IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 4, No 3, July 2010
ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

Design of Radial Basis Function Neural Networks for Software Effort Estimation

Ali Idri[1], Abdelali Zakrani[1] and Azeddine Zahi[2]

[1] Department of Software Engineering, ENSIAS, Mohammed V[th] –Souissi University, BP. 713, Madinat Al Irfane, Rabat, Morocco

[2] Department of Computer Science, FST, Sidi Mohamed Ben Abdellah University, B.P. 2202, Route d'Imouzzer, Fès, Morocco

Abstract

In spite of the several software effort estimation models developed over the last 30 years, providing accurate estimates of the software project under development is still unachievable goal. Therefore, many researchers are working on the development of new models and the improvement of the existing ones using artificial intelligence techniques such as: case-based reasoning, decision trees, genetic algorithms and neural networks. This paper is devoted to the design of Radial Basis Function Networks for software cost estimation. It shows the impact of the RBFN network structure, especially the number of neurons in the hidden layer and the widths of the basis function, on the accuracy of the produced estimates measured by means of MMRE and Pred indicators. The empirical study uses two different software project datasets namely, artificial COCOMO'81 and Tukutuku datasets.

Keywords: *software effort estimation, RBF Neural Networks, COCOMO'81, Tukutuku dataset.*

1. Introduction

As demand for software computer increases continually and the software scope and complexity become higher than ever, the software companies are in real need of accurate estimates of the project under development. Indeed, good software effort estimates are critical to both companies and clients. They can be used for making request for proposals, contract negotiations, planning, monitoring and control [1]. Unfortunately, many software development estimates are quite inaccurate. *Molokken and Jorgensen* report in recent review of estimation studies that software projects expend on average 30-40% more effort than is estimated [2].

In fact, if the project is underestimated, once it seems running out of the schedule, the project manager and his team are put under high pressure to finish the software. As a result, the manager may skimp on verification and validation or quality assurance. For example, developers might scrap unit testing or software integration and test or rush a premature design into production [3]. Consequently, the delivered product may be with poor quality and many underdeveloped functions. On the other side, if the project is overestimated, hence according to Parkinson's Law, the work will expand to fill available time, which leads the company to miss opportunities to take on other projects.

In order to help solving the problems of making some accurate software project predictions and supporting managers in their task, many estimation models have been developed over the last three decades [4], falling into three general categories [5][6]:

Expert judgment: This technique evolves the consultation of one expert in order to derive an estimate for the project based on his experience and available information about the project under development. This technique has been used extensively. However, the estimates are produced in intuitive and non-explicit way. Therefore, it is not repeatable. According to *Gray et al.* [7] although expert judgment is always difficult to quantify, it can be an effective estimate tool to adjust algorithmic models.

Algorithmic models: These are the most popular models in the literature. They are based on mathematical formulae linking effort with effort drivers to produce an estimate of the project. Usually the principal effort driver used in these models is software size (FP, source lines of code…). They need to be calibrated to local circumstances.

Machine learning: The machine learning approaches were appeared at the beginning of nineties to overcome the drawbacks of the two previous categories. Examples of these approaches include regression trees [8] [9], fuzzy logic [1] [10], case based reasoning [6] [11], and artificial neural networks [12] [13] [14] [15].

This paper is concerned with the design of the neural networks approach, especially radial basis function neural

networks, for development effort estimation models. Artificial neural networks are recognized for their ability to produce good results when dealing with problems where there are complex relationships between inputs and outputs, and where the inputs are distorted by high noise levels [14]. The use of Radial Basis Function neural Networks (RBFN) in software effort estimation requires the determination of the structure of these latter and the adjustment of their parameters.

A critical step among the RBF networks configuration is the determination of the optimal structure of the hidden layer. In particular the number of hidden neurons and the formulae used to compute the widths of these kernels functions. In our earlier work [15], we have conducted an empirical study using two different designs of RBFN networks, one time using APC-III clustering algorithm to construct the hidden layer and another time using the well-known algorithm K-means. The results obtained suggest that RBFN with K-means performs better than that with APC-III in terms of accuracy measures MMRE and Pred. In addition, the results showed that the accuracy of RBFN depends also on classification used in the hidden layer. For instance, the classification obtained by minimizing objective function J leads to better estimates than that obtained by maximizing Dunn index D1.

The main goal of the present paper is to experiment two designs of RBF neural networks for software effort estimation. It focuses on the impact of the widths of the Gaussian functions on accuracy of estimates generated by RBFN network.

The remaining of this paper is organized as follows. Section 2 reviews the basic principles of a RBF neural network. Section 3 tackles the description of datasets used to perform our empirical study and evaluation criteria adopted to compare the predictive accuracy of the designed models. Section 4 focuses on the configuration of the proposed RBF network. Section 5 presents and discusses the obtained results, and finally, in Section 6 we provide a number of concluding comments.

2. RBF Neural Networks

The architecture of RBFN Neural Network is quite simple. It involves three different layers. An input layer which consists of sources nodes (cost drivers); a hidden layer in which each neuron computes its output using a radial basis function, which is in general a Gaussian function, and an output layer which builds a linear weighted sum of hidden neuron outputs and supplies the response of the network (effort). A RBF neural network configured for software effort estimation has only one output neuron. So, it implements the output-input relation in Eq. (1) which is indeed a composition of the nonlinear mapping realized by the hidden layer with the linear mapping realized by the output layer

$$F(x) = \sum_{j=1}^{M} \beta_j \phi_j(x) \quad (1)$$

where M is the number of hidden neurons, $x \in \mathbb{R}^p$ is the input, β_j are the output layer weights of the RBFN networks and $\phi(x)$ is Gaussian radial basis function given by:

$$\phi_j(x) = e^{\left(-\frac{\|x-c_j\|^2}{\sigma_j^2}\right)} \quad (2)$$

where $c_j \in \mathbb{R}^p$ and σ_j are the center and the width of j^{th} hidden neuron respectively and $\|.\|$ denotes the Euclidean distance.

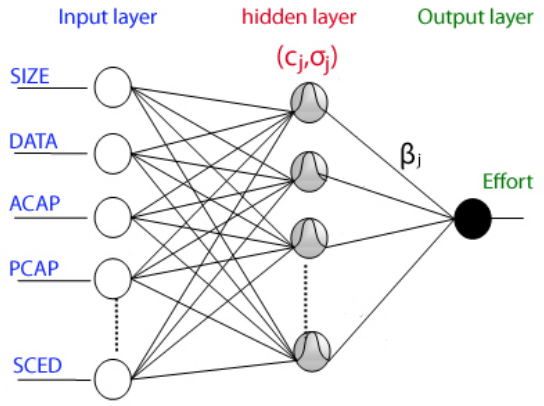

Fig. 1 Radial Basis Function Network architecture for software development effort estimation

3. Data Description and Evaluation Criteria

This section describes the datasets used to perform the empirical study, and presents also the evaluation criteria adopted to compare the estimating capability of the developed models.

3.1 Data Description

The data used in the present study comes from two sources namely, from the COCOMO'81 dataset published by Boehm in his seminal book "software engineering economics" [16] and from Tukutuku dataset.

- **The Artificial COCOMO'81 dataset** is generated artificially from the original one. It contains 252 software projects which are mostly scientific applications developed by Fortran [16]; Each project is described by 13 attributes (see Table 1) : the software size measured in KDSI (Kilo

Delivered Source Instructions) and the remaining 12 attributes are measured on a scale composed of six linguistic values: 'very low', 'low', 'nominal', 'high', 'very high' and 'extra high'. These 12 attributes are related to the software development environment such as the experience of the personnel involved in the software project, the method used in the development and the time and storage constraints imposed on the software.

Table 1: Software attributes for COCOMO'81dataset

Software attributes	*Description*
SIZE	Software Size
DATA	Database Size
TIME	Execution Time Constraint
STOR	Main Storage Constraint
VIRTMIN	Virtual Machine Volatility
VIRT MAJ	Virtual Machine Volatility
TURN	Computer Turnaround
ACAP	Analyst Capability
AEXP	Applications Experience
PCAP	Programmer Capability
VEXP	Virtual Machine Experience
LEXP	Programming Language Experience
SCED	Required Development

▪ **The Tukutuku dataset** contains 53 Web projects [17]. Each Web application is described using 9 numerical attributes such as: the number of html or shtml files used, the number of media files and team experience (see Table 2). However, each project volunteered to the Tukutuku database was initially characterized using more than 9 software attributes, but some of them were grouped together. For example, we grouped together the following three attributes: the number of new Web pages developed by the team, the number of Web pages provided by the customer and the number of Web pages developed by a third party (outsourced) in one attribute reflecting the total number of Web pages in the application (Webpages).

Table 2: Software attributes for Tukutuku dataset

Attributes	*Description*
Teamexp	Average number of years of experience the team has on Web development
Devteam	Number of people who worked on the software project
Webpages	Number of web pages in the application
Textpages	Number of text pages in the application (text page has 600 words)
Img	Number of images in the application
Anim	Number of animations in the application
Audio/video	Number of audio/video files in the application
Tot-high	Number of high effort features in the application
Tot-nhigh	Number of low effort features in the application

3.2 Evaluation Criteria

We employ the following criteria to assess and compare the accuracy of the effort estimation models. A common criterion for the evaluation of effort estimation models is the magnitude of relative error (MRE), which is defined as

$$MRE = \left|\left(\frac{Effort_{actual} - Effort_{estimated}}{Effort_{actual}}\right)\right| \quad (3)$$

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

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i \quad (4)$$

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 the literature and is the proportion of the projects for a given level of accuracy. The definition of Pred(*l*) is given as follows:

$$Pred(l) = \frac{k}{N} \quad (5)$$

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

4. RBF Network Construction

As we have mentioned earlier in this paper, the use of RBF neural network to estimate software development effort requires the determination of the network architecture and its parameters, namely the number of input and hidden neurons, the centers c_j, widths σ_j, and the weights β_j.

The number of the input neurons is, usually and simply, the number of the attributes describing the historical

software projects in the used dataset. Therefore, when applying RBF network to COCOMO'81, the number of input neurons is equal to 9 and in the case of Tukutuku dataset, the number of input neurons is equal to 13. Concerning the number of hidden neurons, C, is determined using clustering algorithms. The following subsection deals with the initialization of the centers of hidden layer by means of k-means algorithm.

4.1 Construction of the Hidden Layer of RBFN

The construction of hidden layer of the proposed network is achieved using clustering techniques. The role of clustering in the design of RBFN is to set up an initial distribution of receptive fields (hidden neurons) across the input space of the input variables (software attributes). In literature, clustering techniques have been successfully used in the configuration of the hidden layer of RBFN [18] [19].

K-means clustering algorithm, developed by MacQueen in 1967 [20], is a popular clustering technique which is used in numerous applications. It is a multi-pass clustering algorithm. The k-means algorithm partitions a collection of N vectors into c clusters C_i, i=1,..,*c*. The aim is to find cluster centers (centroids) by minimizing a dissimilarity (or distance) function which is given in below.

$$J = \sum_{i=1}^{C} \sum_{x_k \in C_j} d\,(x_k, c_i) \qquad (6)$$

where c_i is the center of cluster C_i; $d(x_k, c_i)$ is the distance between i^{th} center (c_i) and k^{th} data point (x_k). For simplicity, the Euclidean distance is used as dissimilarity measure and the overall dissimilarity function is expressed as follows.

$$J = \sum_{i=1}^{C} \sum_{x_k \in C_j} \|x_k - c_i\|^2 \qquad (7)$$

The outline of the k-means algorithm can be stated as follows:

1- Define the number of the desired clusters, *C*.
2- Initialize the centers C_i , i=1..C. This is typically achieved by randomly selecting *c* points from among all of the data points.
3- Compute the Euclidean distance between x_i and c_i, *j=1..N* and *i*=1..*c*
4- Assign each x_i to the most closer cluster C_i
5- Recalculate the centers c_i
6- Compute the objective function *J* given in Eq. (7). Stop if its improvement over previous iteration is below a threshold.
7- Iterate from step 3.

The above k-means based initialization process produces a set of centers (c_i), which are the kernels of Gaussian basis function.

4.2 Computation of the widths

One of the most important aspects to be addressed is the determination of the widths of unit function and its inherent complexity [21]. These parameters control the amount of overlapping of kernels functions as well as the network generalization. Small values lead a rapidly decreasing function while a larger values result in more varying function. Hence, it is clear that the setting of the widths (σ_j) is critical step to the RBF network generalization. To address this issue, many heuristics and techniques based on solid mathematical concepts have been proposed in the literature [21] [22] [23]. The main idea is to determine (σ_j) in order to cover the input space as uniformly as possible [19]. However, covering the historical software project space uniformly implies that the RBFN will be able to generate an estimate for a new project even though it is not similar to any historical project. In such a situation, we prefer that the RBFN does not provide any estimate than one that may easily lead to wrong managerial decisions and project failure. In [24] and [25], we have adopted a simple strategy based primarily on assigning one value to all (σ_j). In the present paper we are interested in studying the impact of the widths on the accuracy of RBFN. The empirical study uses the following formulae:

$$\sigma_i(max) = \begin{cases} \max\limits_{x_j \in C_i} d(x_j, c_i), & if\ card(C_k) > 1 \\ \max\limits_{k/card(C_k)>1} \sigma_k, & if\ card(C_k) = 1 \end{cases} \qquad (8)$$

$$\sigma_i(min) = \begin{cases} \max\limits_{x_j \in C_i} d(x_j, c_i), & if\ card(C_k) > 1 \\ \min\limits_{k/card(C_k)>1} \sigma_k, & if\ card(C_k) = 1 \end{cases} \qquad (9)$$

where $\mathrm{Card}(\mathrm{C_k})$ is the cardinality of cluster C_k.

4.3 Computation of weights

Provided that the centers c_i and the widths σ_j of the basis functions have been determined, the weights of the output layer can be calculated and adjusted using a well-known supervised learning algorithm *Backpropagation*, which based on following formula:

$$\Delta\beta_j = \eta(effort_{estimated} - effort_{actual})\phi_j \qquad (10)$$

where η is the learning rate and ϕ_j is the output of the j^{th} basis function of the hidden layer.

5. Overview of Empirical Results

The following section presents and discusses the results obtained when applying the RBFN to the artificial COCOMO'81 and Tukutuku datasets. The calculations were made using two software prototypes developed with C language under a Microsoft Windows PC environment. The first software prototype implements the k-means clustering algorithm, providing the clusters ($C\ clusters$) and their centers (c_j) from the used dataset. The second software prototype implements a cost estimation model based on an RBFN architecture in which the hidden layer parameters are determined by means of the first software prototype.

The two datasets used to carry out the empirical study were prepared and analyzed. For example all numeric types of effort drivers were normalized within 0 to 1 in both datasets in order to have the same degree of influence. Afterwards, we conducted several experiments using different configurations of RBFN. These experiments use the full dataset for training and testing.

5.1 COCOMO'81 dataset

Our first experiment is performed using COCOMO'81 dataset containing 252 historical software projects. Two models of RBF networks were designed to examine the impact of basis function widths on accuracy of estimates produced by the network. The first RBFN effort estimation model uses the formula of widths given in Eq. (8) and the second model uses the formula of widths given in Eq. (9).

$$\textbf{\textit{Model 1}}: \quad RBFN(\sigma_i(max), C)$$
$$\textbf{\textit{Model 2}}: \quad RBFN(\sigma_i(min), C)$$

From each model, different configurations have been obtained by varying the number of hidden neurons *C*. the aim was to determine which widths formula improve the estimates. So, we have trained and tested the above models using COCOMO'81 dataset. The learning rate was set to 0.03 for the all experiments in this paper.

Fig. 2 shows and compares the results of the two models. From this figure we note that the RBFN models using $\sigma_i(min)$ generate a lower MMRE than that using $\sigma_i(max)$ for each number of hidden neurons. For example, for C=120 the *model 1* made a higher prediction error (MMRE=60.81) than the *Model 2* (MMRE=34.96). Then, in terms of MMRE, *Model 2* performs better than *Model 1*.

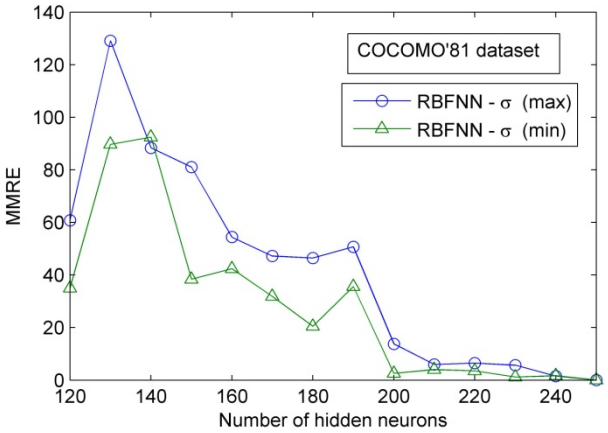

Fig. 2 Relationship between the accuracy of RBFN (MMRE) and used classification k-means for COCOMO'81 dataset.

Fig. 3 compares the accuracy of the two above models, in terms of Pred(0.25), when varying the number of hidden neurons. As can be seen, the accuracy of RBF networks using $\sigma_i(min)$ performs much better than RBF Networks using $\sigma_i(max)$. Indeed, the *Model 1* produced poor estimates even when increasing the number of hidden neurons until 160. Whereas *Model 2* generates acceptable effort estimates with a number of hidden neurons C=120.

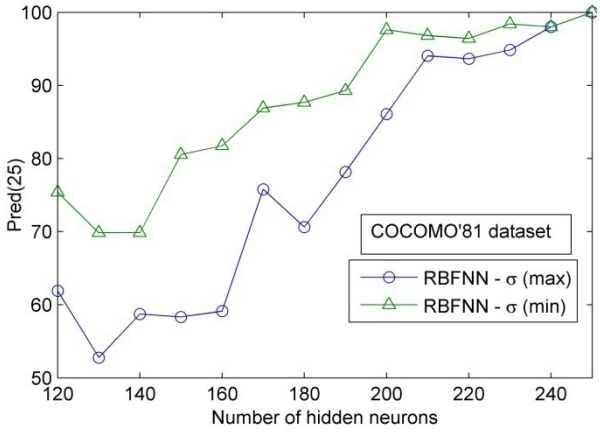

Fig. 3 Relationship between the accuracy of RBFN (Pred) and the used formula of σ according to the number of hidden neurons.

Table 3 shows the results obtained using different configurations of RBF networks.

Table 3: MMRE and Pred results of different RBFN configurations for COCOMO'81 dataset

Number of hidden neurons	*Accuracy of RBFN using $\sigma_i(max)$ Eq. 8*		*Accuracy of RBFN using $\sigma_i(min)$ Eq. 9*	
	MMRE	*Pred(0.25)*	*MMRE*	*Pred(0.25)*
120	60.81	61.9	34.96	75.4
130	129.13	52.78	89.69	69.84
140	88.35	58.73	92.46	69.84
150	81.08	58.33	38.34	80.56
160	54.41	59.13	42.38	81.75
170	47.2	75.79	31.81	86.9
180	46.47	70.63	20.48	87.7
190	50.74	78.17	35.61	89.29
200	13.78	86.09	2.59	97.62
210	5.99	94.05	4.03	96.83
220	6.49	93.65	3.56	96.43
230	5.68	94.84	1.21	98.41
240	1.55	98.02	1.66	98.02
250	0.02	100	0.01	100

5.1 Tukutuku dataset

In [22], the authors state that the problem of fixing width of Gaussian kernels is not evident since it is largely data-dependent and depends also on the dimension of input space. In this regard, we have replicated the previous empirical study using Tukutuku dataset to verify how much the Eq. (9) improves the accuracy of RBF Network in estimating software effort.

So, we have conducted several experiments, one time using RBFN with $\sigma_i(max)$ and another time using RBFN with $\sigma_i(min)$. In addition, for each RBFN design we varied the number of hidden neurons from 32 to 52. Fig. 4 and Fig. 5 show the accuracy of both models (1 and 2), measured in terms of MMRE and Pred, on Tukutuku dataset. These figures confirm the superiority of the RBFN network that uses Eq. (9) to compute Gaussian kernels widths over that one using Eq. (8).

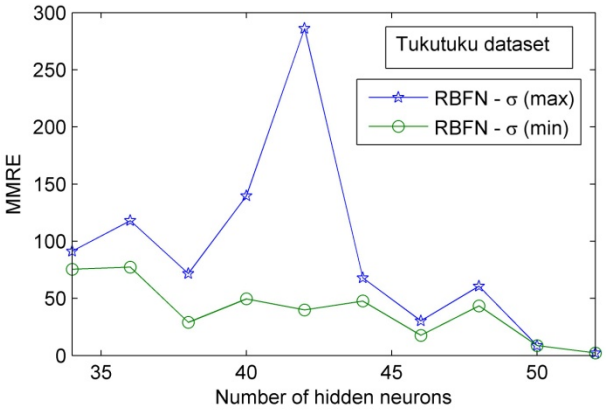

Fig. 4 Relationship between the accuracy of RBFN (MMRE) and the used formula of σ according to the number of hidden neurons.

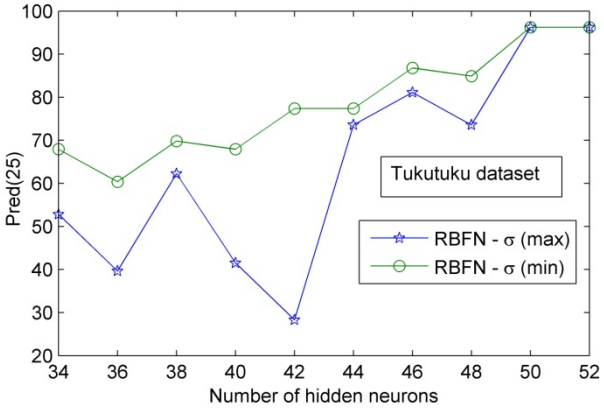

Fig. 5 Relationship between the accuracy of RBFN (Pred) and the used formula of σ according to the number of hidden neurons.

The following table (4) summarizes the results produced by RBFN network using the above formulae to calculate the Gaussian kernels widths.

Table 4: MMRE and Pred results of different RBFN configurations for Tukutuku dataset

Number of hidden neurons	*Accuracy of RBFN using $\sigma_i(max)$ Eq. 1*		*Accuracy of RBFN using $\sigma_i(min)$ Eq. 2*	
	MMRE	*Pred(0.25)*	*MMRE*	*Pred(0.25)*
34	91.24	52.83	75.46	67.92
36	117.99	39.62	77.3	60.38
38	71.65	62.26	28.96	69.81
40	139.67	41.51	49.56	67.92
42	286.16	28.3	39.91	77.36
44	67.84	73.58	47.69	77.36
46	30.35	81.13	17.64	86.79
48	60.63	73.58	43.47	84.91
50	8.68	96.23	8.67	96.23
52	2.23	96.23	2.23	96.23

6. Conclusion

The aim of this work was to examine the impact of the radial basis widths on the accuracy of RBF Network in the context of software effort estimation. Some empirical studies in the literature have shown that in many situations, a bad choice of the widths can easily lead to poor approximation ability.

In this paper, we have designed two RBF networks for software effort estimation. Each one of these networks used a different formula to compute the widths of the Gaussian kernels. These RBFN models were trained and tested using two software projects datasets. The results show that the use of an adequate formula of width which controls the overlap between Gaussian kernels, according to the number of projects placed in the region covered by centers, improves greatly the estimates generated by RBFN network model.

References

[1] M. W. Nisar, and Y.-J. Wang, and M. Elahi, "Software Development Effort Estimation Using Fuzzy Logic - A Survey", in 5th International Conference on Fuzzy Systems and Knowledge Discovery, 2008, pp. 421-427.

[2] K. Molokken, and M. Jorgensen, "A Review of Surveys on Software Effort Estimation", in International Symposium on Empirical Software Engineering, 2003, pp. 223-231.

[3] T. Menzies, and J. Hihn, "Evidence-Based Cost Estimation for Better-Quality Software", IEEE Software, Vol. 23, No. 4, 2006, pp. 64-66.

[4] M. Jorgensen, and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies", IEEE Transactions on Software Engineering, Vol. 33, No. 1, 2007, pp. 33-53.

[5] M. J. Shepperd, and C. Schofield, and B. Kitchenham, "Effort Estimation Using Analogy", in International Conference on Software Engineering, 1996, pp. 170-178.

[6] E. Mendes, and I. D. Watson, and C. Triggs, and S. Counsell, "A Comparative Study of Cost Estimation Models for Web Hypermedia Applications", Empirical Software Engineering, Vol. 8, No. 2, 2003, pp. 163-196.

[7] A. R. Gray, and S. MacDonell, and M. Shepperd, "Factors Systematically Associated with Errors in Subjective Estimates of Software Development Effort: The Stability of Expert Judgment", in 6th IEEE International Software Metrics Symposium, 1999, pp. 216-227.

[8] R. W. Selby, and A.A. Porter, "Learning from examples: generation and evaluation of decision trees for software resource analysis", IEEE Transactions on Software Engineering, Vol. 14, No. 12, 1988, pp. 1743-1757.

[9] A. S. Andreou, and E. Papatheocharous, "Software Cost Estimation using Fuzzy Decision Trees", in 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, pp. 371-374.

[10] V. Sharma, and H. K. Verma, "Optimized Fuzzy Logic Based Framework for Effort Estimation in Software Development", Computer Science Issues, Vol. 7, Issue 2, No. 2, 2010, pp. 30-38.

[11] A. Idri, and T. M. Khoshgoftaar, and A. Abran, "Investigating Soft Computing in Case-based Reasoning for Software Cost Estimation", Engineering Intelligent Systems for Electrical Engineering and Communications, Vol. 10, No. 3, 2002, pp. 147-157.

[12] K. Srinivasan, and D. Fisher, "Machine Learning Approaches to Estimating Software Development effort", IEEE Transactions on Software Engineering, Vol. 21, No. 2, 1995, pp. 126-137.

[13] G. R. Finnie, and G. Witting, and J.-M. Desharnais, "A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models", Systems and Software, Vol. 39, No. 3, 1997, pp. 281-289.

[14] G. R. Finnie, and G. Witting, "AI Tools for Software Development Effort", In International Conference on Software Engineering: Education and Practice, 1996, pp. 346-289.

[15] A. Idri, and A. Abran, and S. Mbarki, "An Experiment on the Design of Radial Basis Function Neural Networks for Software Cost Estimation", in 2nd IEEE International Conference on Information and Communication Technologies: from Theory to Applications, 2006, Vol. 1, pp. 230-235.

[16] B.W. Boehm, Software Engineering Economics, Place: Prentice-Hall, 1981.

[17] B.A. Kitchenham, and E. Mendes, "A Comparison of Cross-company and Within-company Effort Estimation Models for Web Applications", In 8th International Conference on Empirical Assessment in Software Engineering, 2004, pp. 47-56.

[18] W. Pedrycz, "Conditional Fuzzy Clustering in the Design of Radial Basis Function Neural Networks", IEEE Transactions on Neural Networks, Vol. 9, No. 4, 1998, pp. 601-612.

[19] Y.-S. Hwang, and S.-Y. Bang, "An Efficient Method to Construct a Radial Basis Function Network Classifier", Neural Networks, Vol. 10, No. 8, 1997, pp. 1495-1503.

[20] J. B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations", in Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281-297.

[21] F. Ros, and M. Pintore, and J. R. Chretien, "Automatic design of growing radial basis function neural networks based on neighboorhood concepts", Chemometrics and Intelligent Laboratory Systems, Vol. 81, Issue 2, 2007, pp. 231-240.

[22] N. Benoudjit, and M. Verleysen, "On the Kernel Widths in Radial-Basis Function Networks", Neural Processing Letters, Vol. 18, No. 2, 2003, pp. 139-154.

[23] F. Schwenker, and C. Dietrich, "Initialisation of Radial Basis Function Networks Using Classification Trees", Neural Networks World, Vol. 10, 2000, pp. 476-482.

[24] A. Idri, and A. Zahi, and E. Mendes, and A. Zakrani, "Software Cost Estimation Models Using Radial Basis Function Neural Networks", in International Conference on Software Process and Product Measurement, 2007, pp. 21-31.

[25] A. Idri, and A. Zakrani, and A. Abran, "Functional Equivalence between Radial Basis Function Neural Networks and Fuzzy Analogy in Software Cost Estimation", in 3rd IEEE International Conference on Information and Communication Technologies: from Theory to Applications, 2008, pp. 1-5.

A. Idri is a Professor at Computer Science and Systems Analysis School (ENSIAS, Rabat, Morocco). He received DEA (Master) (1994) and Doctorate of 3rd Cycle (1997) degrees in Computer Science, both from the University Mohamed V of Rabat. He has received his Ph.D. (2003) in Cognitive Computer Sciences from ETS, University of Quebec at Montreal. His research interests include software cost estimation, software metrics, fuzzy logic, neural networks, genetic algorithms and information sciences

A. Zakrani received the B.Sc. degree in Computer Science from Hassan II University, Casablanca, Morocco, in 2003, and his DESA degree (M.Sc.) in the same major from University Mohammed V, Rabat, in 2005. Currently, he is preparing his Ph.D. in computer science in ENSIAS. His research interests include software cost estimation, software metrics, fuzzy logic, neural networks, decision trees.

A. Zahi is an assistant professor at faculty of Sciences and techniques (FST, Fès, Morocco). He received his Doctorate of 3rd Cycle (1997) degree in Computer Science from Mohamed V University in Rabat. His research interests include software cost estimation, software measurement, fuzzy analogy, neural networks.