DOI: 10.21917/ijsc.2013.0077

SENSITIVITY-BASED LINEAR LEARNING METHOD AND EXTREME LEARNING MACHINES COMPARED FOR SOFTWARE MAINTAINABILITY PREDICTION OF OBJECT-ORIENTED SOFTWARE SYSTEMS

Sunday Olusanya Olatunji
Department of Computer Science, Adekunle Ajasin University, Akungba Akoko, Ondo State, Nigeria
E-mail: oluolatunji.aadam@gmail.com

Abstract

This paper presented two maintainability prediction models that are developed and compared for object-oriented software systems based on the recently introduced learning algorithm called Sensitivity Based Linear Learning Method (SBLLM) and extreme learning machines (ELM). As the number of object-oriented software systems increases, it becomes more important for organizations to maintain those systems effectively. However, currently only a small number of maintainability prediction models are available for object oriented systems. The model was constructed using popular object-oriented metric datasets, collected from different object-oriented systems. Prediction accuracy of the models were evaluated and compared with each other and with other commonly used regression-based models and also with Bayesian network based model which were earlier developed using the same datasets. Empirical results from simulation show that the proposed ELM and SBLLM based models produced promising results in term of prediction accuracy measures authorized in OO software maintainability literatures, better than most of the other earlier implemented models on the same datasets.

Keywords:

Sensitivity Based Linear Learning Method (SBLLM), Extreme Learning Machines, Object Oriented Software Systems, Software Metrics, Software Maintainability Prediction Models

1. INTRODUCTION

Software maintainability is the process of modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment. Maintaining and enhancing the reliability of software during maintenance requires that software engineers understand how various components of a design interact. People usually think of software maintenance as beginning when the product is delivered to the client. While this is formally true, in fact decisions that affect the maintenance of the product are made from the earliest stage of design.

Software maintenance is classified into four types: corrective, adaptive, perfective and preventive. Corrective maintenance refers to fixing a program. Adaptive maintenance refers to modifications that adapt to changes in the data environment, such as new product codes or new file organization or changes in the hardware of software environments. Perfective maintenance refers to enhancements: making the product better, faster, smaller, better documented, cleaner structured, with more functions or reports. The preventive maintenance is defined as the work that is done in order to try to prevent malfunctions or improve maintainability.

When a software system is not designed for maintenance, it exhibits a lack of stability under change. A modification in one part of the system has side effects that ripple throughout the system. Thus, the main challenges in software maintenance are to understand existing software and to make changes without introducing new bugs.

It is arguable that many object-oriented (OO) software systems are currently in use. It is also arguable that the growing popularity of OO programming languages, such as Java, as well as the increasing number of software development tools supporting the Unified Modelling Language (UML), encourages more OO systems to be developed at present and in the future. Hence it is important that those systems are maintained effectively and efficiently. A software maintainability prediction model enables organizations to predict maintainability of a software system and assists them with managing maintenance resources.

In addition, if an accurate maintainability prediction model is available for a software system, a defensive design can be adopted. This would minimize, or at least reduce future maintenance effort of the system. Maintainability of a software system can be measured in different ways. Maintainability could be measured as the number of changes made to the code during a maintenance period or be measured as effort to make those changes. The predictive model is called a maintenance effort prediction model if maintainability is measured as effort. Unfortunately, the number of software maintainability prediction models including maintenance effort prediction models, is currently very small in the literature.

In this research work, two maintainability predictions models are developed and compared for an object-oriented software system based on the recently introduced learning algorithm called Sensitivity Based Linear Learning Method (SBLLM) [1, 2] and extreme learning machines (ELM) [3-6]. They are variants of the classical neural network models, as will be discussed in the sections that follow shortly. Empirical results gotten demonstrated that both SBLLM and ELM models can produce good generalization performance in most cases and performed better than earlier used methods.

Despite the importance of software maintenance, little work has been done as regards developing predictive models for software maintainability, particularly object-oriented software system, which is evident in the fewer number of software maintainability prediction models, that are currently found in the literature.

In view of this, we have proposed and compared two maintainability prediction model for an object-oriented software system based on each of the two recently introduced learning algorithms known as Sensitivity Based Linear Learning Method (SBLLM) and extreme learning machines (ELM).

Implementation was carried out on representative datasets related to the target systems. Furthermore, we performed comparative analysis between our models and the models presented in Koten [7], which include Regression-Based and Bayesian Network Based models, in terms of their performance measures, as recommended in the literatures.

The rest of this paper is organized as follows. Section 2 contains review of related earlier works. Section 3 discusses Sample predictive modelling techniques, and also describes the two proposed models using ELM and SBLLM. Section 4 presents the OO software data sets and the metrics used in our study and their descriptions. Section 5 contains model evaluation that includes model validation approach and prediction accuracy measures used. Section 6 contains empirical results, comparison with other models and discussions. Section 7 concludes the paper.

2. RELATED WORK

Several object oriented software maintainability prediction models were developed of recent; and they are mostly characterized by low prediction accuracies Lucia et al. [8]. Regression techniques have been thoroughly utilized by Li and Henry [9], Fioravanti and Nesi [10] to predict maintainability of object oriented software systems. Some recent work have been done using artificial neural networks and some other artificial intelligence techniques such as Bayesian Belief Networks (BBN), Van and Gray [7] and Multivariate adaptive regression splines (MARS), Zhou and Leung [11].

Variants of artificial neural networks were also employed in predicting the maintainability effort of object oriented software systems. Feed forward neural network and General Regression neural network (GRNN) were used by [12] to predict the maintainability effort for object oriented software systems using object oriented metrics.

Bayesian Belief Networks (BBN) was first suggested as a novel approach for software quality prediction by Fenton et al. [13] and Fenton and Neil [14] and [15]. They build their conjecture based on Bayesian Belief Networks' ability in handling uncertainties, incorporating expert knowledge, and modeling the complex relationships among variables. However, a number of researchers have pointed out several limitations of Bayesian Belief Networks when they are applied as a model for object oriented software quality and maintainability prediction [7, 11, 14]. Later on, a special type of Bayesian Belief Networks called Naïve-Bayes classifier was used by Van and Gray [7] to implement a Bayesian Belief Networks based software maintainability prediction model. Although their results showed that their model give better results than regression-based techniques for some datasets, the model is still inferior to regression-based techniques for some other datasets.

3. SAMPLE MODELING TECHNIQUES

3.1 REGRESSION BASED MODELS

Regression models are used to predict one variable from one or more other variables. Regression models provide the scientist with a powerful tool, allowing predictions about past, present, or future events to be made with information about past or present events.

3.1.1 Multiple Linear Regression Model:

Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. The regression line for p explanatory variables x_1, x_2,..... x_p is defined to be $\mu_y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots\ldots \beta_p x_p$. This line describes how the mean response μ_y changes with the explanatory variables. The observed values for y vary about their means μ_y and are assumed to have the same standard deviation σ. Formally, the model for multiple linear regressions given observations is,

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots\ldots \beta_p x_{ip} + \varepsilon_i \text{ for } i = 1, 2, \ldots.\ n$$

where, ε_i is notation for model deviation.

One approach to simplifying multiple regression equations is the stepwise procedures. These include forward selection, backwards elimination, and stepwise regression. They add or remove variables one-at-a-time until some stopping rule is satisfied.

Forward selection: Forward selection starts with an empty model. The variable that has the smallest *P* value when it is the only predictor in the regression equation is placed in the model. Each subsequent step adds the variable that has the smallest *P* value in the presence of the predictors already in the equation. Variables are added one-at-a-time as long as their *P* values are small enough, typically less than 0.05 or 0.10.

Backward elimination: It starts with all of the predictors in the model. The variable that is least significant that is, the one with the largest *P* value is removed and the model is refitted. Each subsequent step removes the least significant variable in the model until all remaining variables have individual *P* values smaller than some value, such as 0.05 or 0.10.

Stepwise regression: This approach is similar to forward selection except that variables are removed from the model if they become non significant as other predictors are added.

Backwards elimination: has an advantage over forward selection and stepwise regression because it is possible for a set of variables to have considerable predictive capability rather than any individual subset. Forward selection and stepwise regression will fail to identify them because sometimes variables don't predict well individually and backward elimination starts with everything in the model, so their joint predictive capability will be seen.

3.2 BAYESIAN NETWORKS

A Bayesian network consists of nodes interconnected by the directed links forming directed acyclic graph. In this graph, nodes represent random variables (RVs) and links correspond to direct probabilistic influences. The RVs correspond to important attributes of the modeled system which exemplifying the system's behavior. Directed connection between the two nodes indicates a casual effect between RVs which associated with these nodes.

The structure of directed acyclic graph states that each node is independent of all its non descendants conditioned on its parent nodes. In other words, the Bayesian Network represents the conditional probability distribution $P(Y/X_1,\ldots, X_n)$ which is used to quantify the strength of variables X_i on the variable Y, Nodes X_i are called the parents of Y and Y is called a child of each X_i. This should be noted that outcomes of the events for the variables X_i have an influence on the outcome of the event Y.

3.3 SENSITIVITY BASED LINEAR LEARNING METHOD (SBLLM)

Castillo et al. [2], proposed a new learning scheme in order to both speed up and avoid local minima convergence of the existing backpropagation learning technique. This new learning strategy is called the Sensitivity Based Linear Learning (SBLLM) scheme. It is a learning technique for two-layer feedforward neural networks based on sensitivity analysis, which uses a linear training algorithm for each of the two layers. First, random values are assigned to the outputs of the first layer; later, these initial values are updated based on sensitivity formulas, which use the weights in each of the layers; the process is repeated until convergence. Since these weights are learnt solving a linear system of equations, there is an important saving in computational time. The method also gives the local sensitivities of the least square errors with respect to input and output data, with no extra computational cost, because the necessary information becomes available without extra calculations. This new scheme can also be used to provide an initial set of weights, which significantly improves the behavior of other learning algorithms. The full theoretical basis for SBLLM and its performance has been demonstrated in Castillo et al. [2], which contained its application to several learning problems examples in which it is compared with several learning algorithms and well known data sets. The results have shown a learning speed generally faster than other existing methods. In addition, it can be used as an initialization tool for other well known methods with significant improvements.

Sensitivity analysis is a very useful technique for deriving how and how much the solution to a given problem depends on data, see Castillo et al. [16]-[19] and the references therein for more details. However, in Castillo et al. [2] it was shown that sensitivity formulas can also be used as a novel supervised learning algorithm for two-layer feedforward neural networks that presents a high convergence speed. Generally, SBLLM process is based on the use of the sensitivities of each layer's parameters with respect to its inputs and outputs and also on the use of independent systems of linear equations for each layer to obtain the optimal values of its parameters. In addition, it gives the sensitivities of the sum of squared errors with respect to the input and output data.

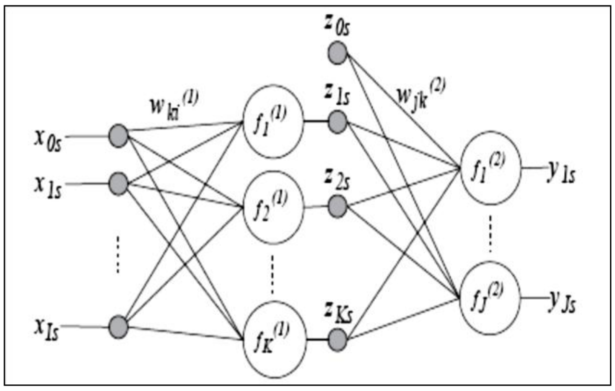

Fig.1. Two-layer feedforward neural network [2]

3.3.1 How Sensitivity Based Linear Learning Method (SBLLM) Works:

Consider the two-layer feedforward neural network in Fig.1, where, i is the number of inputs, j the number of outputs, k the number of hidden units, $x_{0s} = 1$, $z_{0s} = 1$, s the number of data samples and the superscripts (1) and (2) are used to refer to the first and second layer, respectively. This network can be considered to be composed of two one-layer neural networks as it is shown in Fig.2. For this one layer neural network, to learn the weights w_{ji}, one can minimize the sum of squared errors before the nonlinear activation functions Castillo et al. [20], that is,

$$\begin{aligned} Q &= \sum_{s=1}^{S}\sum_{j=1}^{J}\varepsilon_{js}^2 \\ &= \sum_{s=1}^{S}\sum_{j=1}^{J}\left(\sum_{i=0}^{I} w_{ji}x_{is} - f_j^{-1}\left(y_{js}\right)\right)^2 \end{aligned} \tag{1}$$

this leads to the system of equations:

$$\sum_{i=0}^{I} A_{pi}w_{ji} = b_{pj};\ p = 0,1,\ldots, I;\ \forall j$$

where,

$$A_{pi} = \sum_{s=1}^{S} x_{is}x_{ps};\ p = 0,1,\ldots I;\quad \forall i,$$

$$b_{pj} = \sum_{s=1}^{S} f_j^{-1}\left(y_{js}\right)x_{ps};\ p = 0,1,\ldots I;\quad \forall j$$

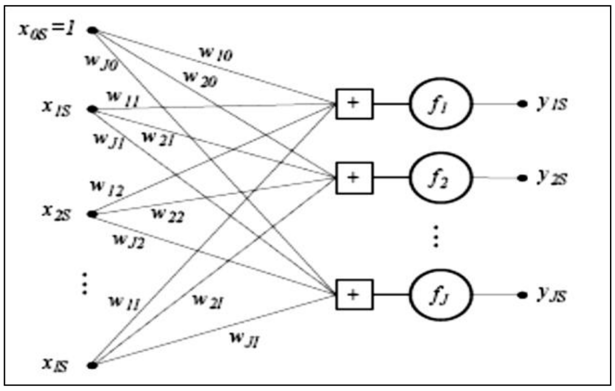

Fig.2. One-layer feedforward neural network [2]

Therefore, assuming that the intermediate layer outputs z are known, using Eq.(1), a new cost function for the two-layer feedforward neural network in Fig.1 is defined as,

$$Q(z)=Q^{(1)}(z)+Q^{(2)}(z)$$
$$=\sum_{s=1}^{S}\left[\sum_{k=1}^{K}\left(\sum_{i=0}^{I}w_{ki}^{(1)}x_{is}-f_k^{(1)^{-1}}(z_{ks})\right)^2+\sum_{j=1}^{J}\left(\sum_{k=0}^{K}w_{jk}^{(2)}z_{ks}-f_k^{(2)^{-1}}(y_{js})\right)^2\right]$$

Thus, using the outputs z_{ks} we can learn, for each layer independently, the weights $w_{ki}^{(1)}$ and $w_{jk}^{(2)}$ by solving the corresponding linear system of Eq.(2). For the neural network shown in Fig.1, according to Castillo et al. [2, 17, 21], the sensitivities of the new cost function, Q, with respect to the output and input data can be obtained as,

$$\frac{\partial Q}{\partial y_{pq}}=-\frac{2\left(\sum_{i=0}^{I}w_{pi}x_{iq}-f_p^{-1}(y_{pq})\right)}{f_p^{'}(y_{pq})};\ \forall p,q$$

$$\frac{\partial Q}{\partial x_{pq}}=2\sum_{j=1}^{J}\left(\sum_{i=0}^{I}w_{ji}x_{iq}-f_j^{-1}(y_{jq})\right)w_{jp};\ \forall p,q.$$

Thus, the sensitivities with respect to z_{ks} for the two-layer feedforward neural network in Fig.1 are calculated as,

$$\frac{\partial Q}{\partial z_{ks}}=\frac{\partial Q^{(1)}}{\partial z_{ks}}+\frac{\partial Q^{(2)}}{\partial z_{ks}}$$
$$=-\frac{2\left(\sum_{i=0}^{I}w_{ki}^{(1)}x_{is}-f_k^{(1)^{-1}}(z_{ks})\right)}{f_k^{'(1)}(z_{ks})}+2\sum_{j=1}^{J}\left(\sum_{r=0}^{K}w_{jr}^{(2)}z_{rs}-f_j^{(2)^{-1}}(y_{js})\right)w_{jk}^{(2)}$$

with k = 1, . . . K, as z_{0s} = 1, $\forall s$. After this, the values of the intermediate outputs z are modified using the Taylor series approximation,

$$Q(z+\Delta z)=Q(z)+\sum_{k=1}^{K}\sum_{s=1}^{S}\frac{\partial Q(z)}{\partial z_{ks}}\Delta z_{ks}\approx 0,$$

this leads to the following increments,

$$\Delta z=-\rho\frac{Q(z)}{\|\nabla Q\|^2}\nabla Q,$$

where, ρ is a relaxation factor or step size. The Sensitivity-Based Linear Learning scheme is summarized in the following algorithmic steps.

3.3.2 SBLLM Learning Process:

The training algorithm of the SBLLM technique can be summarized in the following algorithmic steps:

Input – The inputs to the system, which is the available or simulated data (training) set (input, x_{is}, and desired data, y_{js}), two threshold errors (ε and $\varepsilon^{'}$) to control both convergence and a step size ρ.

Output – The output results of the SBLLM system are the weights of the two layers and the sensitivities of the sum of squared errors with respect to input and output data.

Step 0: Initialization.

Assign to the outputs of the intermediate layer the output associated with some random weights $w^{(1)}(0)$ plus a small random error, that is,

$$z_{ks}=f_k^{(1)}\left(\sum_{i=0}^{I}w_{ki}^{(1)}(0)x_{is}\right)+\varepsilon_{ks};\varepsilon_{ks}\to U(-\eta,\eta);k=1,...K$$

where, η is a small number, and initialize the sum of squared errors $(Q)_{\text{previous}}$ and mean-squared errors $(\text{MSE})_{\text{previous}}$ to some large number, where MSE measures the error between the obtained and the desired output.

Step 1: Sub-problem solution.

Learn the weights of layers 1 and 2 and the associated sensitivities solving the corresponding systems of equations, that is,

$$\sum_{i=0}^{I}A_{pi}^{(1)}w_{ki}^{(1)}=b_{pk}^{(1)}$$
$$\sum_{k=0}^{K}A_{qk}^{(2)}w_{jk}^{(2)}=b_{qj}^{(2)}$$

where,

$$A_{pi}^{(1)}=\sum_{s=1}^{S}x_{is}x_{ps};\ b_{pk}^{(1)}=\sum_{s=1}^{S}f_k^{(1)^{-1}}(z_{ps})x_{ps};$$
$$p=0,1,....,I;\ k=1,2,....K$$

and

$$A_{qk}^{(2)}=\sum_{s=1}^{S}z_{ks}z_{qs};\ b_{qj}^{(2)}=\sum_{s=1}^{S}f_k^{(2)^{-1}}(y_{js})z_{qs};$$
$$q=0,1,....,K;\quad\forall j$$

Step 2: Evaluate the sum of squared errors.

Evaluate Q using,

$$Q(z)=Q^{(1)}(z)+Q^{(2)}(z)$$
$$=\sum_{s=1}^{S}\left[\sum_{k=1}^{K}\left(\sum_{i=0}^{I}w_{ki}^{(1)}x_{is}-f_k^{(1)^{-1}}(z_{ks})\right)^2+\sum_{j=1}^{J}\left(\sum_{k=0}^{K}w_{jk}^{(2)}z_{ks}-f_k^{(2)^{-1}}(y_{js})\right)^2\right]$$

and evaluate also the MSE.

Step 3: Convergence checking.

If $|Q - Q_{\text{previous}}| < \varepsilon$ or $|\text{MSE}_{\text{previous}} - \text{MSE}| < \varepsilon^{'}$ stop and return the weights and the sensitivities. Otherwise, continue with Step 4.

Step 4: Check improvement of Q.

If $Q > Q_{\text{previous}}$ reduce the value of ρ, that is, $\rho\Delta = \rho/2$ and return to the previous position, that is, restore the weights, $z = z_{\text{previous}}$, $Q = Q_{\text{previous}}$ and go to Step 5. Otherwise, store the values of Q and z, that is, $Q_{\text{previous}} =$

Q, MSE$_{\text{previous}}$ = MSE and $z_{\text{previous}} = z$ and obtain the sensitivities using,

$$\frac{\partial Q}{\partial z_{ks}} = -\frac{2\left(\sum_{i=0}^{I} w_{ki}^{(1)} x_{is} - f_k^{(1)^{-1}}(z_{ks})\right)}{f_k^{'(1)}(z_{ks})} + 2\sum_{j=1}^{J}\left(\sum_{r=0}^{K} w_{jr}^{(2)} z_{rs} - f_j^{(2)^{-1}}(y_{js})\right) w_{jk}^{(2)};\ k = 1,....K.$$

Step 5: Update intermediate outputs.

Using the Taylor series approximation in Eq.(3), update the intermediate outputs as,

$$z = z - \rho \frac{Q(z)}{\|\nabla Q\|^2} \nabla Q,$$

and go to Step 1.

3.4 EXTREME LEARNING MACHINES

An extreme learning machine (ELM) is a learning algorithm for one layer neural network with unique abilities. According to [6], "Unlike the traditional classic gradient-based learning algorithms, like backpropagation method, facing several issues like local minimum, improper learning rate and over-fitting, etc, the ELM is a simple tuning-free three-step algorithm that tends to reach the solutions straightforward without such trivial issues". Also ELM tends to reach the minimum training error as well as it considers magnitude of weights which is opposite to the classic gradient-based learning algorithms which only intend to reach minimum training error but do not consider the magnitude of weights. Also unlike the classical gradient-based learning algorithms which only work for differentiable activation functions ELM learning algorithm can be used to train SLFNs with non-differentiable activation functions [22].

3.4.1 The Learning Process for the Proposed Permeability model based on ELM Framework:

Let us first define the standard SLFN (single-hidden layer feed-forward neural networks). If we have N samples (x_i, t_i), where, $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in \mathbf{R}^n$ and $t_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in \mathbf{R}^m$, then the standard SLFN with $\tilde{N}$ hidden neurons and activation function $g(x)$ is defined as,

$$\sum_{i=1}^{\tilde{N}} \beta_i g(w_i . x_j + b_i) = o_j,\ j = 1,....N, \tag{2}$$

where, $w_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^T$ is the weight vector that connects the ith hidden neuron and the input neurons, $\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the weight vector that connects the ith neuron and the output neurons and b_i is the threshold of the ith hidden neuron. The "." in $w_i . x_j$ means the inner product of w_i and x_j.

SLFN aims to minimize the difference between o_j and t_j. This can be expressed mathematically as,

$$\sum_{i=1}^{\tilde{N}} \beta_i g(w_i . x_j + b_i) = t_j,\ j = 1,....N. \tag{3}$$

Or, more compactly, as:

$$\mathbf{H}\beta = \mathbf{T} \tag{4}$$

where,

$$\mathbf{H}(w_1,...w_{\tilde{N}}, b_1,...b_{\tilde{N}}, x_1,...x_N) = \begin{bmatrix} g(w_1.x_1 + b_1) & \cdots & g(w_{\tilde{N}}.x_{\tilde{N}} + b_{\tilde{N}}) \\ . & & . \\ . & \cdots & . \\ . & & . \\ g(w_1.x_N + b_1) & & g(w_{\tilde{N}}.x_N + b_{\tilde{N}}) \end{bmatrix}_{N\times\tilde{N}},$$

$$\beta = \begin{bmatrix} \beta_1^T \\ . \\ . \\ . \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N}\times m} \quad \text{and } T = \begin{bmatrix} T_1^T \\ . \\ . \\ . \\ T_{\tilde{N}}^T \end{bmatrix}_{\tilde{N}\times m}$$

As proposed by Huang and Babri [24], **H** is called the neural network output matrix.

With the above SLFN specification background, thus the training procedures for the proposed ELM based model for permeability estimation can be summarized in the following algorithmic steps. Refer [5, 6] for further details on the workings of ELM algorithm.

Mathematically, given a training set,

$$N = \{(x_i, t_i) \mid x_i \in \mathbf{R}^n, t_i \in \mathbf{R}^m, i = 1,....,N\},$$

activation function $g(x)$, and the number of hidden neuron = $\tilde{N}$, then, do the following:

Step 0: Initialization.

Assign random values to the input weight wj and the bias b_j, $j = 1,\ldots, \tilde{N}$

Step 1: Find the hidden layer output matrix **H**.

Step 2: Find the output weight β as follows:

$$\beta = \mathbf{H}^\dagger\mathbf{T}$$

where, β, **H** and **T** are defined in the same way they were defined in the SLFN specification above (Eq.(2), Eq.(3) and Eq.(4)).

4. DATA SETS USED

In this work, we made use of OO software datasets published by Li and Henry [9]. In this section, we describe the data set used for this study. We first introduce the metrics under study and then give some statistical analysis of the metrics that were investigated.

4.1 STUDIED METRICS

This study makes use of two OO software data sets published by Li and Henry [9]. These metric data were collected from a total of 110 classes in two OO software systems The first data set, UIMS, contains the metric data of 39 classes collected from a user interface management system (UIMS). The second data set, QUES, contains the metric data of 71 classes collected from a quality evaluation system (QUES). Both systems were implemented in Ada.

The datasets consist of five C&K metrics: DIT, NOC, RFC, LCOM and WMC, and four L&H metrics: MPC, DAC, NOM and SIZE2, as well as SIZE1, which is a traditional lines of code size metric. Maintainability was measured in CHANGE metric by counting the number of lines in the code, which were changed during a three-year maintenance period. Neither UIMS nor QUES datasets contain actual maintenance effort data. The description of each metric is given in the Table.1.

DIT, NOC, RFC, LCOM, WMC, MPC, DAC, NOM, SIZE2, and SIZE1 are the features that are combined and made used of to predict the attribute change. QUES data set has 71 sample cases, whereas UIMS has 39 sample cases.

Table.1. Description of Metrics

Metric	Description
WMC (Weighted methods per class)	The sum of McCabe's cyclomatic complexity of all local methods in a given class
DIT (Depth of inheritance tree)	The length of the longest path from a given class to the root in the inheritance hierarchy
RFC (Response for a class)	The number of methods that can potentially be executed in response to a message being received by an object of a given class
NOC (Number of children)	The number of classes that directly inherit from a given class. i.e. number of direct sub-classes that the class has
LCOM (Lack of cohesion in methods)	The number of pairs of local methods in a given class using no attribute in common. number of disjoint sets of local methods, i.e. number of sets of local methods that do not interact with each other, in the class
MPC (Message-passing coupling)	The number of send statements defined in a given class
DAC (Data abstraction coupling)	The number of abstract data types defined in a given class
NOM (Number of methods)	The number of methods implemented within a given class
SIZE1 (Lines of code)	The number of semicolons in a given class
SIZE2 (Number of properties)	The total number of attributes and the number of local methods in a given class
CHANGE (Number of lines changed in the class)	Insertion and deletion are independently counted as 1, change of the contents is counted as 2

4.2 CHARACTERISTICS OF THE DATASETS

Table.2. Descriptive statistics of the UIMS data set

Metric	Maximum	75%	Median	25%	Minimum	Mean	Standard deviation	Skewness
WMC	69	12	5	1	0	11.38	15.90	2.03
DIT	4	3	2	2	0	2.15	0.90	−0.54
RFC	101	30	17	11	2	23.21	20.19	2.00
NOC	8	1	0	0	0	0.95	2.01	2.24
LCOM	31	8	6	4	1	7.49	6.11	2.49
MPC	12	6	3	1	1	4.33	3.41	0.731
DAC	21	3	1	0	0	2.41	4.00	3.33
NOM	40	13	7	6	1	11.38	10.21	1.67
SIZE1	439	131	74	27	4	106.44	114.65	1.71
SIZE2	61	16	9	6	1	13.97	13.47	1.89
CHANGE	289	39	18	10	2	46.82	71.89	2.29

Table.3. Descriptive statistics of the QUES data set

Metric	Maximum	75%	Median	25%	Minimum	Mean	Standard deviation	Skewness
WMC	83	22	9	2	1	14.96	17.06	1.77
DIT	4	2	2	2	0	1.92	0.53	−0.10
RFC	156	62	40	34	17	54.44	32.62	1.62
NOC	0	0	NA	0	0	0	0.00	NA
LCOM	33	14	5	4	3	9.18	7.34	1.35
MPC	42	21	17	12	2	17.75	8.33	0.88
DAC	25	4	2	1	0	3.44	3.91	2.99
NOM	57	21	6	5	4	13.41	12.00	1.39
SIZE1	1009	333	211	172	115	275.58	171.60	2.11
SIZE2	82	25	10	7	4	18.03	15.21	1.71
CHANGE	217	85	52	35	6	64.23	43.13	1.36

5. MODEL EVALUATION

5.1 MODEL VALIDATION APPROACH

The available data set, for each data set, were divided into two parts. One part was used as a training set, for constructing maintainability prediction models. The other part was used for testing to determine the prediction ability of the developed models. Although there are many different ways to split a given dataset, we have chosen to use the stratify sampling approach in breaking the datasets due to its ability to break data randomly with a resultant balanced division based on the supplied percentage. The division, for instance could be 70% for training set and 30% for testing set. In this work, we selected 70% of the data for building the model (internal validation) and 30% of the data for testing/ validation (external validation or cross-validation criterion). We repeat both internal and external validation processes for 1000 times to have a fair partition through the entire process operations.

We also evaluate and compare our developed models with other OO software maintainability prediction models, sited earlier, quantitatively, using the following prediction accuracy measures recommended in the literatures: absolute residual (Ab.Res.), the magnitude of relative error (MRE) and the proportion of the predicted values that have MRE less than or equal to a specified value suggested in the literatures (pred measures). Details of all these measures of performance will be provided shortly.

5.2 PREDICTION ACCURACY MEASURES

In this paper, we compared the software maintainability prediction models using the following prediction accuracy measures: absolute residual (Abs Res), the magnitude of relative error (MRE) and Pred measures.

The Ab.Res. is the absolute value of residual evaluated by,

Ab.Res. = abs (actual value − predicted value)

In this paper, the sum of the absolute residuals (Sum Ab.Res.), the median of the absolute residuals (Med.Ab.Res.) and the standard deviation of the absolute residuals (SD Ab.Res.) are used. The Sum Ab.Res. measures the total residuals over the dataset. The Med.Ab.Res. measures the central tendency of the residual distribution. The Med.Ab.Res. is chosen to be a measure of the central tendency because the residual distribution is usually skewed in software datasets. The SD Ab.Res. measures the dispersion of the residual distribution.

MRE is a normalized measure of the discrepancy between actual values and predicted values given by,

MRE = abs (actual value − predicted value) / actual value

The Max.MRE measures the maximum relative discrepancy, which is equivalent to the maximum error relative to the actual effort in the prediction. The mean of MRE, the mean magnitude of relative error (MMRE):

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i$$

According to Fenton and Pfleeger [15], Pred is a measure of what proportion of the predicted values have MRE less than or equal to a specified value, given by,

Pred (q) = k / n

where, q is the specified value, k is the number of cases whose MRE is less than or equal to q and n is the total number of cases in the dataset.

According to Conte and Dunsmore [24] and MacDonell [25], in order for an effort prediction model to be considered accurate, MMRE < 0.25 and/or either pred(0.25) > 0.75 or pred(0.30) > 0.70 . These are the suggested criteria in literature as far as effort prediction is concerned.

6. EMPIRICAL RESULTS AND DISCUSSIONS

Below are tables showing the results of the models compared in comparison to the other earlier models used on the same datasets.

6.1 RESULTS FROM QUES DATASET

Table.4 shows the values of the prediction accuracy measures achieved by each of the maintainability prediction models for the QUES dataset. In order for an effort prediction model to be considered accurate, either MMRE < 0.25 and/or either pred(0.25) > 0.75 or pred(0.30) > 0.70, needed to be achieved, Conte and Dunsmore [24], and MacDonell [25]. Hence the closer a model's prediction accuracy measure value is to these baseline values, the better. Since Table.4 shows that the SBLLM and ELM models respectively have achieved MMRE values of 0.348 and 0.3502, the pred(0.25) value of 0.5 and 0.368 and the pred(0.30) value of 0.56 and 0.38. It is clear from these, that the proposed ELM and SBLLM models are the those that are very close to the required values for all the three essential prediction measures recommended in literature, hence they are the best among all the presented models. They outperform all the other models in terms of all the predictive measures used. Although, SBLLM performance values are better than those of ELM for this case but their results are closer together when compared with other methods.

In comparison with the UIMS dataset, the performance on QUES dataset is far better than that on UIMS. This indicates that the performance of the proposed ELM and SBLLM models may vary depending on the characteristics of the dataset and/or depending on what prediction accuracy measure is used.

6.2 RESULTS FROM UIMS DATASET

Table.5 shows the values of the prediction accuracy measures achieved by each of the maintainability prediction models for the UIMS dataset. From the results presented, the SBLLM and ELM models respectively achieved the MMRE value of 1.966 and 0.968, the pred (0.25) value of 0.179 and 0.392 and the pred (0.30) value of 0.25 and 0.45. These values compete favorably among all the models presented. Specifically in term of Sum.Abs.value, ELM and SBLLM are the best among all the models and we can see that there is strong evidence that the ELM and SBLLM model's values are significantly lower and thus, better than those of the other models. In term of pred(0.30), ELM and SBLLM are the second and third best models after Bayesian network. In addition, in terms of standard deviation, both ELM and SBLLM achieved the least values, which indicated their stability in predictions better than other methods.

Although, ELM performance values are better than those of SBLLM for this case but their results are closer together when compared with other methods.

Though the performance of SBLLM on UIMS dataset is low compared to its performance on QUES dataset, yet its performance compared to other models on same dataset is competitive and encouraging.

Table.4. Prediction accuracy for the QUES dataset

Model	Max. MRE	MMRE	Pred (0.25)	Pred (0.30)	Sum Ab.Res.	Med. Ab.Res.	SD Ab.Res.
Bayesian network	1.592	0.452	0.391	0.430	686.610	17.560	31.506
Regression Tree	2.104	0.493	0.352	0.383	615.543	19.809	25.400
Backward Elimination	1.418	0.403	0.396	0.461	507.984	17.396	19.696
Stepwise Selection	1.471	0.392	0.422	0.500	498.675	16.726	20.267
SBLLM	1.713	0.348	0.5	0.56	778.437	11.274	16.258
ELM	1.803	0.3502	0.368	0.380	56.122	28.06	22.405

Table.5. Prediction accuracy for the UIMS dataset

Model	Max. MRE	MMRE	Pred (0.25)	Pred (0.30)	Sum Ab.Res.	Med. Ab.Res	SD Ab.Res.
Bayesian Network	7.039	0.972	0.446	0.469	362.300	10.550	46.652
Regression Tree	9.056	1.538	0.200	0.208	532.191	10.988	63.472
Backward Elimination	11.890	2.586	0.215	0.223	538.702	20.867	53.298
Stepwise Selection	12.631	2.473	0.177	0.215	500.762	15.749	54.114
SBLLM	13.053	1.966	0.179	0.25	240.583	7.7452	18.095
ELM	4.918	0.968	0.392	0.450	39.625	18.768	16.066

7. CONCLUSION

Two models based on ELM and SBLLM for OO software maintainability prediction have been constructed and compared using the OO software metric data in Li and Henry datasets, Li and Henry [9]. The prediction accuracy of the model is evaluated and compared with the Bayesian network model, regression tree model and the multiple linear regression models using the prediction accuracy measures: the absolute residuals, MRE and pred measures. The results indicate that ELM and SBLLM model can reliably predict maintainability of the OO software systems to acceptable degrees compared to other. However, it is reported that prediction accuracy of software maintenance effort prediction models are often low and thus, it is very difficult to satisfy the established accuracy criteria, Lucia et al. [8]. Nevertheless, the proposed ELM and SBLLM models have achieved significantly better prediction accuracy, closer to the stated criteria in literatures, than the other models. For the QUES dataset, SBLMM performed better than ELM while for the UIMS datasets, ELM outperformed SBLLM model. Whenever the ELM and SBLLM models' prediction accuracy measure are not better than the best among the other models, they have been reasonably competitive against the best model among others.

Therefore, it is concluded here that the prediction accuracy of the proposed ELM and SBLLM models are better than, or at least, are competitive against the Bayesian network model and the regression based models. These outcomes have confirmed that ELM and SBLLM are indeed useful modeling techniques for software maintainability prediction, although further research studies are required to realize their full potentials in the general area of software engineering prediction tasks.

The results emanating from these studies indicated that the prediction accuracy of the proposed and compared ELM and SBLLM models may vary depending on the characteristics of dataset and/or the prediction accuracy measure used. This provides an interesting direction for future studies.

REFERENCES

[1] S. O. Olatunji, A. Selamat, A. A. A. Raheem and S. Omatu, "Modeling the correlations of crude oil properties based on sensitivity based linear learning method", *Engineering Applications of Artificial Intelligence*, Vol. 24, No. 4, pp. 686-696, 2011.

[2] Enrique Castillo, Bertha Guijarro-Berdiñas, Oscar Fontenla-Romero and Amparo Alonso-Betanzos, "A Very Fast Learning Method for Neural Networks Based on Sensitivity Analysis", *Journal of Machine Learning Research*, Vol. 7, pp. 1159-1182, 2006.

[3] S. O. Olatunji, Z. Rasheed, K.A. Sattar, A. M. Al-Mana, M. Alshayeb and E.A. El-Sebakhy, "Extreme Learning Machine as Maintainability Prediction model for Object-Oriented Software Systems", *Journal of Computing*, Vol. 2, No. 8, pp. 42-56, 2010.

[4] S. O. Olatunji, A. Selamat, A. A. A. Raheem, "Modeling Permeability Prediction Using Extreme Learning Machines", *IEEE Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation*, pp. 29-33, 2010.

[5] Guang-Bin Huang, Qin-Yu Zhu and Chee-Kheong Siew, "Extreme learning machine: Theory and applications", *Neurocomputing*, Vol. 70, No. 1-3, pp. 489-501, 2006.

[6] Guang-Bin Huang, Qin-Yu Zhu and Chee-Kheong Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks", *IEEE International Joint Conference on Neural Networks*, pp. 985-990, 2004.

[7] C. Van Koten and A. R. Gray, "An application of Bayesian network for predicting object-oriented software maintainability", *Information and Software Technology*, Vol. 48, No. 1, pp. 59-67, 2006.

[8] Andrea De Lucia, Eugenio Pompella and Silvio Stefanucci, "Assessing effort estimation models for corrective maintenance through empirical studies", *Information and Software Technology*, Vol. 47, No. 1, pp. 3-15, 2005.

[9] Wei Li and Sallie Henry, "Object-oriented metrics that predict maintainability", *Journal of Systems and Software*, Vol. 23, No. 2, pp. 111-122, 1993.

[10] F. Fioravanti and P. Nesi, "Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems", *IEEE Transactions on Software Engineering*, Vol. 27, No. 12, pp. 1062-1084, 2001.

[11] Yuming Zhou and Hareton Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines", *Journal of Systems and Software*, Vol. 80, No. 8, pp. 1349-1361, 2007.

[12] Tong-Seng Quah and Mie Mie Thet Thwin, "Application of neural networks for software quality prediction using object-oriented metrics", *Proceedings of International Conference on Software Maintenance*, pp. 116-125, 2003.

[13] N. Fenton, P. Krause and M. Neil, "Software measurement: uncertainty and causal modeling", *IEEE Software*, Vol. 19, No. 4, pp. 116-122, 2002.

[14] N. E. Fenton and M. Neil, "A critique of software defect prediction models", *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, pp. 675-689, 1999.

[15] Norman E. Fenton and Shari Lawrence Pfleeger, "*Software Metrics: A Rigorous and Practical Approach*", PWS Publishing Co., 1998.

[16] E. Castillo, A. J. Conejo, C. Castillo, R. Minguez and D. Ortigosa, "Perturbation approach to sensitivity analysis in Mathematical programming", *Journal of Optimization Theory and Applications*, Vol. 128, No. 1, pp, 49-74, 2006.

[17] Enrique Castillo, Ali S. Hadi, Antonio Conejo and Alfonso Fernandez-Canteli, "A general method for local sensitivity analysis with application to regression models and other optimization problems", *Technometrics*, Vol. 46, No. 4, pp. 430-444, 2004.

[18] Enrique Castillo, Angel Cobo, Jose Manuel Gutierrez, and Eva Pruneda, "Working with differential, functional and difference equations using functional networks", *Applied Mathematical Modelling*, Vol. 23, No. 2, pp. 89 -107, 1999.

[19] E. Castillo, J. M. Gutierrez and A. S. Hadi, “Sensitivity analysis in discrete Bayesian networks”, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 27, No. 4, pp. 412-423, 1997.

[20] E. Castillo, O. F. Romero, B. G. Berdinas and A. A. Betanzos, “A global optimum approach for one-layer neural networks”, *Neural Computation*, Vol. 14, No. 6, pp. 1429-1449, 2002.

[21] Enrique Castillo, Antonio J. Gonejo, Pablo Pederegal, Ricardo Garcia and Natalia Alguacil, “*Building and Solving Mathematical Programming Models in Engineering and Science*”, John Wiley & Sons Inc., 2002.

[22] Guang-Bin Huang, Qin-Yu Zhu, K. Z. Mao and Chee-Kheong Siew, “Can threshold networks be trained directly?”, *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 53, No. 3, pp. 187-191, 2006.

[23] Guang-Bin Huang and H. A. Babri, “Feed forward networks with arbitrary bounded nonlinear activation functions”, *IEEE Transactions on Neural Networks*, Vol. 9, No. 1, pp. 224-229, 1998.

[24] Samuel Daniel Conte, H. E. Dunsmore and V. Y. Shen, “*Software engineering metrics and models*”, Benjamin-Cummings Publishing Co., Inc., 1986.

[25] Andrew R. Gray and Stephen G. MacDonell, “A comparison of techniques for developing predictive models of software metrics”, *Information and Software Technology*, Vol. 39, No. 6, pp. 425-437, 1997.