_i - \overline{\mathbf{e}}_L)^2 + \sum_{i \in R} (e_i - \overline{\mathbf{e}}_R)^2 \right] \tag{4}$$

The stopping criterion is set as a minimum of 10 observations at the terminal nodes. Fig. 1 presents a binary regression tree constructed by applying CART to the ISBSG R11 data set.

The good comprehensibility of regression trees can be considered a strong point of this technique. To determine the effort needed for a new project, it is sufficient to select the appropriate branches based on the characteristics of the new project. It is possible to construct an equivalent rule set based on the obtained regression tree (Fig. 1, bottom).



Equivalent rule set
IF Teamsize >= 31.5 AND Size < 773.5 THEN effort = 13617
IF Teamsize >= 31.5 AND Size >= 773.5 THEN effort = 54947
IF Teamsize < 11.5 THEN effort 2324.71
IF Teamsize >= 11.5 AND Teamsize < 31.5 AND
Size < 391.5 THEN effort = 5567.93
ELSE effort = 13282.49

Fig. 1. Pruned CART tree induced on the ISBSG R11 data set

C. Radial basis function networks

Radial Basis Function Networks (RBFN) are a special case of neural networks, rooted in the idea of biological receptive fields [28]. A RBFN is a three-layer feedforward network consisting of an input layer, a hidden layer typically containing multiple neurons, and a linear output layer. Due to the continuous target, a special type of RBFN is used, called Generalized Regression Neural Networks [29]. The output of the hidden units is calculated by a radial symmetric gaussian transfer function, radbas(x_i):

$$\text{radbas}(x_i) = e^{-||c_k - x_i|| \times b^2} \tag{5}$$

where c_k is the k^{th} cluster centroid, $||.||$ the Euclidian distance between two points, and b a bias term. Hence, each i^{th} neuron has its own receptive field in the input domain: a region centered on c_k with size proportional to the bias term, b. The final effort estimates are obtained by multiplying the output of the hidden units with the vector consisting of the targets associated with the cluster centroids c_k, and then inputting this result into a linear transfer function.

D. Neural networks

Another type of neural networks often considered in the field of software effort prediction are multilayer perceptron (MLP) networks. Again, a MLP network is typically a three-layer feedforward network with each layer consisting of several neurons. With the inputs of each of the neurons, a weight is associated. Assume $\mathbf{w}_m$ is the vector of weights associated with the inputs of the mth hidden unit and v_m is the scalar representing the weight associated with the output of the mth hidden unit.

A hyperbolic tangent transfer function, tanh(.), is adopted in the hidden nodes such that the final effort estimation is given by the following equation:

$$\hat{e}_i = \sum_{m=1}^{H} \tanh(\mathbf{x}_i'\mathbf{w}_m) v_m + \tau \tag{6}$$

where τ is the output bias term. Once the network has been trained, irrelevant and redundant hidden units and input units are removed from the network by applying the N2PFA (Neural Network Pruning for Function Approximation) algorithm [30].

E. Regression rule extraction

A set of regression rules can be induced from the trained NN by applying REFANN (Rule Extraction from Function Approximating Neural Networks) [31, 32]. This technique uses a piece-wise linear approximation, L(ξ), of the hyperbolic tangent activation function, tanh(ξ), consisting of three line segments, Fig. 2.

The 3-piece linear approximation, L(ξ), is obtained by minimizing the sum of the squared deviations:

$$\min_{\xi_0,\beta_0,\beta_1} \sum_{i=1}^{K} \left(\tanh(\xi_i) - L(\xi_i)\right)^2, \tag{7}$$

where $\xi_i = \mathbf{x}_i'\mathbf{w}$, the weighted input of sample $i, i = 1, 2, \ldots, N$ and

$$L(\xi) = \begin{cases} -\alpha_1 + \beta_1 \xi & \text{if} \quad \xi < -\xi_0 \\ \beta_0 \xi & \text{if} \quad -\xi_0 \le \xi \le \xi_0 \\ \alpha_1 + \beta_1 \xi & \text{if} \quad \xi > \xi_0 \end{cases}$$

As tanh(0) = 0, L(0) is also fixed at 0. Due to symmetry of the tanh(ξ) function, the slope of the first and third line segment are equal, and the intercept between the first (third) and the middle line segment is $-\xi_0$ (ξ_0).

A set of comprehensible IF-THEN regression rules can be extracted as the hyperbolic tangent transfer function for each hidden neuron $m = 1 \ldots H$ is approximated by a 3-piece linear function. The procedure for rule extraction is as follows.

- The input space is divided into 3^H subregions by using the pair of intercepts (ξ_0 and $-\xi_0$) from the function $L_m(\xi)$.
- For each non-empty subregion, a rule is generated as follows:
  1) Define a linear equation that approximates the network's output for input sample i in this subregion as the *consequence* of the rule:

$$\begin{aligned} \hat{e}_i &= \sum_{m=1}^{H} v_m L_m(\xi_{mi}) + \tau \\ \xi_{mi} &= \mathbf{x}_i'\mathbf{w}_m \end{aligned}$$

  2) Generate the rule *condition*: ($\mathcal{C}_1$ and $\mathcal{C}_2$ and $\cdots$ $\mathcal{C}_H$), where $\mathcal{C}_m$ is either $\xi_{mi} < -\xi_{m0}$, $-\xi_{m0} \le \xi_{mi} \le \xi_{m0}$, or $\xi_{mi} > \xi_{m0}$.

Thus, a rule set which is equivalent to the pruned NN with 3-piece linear approximation, can be represented in the following form:

$\xi_{m1}, \xi_{m2} \ldots \xi_{mH} \in \mathbb{R}$, the intercepts of the 3-piece linear approximation of the H hidden neurons which determine the regions.

Rule 1: IF Region 1 THEN $\hat{e}_i = E_1$
Rule 2: IF Region 2 THEN $\hat{e}_i = E_2$
...
Rule P: IF Region P THEN $\hat{e}_i = E_P$

with P the number of non empty regions and

$$\begin{aligned} E_1 &= \textstyle\sum_{m=1}^{H} v_m L_m(\mathbf{x}_i'\mathbf{w}_m) + \tau \\ E_2 &= \textstyle\sum_{m=1}^{H} v_m L_m(\mathbf{x}_i'\mathbf{w}_m) + \tau \\ &\ldots \\ E_P &= \textstyle\sum_{m=1}^{H} v_m L_m(\mathbf{x}_i'\mathbf{w}_m) + \tau \end{aligned}$$

with $\mathbf{x}_i$ the input samples that lies within the associated region.

IV. Empirical setup

In this section, background information concerning the used data set is given, and the data preprocessing steps are detailed. The setup of the study and the different evaluation criteria to assess the techniques are also discussed.

A. Data set

The International Software Benchmarking Standards Group (ISBSG) is a not-for-profit organization that maintains a large data set of software project data [3]. This data set is often used by researchers (e.g. see [33, 34, 35, 36]). In this study, ISBSG R11 (May 2009) is used, containing 5052 projects collected from companies worldwide; Fig. 3 gives a breakdown of the origin of the projects. The data relates to projects collected from 1992 until 2009.

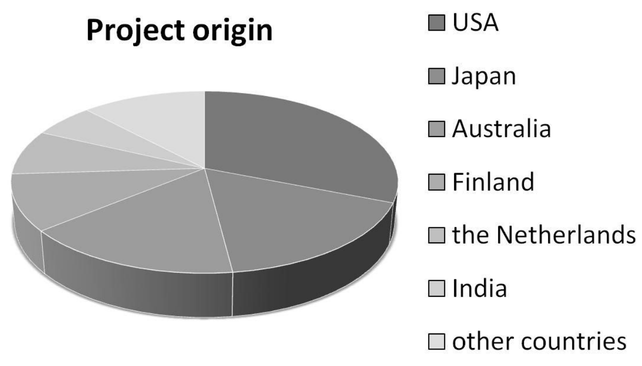


Fig. 3. Breakdown of the origin of the projects in the ISBSG data set

For this study, 721 projects were selected according to the following criteria.

- Only projects with an overall data quality of A or B, and a function point quality of A were selected.
- The function points needed to be counted by the IFPUG 4 standard [4].
- Projects with missing values for team size, function point count or effort were discarded.

[3]www.isbsg.org
[4]www.ifpug.org/publications/manual.htm

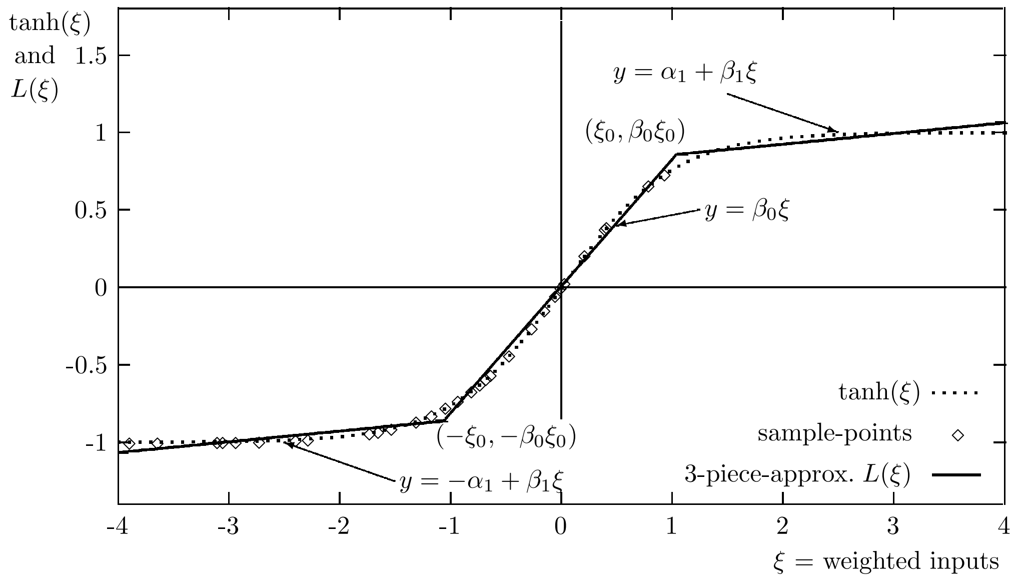


Fig. 2. The 3-piece linear approximation of the hidden unit activation function tanh(ξ) given 30 training samples ($\diamond$).

TABLE II
OVERVIEW OF THE SELECTED ATTRIBUTES

Variable name	Variable description	Type
Effort	total effort of the project in man hours	cont.
TeamSize	number of developers working on the project	cont.
FunctionCount	size of the project, expressed in function points	cont.
ApplType	application type (e.g. financial application)	cat.
Arch	architecture (e.g. client-server)	cat.
Database	primary database used in the project	cat.
DevPlat	development platform (e.g. mainframe)	cat.
DevType	development type (e.g. new development)	cat.
LanType	language generation	cat.
Lan	language used for the project	cat.
OrgType	organization type	cat.
Meth	methodology used during development	cat.

- The total effort recorded must be related only to development effort.

Table II provides an overview of the parameters included in the data set. The ISBSG data set contains a large number of missing values for a number of important attributes. Therefore, we decided only to include attributes with less then 40 % missing values. Additionally, no attributes contributing to an aggregated variable are considered. In case of categorical variables with more than 8 possible values, some of the levels were merged based on semantic similarity to obtain a more concise data set. Levels with less than 15 observations, were put in a category named ‘Other’, in line with a study done by Jeffrey *et al.* [36]. The categorical variables were transformed into binary variables by applying dummy encoding as the categorical attributes are nominal by nature [37]. A missing value flag was created to account for missing data. No outlier removal or other preprocessing steps were applied.

B. Study setup

To obtain a fair estimation of performance of the various techniques, the data is randomly split into a training and a test set containing 649 projects and 72 projects respectively. The models are induced on the training data set, and afterwards evaluated using the previously unseen data from the test set.

The results obtained on this independent test set are tested using a one way ANOVA (ANalysis Of VAriance). This statistical technique assesses the question whether the means across multiple groups are different by taking the variances of the groups into account, hence its name. The results are tested at statistical significance level α = 0.01. Following the ANOVA test, Tukey’s honest significance test is utilized to perform a pairwise comparison of the techniques in order to investigate whether the differences between two specific techniques are statistically significant. A statistical significance level of α = 0.05 is used for this second test.

C. Evaluation metrics

A key question of any estimation method is whether the predictions are accurate; the difference between the actual effort, e_i, and the predicted effort, $\hat{e}_i$, should be as small as possible. Two of the most commonly used criteria in the context of software effort prediction are MMRE and Pred$_{25}$. Both are derived from the Magnitude of Relative Error (MRE) [38]. The MRE is calculated for each observation and is defined as:

$$MRE_i = \frac{|e_i - \hat{e}_i|}{e_i} \tag{8}$$

The MMRE (Mean MRE) is then defined as:

$$MMRE = \frac{100}{N}\sum_{i=1}^{N}\frac{|e_i - \hat{e}_i|}{e_i} \tag{9}$$

A complementary accuracy measure is Pred$_k$ [38], the fraction of observations for which the predicted effort, $\hat{e}_i$, falls within

k% of the actual effort, e_i:

$$Pred_k = \frac{100}{N} \sum_{i=1}^{N} \begin{cases} 1 & \text{if } MRE_i \leq \frac{k}{100} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Typically, the Pred$_{25}$ measure is considered, looking at the percentage of predictions that fall within 25% of the actuals.

While both measures are based on the MRE, they have a slightly different focus; Pred$_k$ is favoring models which are generally accurate but occasionally widely inaccurate. MMRE on the other hand can be highly affected by outliers [39]. To address this shortcoming in the MMRE measure, the MdMRE is also considered. The MdMRE is the median of all MREs, and thus can be considered more robust against outliers.

$$MdMRE = 100 \times \text{median}(MRE) \tag{11}$$

Additionally, models are compared using a rank correlation measure, which measures the monotonic relationship between e_i and $\hat{e}_i$. More specifically, the Spearman's rank correlation r_s coefficient is used, since this non-parametric correlation coefficient does not assume a normal distribution of the underlying data [40]. The Spearman's rank correlation takes on a value between -1 and 1 with 1 (-1) indicating a perfect positive (negative) monotonic relationship between the actual values and the predicted values. The Spearman's rank correlation is defined as:

$$r_s = 1 - \frac{6 \sum_{i=1}^{N} d_i^2}{N(N^2-1)} \tag{12}$$

whereby d_i represents the difference between the ordinal ranks assigned to each of the variable values. In case of equal ranks, the average rank is assigned.

V. Results

In a first part of the experiment, the four techniques (OLS, CART, RBFN, and Rule set) are compared in terms of MMRE, MdMRE, Pred$_{25}$, and r_s. The results of the experiments are displayed in Table III. It can be seen that CART performs best, while the rule set extracted from the trained NN performs second best. However, analysis of the CART tree shows that the resulting tree consists of 47 splitting nodes. Hence, the induced rule set contains 48 rules with in some cases up to 10 rule antecedents. Therefore, the CART tree can be considered too elaborate to be easily comprehensible by end users. The extracted rule set is considerably smaller, only containing 5 rules with at most 2 rule antecedents. The rule set obtained from the pruned NN is given in Table IV.

TABLE III
Results of the different techniques on the ISBSG R11 data set

Technique	Pred$_{25}$	MMRE	MdMRE	r_s
OLS	22.22	135.03	57.06	0.742
CART	**37.50**	**61.16**	**39.87**	**0.871**
RBFN	16.67	113.07	64.35	0.343
CART Pruned	16.67	186.45	77.05	0.452
Rule set	30.56	95.39	57.25	0.813

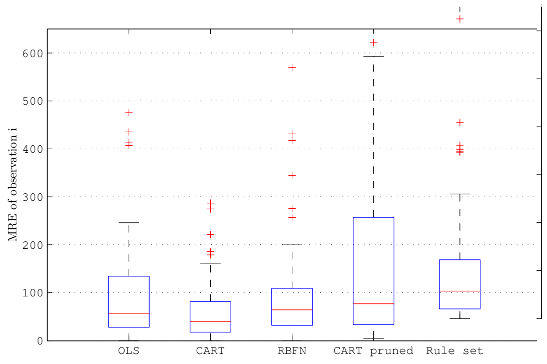


Fig. 4. Box plot of MRE for the different techniques

In a second part of the experiment, the CART tree is further pruned in order to obtain the same complexity of rule set (the same number of rules and rule antecedents) as the rule set extracted from the pruned NN. The results are indicated in Table III as 'CART pruned', and the CART tree as well as the induced rule set are shown in Fig. 1. The results indicate that the rule set performs better than the pruned CART tree in terms of MMRE, MdMRE, Pred$_{25}$, and r_s. Fig. 4 shows box plots of the MRE for the five techniques. In each box plot, the central line indicates the median MRE (MdMRE), while the edges of the box represent for each technique the 25th and the 75th percentiles. The whiskers extend to the most extreme MRE values that are not considered to be outliers. Outliers finally are represented by crosses.

To further assess the statistical significance of the results, a one way ANOVA analysis is performed to test whether the MMRE is significantly different across the five techniques (OLS, CART, CART pruned, RBFN, and Rule set). The null hypothesis is the following:

H_0 : $\text{MMRE}_{\text{CART}} = \text{MMRE}_{\text{OLS}} = \text{MMRE}_{\text{CARTpruned}} = \text{MMRE}_{\text{RBFN}} = \text{MMRE}_{\text{Ruleset}}$

The results of the ANOVA analysis are given in Table V. The test is significant at 1 % ($p < 0.001$) indicating that the MMRE across the five techniques is statistically different. Following the ANOVA, a Tukey's honest significance test is performed. The null hypothesis is given below:

$H_0 : \text{MMRE}_\text{k} = \text{MMRE}_\text{l} \text{ with k} \neq \text{l}$

The results of this test are displayed in Table VI. The significant pairwise differences are given in boldface font (critical value = 0.775).

It can be concluded from these tests that the extracted regression rules are significantly better performing than the pruned CART tree. The CART tree without additional pruning is the best performing technique at the expense of a larger and thus less comprehensible rule set, although this result is only statistically significant when compared to the pruned CART tree. When CART is compared with OLS, the result is very close to the critical value (0.7388 as compared to the critical value = 0.775).

TABLE IV
RULE SET EXTRACTED FROM NEURAL NETWORK

Equivalent rule set

Rule 1 IF Region 1 AND DevType = Enhan THEN
$496.98 + 1.74 \times$ FunctionCount $+ 428.32 \times$ TeamSize

Rule 2 IF Region 1 AND DevType $\neq$ Enhan THEN
$-776.42 + 1.74 \times$ FunctionCount $+ 428.32 \times$ TeamSize

Rule 3 IF Region 2 THEN
$-15705.45 + 10.59 \times$ FunctionCount $+ 778.27 \times$ TeamSize

Rule 4 IF Region 3 THEN
$5859.29 + 9.02 \times$ FunctionCount $+ 365.39 \times$ TeamSize

Rule 5 IF Region 4 THEN
$55637.90 + 0.16 \times$ FunctionCount $+ 15.50 \times$ TeamSize

Intercepts of the hidden neurons, ξ_{mi}

$\xi_{10} = 92.63$

$\xi_{20} = 140.18$

Region demarcation

Region 1 $\mathbf{x}_i'\mathbf{w}_1 \leq \xi_{10}$ AND $\mathbf{x}_i'\mathbf{w}_2 \leq -\xi_{20}$

Region 2 $\mathbf{x}_i'\mathbf{w}_1 \leq \xi_{10}$ AND $\mathbf{x}_i'\mathbf{w}_2 \leq \xi_{20}$

Region 3 $\mathbf{x}_i'\mathbf{w}_1 \geq \xi_{10}$ AND $\mathbf{x}_i'\mathbf{w}_2 \leq \xi_{20}$

Region 4 $\mathbf{x}_i'\mathbf{w}_1 \geq \xi_{10}$ AND $\mathbf{x}_i'\mathbf{w}_2 \geq -\xi_{20}$

TABLE V
ONE WAY ANOVA ANALYSIS RESULTS

Source of Variation	DF	Type III SS
Between Groups	4	62.7
Within Groups	355	849.3
F value	F value$_{critical}$	p value
5.4157	3.3724	0.0003

VI. CONCLUSIONS

Neural network based approaches are often considered to be less suitable in a business environment due to a lack of comprehensibility. Instead, techniques such as CART are adopted to induce a set of understandable rules.

This experimental study assessed the feasibility of regression rule extraction from neural networks in the context of software effort prediction. The algorithm generates a small number of linear equations from a neural network trained for regression by approximating each hidden unit activation function by a 3-piece linear function. The best performing technique was found to be CART, although it did not statistically outperform the extracted rule set and other techniques. However, due to the size of the CART tree, the obtained decision model lacks comprehensibility. The extracted rule set was found to provide a more concise rule set to the end user while still attaining high performance. Further pruning the CART tree yields a statistical significantly less accurate model as compared to the extracted rule set. Therefore, we consider the idea of regression rule extraction highly relevant to the practical implementation of software effort prediction systems in a business environment where comprehensibility is important.

Further research on other software effort data sets is needed to assess this technique more in depth. Also other types of rule extraction algorithms are currently investigated by the authors.

ACKNOWLEDGMENTS

This research was supported by the Odysseus program (Flemish Government, FWO) under grant G.0915.09.

TABLE VI
TUKEY'S HONEST SIGNIFICANCE TEST RESULTS

	OLS	**CART**	**RBFN**	**CART Pruned**	**Rule set**
OLS	▨	0.7388	0.2196	0.5142	0.3964
CART	0.7388	▨	0.5192	**1.2530**	0.3423
RBFN	0.2196	0.5192	▨	0.7338	0.1768
CART Pruned	0.5142	**1.2530**	0.7338	▨	**0.9106**
Rule set	0.3964	0.3423	0.1768	**0.9106**	▨

REFERENCES

[1] M. Jørgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33–53, 2007.

[2] E. A. Nelson, *Management Handbook for the Estimation of Computer Programming Costs*. System Developer Corp., 1966.

[3] B. Boehm, *Software Engineering Economics*. New Jersey: Prentice Hall, 1981.

[4] L. H. Putnam, "A general empirical solution to the macro software sizing and estimation problem," *IEEE Transactions on Software Engineering*, vol. 4, no. 4, pp. 345–381, 1978.

[5] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," *IEEE Transactions on Software Engineering*, vol. 9, no. 6, pp. 639–648, 1983.

[6] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Boston, MA: Addison Wesley, 2005.

[7] G. Kateman and J. R. M. Smits, "Colored information from a black box ? Validation and evaluation of neural networks," *Analytica Chimica Acta*, vol. 277, no. 2, pp. 179–188, 1993.

[8] R. Setiono, B. Baesens, and C. Mues, "Recursive neural network rule extraction for data with mixed attributes," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 299–307, 2008.

[9] H. Wang, T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Mining Data from Multiple Software Development Projects," in *ICDM Workshops*, 2009, pp. 551–557.

[10] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 126–137, 1995.

[11] G. Finnie, G. Wittig, and J.-M. Desharnais, "A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models," *Journal of Systems and Software*, vol. 39, pp. 281–289, 1997.

[12] G. Wittig and G. Finnie, "Estimating software development effort with connectionist models," *Information and Software Technology*, vol. 39, no. 7, pp. 469–476, 1997.

[13] A. Heiat, "Comparison of artificial neural networks and regression models for estimating software development effort," *Information and Software Technology*, vol. 44,

no. 15, pp. 911–922, 2002.

[14] R. Shukla and A. Misra, "Estimating software maintenance effort- a neural network approach," in *Proceedings of the 1st conference on India software engineering*, Hyderabad, India, February 2008, pp. 107–112.

[15] A. Idri, T. M. Khoshgoftaar, and A. Abran, "Can Neural Networks be easily Interpreted in Software Cost Estimation ?" in *Proceedings of the IEEE International Conference on Fuzzy Systems*, Honolulu, Hawaii, 2002, pp. 1162–1167.

[16] C. F. Kemerer, "An empirical validation of software cost estimation models," *Communications of the ACM*, vol. 30, no. 5, pp. 416–429, 1987.

[17] B. Rao and N. Sarda, "Effort drivers in maintenance outsourcing: an experiment using Taguchi's methodology," in *Proceedings of the 7th IEEE European Conference on Software Maintenance and Reengineering*, 2003, pp. 1–10.

[18] J. M. Desharnais, "Analyse statistique de la productivities des projets de developpement en informatique apartir de la techniques des points de fonction," Ph.D. dissertation, University du Quebec, Quebec, Canada, 1988.

[19] C. M. Bishop, *Neural networks for pattern recognition*. Oxford University Press, 1995.

[20] D. Treigueiros and R. Berry, "The application of neural network based methods to the extraction of knowledge from accounting reports," in *Proceedings of 24th Annual Hawaii International Conference on Systems Sciences*, vol. 4, Hawaii, 1991, pp. 137–146.

[21] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[22] O. Vandecruys, D. Martens, B. Beasens, C. Mues, M. De Backer, and R. Haesen, "Mining software repositories for comprehensible software fault prediction models," *The Journal of Systems and Software*, vol. 81, no. 5, p. 823, 2008.

[23] M. Pazzani, S. Mani, and W. Shankle, "Acceptance by medical experts of rules generated by machine learning," *Methods of Information in Medicine*, vol. 40, no. 5, pp. 380–385, 2001.

[24] D. Martens, B. Baesens, T. Van Gestel, and J. Vanthienen, "Comprehensible credit scoring models using rule extraction from support vector machines," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1466–1476, 2007.

[25] B. Baesens and C. Mues, "Recursive neural network rule extraction for data with mixed attributes," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 299–307, 2008.

[26] L. Torgo and J. Gama, "Search-based class discretization," in *Proceedings of the 9th European Conference on Machine Learning, Lecture Notes in AI*, vol. 1224. Prague, Czech Republic: Springer, 1997, pp. 266–273.

[27] L. Breiman, J. H. Friedman, R. A. Olsen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth & Books/Cole Advanced Books & Software, 1984.

[28] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computing*, vol. 1, pp. 281–294, 1989.

[29] D. Specht, "A general regression neural network," *IEEE Transactions on Neural Networks*, vol. 2, no. 6, pp. 568–576, 1991.

[30] R. Setiono and W. K. Leow, "Pruned neural networks for regression," in *Proceedings of the 6th Pacific Rim Conference on Artificial Intelligence, Lecture notes in AI*, vol. 1886, San Mateo, CA, 2000, pp. 500–509.

[31] R. Setiono, "Generating linear regression rules from neural networks using local least squares approximation," in *Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks*, vol. 1, Granada, Spain, 2001, pp. 277–284.

[32] R. Setiono, W. K. Leow, and J. M. Zurada, "Extraction of rules from artificial neural networks for nonlinear regression," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 564–577, 2002.

[33] S.-J. Huang and N.-H. Chiu, "Optimization of analogy weights by genetic algorithm for software effort estimation," *Information and Software Technology*, vol. 48, no. 11, pp. 1034–1045, 2006.

[34] J. Li, G. Ruhe, A. Ak-Emran, and M. Richter, "A flexible method for software effort estimation by analogy," *Empirical Software Engineering*, vol. 12, pp. 65–107, 2007.

[35] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris, "Software productivity and effort prediction with ordinal regression," *Information and Software Technology*, vol. 47, pp. 17–29, 2005.

[36] R. Jeffery, M. Ruhe, and I. Wieczorek, "Using public domain metrics to estimate software development effort," in *Proceedings of the 7th International Software Metrics Symposium*, London, UK, April 2001, pp. 16–27.

[37] B. Baesens, R. Setiono, C. Mues, and J. Vanthienen, "Using neural network rule extraction and decision tables for credit-risk evaluation," *Management Science*, vol. 49, no. 3, pp. 312–329, 2003.

[38] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software engineering metrics and models*. Redwood City, CA, USA: The Benjamin/Cummings Publishing Company, Inc, 1986.

[39] D. Port and M. Korte, "Comparative Studies of the Model Evaluation Criterions MMRE and PRED in Software Cost Estimation Research," in *Proceedings of the 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, Germany, October 2008, pp. 51–60.

[40] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning, Data Mining, Inference, and Prediction*. Springer, 2001.