Available online at www.sciencedirect.com

ScienceDirect

The Journal of Systems and Software 81 (2008) 1853–1867

The Journal of Systems and Software

www.elsevier.com/locate/jss

Software development cost estimation using wavelet neural networks

K. Vinay Kumar, V. Ravi *, Mahil Carr, N. Raj Kiran

Institute for Development and Research in Banking Technology, Castle Hills Road #1, Masab Tank, Hyderabad 500 057, AP, India

Received 30 May 2007; received in revised form 28 December 2007; accepted 28 December 2007
Available online 8 January 2008

Abstract

Software development has become an essential investment for many organizations. Software engineering practitioners have become more and more concerned about accurately predicting the cost and quality of software product under development. Accurate estimates are desired but no model has proved to be successful at effectively and consistently predicting software development cost. In this paper, we propose the use of wavelet neural network (WNN) to forecast the software development effort. We used two types of WNN with Morlet function and Gaussian function as transfer function and also proposed threshold acceptance training algorithm for wavelet neural network (TAWNN). The effectiveness of the WNN variants is compared with other techniques such as multilayer perceptron (MLP), radial basis function network (RBFN), multiple linear regression (MLR), dynamic evolving neuro-fuzzy inference system (DENFIS) and support vector machine (SVM) in terms of the error measure which is mean magnitude relative error (MMRE) obtained on Canadian financial (CF) dataset and IBM data processing services (IBMDPS) dataset. Based on the experiments conducted, it is observed that the WNN-Morlet for CF dataset and WNN-Gaussian for IBMDPS outperformed all the other techniques. Also, TAWNN outperformed all other techniques except WNN.
© 2008 Elsevier Inc. All rights reserved.

Keywords: Software development effort; Software cost estimation; Wavelet neural networks; Threshold accepting based wavelet neural network

1. Introduction

Software development has become an important activity for many modern organizations. In fact the quality, cost, and timeliness of developed software are often crucial determinants of an organization's success. There are significant financial and strategic implications for development projects in terms of activity scheduling and cost estimation. Software cost estimation is one of the most critical tasks in managing software projects. Development costs tend to increase with project complexity and hence accurate cost estimates are highly desired during the early stages development (Xu and Khoshgoftaar, 2004). A major problem of the software cost estimation is first obtaining an accurate size estimate of the software to be developed (Kitchenham et al., 2003). An important objective of the software engineering community has been to develop useful models that constructively explain the software development life cycle and accurately estimate the cost of software development.

In order to effectively develop software in an increasingly competitive and complex environment many firms use software metrics as part of their project management process. The field concerned with managing software development projects using empirical models is referred to as software metrics (Fenton and Pleeger, 1997). Software metrics are aspects of software development (either the software product itself, or the development process producing it) that can be measured. These metrics can be used as variables in models to predict some aspect(s) of the development process or product.

Estimating development effort and schedule, can include activities such as assessing and predicting system quality, measuring system performance, estimating user satisfaction and in fact any modeling task involving measurable

* Corresponding author. Tel.: +91 40 2353 4981x2042; fax: +91 40 2353 5157.
E-mail addresses: vinaykadiyam@gmail.com (K. Vinay Kumar), rav_padma@yahoo.com (V. Ravi), mahil.carr@gmail.com (M. Carr), nrajkiran@gmail.com (N. Raj Kiran).

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.12.793

attributes of interest within the software development sphere (Gray, 1999). However, the most researched area has been effort estimation as it carries the greatest promise of benefit for project management. Software effort estimates are crucial for estimating the amount of manpower needed for the project. This estimate determines staff allocation and schedule for a software project. Since human effort is the major cost driver in a software project, the effort estimate determines the budget of the project. Accurate effort estimates help software consultancies to make appropriate bids when quoting for tenders – a lower estimate than the actual will lead to a loss and an unreasonably high estimate will loose the bid. Such estimation models are developed using a set of measures that describe the software development process, product and resources such as developer experience, system size and complexity and the characteristics of the development environment, respectively. The output of the model is some measure of effort in terms of person hours (months or years).

There are many models and tools used in software cost estimation that provide invaluable information regarding efforts and expenditure to the management to bid for a project (Kitchenham et al., 2003). The most commonly used methods for predicting software development effort have been based on linear-least-squares regression such as COCOMO (Boehm, 1981; Fenton and Pleeger, 1997; Pressman, 1997). As such, COCOMO is extremely susceptible to local variations in data points (Miyazaki et al., 1994). Additionally, the models have failed to deal with implicit nonlinearities and interactions between the characteristics of the project and effort (Gray, 1999). Software cost estimation models are deemed to be acceptably accurate if they yield estimates with 25% mean relative error to the actual and this must be true at least 75% of the time. There is always scope for developing effort estimation models with better predictive accuracy (Kemerer, 1987).

In recent years, a number of alternative modeling techniques have been proposed. Existing datasets have their performance examined with some success including those in Gray and MacDonell (1997). Alternative models include artificial neural networks (Verkatachalm, 1993), analogy-based reasoning, regression trees and rule induction models. Gray and MacDonell (1997) applied fuzzy logic to software metric models for development effort estimation. They outlined the use of fuzzy logic for defining software metrics as linguistic variables and for modeling process. They made comparison of results obtained from an elementary fuzzy inference system with other techniques such as linear regression and neural network techniques and found that it outperformed. Gray (1999) presented several different predictive model-building techniques such as robust statistical procedures, various forms of neural network models, fuzzy logic, case-based reasoning and regression trees. He also described a simulation-based study on the performance of these empirical modeling techniques using size and effort software metric dataset and observed that M-estimation regression outperformed all other parametric and non-parametric techniques. Xu and Khoshgoftaar (2004) presented an innovative fuzzy identification software cost estimation modeling technique, which is an advanced fuzzy logic technique that integrates fuzzy clustering, space projection, fuzzy inference and defuzzification. Based upon their case study on the COCOMO’81 database it was observed that the fuzzy identification model provided significantly better cost estimations than the three COCOMO models, i.e. basic, intermediate and detailed. Many researchers have applied the neural networks approach to estimate software development effort (Hughes, 1996; Jorgerson, 1995; Samson et al., 1997; Schofield, 1998; Seluca, 1995; Heiat, 2002; Srinivasan and Fisher, 1995; Wittig and Finnie, 1997). Most of their investigations have focused more attention on the accuracy of the other cost estimation techniques such as COCOMO and Function Point Analysis. Idri et al. (2002) have also done research on estimating software cost using the neural networks approach and fuzzy if–then rules on the COCOMO’81 dataset. The use of software effort estimations by means of analogy have been evaluated and confirmed in several studies (Angelis and Stamelo, 2000; Jorgenson et al., 2003; Shepperd and Schofield, 1997). Wolverton (1974) explained the use of estimation by analogy and described the similarities and differences of existing software cost estimating techniques. Mukhopadhyay et al. (1992) utilized analogy for software effort estimation by retrieving the most similar cases. Their results showed that the analogy approach is more accurate and consistent than the function point and COOCMO models. Chiu and Huang (2007) used adjusted analogy-based software effort estimation based on similarity distances between pairs of projects. They demonstrated that applying effort adjustment to the analogy-based software effort estimations is a feasible approach to improve estimating using the three distance metrics. In addition, they demonstrated that the proposed adjusted analogy-based estimations are also compatible to the widely used estimation models of ANN, CART and OLS.

The rest of the paper is organized as follows: In Section 2, the techniques applied to software cost estimation are described briefly. Section 3 introduces wavelet neural networks. Section 4 describes the datasets and data preparation for our empirical study, while Section 5 presents our experimental methodology and compares the estimation performances with the most often used software effort estimation methods in the literature. Finally, Section 6 summarizes our work and outlines further directions.

2. Brief overview of the techniques employed

2.1. Multilayer perceptron (MLP)

Multilayer perceptrons (MLPs) are feed-forward neural networks trained with the standard back propagation algorithm. They are supervised networks so they require a desired response to be trained. They learn how to transform input data into a desired response, so they are widely used for pattern classification and prediction. A multilayer perceptron is made up of several layers of neurons. Each layer is fully connected to the next one. With one or two hidden layers, they can approximate virtually any input–output map. They have been shown to approximate the performance of accurate predictions in difficult problems. Most neural network applications involve MLPs.

2.2. Radial basis function network (RBFN)

RBFN (Moody and Darken, 1989), another member of the feed-forward neural networks, has both unsupervised and supervised phases. In the unsupervised phase, input data are clustered and cluster details are sent to hidden neurons, where radial basis functions of the inputs are computed by making use of the center and the standard deviation of the clusters. The radial basis functions are similar to kernel functions in kernel regression. The activation function or the kernel function can use a variety of functions, though Gaussian radial basis functions are the most commonly used. The learning between hidden layer and output layer is of supervised learning type where ordinary least squares technique is used. As a consequence, the weights of the connections between the kernel layer (also called hidden layer) and the output layer are determined. Thus, it comprises of hybrid of unsupervised a supervised learning. RBFN can solve many complex prediction problems with quite satisfying performance. Like MLP, RBFN is also a universal approximator. However, training an RBFN is very fast. Depending on dataset, MLP and RBFN are capable of outperforming each other.

2.3. Dynamic evolving neuro-fuzzy inference system (DENFIS)

DENFIS was introduced by Kasabov and Song (2002). DENFIS evolve through incremental, hybrid (supervised and unsupervised) learning, and accommodate new input data, including new features, new classes, etc., through local element tuning. New fuzzy rules are created and updated during the operation of the system. At each time moment, the output of DENFIS is calculated through a fuzzy inference system based on most activated fuzzy rules, which are dynamically chosen from a fuzzy rule set. A set of fuzzy rules can be inserted into DENFIS before or during its learning process. Fuzzy rules can also be extracted during or after the learning process.

2.4. Support vector machines (SVM)

The SVM is a powerful learning algorithm based on recent advances in statistical learning theory (Vapnik, 1998). SVMs are learning systems that use a hypothesis space of linear functions in a high-dimensional space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory (Cristianini and Shawe-Taylor, 2000). SVMs have recently become one of the popular tools for machine learning and data mining and can perform both classification and regression. SVM uses a linear model to implement nonlinear class boundaries by mapping input vectors nonlinearly into a high-dimensional feature space using kernels. The training examples that are closest to the maximum margin hyper plane are called support vectors. All other training examples are irrelevant for defining the binary class boundaries. The support vectors are then used to construct an optimal linear separating hyper plane (in case of pattern recognition) or a linear regression function (in case of regression) in this feature space. The support vectors are conventionally determined by solving a quadratic programming (QP) problem. SVMs have the following advantages: (i) they are able to generalize well even if trained with a small number of examples (ii) they do not assume prior knowledge of the probability distribution of the underlying dataset. SVM is simple enough to be analyzed mathematically. In fact, SVM may serve as a sound alternative combining the advantages of conventional statistical methods that are more theory-driven and easy to analyze and machine learning methods that are more data-driven, distribution-free and robust. Recently, SVM are used in financial applications such as credit rating, time series prediction and insurance claim fraud detection.

3. Overview of wavelet neural networks

The word *wavelet* is due to Grossmann and Morlet (1984). Wavelets are a class of functions used to localize a given function in both space and scaling (http://mathworld.wolfram.com/wavelet.html). They have advantages over traditional Fourier methods in analyzing physical situations where the signal contains discontinuities and sharp spikes. Wavelets were developed independently in the fields of mathematics, quantum physics, electrical engineering, and seismic geology. Interchanges between these fields during the last few years have led to many new wavelet applications such as image compression, turbulence, human vision, radar, and earthquake prediction.

A family of wavelets can be constructed from a function $\psi(x)$, sometimes known as a "mother wavelet," which is confined in a finite interval. "Daughter wavelets" $\psi^{a,b}(x)$ are then formed by translation (b) and dilation (a). Wavelets are especially useful for compressing image data, since a wavelet transform has properties that are in some ways superior to a conventional Fourier transform (http://mathworld.wolfram.com/wavelet).

An individual wavelet is defined by (http://mathworld.wolfram.com/wavelet.html)

$$\psi^{a,b}(x) = |\alpha|^{-1/2}\psi\left(\frac{x-b}{a}\right). \tag{1}$$

Industrial processes can impose a number of problems upon the structures adopted for neural network dynamic modeling due to varying sampling times, sparse and dense data in different operating regions and the inherent presence of both large and small dynamics. In the case of non-uniformly distributed training data, an efficient way of solving this problem is by learning at multiple resolutions. A higher resolution of input space is used if the data is dense and a lower resolution when it is sparse. Recently, due to the similarity between the discrete inverse wavelet transform and a one-hidden-layer neural network, the idea of combining both wavelets and neural networks has been proposed. This has resulted in the Wavelet neural network – a feed-forward neural network with one-hidden-layer of nodes, whose basis functions are drawn from a family of orthonormal wavelets (http://www.ncl.ac.uk/pat/neural-networks.html).

Wavelets, in addition to forming an orthogonal basis, are capable of explicitly representing the behavior of a function at various resolutions of input variables. Consequently, a wavelet network is first trained to learn the mapping at the coarsest resolution level. In subsequent stages, the network is trained to incorporate elements of the mapping at higher and higher resolutions. Such hierarchical, multi-resolution training has many attractive features for solving engineering problems, resulting in a more meaningful interpretation of the resulting mapping and more efficient training and adaptation of the network compared to conventional methods. The wavelet theory provides useful guidelines for the construction and initialization of networks and consequently, the training times are significantly reduced (http://www.ncl.ac.uk/pat/neural-networks.html).

Wavelet networks employ activation functions that are dilated and translated versions of a single function $\psi : R^d \to R$, where d is the input dimension (Zhang and Benvniste, 1992; Zhang, 1997). This function called the 'mother wavelet' is localized both in the space and frequency domains (Becerra et al., 2005). Based on wavelet theory, the wavelet neural network (WNN) was proposed as a universal tool for functional approximation, which shows surprising effectiveness in solving the conventional problem of poor convergence or even divergence encountered in other kinds of neural networks. It can dramatically increase convergence speed (Zhang et al., 2001). The structure of the WNN with two input and six hidden nodes is shown in Fig. 1.

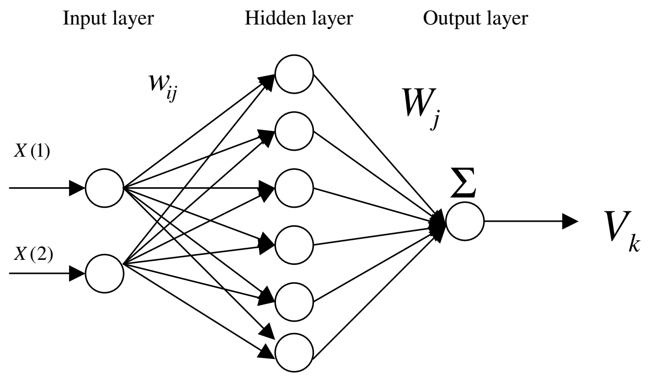

Fig. 1. The structure of a wavelet neural network.

The WNN consists of three layers: input layer, hidden layer and output layer. All the units in each layer are fully connected to the nodes in the next layer. The output layer contains a single unit. Based on the use of activation functions in the hidden nodes, there are two variants of WNN that are implemented here. One based on the Morlet wavelet function (Rioul and Vetterli, 1991) and other being the Gaussian wavelet function. The training algorithm for a WNN is as follows (Zhang et al., 2001):

(1) Select the number of hidden nodes required. Initialize the dilation and translation parameters for these nodes to some random values. Also initialize the weights for the connections between the input and hidden layer and also for the connections between the hidden and the output layer. It should be taken note that the random values should be limited to the interval $(0,1)$.

(2) The output value of the sample V_k, $k = 1,2,\dots,np$, where np is the number of sample s, is calculated with the following formula:

$$V_k = \sum_{j=1}^{nhn} W_j f\left(\frac{\sum_{i=1}^{nin} w_{ij}x_{ki} - b_j}{a_j}\right), \tag{2}$$

where nin is the number of input nodes and nhn is the number of hidden nodes and $k = 1,2,\dots,np$.
In (2) when $f(t)$ is taken as a Morlet mother wavelet (Grossmann and Morlet, 1984) it has the following form:

$$f(t) = \cos(1.75t)\exp\left(-\frac{t^2}{2}\right) \tag{3}$$

and when taken as Gaussian wavelet it becomes

$$f(t) = \exp(-t^2). \tag{4}$$

(3) Reduce the error of prediction by adjusting W_j, w_{ij}, a_j, b_j using $\Delta W_j, \Delta w_{ij}, \Delta a_j, \Delta b_j$ (see formulas (5)–(8)). In the WNN, the gradient descend algorithm is employed:

$$\Delta W_j(t+1) = -\eta\frac{\partial E}{\partial W_j(t)} + \alpha\Delta W_j(t), \tag{5}$$

$$\Delta w_{ij}(t+1) = -\eta\frac{\partial E}{\partial w_{ij}(t)} + \alpha\Delta w_{ij}(t), \tag{6}$$

$$\Delta a_j(t+1) = -\eta\frac{\partial E}{\partial a_j(t)} + \alpha\Delta a_j(t), \tag{7}$$

$$\Delta b_j(t+1) = -\eta\frac{\partial E}{\partial b_j(t)} + \alpha\Delta b_j(t), \tag{8}$$

where the error function E is taken as

$$E = \frac{1}{2}\sum_{k=1}^{np}(V_k - \widehat{V}_k)^2, \tag{9}$$

where η and α are the learning and the momentum rates, respectively.

(4) Return to step (2), the process is continued until E satisfies the given error criteria, and the whole training of the WNN is completed.

3.1. Training WNN with TA learning algorithm

It should be noted that gradient descent algorithm is used for training the original WNN.

In this paper, we developed threshold accepting based learning algorithm for wavelet neural networks and we call the resulting WNN as threshold accepting based wavelet neural network (TAWNN). Threshold accepting (TA) algorithm proposed by Dueck and Scheuer (1990), is a faster variant of the original simulated annealing algorithm wherein the acceptance of a new solution is determined by a deterministic criterion rather than a probabilistic one. The important feature of TA is that it accepts any new solution, which is not much worse than the current one. The idea behind the TA-based training algorithm used in building threshold accepting neural networks (TANN) (Ravi et al., 2005; Ravi and Zimmermann, 2001, 2003) is that the 'forward pass' of the back propagation algorithm is not disturbed. But, in the backward pass, TA is used to update all the weights instead of the steepest descent algorithm used in back propagation. In this context, the set of weights of the neural network (between input layer hidden and hidden layer and output layer) becomes the vector of decision variables. In this paper, TA algorithm is proposed to train the WNN. The resulting network is henceforth referred to as TAWNN. The flowchart for the TA-based training algorithm for WNN is presented in Fig. 2. In TAWNN, similar to TANN, weights connecting the input to hidden nodes and hidden to output nodes and the dilation and translation parameters specific to original WNN are all considered as decision variables. The entire TAWNN algorithm is not presented here for the sake of brevity. However, TAWNN in its structure exactly resembles TANN but for the presence of additional variables i.e. dilation and translation parameters. Interested reader is referred to Ravi and Zimmermann (2003) and Ravi et al. (2005) for further details on TANN. When Morlet wavelet is used as the activation function, the resulting TAWNN is named TAWNN-Morlet and when Gaussian wavelet is used as the activation function, the resulting TAWNN is named TAWNN-Gaussian. The objective function computation in TAWNN algorithm is described as follows:

Step 1: Normalize the input data to make sure that units of measurement (for e.g., kilograms, kilometers, etc.) are removed and all the attributes are in the same range [0, 1] as follows:

$$x_{ki} = \frac{x_{ki} - x_{\min_i}}{x_{\max_i} - x_{\min_i}}; \quad k = 1, 2, \ldots, np \text{ and } i = 1, 2, \ldots, nin,$$

where $x_{\min_j}$, $x_{\max_j}$ are the minimum and maximum of the jth input variable and np, nin are the number of patterns and the input variables, respectively. Thus, all the variables become dimensionless after effecting this transformation.

Step 2: Select the number of hidden nodes *nhn*.

Step 3: The output value of the sample V_n is calculated with the following formula:

$$V_k = \sum_{j=1}^{nhn} W_j f\left(\frac{\sum_{i=1}^{nin} w_{ij} x_{ki} - b_j}{a_j}\right), \tag{2}$$

where *nin* is the number of input nodes and *nhn* is the number of hidden nodes and $k = 1, 2, \ldots, np$. The activation function $f(t)$ when taken as a Morlet mother wavelet (Grossmann and Morlet, 1984) becomes

$$f(t) = \cos(1.75t)\exp\left(-\frac{t^2}{2}\right) \tag{3}$$

and Gaussian wavelet

$$f(t) = \exp(-t^2). \tag{4}$$

Now using these predicted outputs V_k we calculate the objective function.

Step 4: Repeat step 3 for all patterns. Then calculate the objective function value, which is the mean squared error (MSE) for the current iteration as follows:

$$\text{MSE} = \sum_{k=1}^{np} (V_k - \widehat{V}_k)^2.$$

3.2. Neighborhood search scheme

The inner iterations of TAWNN perform a neighborhood search, which is the essential ingredient of any TA implementation. For neighborhood search, the formula used for all the decision variables in TAWNN is as follows:

$$w_{ij}^{c} = w_{ij}^{o} + (wul - wll) * (2u - 1)^p, \quad i = 1, \ldots, nin \text{ and } j = 1, \ldots, nhn,$$
$$W_j^{c} = W_j^{o} + (wul - wll) * (2u - 1)^p, \quad j = 1, \ldots, nhn,$$
$$a_j^{c} = a_j^{o} + (wul - wll) * (2u - 1)^p, \quad j = 1, \ldots, nhn,$$
$$b_j^{c} = b_j^{o} + (wul - wll) * (2u - 1)^p, \quad j = 1, \ldots, nhn,$$

where superscripts c and o stand for *current* and *old* solutions, respectively, u is the random number drawn from a uniform distribution in [0, 1], p is an odd integer.

In the flowchart of TAWNN, parameter *thresh* is very important and is used in accepting or rejecting a neighborhood solution. Parameter *limit* suggests the number of neighborhood searches to be made in a given global iteration. Parameter *eps* is the factor by which thresh is reduced in each global iteration.

4. Dataset description

The data used in the present study comes from two sources namely, from the IBM data processing services

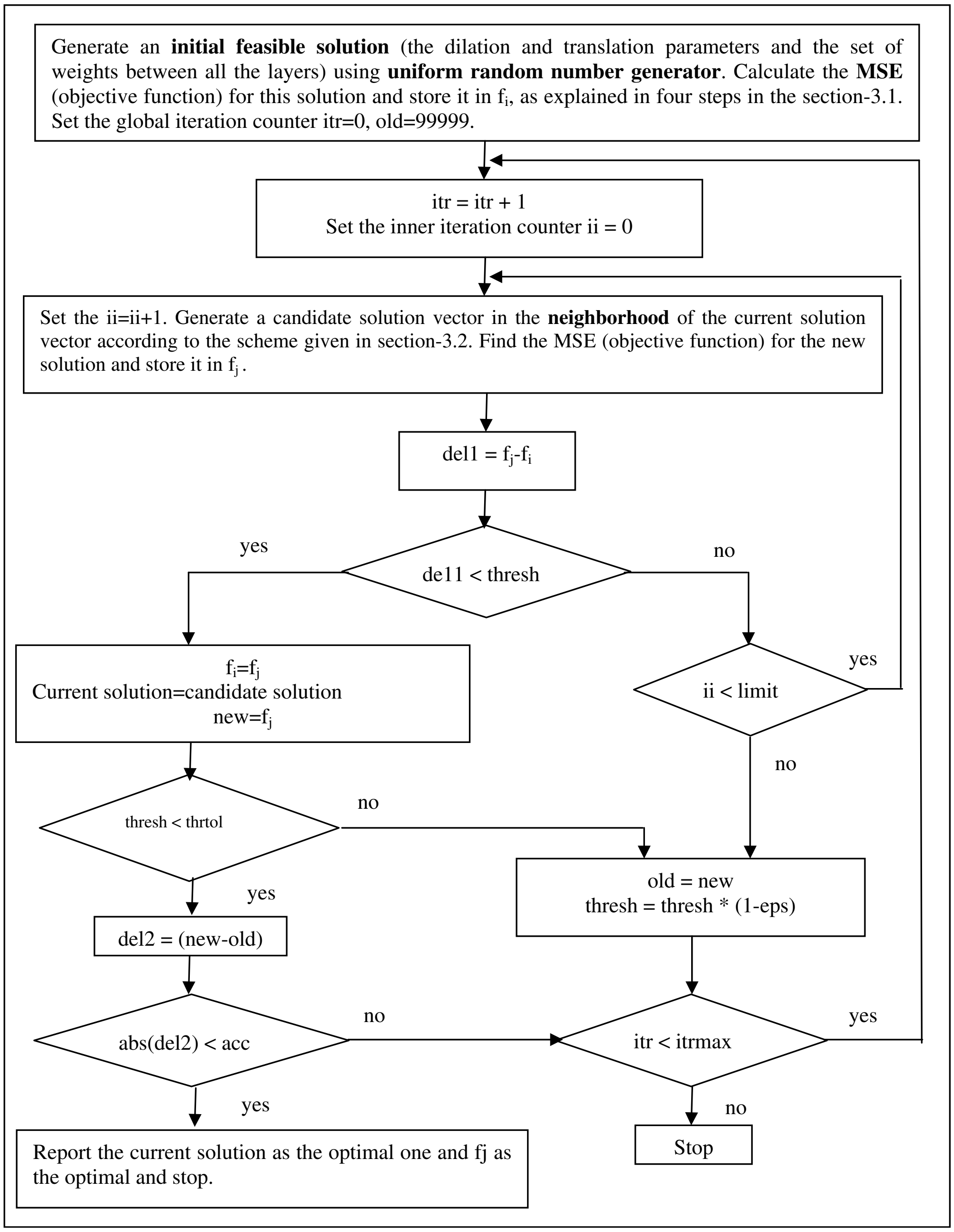
Generate an initial feasible solution (the dilation and translation parameters and the set of weights between all the layers) using uniform random number generator. Calculate the MSE (objective function) for this solution and store it in fi, as explained in four steps in the section-3.1. Set the global iteration counter itr=0, old=99999.
itr = itr + 1
Set the inner iteration counter ii = 0
Set the ii=ii+1. Generate a candidate solution vector in the neighborhood of the current solution vector according to the scheme given in section-3.2. Find the MSE (objective function) for the new solution and store it in fj.
del1 = fj-fi
de11 < thresh
yes
no
fi=fj
Current solution=candidate solution
new=fj
ii < limit
yes
no
thresh < thrtol
no
yes
old = new
thresh = thresh * (1-eps)
del2 = (new-old)
abs(del2) < acc
no
itr < itrmax
yes
yes
no
Report the current solution as the optimal one and fj as the optimal and stop.
Stop

Fig. 2. TA-based training algorithm for WNN (TAWNN).

(IBMDPS) organization (Matson et al., 1994) and from a major Canadian financial organization (Abran and Robillard, 1996). Effort drivers that have potential impacts on the project effort were selected. In IBMDPS dataset, five effort drivers of input count (IC), output count (OC), query count (QC), file count (FC) and adjustment factors (AF) were considered as explanatory variables for the model construction and effort, considered as dependent variable, is estimated from these drivers. Twenty four projects are present in the dataset. Another historical database used to carry out the empirical experiments came from Canadian financial (CF) datasets. During the data collection stage, this organization used IFPUG 90 rules in the data collection process (Chiu and Huang, 2007). The complete data of these projects are presented in Abran and Robillard (1996). Chiu and Huang (2007) considered effort drivers of input count (IC), output count (OC), inquiry count (IQC), internal logical files count (ILF), external interface files (EIF) and adjustment factor (AF) as explanatory variables for model construction and effort was considered as dependent variable.

5. Methodology

We divided the datasets into training and test sets in the ratio of 80%:20% in holdout method. All the models are

Table 1
MMRE results of different models for IBMDPS database in holdout mode

Technique	MMRE value
MLP	1.82
MLR	3.283541
RBF	0.363952
DENFIS	3.283541
SVM	0.929062
WNN-Morlet	0.113046
WNN-Gaussian	0.032787
TAWNN-Gaussian	0.84
TAWNN-Morlet	0.730461

Table 2
MMRE results of different models for IBMDPS database in 3-fold cross-validation

Technique	FOLD			Average
	FOLD-1	FOLD-2	FOLD-3	
WNN-Morlet	0.132475	0.107904	0.125562	0.12198
WNN-Gaussian	0.054819	0.052327	0.108325	0.071824
TAWNN-Gaussian	0.396261	0.460999	517,412	0.458224
TAWNN-Morlet	0.259352	0.815123	0.663326	0.579267
MLR	0.8261	1.032298	1.18848	1.015626
DENFIS	0.690414	1.032298	1.18848	0.970397
MLP	0.247004	1.337848	0.595998	0.72695
RBF	0.323558	1.678181	1.064399	1.022046
SVM	0.522356	1.403385	1.47084	1.1321

Table 3
Results with different models for IBMDPS on test data

Model	MMRE	Pred(0.25)	MdMRE
WNN-Morlet	0.12198	0.70833	0.177521
WNN-Gaussian	0.071824	0.875	0.097048
TAWNN-Morlet	0.579267	0.291667	0.5446
TAWNN-Gaussian	0.458224	0.416667	0.360288
MLR	1.015626	0.33	0.53
RBF	1.022046	0.375	0.35
DENFIS	0.970397	0.33	0.53
SVM	1.1321	0.375	0.57
AE[a]	0.59	0.39	0.37
AMH[a]	0.56	0.43	0.35
AMK[a]	0.46	0.43	0.35
AAE[a]	0.38	0.57	0.19
AAMH[a]	0.36	0.52	0.24
AAMK[a]	0.43	0.61	0.20
ANN[a]	0.90	0.22	0.61
CART[a]	0.77	0.26	0.50
OLS[a]	0.72	0.33	0.41

[a] From Chiu and Huang (2007).

trained with training data and tested with the test data. In addition, 3-fold cross-validation was also performed.

5.1. Evaluation criteria

We employ the following criteria to assess and compare the performance of cost estimation models. A common

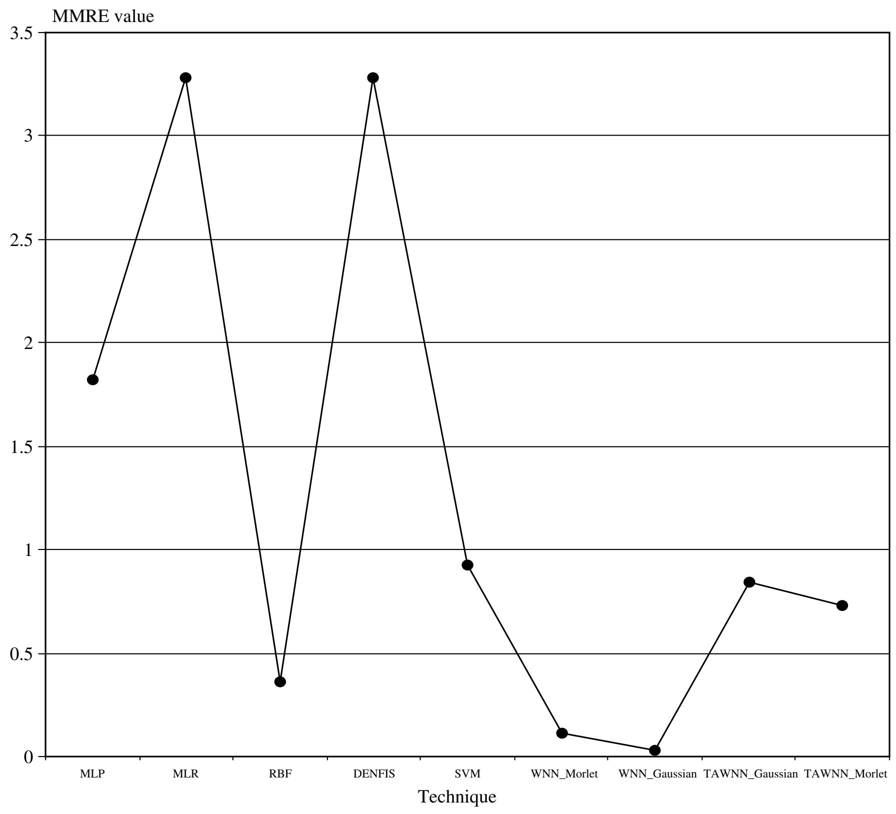

Fig. 3. MMRE results of different models in holdout mode for IBMDPS database.

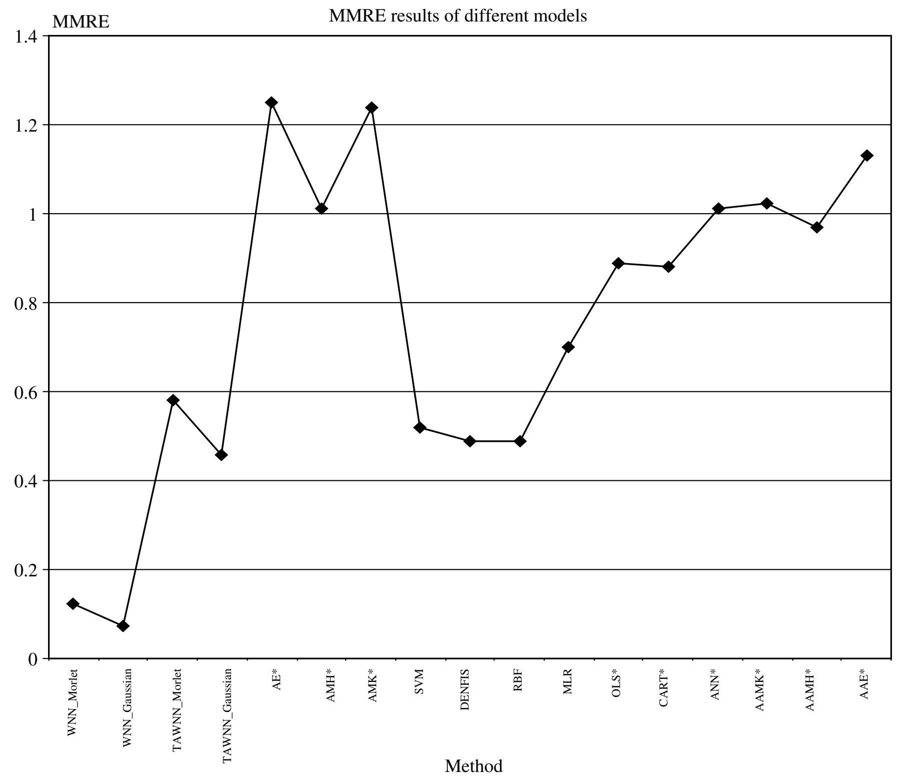

Fig. 4. MMRE results of different models in 3-fold cross-validation for IBMDPS database.

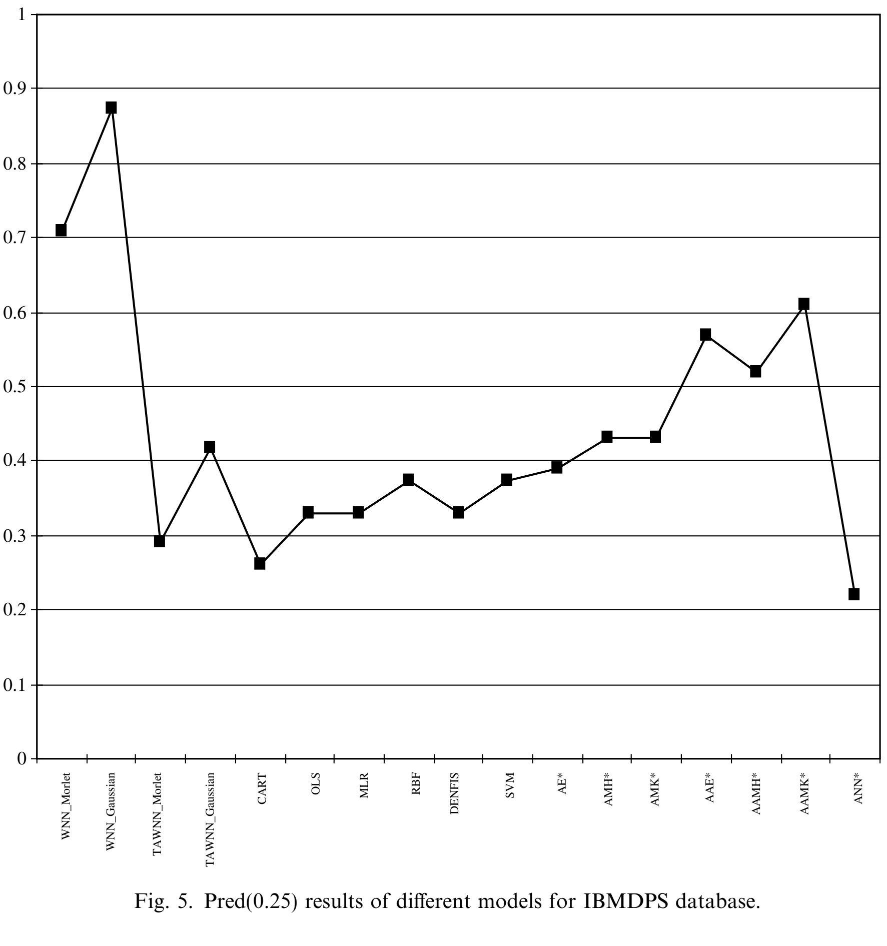

Fig. 5. Pred(0.25) results of different models for IBMDPS database.

criterion for the evaluation of cost estimation model is the magnitude of relative error (MRE), which is defined as

$$\mathrm{MRE} = \left|\left(\frac{\mathrm{Effort}_{\mathrm{actual}} - \mathrm{Effort}_{\mathrm{estimated}}}{\mathrm{Effort}_{\mathrm{actual}}}\right)\right|.$$

The MRE values are calculated for each project in the datasets, while mean magnitude of relative error (MMRE) computes the average of MRE over N projects

$$\mathrm{MMRE} = \frac{1}{N}\sum_{X=1}^{N}\mathrm{MRE}x.$$

Generally, the acceptable target value for MMRE is 25%. This indicates that on the average, the accuracy of the established estimation models would be less than 25%.

Another widely used criterion is the Pred(l) which represents the percentage of MRE that is less than or equal to the value l among all projects. This measure is often used in literature and is the proportion of the projects for a given level of accuracy. The definition of Pred(l) is given as follows:

$$\mathrm{P}red(l) = \frac{k}{n},$$

where n is the total number of observations and k is the number of observations whose MRE is less than or equal to l.

A common value for l is 0.25, which is also used in the present study. The Pred(0.25) represents the percentage of projects whose MRE is less than or equal to 25%. The Pred(0.25) value identifies the effort estimates that are generally accurate while the MMRE is fairly conservative with a bias against overestimates.

Another evaluation criterion is the MdMRE, which measures the median of all MREs. MdMRE is less sensitive to extreme values. It exhibits a similar pattern to MMRE

Table 4
MMRE results of different models for CF database in holdout mode

Technique	MMRE value
MLP	1.38
MLR	1.58332
RBF	0.50971
DENFIS	1.131356
SVM	0.661171
WNN-Morlet	0.217086
WNN-Gaussian	0.147292
TAWNN-Gaussian	0.391968
TAWNN-Morlet	0.69

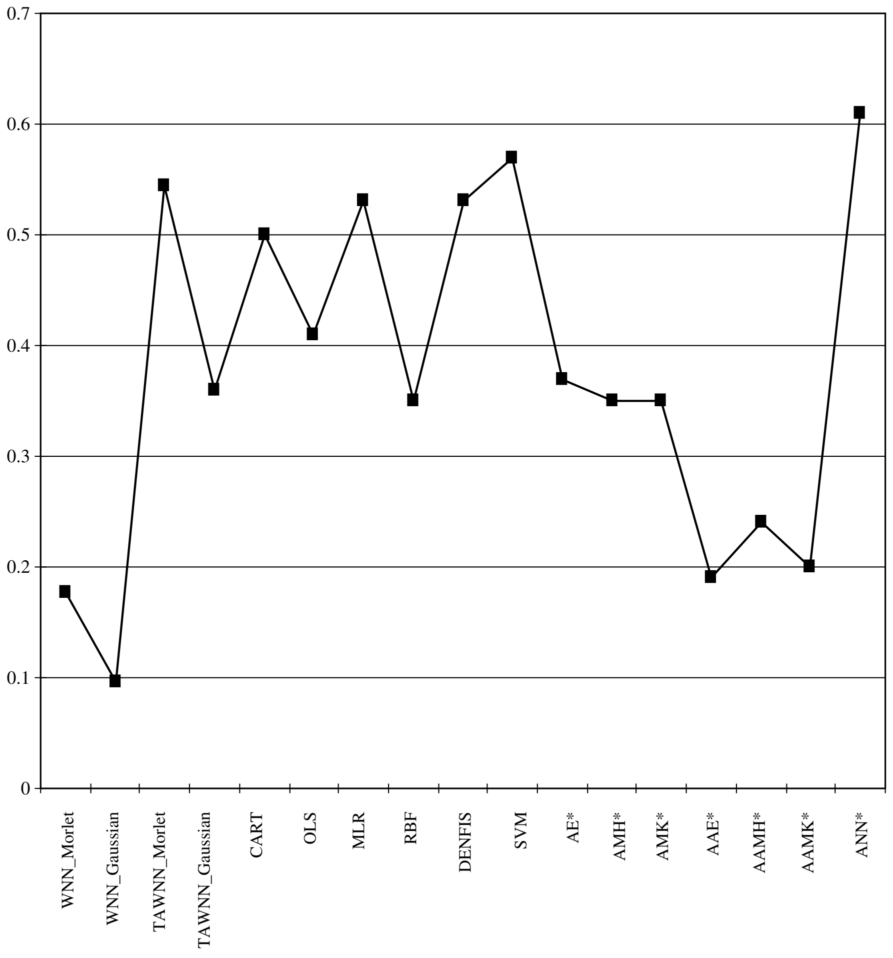

Fig. 6. MdMRE results of different models for IBMDPS database.

but it is more likely to select the true model if the underestimation is served (Foss et al., 2003). Since MdMRE, MMRE and Pred(0.25) are well-known evaluation criteria, they are adopted as performance indicators in the present paper. If one constructs an effort estimation model using a particular dataset and then computes the accuracy of the model using the same datasets, the accuracy evaluation may be too optimistic (Chiu and Huang, 2007). The estimation error may be artificially low and does not reflect the performance of the model on another unseen dataset (Hays, 1994). A cross-validation approach gives a more realistic assessment of accuracy. It involves dividing the whole dataset into multiple training and test sets. The results in the training stages present the estimation accuracies from the model construction datasets. The test stage evaluates the estimation accuracies of the models using the other unseen datasets. In the training stage, a cross-validation approach calculates the accuracy for each training dataset, and then computes the average accuracy across all training sets. In the test stage, it calculates the accuracy for each test set, and then computes the average accuracy across all test sets. We performed the testing by using 3-fold cross-validation since it is a widely used method (Chiu and Huang, 2007).

6. Results and discussion

Four types of models using wavelet neural network are constructed in order to estimate the effort of a software development. The other widely used methods of the intelligent techniques MLR, MLP, RBFN, DENFIS and SVM were also used for constructing their respective estimation models in order to compare their estimation capabilities.

6.1. Adjustment factors in function points

The value adjustment factor (VAF) is based on 14 general system characteristics (GSC's). The degrees of influence range on a scale of zero to five, from no influence to strong influence. The GSCs are listed below (http://www.c-sharpcorner.com/UploadFile/imtiyazmulla/FunctionalPointAnalysisP211302005012541AM/FunctionalPointAnalysisP2.aspx):

Table 5
MMRE results of different models for CF database in 3-fold cross-validation

Technique	FOLD			Average
	FOLD-1	FOLD-2	FOLD-3	
WNN-Morlet	0.33509	0.258721	0.112134	0.235315
WNN-Gaussian	0.430127	0.256229	0.072338	0.252898
TAWNN-Gaussian	1.1	0.588798	0.434011	0.707603
TAWNN-Morlet	0.971276	0.68	0.604639	0.751972
MLR	1.645957	1.008544	0.439373	1.031291
DENFIS	1.645957	1.008544	0.439373	1.031291
MLP	1.4866	0.62	0.358418	0.821672
RBF	1.504822	0.580015	0.296616	0.793835
SVM	0.647289	0.730222	0.319739	0.56575

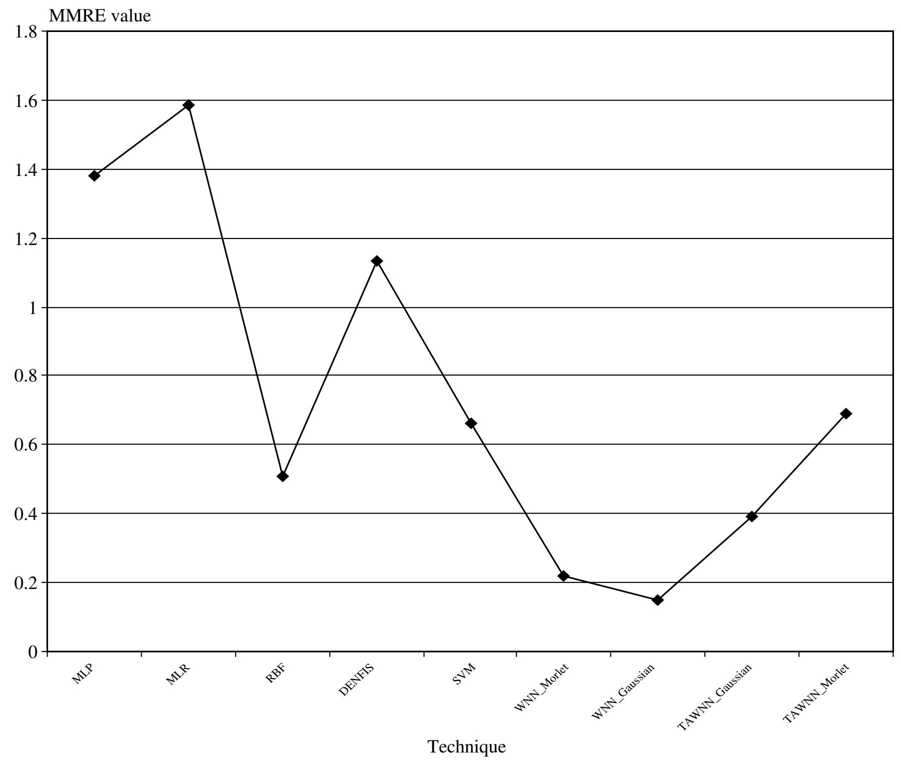

Fig. 7. MMRE results of different models in holdout mode for CF database.

(i) Data communication; (ii) distributed functions; (iii) performance objectives; (iv) heavily used configuration; (v) transaction rate; (vi) online data entry; (vii) end-user efficiency; (viii) online update; (ix) complex processing; (x) reusability; (xi) installation ease; (xii) operational ease; (xiii) multiple sites; (xiv) facilitate change.

VAF is computed as follows:

VAF = 0.65 + (Sum of degrees of Influence of the 14 GSCs)/100.

The final function point count (FP) is obtained by multiplying VAF by the unadjusted function point (UAF) as follows: FP = UAF * VAF.

6.2. The IBMDPS database

In this estimation model using the four components as predictors (excluding FP) we take the position that the adjustment factor and weights act multiplicatively with each variable. Albrecht and Gaffeny (1983) used the following average weights to determine effort: number of inputs (IN) * 4; number of outputs (OUT) * 5; number of inquiries (INQ) * 4; number of master files (FILE) * 10, i.e. the inputs for our estimation model are

$$IN^{\sim} = IN * 4 * \text{Adjustment Factor (AF)},$$
$$OUT^{\sim} = OUT * 5 * \text{Adjustment Factor (AF)},$$
$$INQ^{\sim} = INQ * 4 * \text{Adjustment Factor (AF) and}$$
$$FILE^{\sim} = FILE * 10 * \text{Adjustment Factor (AF)},$$

where $IN^{\sim}$, $OUT^{\sim}$, $INQ^{\sim}$ and $ILE^{\sim}$ are inputs for our estimation models.

We considered these as explanatory variables to estimation models that are constructed from intelligent techniques, neuro-fuzzy techniques and other regression techniques. Tables 1 and 2 present the results of the different software effort estimation models for the IBMDPS dataset. These results are for four WNN estimation models and other widely used intelligent techniques MLP, RBFN, DENFIS, SVM and multiple linear regression (MLR) using MMRE as evaluation criterion on the test data.

Table 6
Results of different models on test stage for CF database

Model	MMRE	Pred(0.25)	MdMRE
WNN-Morlet	0.198213	0.66637	0.16376
WNN-Gaussian	0.2313013	0.56667	0.17747
TAWNN-Gaussian	0.707603	0.14	0.580491
TAWNN-Morlet	0.751972	0.33	0.613595
MLR	1.031291	0.14	0.99
RBF	0.793835	0.19	0.65
DENFIS	1.08	0.42	0.45
SVM	0.56575	0.24	0.53
AE[a]	1.25	0.19	0.58
AMH[a]	1.01	0.29	0.51
AMK[a]	1.24	0.24	0.55
AAE[a]	0.52	0.43	0.36
AAMH[a]	0.49	0.38	0.31
AAMK[a]	0.49	0.38	0.38
ANN[a]	0.70	0.10	0.36
CART[a]	0.89	0.29	0.43
OLS[a]	0.88	0.33	0.41

[a] From Chiu and Huang (2007).

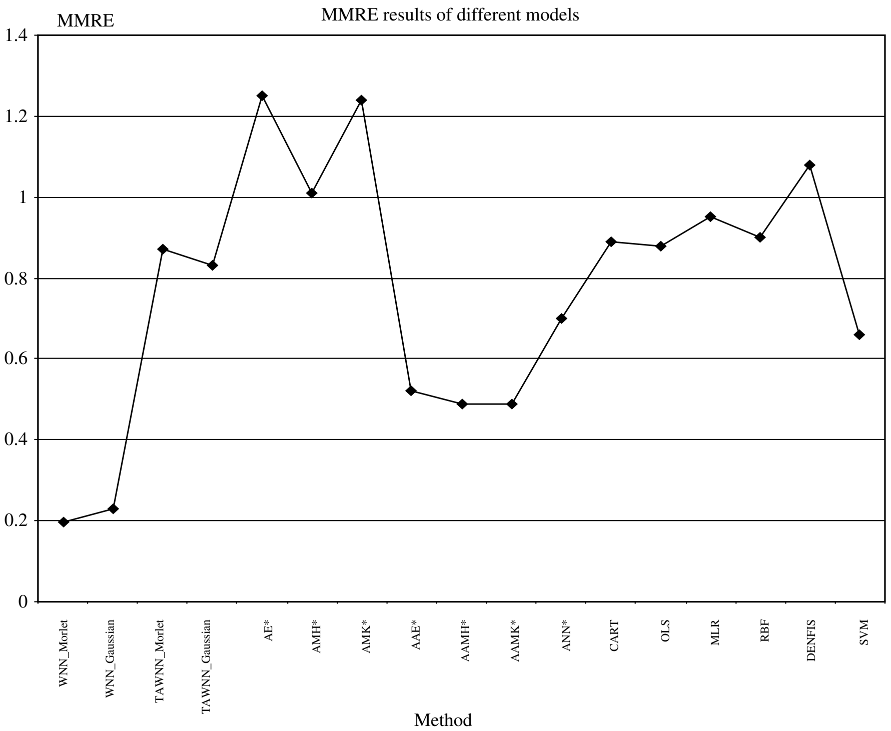

Fig. 8. MMRE results of different models in 3-fold cross-validation for CF database.

While comparing these effort estimation models, it should be noted that a model with a lower value of MMRE presents an accurate estimate of the software development effort. The test stage for each evaluation criterion computes the estimation accuracy of the model using the test dataset. For IBMDPS dataset, Table 1 presents the results of the different software effort estimation models in the holdout mode. Fig. 3 shows the results of all models in the test stage. WNN-Gaussian with an MMRE value of 0.091503, in holdout mode, outperformed all other techniques in the test data.

Results of MMRE for the IBMDPS dataset in the 3-fold cross-validation for all the methods are presented in Table 2 and Fig. 3. In Table 3, presents the average results of MMRE, MdMRE and Pred(0.25) obtained in 3-fold cross-validation method for all methods. Further, it is observed that WNN-Gaussian performed equally well. Other models such as TAWNN-Morlet and TAWNN-Gaussian have also yielded good results but those are not comparable with other methods such as analogy-based estimations without adjustment using Euclidean distance (AE), Manhattan distance (AMH), Minikowski distance (AMK), analogy-based estimations with adjustment using Euclidean distance (AAE), Manhattan distance (AAMH) and Minikowski distance (AAMK), which are employed by Chiu and Huang (2007).

Fig. 4 depicts MMRE results of different models in 3-fold cross-validation for IBMDPS dataset. Then, Fig. 5 depicts the percentage of projects whose MRE are equal to or less than 0.25, also know as Pred(0.25). In terms of Pred(0.25) also, WNN-Morlet, WNN-Gaussian outperformed other techniques including the models of Chiu and Huang (2007) in 3-fold cross-validation. Fig. 6 presents the results in terms of MdMRE. In terms of MdMRE also, WNN-Morlet, WNN-Gaussian outperformed other techniques including the models of Chiu and Huang (2007) in 3-fold cross-validation. Thus, based on the MMRE, Pred(0.25) and MdMRE evaluation criteria, WNN-Morlet, WNN-Gaussian outperformed all other techniques. In general, these encouraging results show that WNN is an effective technique to improve the efficiency.

6.3. The CF database

In this dataset also using the four components as explanatory variables (rather than FP) we take the position that the adjustment factor and weights, act multiplicatively with

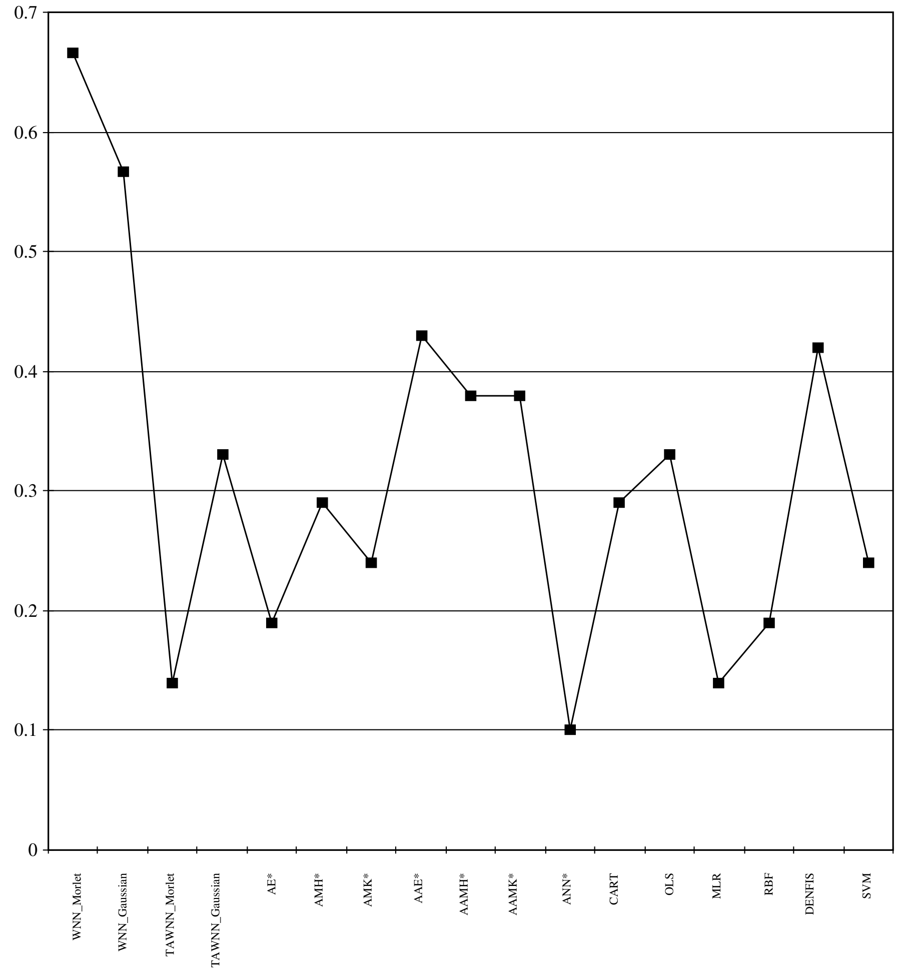

Fig. 9. Pred(0.25) results of different models for CF database.

each variable. We used the following average weights to determine effort (Chiu and Huang, 2007): input count (IC) * 4; output count (OC) * 5; inquiry count (IQC) * 4; (internal logical files count (ILF) + external interface files (EIF)) * 10, i.e. the inputs for our estimation model are

$$IC^{\sim} = IC * 4 * \text{Adjustment Factor (AF)}/100,$$
$$OC^{\sim} = OC * 5 * \text{Adjustment Factor (AF)}/100,$$
$$IQC^{\sim} = IQC * 4 * \text{Adjustment Factor (AF)}/100,$$
$$FILE^{\sim} = (ILF + ELF) * 10 * \text{Adjustment Factor (AF)}/100.$$

Thus, we employed these explanatory variables to estimation models that are constructed from intelligent techniques, neuro-fuzzy techniques and other regression techniques. For CF dataset, Table 4 presents the results of the different software effort estimation models in the holdout mode. The models used here are WNN-Morlet, WNN-Gaussian, TAWNN-Morlet and TAWNN-Gaussian and other intelligent techniques such as MLR, MLP, RBFN, DENFIS and SVM. Fig. 7 depicts the MMRE values of all models in the test stage. WNN-Gaussian with an MMRE value of 0.147292, in holdout mode, outperformed all other techniques in this case.

Results of the MMRE in the test stage in 3-fold cross-validation are presented in Table 5 and Fig. 8. In Table 6, results obtained by WNN-Morlet, WNN-Gaussian, TAWNN-Morlet and TAWNN-Gaussian and other intelligent techniques RBF, BPNN, MLR and SVM using three evaluation criteria viz., MMRE, Pred(0.25) and MdMRE in testing stage are presented. Among these models, WNN-Morlet, produced best results in hold mode and 3-fold cross-validation method. Further, WNN-Gaussian performed equally well. Other models such as TAWNN-Morlet and TAWNN-Gaussian have also yielded good results but those are not comparable with other methods such as AAE, AAMH and AAMK, which are applied by Chiu and Huang (2007).

Fig. 9 depicts the results of Pred(0.25). In terms of Pred(0.25) also, WNN-Morlet, WNN-Gaussian outperformed other techniques including the models of Chiu and Huang (2007) in 3-fold cross-validation. Fig. 10 depicts the results in terms of MdMRE. In terms of MdMRE also, WNN-Morlet, WNN-Gaussian outperformed other techniques including the models of Chiu and Huang (2007) in 3-fold cross-validation. Thus, based on the MMRE, Pred(0.25) and MdMRE evaluation criteria, WNN-Morlet, WNN-Gaussian outperformed all other techniques. In general, these encouraging results show that applying WNN to effort estimation is an effective approach to improve the efficiency.

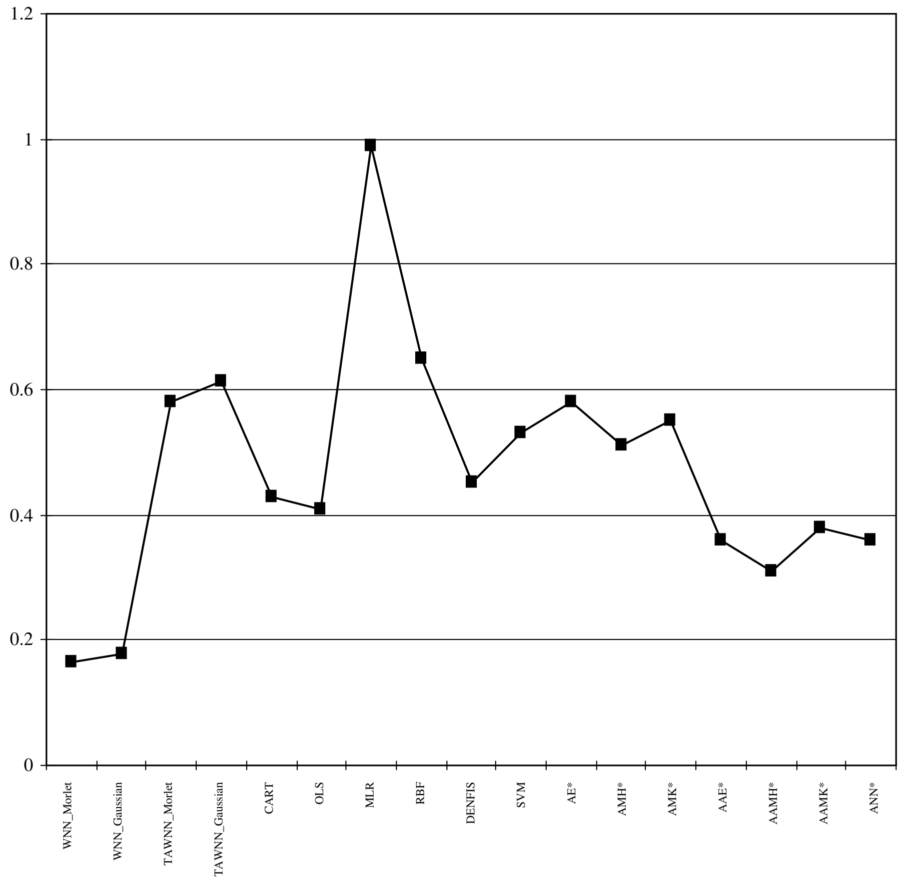

Fig. 10. MdMRE results of different models for CF database.

7. Conclusions

A reliable and accurate estimate of software development effort has always been a challenge for both the software industrial and academic communities. It is well-known that software project management teams can greatly benefit from knowing the estimated cost of their software projects. The benefits can have greater impact if accurate cost estimations are deduced during the early stages of the project life cycle. Estimating the process and performance behavior of a software project, early in the software life cycle, is very challenging and often very difficult. The model incorporates data collected from several projects, and uses it for its effort estimations. In the paper, we used two types of WNN – one each with Morlet function and Gaussian function as transfer function and proposed two variants of TAWNN. The effectiveness of the WNN is compared with that of MLP, RBFN, MLR, DENFIS and SVM in terms of MMRE obtained on Canadian financial dataset and IBM Data processing services dataset. Based on the experiments performed, it is observed that the WNN-Morlet and WNN-Gaussian for CF and IBMDPS datasets outperformed all the other techniques. The WNN-Morlet function and WNN-Gaussian outscored other techniques such as RBF, BPNN, MLR, SVM and adjusted analogy-based estimation models with respect to the evaluation criteria MMRE, MdMRE and Pred(0.25). The results demonstrated that applying WNN to software effort estimation is a feasible approach to improve estimation accuracy. In addition, the proposed TAWNN is also comparable to widely used estimation models. Finally, we conclude that the present work provides accurate forecasts for the software development effort. Future research can replicate and confirm this estimation technique with fresh datasets.

References

Abran, A., Robillard, P.N., 1996. Function points analysis: an empirical measurement processes. IEEE Transactions on Software Engineering 22 (12), 895–910.

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

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

Becerra, V.M., Galvao, R.K.H., Abou-Seada, M., 2005. Neural and wavelet network models for financial distress classification. Data Mining and Knowledge Discovery 11 (1), 35–55.

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

Chiu, N.H., Huang, S.J., 2007. The adjusted analogy-based software effort estimation based on similarity distances. Journal of Systems and Software 80 (4), 628–640.

Cristianini, N., Shawe-Taylor, J., 2000. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge, UK.

Dueck, G., Scheuer, T., 1990. Threshold accepting: a general-purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics 90, 161–175.

Fenton, N.E., Pleeger, S.L., 1997. Software Metrics: A Rigorous and Practical Approach, second ed. PWS, Boston, MA, USA.

Foss, T., Stensrud, E., Kitchenham, B., Myrtveit, I., 2003. A simulation study of the model evaluation criterion MMRE. IEEE Transactions on Software Engineering 29 (11), 985–995.

Gray, A.R., 1999. A simulation-based comparison of empirical modeling techniques for software metric models of development effort. In: Proceedings of ICONIP, Sixth International Conference on Neural Information Processing, Perth, WA, Australia, vol. 2, pp. 526–531.

Gray, A. MacDonell, S., 1997. Applications of fuzzy logic to software metric models for development effort estimation. In: Proceedings of the 1997 Annual meeting of the North American Fuzzy Information Proceeding Society – NAFIPS'97, pp. 394–399.

Grossmann, A., Morlet, J., 1984. Decomposition of Hardy functions into square integrable wavelets of constant shape. SIAM Journal of Mathematical Analysis 15 (4), 723–726.

Hays, W.L., 1994. Statistics, fifth ed. Harcourt Brace, Orlando.

Hughes, R.T., 1996. An evaluation of machine learning techniques for software effort estimation. University of Brighton.

Heiat, A., 2002. Comparison of artificial neural network and regression models for estimating software development effort. Information and Software Technology 44 (15), 911–922.

Idri, A., Khosgoftaar, T.M., Abran, A., 2002. Can neural networks be easily interpreted in software cost estimation? In: 2002 World Congress on Computational Intelligence, Honolulu, Hawaii, USA, pp. 12–17.

Jorgerson, M., 1995. Experience with accuracy of software maintenance task effort prediction models. IEEE Transaction on Software Engineering 21 (8), 674–681.

Jorgenson, M., Indahl, U., Sjoberg, D., 2003. Software effort estimation by analogy and regression toward the mean. Journal of Systems and Software 68 (3), 253–262.

Kasabov, N.K., Song, Q., 2002. DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction. IEEE Transactions on Fuzzy Systems 10 (2), 144–154.

Kemerer, C., 1987. An empirical validation of software cost estimation models. Communications of the ACM 30 (5), 416–429.

Kitchenham, B., Pickard, L.M., Linkman, S., Jones, P.W., 2003. Modeling software bidding risks. IEEE Transactions on Software Engineering 29 (6), 542–554.

Matson, J.E., Barrett, B.E., Mellichamp, J.M., 1994. Software development cost estimation using function points. IEEE Transactions on Software Engineering 20 (4), 275–287.

Miyazaki, Y., Terakado, Y., Ozaki, K., Nozaki, N., 1994. Robust regression for developing Software estimation models. Journal of System and Software 27 (1), 16–35.

Moody, J., Darken, C.J., 1989. Fast learning in networks of locally tuned processing units. Neural Computation 1 (2), 281–294.

Mukhopadhyay, T., Vicinanza, S.S., Prietula, M.J., 1992. Examining the feasibility of a case-based reasoning model for software effort estimation. MIS Quarterly 4 (1), 155–171.

Pressman, R.S., 1997. Software Engineering: A Practitioner's Approach, fourth ed. McGraw-Hill, New York, NY, USA.

Ravi, V., Zimmermann, H.J., 2001. A neural network and fuzzy rule base hybrid for pattern classification. Soft Computing 5 (2), 152–159.

Ravi, V., Zimmermann, H.J., 2003. Optimization of neural networks via threshold accepting: a comparison with back propagation algorithm. In: Conference on Neuro-Computing and Evolving Intelligence, Auckland, New Zealand.

Ravi, V., Murty, B.S.N., Dutt, N.V.K., 2005. Forecasting the properties of carboxylic acids by a threshold accepting-trained neural network. Indian Chemical Engineer 47 (3), 147–151.

Retrieved from the URL: http://mathworld.wolfram.com/wavelet.html on 23.03.07.

Retrieved from the URL: http://www.ncl.ac.uk/pat/neural-networks.html on 23.03.07.

Rioul, O., Vetterli, M., 1991. Wavelets and signal processing. IEEE Signal Processing Magazine 8 (4), 14–38.

Samson, B., Ellison, D., Dugard, P., 1997. Software cost estimation using an Albus perceptron (CMAC). Information and Software Technology 39 (1), 55–60.

Schofield, C., 1998. Non-algorithmic effort estimation techniques. Technical Report TR98-01.

Seluca, C., 1995. An investigation into software effort estimation using a back-propagation neural network, M.Sc.Thesis, Bournemouth University, UK.

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

Srinivasan, K., Fisher, D., 1995. Machine learning approaches to estimating software development effort. IEEE Transaction on Software Engineering 21 (2), 126–136.

Vapnik, V.N., 1998. Statistical Learning Theory. John Wiley, New York, USA.

Verkatachalm, A.R., 1993. Software cost estimation using artificial neural networks. International Joint Conference on Neural Networks, vol. 1. IEEE, Nagoya, pp. 987–990.

Wittig, G., Finnie, G., 1997. Estimating software development effort with connectionist models. Information and Software Technology 39 (7), 469–476.

Wolverton, R.W., 1974. The cost of developing large-scale software. IEEE Transactions on Computers 23 (6), 615–636.

Xu, Z., Khoshgoftaar, T.M., 2004. Identification of fuzzy models of cost estimation. Fuzzy Sets and Systems 145, 141–163.

Zhang, Q., 1997. Using wavelet network in nonparametric estimation. IEEE Transactions on Neural Networks 8 (2), 227–236.

Zhang, Q., Benvniste, A., 1992. Wavelet networks. IEEE Transactions on Neural Networks 3 (6), 889–898.

Zhang, X., Qi, J., Zhang, R., Liu, M., Hu, Z., Xue, H., Fan, B., 2001. Prediction of programmed-temperature retention values of naphthas by wavelet neural networks. Computers and Chemistry 25 (2), 125–133.

Vinay Kumar holds M.Tech (IT) with Banking Technology and Information Security as specialization from University of Hyderabad. He is currently working for Intoto India Pvt. Ltd as Software Engineer. His research interests include software engineering, neural networks, computer networks, data structures and operating systems.

Ravi is working as an Assistant Professor in the Institute for Development and Research in Banking Technology (IDRBT), Hyderabad, since April 2005. He obtained his Ph.D. in the area of Soft Computing from Osmania University, Hyderabad and RWTH Aachen, Germany (2001); MS (Science and Technology) from BITS, Pilani (1991) and M.Sc. (Statistics & Operations Research) from IIT, Bombay (1987). Prior to joining IDRBT, he worked as a Faculty at the Institute of Systems Science (ISS), National University of Singapore (April 2002–March 2005). Before leaving for Singapore, he worked as Assistant Director at the Indian Institute of Chemical Technology (IICT), Hyderabad. He was deputed to RWTH Aachen (Aachen University of Technology) Germany under the DAAD Long Term Fellowship to carry out advanced research during 1997–1999. He is listed as an expert in Soft Computing by TIFAC, Government of India. In a career spanning 19 years, Dr. Ravi has worked in Financial Engineering, Software Engineering, Reliability Engineering, Chemical Engineering, Environmental Engineering, Chemistry, Medical Entomology, Bioinformatics and Geotechnical Engineering. He published more than 45 papers in refereed International / National Journals / Conferences and invited chapters in edited volumes. He edited a Book on "Advances in Banking Technology and Management: Impact of ICT and CRM", published by IGI Global, USA, 2007. Further, he is a referee for 12 International Journals in Computer Science, Operations Research and Economics. Moreover, he is an Editorial board member of *International Journal of Information Systems in the Service Sector (IJISSS), IGI Global, USA, International Journal of Data Analysis Techniques and Strategies (IJDATS), Inderscience Publications, Switzerland, International Journal of Information and Decision Sciences (IJIDS), Inderscience Publications, Switzerland, International Journal of Information Technology Project Management (IJITPM), IGI Global, USA*. Dr. Ravi's research interests include Fuzzy Computing, Neuro Computing, Soft Computing, Data Mining, Global/Multi-Criteria/Combinatorial Optimization, Bankruptcy Prediction, Risk Measurement and Asset Liability Management through Optimization.

Mahil Carr has a bachelor's degree in Mathematics from the American College Madurai, a Master of Computer Applications from St. Joseph's College, Trichy and was awarded a doctoral degree in Information Systems from the City University of Hong Kong. At present, he holds the position as Assistant Professor at the Institute for Development and Research in Banking Technology. His current research interests are in the areas of software engineering and electronic commerce. He has published in several conferences and in Information Technology and Management, the Journal of Services Research, CAB Calling and the Journal of Information System Security. Dr. Carr is on the editorial board of the International Journal of E-Services and Mobile Applications (IJESMA) and the International Journal of Information Systems and Social Change (IJISSC).

Raj Kiran holds M.Tech (IT) with Banking Technology and Information Security as specialization from University of Hyderabad. He is currently working for Motorola company.