Arabian Journal for Science and Engineering (2020) 45:2605–2618
https://doi.org/10.1007/s13369-019-04250-6

RESEARCH ARTICLE - SPECIAL ISSUE - INTELLIGENT COMPUTING AND INTERDISCIPLINARY APPLICATIONS

A Comparative Analysis on Effort Estimation for Agile and Non-agile Software Projects Using DBN-ALO

Anupama Kaushik[1,2] · Devendra Kr. Tayal[3] · Kalpana Yadav[4]

Received: 12 June 2019 / Accepted: 11 November 2019 / Published online: 19 November 2019

Abstract
At present, in the software industry, agile and non-agile software development approaches are followed and effort estimation is an intrinsic part of both the approaches. This work investigates the application of deep belief network (DBN) along with antlion optimization (ALO) technique for effort prediction in both agile as well as non-agile software development environment. The study also provides a prediction interval of effort to handle uncertainty in estimation. This will help the project managers to estimate the effort in ranges instead of a crisp value. The proposed DBN-ALO approach is applied on four promise repository datasets for traditional software development (non-agile), and on three agile datasets. It provides the best results in all the evaluation criteria used. The proposed approach is also statistically validated using nonparametric tests, and it is found that DBN-ALO worked best for both agile and non-agile development approaches.

Keywords Software development effort · Deep belief network · Antlion optimization · Agile software development · Non-agile software development

1 Introduction

Software effort estimation is the requirement of all the software development firms. The project managers cannot evade this activity, and an accurate estimation is the need of the hour for software development. The accurate estimations will lead the project to success, and inaccurate estimations will result in its failure. There are many effort estimation approaches available in the literature for non-agile software development. These approaches are based on neural networks, fuzzy logic [1–3], and various other soft computing techniques [4, 5]. Recently, nature-inspired algorithms are getting attention from the researchers which comprises of firefly algorithm (FA), cuckoo search (CS), bat algorithm (BA), particle swarm optimization (PSO), etc. In software estimation studies also, these algorithms have been used by many researchers [6–9]. These algorithms are simpler to understand, implement, and also require less operators than evolutionary algorithms. Most importantly, when these algorithms are integrated with machine learning techniques, they increase the strength of the machine learning technique.

Nowadays, the software development paradigm is shifting toward agile software development, which was incepted in 2001. These agile methodologies have advantages such as flexibility, respond to changes even in later stages of development, customer friendly, and delivers the working software rapidly [10]. The effort prediction aspect of agile software development is slow in its evolution in comparison with non-agile software development, and very fewer studies exist based on various machine learning approaches [11, 12]. This can be due to the lack of publicly available agile data in comparison with non-agile data, and moreover,

✉ Anupama Kaushik
thisisanupama@gmail.com

Devendra Kr. Tayal
dev_tayal2001@yahoo.com

Kalpana Yadav
kyadav11@yahoo.com

1 Department of IT, Maharaja Surajmal Institute of Technology, New Delhi, India

2 IGDTUW (Indira Gandhi Delhi Technical University for Women), Delhi, India

3 Department of Computer Science, IGDTUW (Indira Gandhi Delhi Technical University for Women), Delhi, India

4 Department of IT, IGDTUW (Indira Gandhi Delhi Technical University for Women), Delhi, India

research in agile is more of a conceptual nature rather than model-driven.

This study provides a model-driven approach of effort estimation for agile and non-agile projects. An accurate estimation is often necessary for agile and non-agile software projects. The model used for effort prediction in both the development approaches is DBN-ALO. Deep belief network (DBN) is a powerful category of deep neural networks. Nowadays, it is extensively used in various applications due to its unique learning capability [13]. The meta-heuristic techniques are primarily used for obtaining optimal solutions. ALO [14] is a meta-heuristic technique based on the hunting nature of antlions. It is chosen over other optimization techniques due to its unique characteristics of exploration and exploitation of the search space. As the algorithm is population-based, it has very few parameters to be adjusted, and local optimum avoidance is also high. To the best of our knowledge, none of the studies have reported DBN-ALO for effort estimation in both traditional and agile software development environments so far.

There are several effort estimation models present, but all these models do not provide 100% accurate results. There are some uncertainty and inaccuracy associated with these estimates. This estimation uncertainty is handled by providing a range of estimation into which the actual effort will presumably fall [15]. This range of estimation is known as prediction interval (PI) proposed by Jorgensen and Sjoberg [16]. Our study also computes a prediction interval (PI) of effort for software projects, which will help the software managers to predict the interval of effort estimate rather than single numeric effort value for a project.

The remaining paper is organized as follows. Section 2 discusses the related work on effort estimation for both agile and non-agile environment. Section 3 briefly describes the background techniques used in the study. Section 4 describes the proposed work. Section 5 discusses the experimental setup and results. Section 6 provides the statistical analysis. Section 7 discusses the threats to validity, and Sect. 8 concludes the paper.

2 Related Work

This section reviews the effort estimation work done in non-agile and agile software development environment in the past few years, which comprises of different techniques and methodologies.

In the non-agile software development environment, the effort estimation studies are broadly classified into parametric and nonparametric studies. The parametric studies are based on mathematical equations, and nonparametric studies use various machine learning approaches. The nonparametric studies are preferred over parametric studies as they are more effective and robust in their results. They can easily model the complex relationships among the contributing factors and provide good reasoning capabilities. Though there are many nonparametric studies available in the literature, we have given only a few of them in brief.

Rijwani and Jain [1] used backpropagation feed-forward neural network for tuning the COCOMO (Constructive Cost Model) parameters. Laqrichi et al. [2] introduced uncertainty in effort estimation using neural networks based on bootstrap technique. They evaluated their technique on ISBSG (International Software Benchmarking Standards Group) dataset, and it provided more realistic effort. Nassif et al. [3] compared Mamdani and Sugeno with constant output, and Sugeno with linear output fuzzy models, and performed regression analysis. They evaluated these models using standard accuracy measures used in effort estimation on ISBSG dataset, and concluded that Sugeno fuzzy inference system with linear output performed best in comparison with other models. Zare et al. [4] proposed a methodology for effort estimation with three levels of Bayesian network, and used fuzzy numbers to depict the interval of all the nodes of the network. They have also used evolutionary algorithms, and their proposed model provided more accurate results than the other models discussed in their work.

Sehra et al. [5] contributed a hybrid model for effort estimation using multi-criteria decision making (MCDM) approach and machine learning algorithm. Their model gave accurate results than bee colony optimization and basic radial basis function (RBF) kernel-based model. Kaushik et al. [6] provided a hybrid effort estimation approach by integrating cuckoo optimization algorithm and the fuzzy inference system. They evaluated their technique on promise datasets for effort estimation, and found their results at par than the other techniques discussed in their work. Kaushik et al. [7] integrated firefly algorithm with radial basis function network and functional link artificial neural network, and predicted software cost. They found their proposed methodology worked best, and used statistical tests to prove that. Sivanageswara Rao et al. [8] proposed multiobjective particle swarm optimization for tuning the COCOMO-based projects, and found their model was better than the COCOMO model. Venkataiah et al. [9] provided ant colony optimization model for software cost prediction, and evaluated their model on three datasets.

Abdelali et al. [17] provided a technique of effort estimation using random forest. They provided this empirical approach by varying its key parameters, and validated their approach using ISBSG Release 8, Tukutuku, and COCOMO datasets. Pai et al. [18] proposed a software effort estimation approach using neural network ensemble and regression analysis. They used 163 software development projects to validate the approach, and found that the project size is the main element in determining the effort of a project. They also observed that neural networks ensemble technique outperforms the regression analysis.

Benala and Mall [19] reported the use of differential evolution for effort estimation in analogy-based software development, and compared their approach with many existing approaches using datasets from promise repository. Ezghari and Zahi [20] proposed Consistent Fuzzy Analogy-based Software Effort Estimation (C-FASEE) model, which overcame the drawbacks of Fuzzy Analogy-based Software Effort Estimation (FASEE). Their model was validated on 13 project datasets, and it provided good estimation accuracy than the other models used for comparison. Abdelali et al. [21] proposed ensemble approach on optimal trees for effort estimation, and found their proposed approach worked well in comparison with random forest and regression trees models. Nguyen et al. [22] proposed a fixed window-based effort estimation model to calibrate COCOMO project data and improved the estimation accuracy of software projects.

In agile software development, the story point estimation is the commonly used effort estimation technique for real-time agile projects, and most of the studies are based on it.

Satapathy and Rath [11] presented a study to improve the story point approach of effort estimation used in agile development environment with different machine learning approaches, and compared the performance of these techniques with the other techniques existing in the literature. Panda et al. [12] contributed to agile effort estimation based on story point approach using different types of neural networks. They also compared the results of these models, and assessed them using standard effort evaluation criteria. Ziauddin et al. [23] provided an effort estimation model for the agile projects based on story point approach, and validated their model using data collected from 21 agile projects. Martínez et al. [24] provided a Bayesian network technique to model the complexity and importance of user stories using planning poker in scrum projects. Their model was based on the data provided by the students and professionals, and found that their model gave better estimates than the traditional planning poker.

Dragicevic et al. [25] proposed a Bayesian model for effort prediction in agile development environment, which can be used during the planning stage of the project development. They evaluated their model on the completed agile projects taken from the single software company, and used standard accuracy measures to assess the model. Tanveer [26] proposed a hybrid methodology of effort estimation for agile projects, which aimed at changing impact analysis for software artifacts. It also included the cost drivers as suggested by the experts for improving the effort estimation. Tanveer et al. [27] contributed to effort estimation of agile projects from agile development team perspective, and presented a case study on three agile development teams of a German multinational software corporation. Bilgaiyan et al. [28] proposed effort estimation for 21 agile projects from six different software houses using feed-forward backpropagation ANN and Elman ANN. Their model was evaluated with three commonly used performance matrices for effort estimation. Britto et al. [29] presented a study on effort estimation of agile global software development by collaborating the results present in the literature survey of effort estimation and global software development context.

Usman et al. [30] investigated distributed large-scale agile projects to explore the effort estimation process, and identified various elements affecting their accuracy. Satapathy et al. [31] optimized the story point approach using support vector regression (SVR) kernel methods, and found that support vector regression with RBF kernel (SVR-RBF) provided the best results for agile projects. Tung and Hanh [32] proposed an integration of artificial bee colony (ABC) and particle swarm optimization (PSO) for effort estimation in agile software development environment. They used velocity and the story points as the main inputs, and found that their technique outperformed the earlier existing studies. Zakrani et al. [33] proposed an improved effort estimation model for agile projects using support vector regression (SVR) optimized by grid search method, and found their model outperformed the SVR kernel methods.

All the techniques available on effort estimation in the literature for agile and non-agile projects are distinct, contributed their best, and have their own merits and demerits. However, there is no standard technique that is commonly accepted for effort estimation. The recent studies in non-agile context used machine learning and various other soft computing approaches for effort prediction, whereas in agile context the studies lack in using the above methods.

The current study is an attempt toward providing a model-driven effort estimation approach for agile and non-agile projects. In order to improve the accuracy of predictions, data from real agile and non-agile projects are used. The novelty here is the integration of DBN and ALO, which is not available in the literature for both agile and non-agile contexts.

3 Background Techniques

This section provides some background concepts used in constructing the proposed framework.

3.1 Deep Belief Network (DBN)

DBN belongs to a set of deep neural networks. The heart of DBN is restricted Boltzmann machine (RBM). All the layers of DBN are placed upon each other as a stack of RBMs. The RBM is a two-layer neural network with a visible layer and a hidden layer as shown in Fig. 1. All the

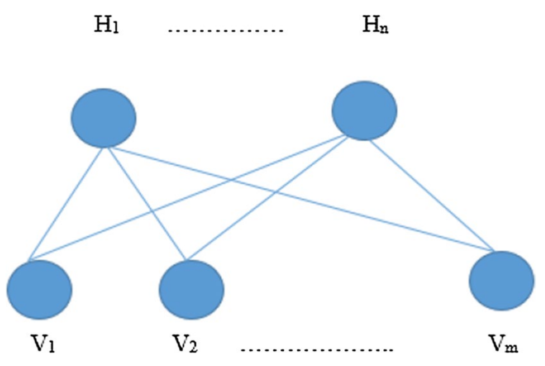


Fig. 1 Structure of RBM

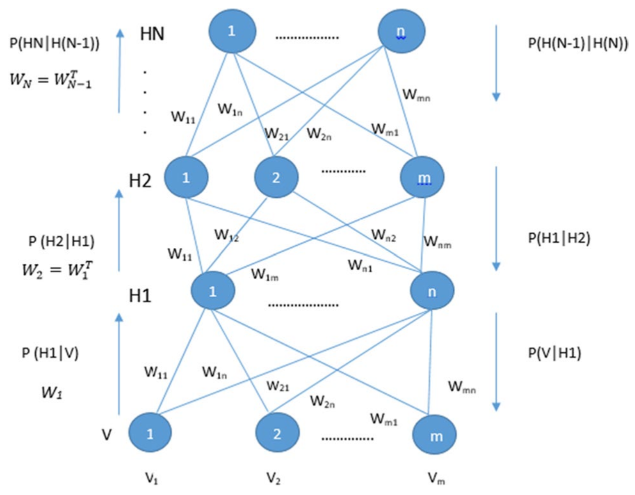


Fig. 2 Structure of DBN

visible layer neurons are connected to the hidden layer neurons, but the nodes of the same layer are not linked to each other. For learning, the inputs are mapped to the visible layer, where they are passed to hidden layer after multiplying with their respective weights, and passing through an activation function producing one output per hidden node.

The structure of DBN is shown in Fig. 2. In DBN, the outputs of hidden layer 1 are passed as input to hidden layer 2, until a final classifying layer is reached. DBN uses a greedy training approach and contrastive divergence method to train the stack of RBMs. Since there is no intra-layer connection in RBM, and also it has the shape of bipartite graph, the hidden neurons are mutually independent given the visible neurons, and vice versa. Given, m visible neurons and n hidden neurons, the conditional probability of hidden neurons H given visible neurons V, is $P\ (H = 1|H)$. This is known as positive phase. Conversely, the conditional probability of V given H, is $P\ (V = 1|H)$. This is known as negative phase.

3.1.1 Training of RBM

1. Map the training dataset to the neurons of the visible layer.
2. *Positive Phase* In this phase, all the hidden neurons are updated in parallel. Compute the positive statistics for edge E_{ij} which is $P\ (H_j = 1|V)$, and can be given as:

$$\text{Positive}\left(E_{ij}\right): \quad P\left(H_j = 1|V\right) = \sigma\left(B_j + \sum_{i=1}^{m} W_{ij}V_i\right) \tag{1}$$

   Here, B_j is the bias associated with the hidden neuron H_j, W_{ij} is the weight associated with the hidden neuron H_j and visible neuron V_i, and σ represents the sigmoid function.
3. *Negative Phase* This phase reconstructs all the visible units. Compute the negative statistics for edge E_{ij} which is $P\ (V_i = 1|H)$, and can be given as:

$$\text{Negative}\left(E_{ij}\right): \quad P\left(V_i = 1|H\right) = \sigma\left(A_i + \sum_{j=1}^{n} W_{ij}H_j\right) \tag{2}$$

   Here, A_i is the bias associated with the visible neuron V_i, W_{ij} is the weight associated with the hidden neuron H_j and visible neuron V_i, and σ represents the sigmoid function.
4. *Update the weights* The previous weight W_{ij} is updated as:

$$\text{Updt}\left(W_{ij}\right) = W_{ij} + L * \left(\text{Positive}\left(E_{ij}\right) - \text{Negative}\left(E_{ij}\right)\right) \tag{3}$$

   where L is the learning weight.
5. Transpose the weights and repeat with all the training examples till the required threshold is achieved.

3.2 Antlion Optimization Algorithm

The antlion optimization algorithm (ALO) was proposed by Mirjalili [14]. It is a meta-heuristic technique based on the hunting mechanism of antlions. In this algorithm, antlions catch the ants by building cone-shaped traps. Once the ants enter the trap, the antlions consume these ants, but many times the ants also try to evade the confinement. For such case, the antlions throw the soil toward the outer edge of the pit, so that the ants trying to escape slide down.

3.2.1 Random Walk of Ants

The location of ants and antlions is stored in two different matrices, and each of them is evaluated using a fitness function. The ants move randomly in the search space which is modeled as:

$$Z(s) = \left[0,\ \mathrm{cs}\left(2y\left(s_1\right) - 1\right),\ \mathrm{cs}\left(2y\left(s_2\right) - 1\right), \ldots,\ \mathrm{cs}\left(2y\left(s_n\right) - 1\right)\right] \tag{4}$$

where cs is the collective sum, n is the maximal number of iterations, s is the step of random walk, and $y(s)$ is defined as:

$$y(s) = \begin{cases} 1 \text{ if } & \mathrm{rno} > 0.5 \\ 0 \text{ if } & \mathrm{rno} \leq 0.5 \end{cases} \tag{5}$$

where rno is the random number in the interval [0 1].

This random walk is restricted within the bounds of the search space which is done using min–max normalization equation given as follows.

$$Z_i^s = \frac{\left(Z_i^s - e_i\right) \times \left(h_i - g_i^s\right)}{\left(h_i^s - e_i\right)} + g_i \tag{6}$$

where e_i provides the minimal value of random walk, g_i^s provides the minimal of the ith variable at the sth iteration, and h_i^s provides the maximal of the ith variable at the sth iteration.

3.2.2 Entrapping in Pits

The random walk of ants in the search space is affected by the antlion's trap which is modeled using the equation:

$$g_i^s = \mathrm{Ant}\ L_j^s + g^s \tag{7}$$

$$h_i^s = \mathrm{Ant}\ L_j^s + h^s \tag{8}$$

where g^s is the minimal value at the sth iteration, h^s is the maximal value at the sth iteration, g_i^s provides the minimal value for the ith ant, h_i^s provides the maximal value for the ith ant, and Ant L_j^s is the location of the chosen jth antlion at the sth iteration.

The ants walk around a selected antlion randomly in a hyperspace as given by Eqs. (7) and (8).

3.2.3 Building Trap and Elitism

The antlions build the traps to catch the ants, but only the fitter antlions can catch the prey. This hunting capability of antlions is modeled by roulette wheel operator. The fittest antlion is obtained in every iteration, and termed as elite. The ants walk around the elite and the selected antlion by the roulette wheel which is given by:

$$\mathrm{Ant}\ _i^s = \frac{Z_A^s + Z_E^s}{2} \tag{9}$$

where Z_A^s is the random walk about the antlion at the sth iteration, Z_E^s is the random walk surrounding the elite at the sth iteration, and Ant_i^s shows the location of the ith ant at the sth iteration.

3.2.4 Sliding Ants Toward Antlion

The ants trying to evade the trap are controlled by antlions by throwing soil outward so that the ants slide inside the trap. This is modeled using the equations:

$$g^s = \frac{g^s}{L} \tag{10}$$

$$h^s = \frac{h^s}{L} \tag{11}$$

where g^s provides the minimal value at the sth iteration, h^s provides the maximal value at the sth iteration, and L is a ratio given by:

$$L = 10^W \frac{s}{T} \tag{12}$$

where s is the current iteration, T is the maximal number of iterations, and w is a constant that regulates the accuracy of exploitation.

3.2.5 Capturing Prey and Rebuilding the Pit

The ants are caught and consumed by the antlions when they reach the bottom of the pit. This behavior of antlions moves them to the current position of the hunted ant. It also enhances their chance of catching a new prey, and is modeled by:

$$\mathrm{Ant}\ L_j^s = \mathrm{Ant}_i^s \quad \text{if}\, f\left(\mathrm{Ant}_i^s > \mathrm{Ant}L_j^s\right) \tag{13}$$

where s is the current iteration, Ant L_j^s is the location of the chosen jth antlion at the sth iteration, and Ant_i^s is the location of the ith ant at the sth iteration.

The ALO algorithm is represented using a flow diagram in Fig. 3.

3.3 Story Point Approach

In an agile development approach, the project teams develop user stories of a project. A user story is the high-level description of the requirement which help the developers to estimate the effort to implement them. The user stories are assigned story points, which is a metric to estimate the effort of implementing a user story. The proposed work uses the user story estimation approach as suggested by Ziauddin et al. [23]. It assigns the story point to a user story based upon its size and

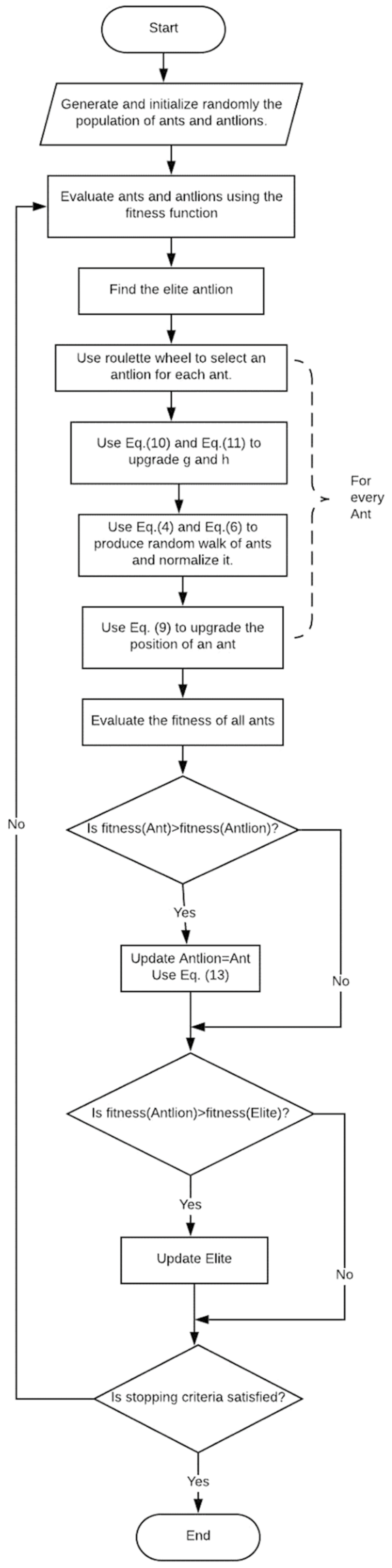

Start
Generate and initialize randomly the population of ants and antlions.
Evaluate ants and antlions using the fitness function
Find the elite antlion
Use roulette wheel to select an antlion for each ant.
Use Eq.(10) and Eq.(11) to upgrade g and h
Use Eq.(4) and Eq.(6) to produce random walk of ants and normalize it.
Use Eq. (9) to upgrade the position of an ant
For every Ant
Evaluate the fitness of all ants
Is fitness(Ant)>fitness(Antlion)?
Yes
No
Update Antlion=Ant Use Eq. (13)
Is fitness(Antlion)>fitness(Elite)?
Yes
No
Update Elite
Is stopping criteria satisfied?
No
Yes
End

Fig. 3 Flow diagram of ALO

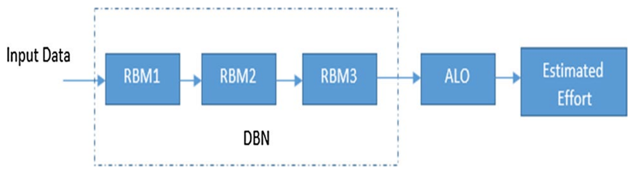


Fig. 4 Block diagram of DBN-ALO

complexity. There is another important factor, velocity. It is the amount of work done in a sprint time, where sprint time is the time allocated for a specific work to get completed and reviewed. In various machine learning studies, the final velocity and the story points of an agile project are taken as input arguments to estimate the effort.

3.4 Prediction Interval Approach of Effort Estimation

Jorgensen and Sjoberg proposed prediction interval (PI) technique in empirical distribution [16]. In this technique, the effort PI of a new project depends on the estimation accuracy of earlier software projects. Each PI has a minimum and maximum effort value, and a confidence level. The confidence level suggests how much the PI comprises the effort.

The confidence level used in the study is 90%. The matrix used for the PI computation is balanced relative error (BRE) computed as:

$$\text{BRE} = \frac{\text{Actual Effort} - \text{Estimated Effort}}{\text{Min(Actual Effort, Estimated Effort)}} \tag{14}$$

To estimate PI of a new software project, completed projects of similar nature are required. We calculate the BRE values of all the completed projects, and find the minimum and maximum BRE. The PI of a new software project is calculated as:

$$\text{PI} = \begin{cases} \frac{\text{Estimated Effort}_{\text{newproject}}}{1-\text{BRE}}, & \text{BRE} \le 0 \\ \text{Estimated Effort} \cdot (1 + \text{BRE}), & \text{BRE} > 0 \end{cases} \tag{15}$$

4 Proposed Work

This work proposes DBN-ALO and investigates whether the same effort estimation technique can be used for both agile and non-agile software development approaches. It also finds an effort prediction interval of a software project (Sect. 3.4). The block diagram of DBN-ALO is given in Fig. 4. The work is evaluated on four datasets for non-agile software development, and three datasets for agile software development. Before beginning the framework, all the data are normalized. The normalized data are fed into DBN-ALO. The key element in DBN-ALO is to determine the number of RBM stacks, input nodes, hidden nodes in each RBM stack, and the output layer. The number of RBM stacks and the nodes at their hidden layer are determined through experiments. The number of nodes at the visible layer of each RBM is equivalent to the number of attributes used from the datasets. In our architecture, the number of nodes at the hidden layer of each RBM is five for non-agile inputs, and three for agile inputs. There is only one node at the output layer which provides the effort. The input data are mapped to the visible layer of DBN and processed according to the training algorithm as given in Sect. 3.1. This is passed through three RBM stacks. At the output layer, the effort is computed as a linear-weighted sum of the final RBM outputs. The backpropagation algorithm is run between the hidden layer of the final RBM and the output layer to reduce the error between the estimated and actual effort. Delta rule is employed to update the weights between them. The ALO is applied to initialize the weights between the hidden layer of the third RBM stack and the output layer as it provides the optimal value. Hence, this reduces the error in minimum time between the estimated value and the actual value of effort. The parameters and their values used in DBN-ALO are given in Table 1. These values are chosen after running the programme code of DBN-ALO multiple times with different values of these parameters, and the values which gave the best results are tabulated. The procedure of DBN-ALO is given in Fig. 5. This whole procedure is implemented in MATLAB R2018b.

Table 1 Parameters of DBN-ALO

Number of search agents	50
Maximum number of iterations	100
Number of variables	1
Upper bound of variable	1
Lower bound of variable	0
Number of RBM stacks	3
Weights between the layers in RBM	0.5
Bias present at all the layers of RBM	0.5

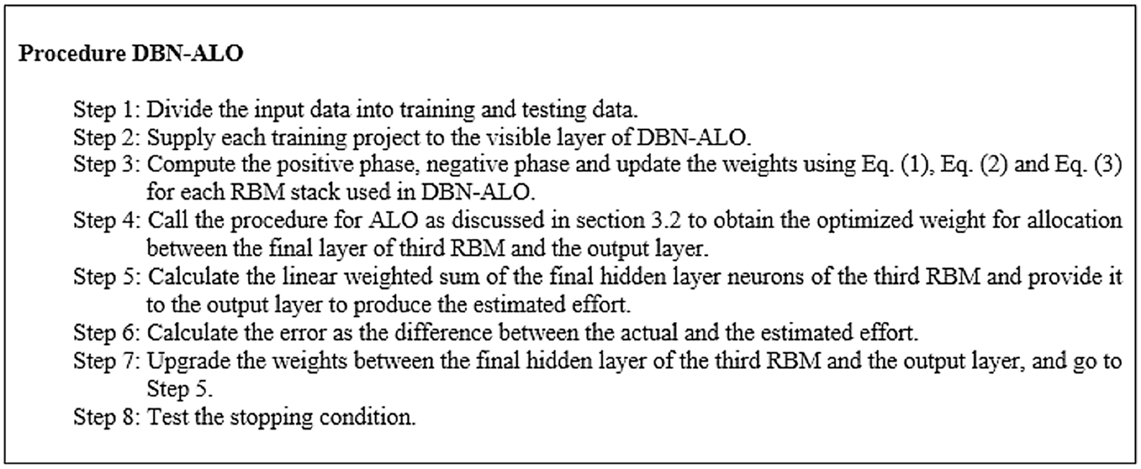
5 Experimental Evaluation

In this section, first the dataset description is presented with its evaluation criteria, and then experimental details are given.

The DBN-ALO technique estimates the effort for both agile and non-agile software development approaches. The non-agile datasets used are COCOMO81, NASA93, CHINA, and MAXWELL. They are available from the public repository of real-world project data [34] for software effort estimation. The statistical details of these datasets are given in Table 2. The agile datasets used are Zia, Company Dataset-1 (CD-1), and Company Dataset-2 (CD-2). The Zia dataset is available from Zia et al. [23], while Company Dataset-1 and Company Dataset-2 are procured from the company on request in Delhi-NCR, India.

To validate the proposed technique, the agile datasets are extended using FF-SMOTE (firefly-synthetic minority oversampling technique) [35] as the original datasets have a very smaller number of projects. Originally, the Zia dataset has only 21 input points, Company Dataset-1 (CD-1) has 23, and the Company Dataset-2 (CD-2) has 25 input points. All these datasets are extended to 100 inputs. The statistical details of these datasets are given in Table 3.

The cost estimation studies suffer from conclusion instability problem [36], in which the studies promote their


Procedure DBN-ALO

Step 1: Divide the input data into training and testing data.
Step 2: Supply each training project to the visible layer of DBN-ALO.
Step 3: Compute the positive phase, negative phase and update the weights using Eq. (1), Eq. (2) and Eq. (3) for each RBM stack used in DBN-ALO.
Step 4: Call the procedure for ALO as discussed in section 3.2 to obtain the optimized weight for allocation between the final layer of third RBM and the output layer.
Step 5: Calculate the linear weighted sum of the final hidden layer neurons of the third RBM and provide it to the output layer to produce the estimated effort.
Step 6: Calculate the error as the difference between the actual and the estimated effort.
Step 7: Upgrade the weights between the final hidden layer of the third RBM and the output layer, and go to Step 5.
Step 8: Test the stopping condition.

Fig. 5 Procedure for DBN-ALO

Table 2 Statistical details for non-agile datasets

Datasets	No. of features	Size	Min. effort	Max. effort	Mean effort	Skew
Zia	4	100	31.74	341	140.477	1.199
Company Dataset 1	4	100	1428.4	57,600	6781.39	6.18
Company Dataset 2	4	100	1119.2	52,880	6310.14	5.96

Table 3 Statistical details for agile datasets

Datasets	No. of features	Size	Min. effort	Max. effort	Mean effort	Skew
Cocomo 81	17	63	5.9	11,400	683.52	4.47
Nasa 93	24	93	8	8211	624	4.2
Maxwell	26	62	583	63,694	8223.21	3.34
China	19	499	26	54,620	3921.04	3.92

contribution as the best estimator, but the sampling methods used in these studies to produce training and testing data are different. The authors, Kocaguneli and Menzies [36] contributed that Leave-One-Out (LOO) method deals with the problem of conclusion instability, whereas N-way methods adds to it. However, a study by Mittas et al. [37] used N-way sampling method, and found no biasness in their results. So, it can be concluded that there is no universal solution to the conclusion instability problem. In the current study, LOO is used for generating train and test datasets. All the data are normalized in the interval [0 1] using min–max normalization approach [36].

The evaluation of a model means how well our model is performing. If the deviation between the actual and the estimated value is large, it will lead to inaccurate estimation, which will impact the cost of software development. The study uses widely accepted evaluation criteria used in the literature [38–40]. They are: MRE (magnitude of relative error), MMRE (mean magnitude of relative error), Pred (prediction), and SA (standard accuracy). A prediction technique is considered better if it has low MRE, and MMRE values, and high Pred, and SA values. These evaluation criteria are calculated as follows:

$$\mathrm{MRE} = \frac{|\text{Actual Effort} - \text{Estimated Estimated}|}{\text{Actual Effort}} \tag{16}$$

$$\mathrm{MMRE} = \frac{1}{m}\sum_{k=1}^{m} \mathrm{MRE}_k \tag{17}$$

where m is the total number of projects in the dataset.

$$\mathrm{MdMRE} = \text{Median (MRE)} \tag{18}$$

$$\mathrm{Pred}(l) = \frac{p}{m} \tag{19}$$

where m is the total number of projects and p is the number of projects whose MRE is less than or equal to l. The value of l in the study is taken as 0.25.

$$\mathrm{SA} = 1 - \frac{\mathrm{MAR}_{\mathrm{PT}}}{\overline{\mathrm{MAR}_{\mathrm{RG}}}} \tag{20}$$

where MAR (mean absolute residual) is calculated as:

$$\mathrm{MAR} = \frac{\sum_1^m |\text{Actual Effort} - \text{Estimated Effort}|}{m} \tag{21}$$

MAR_{PT} is the MAR of the proposed technique, and $\overline{\mathrm{MAR}_{\mathrm{RG}}}$ is the mean of MARs obtained through large runs of random guessing. In random guessing procedure, the estimated effort for test instance is same as that of training instance chosen randomly with equal probability [40].

The experiments with the proposed technique are conducted with agile and non-agile datasets, and using all the four evaluation criteria. To further explain in detail, first the non-agile framework is discussed, and then the agile framework is discussed.

The experiment starts by normalizing all the input datasets, which are fed to DBN-ALO model. The input passes through all the layers of DBN, and is processed as explained in the procedure for DBN-ALO (Fig. 5). We have selected Project 2 from COCOMO dataset to demonstrate a numerical example for the non-agile dataset; the same method is followed for the rest of the datasets. This project has cost driver values as 0.88, 1.16, 0.85, 1, 1.06, 1, 1.07, 1, 0.91, 1, 0.9, 0.95, 1.1, 1 and 1. All these 15 values are mapped to the 15 visible layer neurons of the first RBM stack. The DBN then starts its positive phase to construct the hidden neurons (refer Eq. 1). Here, we have used five hidden neurons for the non-agile input. The values obtained at this stage for these neurons are all 1. The negative phase now starts and updates the visible neurons (refer Eq. 2). The values obtained for the 15 visible neurons are all 0.9959. The weights are now updated (refer Eq. 3), and the values of the hidden neurons are calculated again using these new weights. This completes the training of the first RBM stack. The second RBM is added on top of the first RBM by starting the positive phase, and the weights used are the transpose of the updated weights (refer Fig. 2), which are calculated in the training of the first RBM. This creates the first layer of second RBM stack consisting of 15 neurons. The values of all the neurons obtained are 0.9901. The negative phase now starts, and weights are updated again. This process continues till the addition of the third RBM stack. Now, the ALO procedure is called (refer Fig. 3) and the best value obtained through it for Project 2 is 0.0054. This value is taken as the weight between the final layer of the third RBM stack and the output layer. Now, the weighted sum of all the neurons of hidden layer is calculated and the output is estimated. The process continues till the desired error tolerance is attained, or a certain number of epochs are achieved. The estimated output obtained for effort of Project 2 is 1568, whereas the actual effort value in the dataset is 1600.

In this work, the proposed technique is compared with simple DBN, i.e., without ALO technique, and few of the earlier results [7, 41]. The earlier results by Kaushik et al. [7] are based on two neural networks functional link artificial neural network (FLANN) and radial basis function neural network (RBFN), which are integrated with firefly algorithm (FA), and intuitionistic fuzzy-C means (IFCM) clustering for software effort predictions. Another study by Benala et al. [41] proposed particle swarm-optimized functional link artificial neural network (PSO-FLANN), and evaluated the technique on three datasets COCOMO81, NASA93, and MAXWELL. All the results on non-agile datasets are given in Tables 4, 5, 6, 7 and 8. The results on SA are not available for the techniques, PSO-FLANN, FA-FLANN-IFCM, FA-RBFN-IFCM given by the researchers [7, 41]. So, SA is computed only for the proposed technique. From the results, it is found that DBN-ALO performs best for all the four datasets and on all the evaluation criteria. The power of DBN is enhanced after integrating it with ALO. The results of DBN-ALO technique are far ahead than those of the other techniques listed in the tables.

Table 4 Evaluation on COCOMO dataset

Techniques	MMRE		MdMRE		PRED (25%)	
	Train set	Test set	Train set	Test set	Train set	Test set
DBN-ALO	0.08	0.06	0.04	0.02	0.93	0.94
DBN	0.16	0.12	0.16	0.12	0.91	0.90
PSO-FLANN	0.43	0.37	0.48	0.42	0.39	0.52
FA-FLANN-IFCM	0.14	0.17	0.09	0.11	0.92	0.90
FA-RBFN-IFCM	0.17	0.22	0.18	0.19	0.89	0.87

Table 5 Evaluation on NASA93 dataset

Techniques	MMRE		MdMRE		PRED (25%)	
	Train set	Test set	Train set	Test set	Train set	Test set
DBN-ALO	0.02	0.02	0.04	0.02	0.97	0.98
DBN	0.15	0.18	0.14	0.18	0.93	0.94
PSO-FLANN	0.49	0.34	0.44	0.45	0.39	0.50
FA-FLANN-IFCM	0.18	0.17	0.12	0.13	0.94	0.95
FA-RBFN-IFCM	0.17	0.14	0.15	0.16	0.89	0.90

Table 6 Evaluation on MAXWELL dataset

Techniques	MMRE		MdMRE		PRED (25%)	
	Train set	Test set	Train set	Test set	Train set	Test set
DBN-ALO	0.04	0.05	0.03	0.04	0.96	0.95
DBN	0.19	0.20	0.17	0.23	0.91	0.90
PSO-FLANN	0.55	0.38	0.49	0.42	0.32	0.48
FA-FLANN-IFCM	0.17	0.16	0.15	0.17	0.90	0.89
FA-RBFN-IFCM	0.13	0.14	0.12	0.11	0.90	0.91

Table 7 Evaluation on CHINA dataset

Techniques	MMRE		MdMRE		PRED (25%)	
	Train set	Test set	Train set	Test set	Train set	Test set
DBN-ALO	0.03	0.04	0.02	0.03	0.96	0.94
DBN	0.17	0.16	0.16	0.17	0.94	0.93
FA-FLANN-IFCM	0.21	0.22	0.18	0.20	0.89	0.91
FA-RBFN-IFCM	0.16	0.15	0.17	0.18	0.89	0.90

Table 8 Evaluation of SA on non-agile datasets

Datasets	DBN-ALO		DBN	
	Train set	Test set	Train set	Test set
COCOMO	0.980	0.985	0.931	0.921
NASA93	0.981	0.974	0.942	0.945
MAXWELL	0.982	0.988	0.920	0.910
CHINA	0.971	0.978	0.954	0.942

Table 12 Evaluation of SA on agile datasets

Datasets	DBN-ALO		DBN	
	Train set	Test set	Train set	Test set
ZIA	0.974	0.985	0.961	0.962
CD-1	0.973	0.984	0.931	0.925
CD-2	0.980	0.988	0.929	0.930

To evaluate the proposed technique on agile datasets, the same procedure of DBN-ALO (Fig. 5) is followed. Here, the architecture consists of three RBMs, each with two neurons at the input layer and three neurons at the hidden layer. The inputs from the agile datasets are mapped to the first RBM, and processed in the similar manner as explained for non-agile framework. The inputs from the dataset comprise of story point value and the velocity. The estimated effort obtained for the first project from Zia dataset is 153.85, and its actual value is 156. Tables 9, 10, 11 and 12 show the result on three agile datasets for all the evaluation criteria. Here, the results of DBN-ALO are far better than the result of simple DBN for all the three datasets. Table 13 shows the results of decision tree (DT), stochastic gradient boosting (SGB), random forest (RF), support vector regression with RBF kernel (SVR-RBF), support vector regression with RBF kernel optimized by grid search (SVR-RBF-GS), artificial bee colony–particle swarm optimization (ABC-PSO), and Zia et al.'s regression model, given by the authors [11, 23, 31–33] on Zia dataset. They evaluated these techniques on 21 projects only. In order to do the comparative analysis with these techniques, we also evaluated DBN-ALO on 21 projects. The results on Zia dataset using the proposed work outperform the results given by the other techniques. It is only lacking behind in PRED (25%) by SVR-RBF-GS technique [33]. The authors of ABC-PSO algorithm [32]

Table 9 Evaluation on ZIA dataset

Techniques	MMRE		MdMRE		PRED (25%)	
	Train set	Test set	Train set	Test set	Train set	Test set
DBN-ALO	0.07	0.08	0.06	0.05	0.96	0.97
DBN	0.17	0.14	0.15	0.13	0.95	0.96

Table 10 Evaluation on Company Dataset-1 (CD-1)

Techniques	MMRE		MdMRE		PRED (25%)	
	Train set	Test set	Train Set	Test set	Train set	Test set
DBN-ALO	0.03	0.06	0.02	0.06	0.97	0.97
DBN	0.15	0.15	0.14	0.12	0.90	0.92

Table 11 Evaluation on Company Dataset-2 (CD-2)

Techniques	MMRE		MdMRE		PRED 25%)	
	Train set	Test set	Train set	Test set	Train set	Test set
DBN-ALO	0.04	0.06	0.04	0.05	0.95	0.96
DBN	0.16	0.17	0.15	0.16	0.92	0.91

Table 13 Results on 21 projects of ZIA dataset [11]

Techniques	MMRE	MdMRE	PRED (25%)
DT	0.3820	0.2896	38.0952
SGB	0.1632	0.1151	85.7143
RF	0.2516	0.2033	66.6667
SVR-RBF-GS	0.0620	0.0426	100
SVR-RBF	0.0747	NA	95.9052
ABC-PSO	0.0569	0.0333	NA
Zia et al.'s regression model	0.0719	0.0714	57.14
DBN-ALO	0.0225	0.0222	98.4321

Table 14 PI calculation on COCOMO projects

Project Id	Actual effort	Estimated effort	BRE
1	2040	2378	−0.165
2	1600	1786	−0.116
3	243	230	0.056
4	240	268	−0.116
5	33	28	0.178
6	43	59	−0.372
7	8	15	−0.875
8	1075	899	0.195

also evaluated PRED (8%), R^2, and MAR, and found the values 66.67%, 0.9734, and 3.12, respectively. We also calculated these values for DBN-ALO, and found them to be 99%, 0.9999, and 2.457, respectively, which are better than the results of ABC-PSO algorithm.

So, it can be concluded that the technique DBN-ALO can be used for effort estimation in both agile and non-agile datasets. The difference here lies only in the architecture of DBN-ALO used for both the software development approaches. Figure 6 shows the graphical comparison of agile and non-agile projects on the evaluation criteria, where DBN-ALO emerges out as the best in comparison with other techniques. The test data results of Maxwell and Zia dataset are depicted in the graph.

Now, the prediction interval (PI) calculation as discussed in Sect. 3.4 is demonstrated. The estimated effort value provided by DBN-ALO is the crisp value. To find the PI estimate of any software project, Eqs. (14) and (15) are used. Table 14 lists the actual effort values, estimated values (using DBN-ALO), and BREs of eight COCOMO projects. Using the BRE values of these projects, the PI of ninth COCOMO project is calculated.

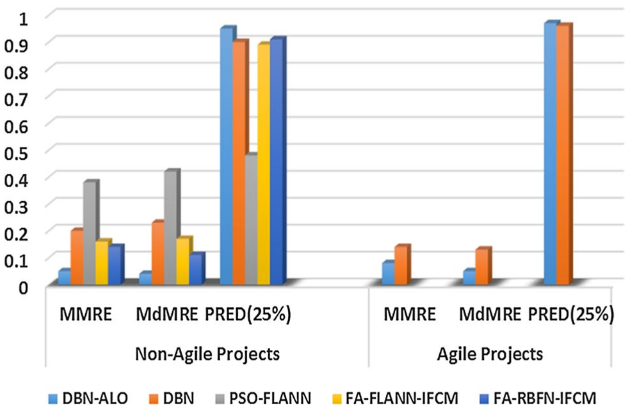


Fig. 6 Graphical analysis of agile and non-agile projects

Here, the minimum BRE is −0.875 and the maximum BRE is 0.195 (Table 14). The estimated effort of the ninth project using DBN-ALO is 390, and the actual effort from the dataset is 423. The range of PI for this project is calculated as [390/ (1 − (−0.875)), 390 * (1 + 0.195)] = [208, 466.05]. In this manner, the PI of any project can be calculated. The PI range of effort estimation of a project is more helpful to the project managers for estimations rather than crisp estimates as it handles uncertainty related to the crisp estimates.

6 Statistical Validations

The statistical validation finally confirms whether the proposed approach is better than the other approaches or not. It is mainly used to judge the performance of a model among various other models with less variation. In software effort estimation studies, the dataset does not belong to any particular distribution, so nonparametric tests are recommended [7]. All these tests are performed on KEEL (Knowledge Extraction based on Evolutionary Learning) tool [42], and the statistical analysis is performed on MMRE metric. We performed Friedman test [43] in order to compare multiple techniques over the non-agile datasets. The Friedman test computation is given by:

$$\mathrm{FT} = \frac{(p-1)\left[\sum_{j=1}^{p} \widehat{L}_j^2 - \left(\frac{pn^2}{4}\right)(pn+1)^2\right]}{\left\{\left[pn(pn+1)(2pn+1)\right]/6\right\} - (1/p)\sum_{i=1}^{n} \widehat{L}_i^2} \tag{22}$$

where p is the total number of techniques, $\widehat{L}_i$ is the total rank of the ith dataset, and $\widehat{L}_j$ is the total rank of the jth technique. The test provides the lowest rank to the best technique. The test statistics FT is computed to know the statistical differences among the techniques. This test is conducted on all the four non-agile datasets using the techniques DBN-ALO, DBN, FA-FLANN-IFCM, and FA-RBFN-IFCM. Our null hypothesis assumed that all the techniques performed equally. The results of Friedman test are given in Table 15.

Table 15 Friedman rank test statistics on non-agile datasets

N	4
χ^2	8.1
Df	3
p value	0.0439

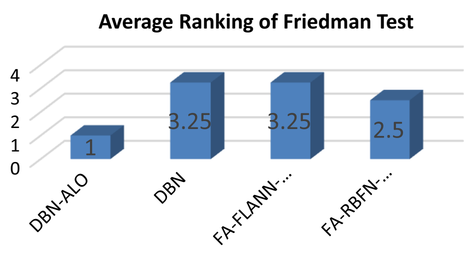


Fig. 7 Friedman rank test comparison on non-agile datasets

Here, *N* is the number of input non-agile datasets and the degree of freedom (*df*) is 3 as we have four techniques to compare. The standard χ^2 value with 3 *df* and significance value α=0.05 is 7.815. Our null hypothesis is rejected as the χ^2 value listed in the test statistic (Table 13) is more than 7.815 and the *p* value is less than 0.05. So, it is deduced that the techniques are different. Figure 7 shows the average ranks of all the techniques as given by Friedman rank test. As our null hypothesis is rejected, Holm post-hoc test [44] is run to find the differences among the techniques with DBN-ALO having rank 1 as the control method. Its test statistics is given in Table 16.

As per the Holm test statistics, DBN-ALO outperformed all other techniques as the Holm values are all less than 0.05.

The results of agile datasets are statistically validated using Wilcoxon signed-rank test [45], which compares the two algorithms based on positive and negative differences of their ranks. The two techniques to analyze on agile datasets are: DBN-ALO and DBN . The null hypothesis assumed here is that the two techniques performed equally. The alternative hypothesis assumed the two techniques performed differently. The test statistics of the test is provided in Table 17.

Here, R^+ shows the sum of ranks for the dataset in which the first algorithm outperformed the second and shows the sum of ranks for the opposite. From the test statistics (Table 17), the *p* value is less than 0.05, so the null hypothesis is rejected, and the first algorithm DBN-ALO outperformed DBN. So, for the agile datasets, DBN-ALO is better than DBN, too.

7 Threats to Validity

The study deploys DBN-ALO for software effort estimation in both agile and non-agile software development environment. The results obtained and evaluated through various evaluation criteria also support the proposed technique. However, there are certain points to ponder upon regarding internal and external validities.

7.1 Internal Validity

The study uses three RBM stacks in DBN construction with three nodes at the hidden layer for the agile inputs, and five nodes for the non-agile inputs. This architecture of DBN gave best results with the chosen ALO parameters. It may not always work, if we vary the ALO parameters, or some other optimization technique is integrated with the chosen DBN structure. The choice of the sampling method used to generate train and test data also affects the result of a particular technique. In this study, LOO is used due to its advantages to generate the train and test data samples. Other sampling techniques can also be used to check the validity of the approach. In addition, the study is limited to numerical attributes of software project datasets, but many historical project datasets contain categorical attributes also. So, the scope of the study can be extended. The experiments are conducted with only Pred (0.25); this value can be varied to further test the proposed approach.

7.2 External Validity

The four non-agile datasets used for experiments are taken from PROMISE Software Engineering Repository [34]. The approach can be tested with real-time and recent non-agile datasets. Due to small size of the agile datasets, they are over-sampled with FF-SMOTE [35]. The actual strength of the technique can be measured with agile datasets, which do not require over sampling.

Table 17 Wilcoxon signed-rank test statistics on agile datasets

Techniques	Rank positive (R^{+})	Rank negative (R^{-})	Hypothesis ($\alpha = 0.05$)	*p* value
DBN-ALO versus DBN	100.0	5.0	Rejected	0.002584

Table 16 Holm test statistics on non-agile datasets

Techniques	*Z*	Holm	Hypothesis ($\alpha = 0.05$)
DBN-ALO versus DBN	2.464752	0.008333	Rejected for DBN-ALO
DBN-ALO versus FA-FLANN-IFCM	2.464752	0.01	Rejected for DBN-ALO
DBN-ALO versus FA-RBFN-IFCM	1.643168	0.0125	Rejected for DBN-ALO

The non-agile input provided to DBN-ALO framework is 15, and the agile input is two. The technique must work well and give good results with varying number of input parameters. The proposed technique can be made robust by applying it on different types of project datasets.

8 Conclusion

The study does the comparative analysis of agile and non-agile development approaches on effort estimation using DBN-ALO technique. It is found that the proposed technique DBN-ALO worked best for both the development approaches. An effort prediction interval is also calculated, which assigns a range of effort estimation to a software project. Four non-agile datasets and three agile datasets are used. The statistical validations using Friedman and Wilcoxon signed-rank test also proved DBN-ALO performed well for both the development approaches, and outperformed other techniques used in the study.

References

1. Rijwani, P.; Jain, S.: Enhanced software effort estimation using multi layered feed forward artificial neural network technique. Procedia Comput. Sci. **89**, 307–312 (2016)
2. Laqrichi, S.; Marmier, F.; Gourc, D.; Nevoux, J.: Integrating uncertainty in software effort estimation using bootstrap based neural networks. IFAC Pap Online **48**(3), 954–959 (2015)
3. Nassif, A.B.; Azzeh, M.; Idri, A.; Abran, A.: Software development effort estimation using regression fuzzy models. J. Comput. Intell. Neurosci. (2019). https://doi.org/10.1155/2019/8367214
4. Zare, F.; Zare, H.K.; Fallahnezhad, M.S.: Software effort estimation based on the optimal bayesian belief network. J. Appl. Soft Comput. **49**, 968–980 (2016)
5. Sehra, S.K.; Brar, Y.S.; Kaur, N.; Sehra, S.S.: Software effort estimation using FAHP and weighted kernel LSSVM machine. J. Soft Comput. (2018). https://doi.org/10.1007/s00500-018-3639-2
6. Kaushik, A.; Verma, S.; Singh, H.J.; Chabbra, G.: Software cost optimization integrating fuzzy system and COA-Cuckoo optimization algorithm. Int. J. Syst. Assur. Eng. Manag. **8**(Suppl. 2), 1461–147 (2017)
7. Kaushik, A.; Tayal, D.K.; Yadav, K.; Kaur, A.: Integrating firefly algorithm in artificial neural network models for accurate software cost predictions. J. Softw. Evol. Process. **28**(8), 665–688 (2016)
8. Sivanageswara Rao, G.; Phani Krishna C.V.; Rajasekhara Rao, K.: Multi objective particle swarm optimization for software cost estimation. In: Proceedings of 48th Annual Convention of Computer Society of India-Vol I. Advances in Intelligent Systems and Computing, vol. 248, pp. 125–132 (2014)
9. Venkataiah, V.; Mohanty, R.; Pahariya, J.S.; Nagaratna, M.: Application of ant colony optimization techniques to predict software cost estimation. Comput. Commun. Network. Internet Secur. **5**, 315–325 (2017)
10. Manifesto for Agile Software Development (2019). https://agilemanifesto.org/principles. Accessed 15 April 2019
11. Satapathy, S.M.; Rath, S.K.: Empirical assessment of machine learning models for agile software development effort estimation using story points. J. Innov. Syst. Softw. Eng. **13**(2–3), 191–200 (2017)
12. Panda, A.; Satapathy, S.M.; Rath, S.K.: Empirical validation of neural network models for agile software effort estimation based on story points. Procedia Comput. Sci. **57**, 772–781 (2015)
13. Karhunen, J.; Raiko, T.; Cho, K.H.: Unsupervised deep learning: a short review. In: Advances in Independent Component Analysis and Learning Machines, pp. 125–142. Academic Press (2015)
14. Mirjalili, S.: The antlion optimizer. Adv. Eng. Softw. **83**, 80–98 (2015)
15. Trendowicz, A.; Jefferey, R.: Software project effort estimation. Foundations and best practice guidelines for success. Springer, Berlin (2014)
16. Jorgensen, M.; Sjoberg, D.I.K.: An effort prediction interval approach based on the empirical distribution of previous estimation accuracy. J. Inf. Softw. Technol. **45**(3), 123–136 (2003)
17. Abdelali, Z.; Mustapha, H.; Abdelwahed, N.: Investigating the use of random forest in software effort estimation. Procedia Comput. Sci. **148**, 343–352 (2019)
18. Pai, D.R.; McFall, K.S.; Subramanian, G.H.: Software effort estimation using a neural network ensemble. J. Comput. Inf. Syst. **53**(4), 49–58 (2013)
19. Benala, T.R.; Mall, R.: DABE: differential evolution in analogy-based software development effort estimation. J. Swarm Evol. Comput. **38**, 158–172 (2018)
20. Ezghari, S.; Zahi, A.: Uncertainty management in software effort estimation using a consistent fuzzy analogy-based method. J. Appl. Soft Comput. **67**, 540–557 (2018)
21. Abdelali, Z., Hicham, M., Abdelwahed, N.: An ensemble of optimal trees for software development effort estimation. In: Smart Data and Computational Intelligence. AIT2S 2018, vol. 66, pp. 55–68. LNNS, Springer, Cham (2018)
22. Nguyen, V.; Boehm, B.; LiGuo, H.: Determining relevant training data for effort estimation using window based COCOMO calibration. J. Syst. Softw. **147**, 124–146 (2019)
23. Ziauddin, S.; Tipu, S.K.; Zia, S.: An effort estimation model for agile software development. J. Adv Comput Sci Appl **2**(1), 314–324 (2012)
24. Martínez, J.L.; Noriega, A.R.; Ramírez, R.J.; Licea, G.; Jiménez, S.: User stories complexity estimation using bayesian networks for inexperienced developers. J. Clust. Comput. **21**(1), 715–728 (2018)
25. Dragicevic, S.; Celar, S.; Turic, M.: Bayesian network model for task effort estimation in agile software development. J. Syst. Softw. **127**, 109–119 (2017)
26. Tanveer, B.: Hybrid effort estimation of changes in agile software development. In: Agile Processes in Software Engineering, and Extreme Programming, vol. 251, pp. 316–320. LNBIP, Springer, Cham (2016)
27. Tanveer, B.; Guzmán, L.; Engel, U.M.: Effort estimation in agile software development: case study and improvement framework. J. Softw. Evol. Process. (2017). https://doi.org/10.1002/smr.1862
28. Bilgayian, S.; Mishra, S.; Das, M.: Effort estimation in agile software development using experimental validation of neural network models. Int. J. Inf. Technol. **11**(3), 569–573 (2019)
29. Britto, R.; Usman, M.; Mendes, E.: Effort estimation in agile global software development context. In: Large-Scale Development, Refactoring, Testing, and Estimation. XP 2014, vol. 199, pp. 182–192. LNBIP (2014)
30. Usman, M.; Britto, R.; Damm, L.O.; Borstler, J.: Effort estimation in large scale software development: an industrial case study. J. Inf. Softw. Technol. **99**, 21–40 (2018)
31. Satapathy, S.M.; Panda, A.; Rath, S. K.: Story point approach based agile software effort estimation using various SVR kernel methods. In: Proceedings of the International Conference on

Software Engineering and Knowledge Engineering, pp. 304–307 (2014)

32. Tung, K.T.; Hanh, L.T.M.: A novel hybrid abc-pso algorithm for effort estimation of software projects using agile methodologies. J. Intell. Syst. **27**(3), 489–506 (2018)
33. Zakrani, A.; Najm, A.; Marzak, A.: Support vector regression based on grid-search method for agile software effort prediction. In: Proceedings of International Congress on Information Science and Technology, pp. 492–497 (2018)
34. Tera promise: Data Categories http://openscience.us/repo/(2019) . Accessed 15 March 2019
35. Kaur, P.; Gossain, A.: FF-SMOTE: a metaheuristic approach to combat class imbalance in binary classification. J. Appl. Artif. Intell. **33**(5), 420–439 (2019)
36. Kocaguneli, E.; Menzies, T.: Software effort models should be assessed via leave-one-out validation. J. Syst. Softw. **86**(7), 1879–1890 (2013)
37. Mittas, N.; Papatheocharous, E.; Angelis, L.; Andreou, A.S.: Integrating non-parametric models with linear components for producing software cost estimations. J. Syst. Softw. **99**, 120–134 (2015)
38. Foss, T.; Stensrud, E.; Kitchenham, B.; Myrtveit, I.: A simulation study of the model evaluation criterion MMRE. IEEE Trans. Softw. Eng. **29**(11), 985–995 (2003)
39. Kaushik, A.; Soni, A.K.; Soni, R.: An improved functional link artificial neural networks with intuitionistic fuzzy clustering for software cost estimation. Int. J. Syst. Assur. Eng. Manag. **7**(1), 50–61 (2016)
40. Shepperd, M.; MacDonell, S.: Evaluating prediction systems in software project estimation. Inf. Softw. Technol. **54**, 820–827 (2012)
41. Benala, T.R.; Korada, C.; Mall, R.; Dehuri, S.: A particle swarm optimized functional link artificial neural networks (PSO-FLANN) in software cost estimation. In: Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA). Advances in Intelligent Systems and Computing, vol. 199, pp. 59–66 (2013)
42. Alcalá-Fdez, J.; Fernández, A.; Luengo, J.; Derrac, J.; García, S.; Sánchez, L.; Herrera, F.: KEEL data-mining software tool: dataset repository, integration of algorithms and experimental analysis framework. J. Mult. Valued Logic Soft Comput. **17**(2), 255–287 (2011)
43. Hodges, J.L.; Lehmann, E.L.: Rank methods for combination of independent experiments in analysis of variance. Ann. Math. Stat. **33**(2), 482–497 (1962)
44. Holm, S.: A simple sequentially rejective multiple test procedure. Scand. J. Stat. **6**(2), 65–70 (1979)
45. Wilcoxon, F.: Individual comparisons by ranking methods. Biom. Bull. **1**(6), 80–83 (1945)