Software Effort Estimation Using Functional Link Neural Networks Tuned with Active Learning and Optimized with Particle Swarm Optimization

Tirimula Rao Benala[1(✉)], Rajib Mall[2], Satchidananda Dehuri[3], and Pala Swetha[1]

[1] Department of Information Technology,
Jawaharlal Nehru Technological University Kakinada,
University College of Engineering, Vizianagaram 535003, India
{b.tirimula,palaswetha}@gmail.com
[2] Department of Computer Science and Engineering,
Indian Institute of Technology Kharagpur, Kharagpur, India
rajib@cse.iitkgp.ernet.in
[3] Department of Information and Communication Technology,
Fakir Mohan University, Vyasa Vihar, Balasore 756019, Odisha, India
satchi.lapa@gmail.com

Abstract. This paper puts forward a new learning model based on the collaborative effort of active learning and particle swarm optimization (PSO) in functional link artificial neural networks (FLANNs) to estimate software effort. The active learning uses quick algorithm to detect the essential content of the datasets by which the dataset is reduced and are processed through PSO optimized FLANN. The PSO uses the inertia weight, which is an important parameter in PSO that significantly affects the convergence and exploration-exploitation in the search space while training FLANN. The Chebyshev polynomial has been used for mapping the original feature space from lower to higher dimensional functional space. The method has been evaluated exhaustively on different test suits of PROMISE repository to study the performance. The computational results show that the active learning along with PSO optimized FLANN greatly improves the performance of the model and its variants for software development effort estimation.

Keywords: Software effort estimation · PSO · Active learning and FLANN

1 Introduction

Software effort estimation is the process of prediction of effort, cost, schedule, and staffing levels for successful project management [10, 11]. Accurate software cost estimation is highly required for the effective software project management. It significantly affects management activities such as resource allocation and creating

B.K. Panigrahi et al. (Eds.): SEMCCO 2014, LNCS 8947, pp. 223–238, 2015.
DOI: 10.1007/978-3-319-20294-5_20

reasonable schedule. The major contributing factor for accurate estimation is effort. This has led researchers to conduct extensive research on software effort estimation methods. Recently, it is boosted to many of the researchers of computational intelligence field to design an intelligent semi-automatic estimator for the aforesaid task [14]. In continuation, this paper investigates the control parameters of FLANN for software effort estimation. The beauty of this approach is that it does not do minimal approximation about the function to be evaluated for effort prediction. Basing on the structure of the dataset the model function parameters are calibrated. The Active Learning algorithm employs tuning of training dataset to characterize the essential content of software effort estimation data; i.e., the least number of features and instances are required to capture the information within software effort estimation data [Active learning]. This work is an improvement and fine tuned of our earlier work PSO-FLANN [11].

FLANN proposed by Pao, is a type of neural network consisting of one input layer and an output layer for forming arbitrarily complex decision regions to guide real world applications. FLANN has been widely used in many application areas of pattern recognition [12], data mining [13], time series forecasting [15], etc. A good survey of FLANN and its variants can be obtained in [13]. FLANN generates output (effort) by expanding the inputs (cost drivers) by nonlinear orthogonal functions like Chebyshev polynomial and then processing the final output layer. Each input neuron corresponds to a component of an input vector. The output layer consists of one output neuron that computes the software development effort as a linear weighted sum of the outputs of the input layer [9, 10]. The non-normal characteristics of the datasets always lead FLANN to low prediction accuracy and high computational complexity. To alleviate these drawbacks the proposed technique has been formulated to exploit the best features of PSO and FLANN. It is named as ACTIVE-**F**LANN-P**S**O (a. k. a FAS).

Functional link neural networks and cost estimation fundamentals are briefly reviewed in Sect. 2. The active learning and particle swarm optimization are described in Sects. 3 and 4. Our approach is presented in Sect. 5. In Sect. 6, numerical examples from Cocomo81 (Coco81), Nasa93, Maxwell dataset is used to evaluate the performance. Section 7 concludes this paper.

2 Background

In this section background of this work is discussed. Software cost estimation and FLANN architecture are described in Sects. 2.1 and 2.2, respectively.

2.1 Software Effort Estimation

Software effort estimation (SEE) is one of the important steps in software project management. It can be defined as the task of estimating the total effort required to develop a software system [6]. SEE is incorporated by a set of attributes (also known

as, cost drivers) representing a software project to predict (or estimate) the cost, in terms of person-months, in turn to predict the required time to develop the software system [2, 7]. SEE assist the project managers to take strategic decisions such as bidding on a new project, managing development, maintenance or customization of the software, planning, and allocation of resource. Software effort estimation has been the process of building regression models. However, regression models may lead to underestimation and overestimation. The underestimation of software effort drops the software product quality as some of the software development life cycle activities are skipped to avoid cost over runs and to meet the deadline. The overestimation leads to wastage of resources and loss of new projects as most of the resources are engaged without any effective output [1]. According to Oliveira [2, 7], the major bottlenecks for software projects are the schedule and effort (cost) to finish it; the ever changing dynamics of project scope make the process of cost estimation complex. Due to the typical characteristics of each project, the accurate measurement of the cost and development time of software can be determined only after the project is completed [3, 7]. However, it is necessary to perform estimations before the project begins. There are various techniques and methods which can be employed to estimate software development effort, cost, and time. This paper introduces an innovative technique aimed to predict (estimate) the software development effort based on Active Learning, PSO, and FLANN.

2.2 Architecture of FLANN

A typical FLANN structure is illustrated in Fig. 1. FLANN is a typical two layer network with an implicit hidden layer. The original input space is mapped into n-dimensional feature space (n is a user defined parameter, varies across the domains) by functional expansion such that the feature space becomes linearly separable in higher space. The Chebyshev polynomial functional expansion is better suited as basis function in software cost estimation as it produces low error estimates [11, 19]. The output of the basis function is multiplied by random weights chosen in the range $[-0.5, 0.5]$ and the summation of all such multiplications is feed to sigmoid function to predict the development effort of 'n' Person-Months [9, 10]. The learning process involves updating the weights of FLANN in order to minimize a given cost function. The weight vector is evolved by PSO, which has unique characteristics like the rapid convergence of global solutions and less number of parameters to be optimized.

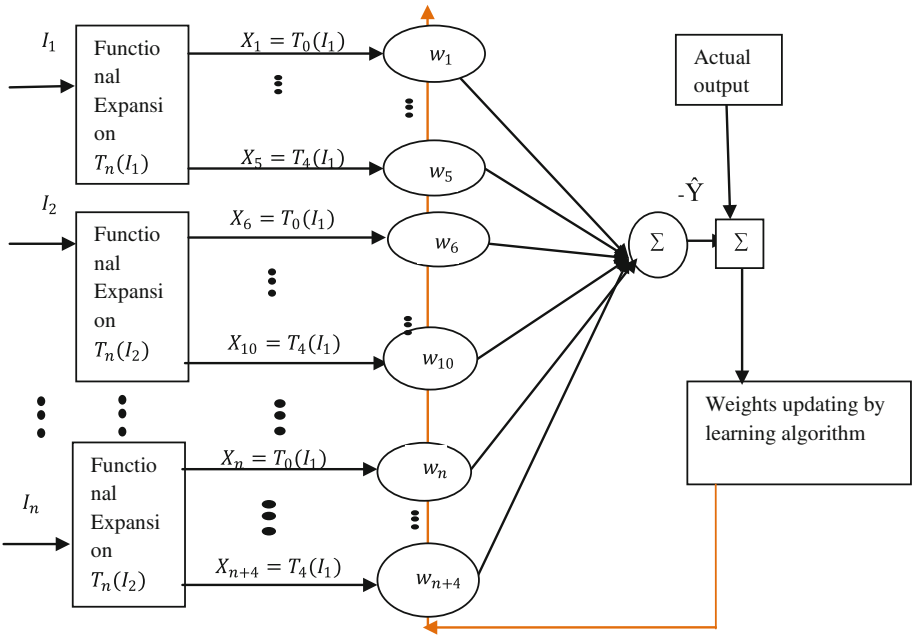

Fig. 1. FLANN architecture

3 Active Learning

Active learning is a special case of semi-supervised machine learning. To characterize the essential content of Software effort estimation data points i.e., the minimum number of features and instances required to capture the information within SEE data. If the essential content is very small then the contained information must be very succinct and the value-added complex learning schemes must be minimal [4].

In active learning, some heuristic (in our case, each row's popularity value) is used to sort instances from most interesting to least interesting. The data is then explored according to that sort order. Learning can terminate early, if the results from all the N instances are not better than from a subset of M instances, where M < N [Active learning].

QUICK is an active learning method that assists in reducing the complexity of data interpretation by identifying the essential content of SEE data sets [4]. QUICK works as follows:

- Group rows and columns by their similarity,
- Discard redundant columns (synonyms) that are too similar,
- Discard outlier rows (outliers) that are too distant, and
- In the remaining data, generate an estimate from the nearest example.

Let us assume that the TrainDataset is the training dataset after sampling the datasets into training and test datasets,

Algorithm-1: Quick$(\boldsymbol{TrainDataset}[\mathbf{1} \ldots \boldsymbol{N-1}, \mathbf{1} \ldots \boldsymbol{M}])$

Repeat until $Pop(j) \leq 1$ *are exhausted or* $MRE < 0.1$ *or*

If the $\Delta\, best(Error) \sim worst(Error)$ *of last n instances in Active Pool is very small*

%% Active Pool is nothing but our obtained reduced size of the Normalized Training Dataset%%

Step 1: Start

Step 2: Synonym Pruning

Step 2.1: $D[1 \ldots N-1, 1 \ldots M] = TrainDataset[1 \ldots N-1, 1 \ldots M]$*% calculate distance matrix for the transposed training dataset. %*

Step 2.2: $DM\,[1 \ldots M, 1 \ldots M] = EuclideanDistance\,(D', D')$ *% Assign a rank based on nearest neighbor elements %*

Step 2.3: $Enn[1 \ldots M, 1 \ldots M] = rank$

Ex: If $DM[i, j] = 3$*, it means j is the* i^{th} *third nearest neighbor*

Step 2.4: Calculate $E(k)[1 \ldots M, 1 \ldots M]$

if $E[1 \ldots M, 1 \ldots M] \leq k\; E[1 \ldots M, 1 \ldots M] = 1$

else $E[1 \ldots M, 1 \ldots M] = 0$

Step 2.5: Calculate Popularity index $Pop[1{,}1 \ldots M]$

$Pop[1{,}1 \ldots M] = SUM(E(K)[1 \ldots M, 1 \ldots M])$

K=Sum of popularity index whose value is not equal to zero

Step 2.6: $SPD[1 \ldots N-1{,}1 \ldots M-k] =$

$TrainDataset(1 \ldots N-1, Index(Pop[1{,}1 \ldots M] == 0))$

%select those features whose popularity index value is Zero%

Step 3: Synonym Pruning Ends

Step 4: Outlier Pruning starts

Step 4.1: $DM[1 \ldots N-1{,}1 \ldots N-1] = EucledianDistance(SPD, SPD)$

%Assign a rank based on nearest neighbor elements%

Step 4.2: $Enn(1 \ldots M, 1 \ldots M) = rank$

Step 4.3: Calculate $E(k)[1 \ldots M, 1 \ldots M]$

if $E[1 \ldots M, 1 \ldots M] \leq k\; E[1 \ldots M, 1 \ldots M] = 1$

else $E[1 \ldots M, 1 \ldots M] = 0$

Step 4.4: Calculate Popularity index $Pop[1{,}1 \ldots M]$

$Pop[1{,}1 \ldots M] = SUM(E(k)[1 \ldots M, 1 \ldots M]$

n=Sum of popularity index whose value is not equal to zero

Step 4.5: $OPD[1 \ldots N-n, 1 \ldots M-k] = SPD(1 \ldots N-1{,}1 \ldots M-k)$

Step 5: Outlier Pruning Ends

4 Particle Swarm Optimization

Particle swarm optimization is a nature inspired algorithm, invented by Kennedy and Eberhart [21], for dealing with problems in which the best solution can be represented as a point or surface in an *n*-dimensional space. Hypothesis are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some fitness criterion after watch time step. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values [11].

In the particle swarm optimization algorithm, particle swarm consists of "m" particles, and the position of each particle represents for the potential solution in D-dimensional space. The particles change its condition according to the following three principles:

(1) To keep its inertia(ω),
(2) To change the condition according to its most optimist position,
(3) To change the condition to the swarm's most optimist position.

The speed and position of each particle changes according to the following equations,

$$v_{id}^{k+1} = \omega v_{id}^{k} + c_1 r_1^k \left(pbest_{id}^{k} - x_{id}^{k}\right) + c_2 r_2^k \left(gbest_{id}^{k} - x_{id}^{k}\right)$$
$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1}$$

Algorithm-2: PSO $(\boldsymbol{Y}, \hat{Y}, \boldsymbol{c_1}, \boldsymbol{c_2}, \boldsymbol{r_1 r_2}, \boldsymbol{velocity}, \boldsymbol{position}, \boldsymbol{\omega}, \boldsymbol{pbest}, \boldsymbol{gbest})$

Step 1: Calculate PSOError=sqrt$\left(\left(Y - \hat{Y}\right)^2\right)$

%Find the minimum Fitness or error %

Step 2: from 1…kk

Step 3: $Fitness(kk) = minimum\left(Fitness(1 \ldots kk)\right) \& MinWtnum = kk$

Step 4: End

Step 5: pbest=wts(MinWtnum)

Step 6: gbest=minimum(pbest)

%%Update weights $wts[1 \ldots 10, 1 \ldots M \times 5]$*(position) and changeWeights* $changewts[1 \ldots kk, 1 \ldots M \times 5]$*(velocity)%%*

Step 7: $changewts[1 \ldots kk, 1 \ldots kk] = iw \times changewts[1 \ldots kk, 1 \ldots M \times 5] + C_1 \times r_1 \times \left(pbest - wts(1 \ldots kk, 1 \ldots kk)\right) + C_2 \times r_2 \times \left(gbest - wts(1 \ldots kk, 1 \ldots kk)\right)$

Step 8: $wts(1..kk, 1 \ldots kk) = wts(1 \ldots kk, 1 \ldots kk) + changewts(1 \ldots kk, 1 \ldots kk)$

$wt1s(1..kk, 1 \ldots kk) = wts(1 \ldots kk, 1 \ldots kk) + changewts(1 \ldots kk, 1 \ldots kk)$

%% Updating Inertia of Weights%%

Step 9: $iw = \lambda \times \mathrm{iw} + (1 - \lambda) \times \mathrm{Variance}$

Step 1: $\lambda = 0.95$

Step 2: $Variance = \sqrt{F}$

% Finding Variance%

$F1 = absolute\left(FitFunn(1 \ldots kk) - Average\left(FitFunn(1 \ldots kk)\right)\right)$

$F2 = \max(F1)$

$if\ F2 < 1$

$F3 = -1$

$else$

$F3 = -absolute(\max(F2))$

$F = sum((F1/F3)^2)$

%Finding variance Ends%

Step 3: End

Step 10: End

% updated weights are send again for training dataset%

5 ACTIVE-FLANN-PSO Algorithm

In this section, we first present the methodology for software cost estimation, next algorithm, and finally performance evaluation metrics.

5.1 Methodology

ACTIVE-FLANN-PSO (a. k. a FAS) is a typical two layer feed forward neural network consisting of an input layer and output layer with an implicit hidden layer. The nodes between input layer and output layer are connected without weight vector and the nodes between hidden layer and output layer are assigned with weight vector. Unlike FLANN, the weight vector is evolved by PSO learning algorithm. There are M input nodes and N data points in every dataset. Our Active learning algorithm computes the Euclidean distance between rows (instances or data points) and columns (features depend on the cost drivers of a particular dataset) of SEE data, then prunes synonyms (similar features) and outliers (distant instances). This reduced size training dataset is processed by FLANN. The input nodes of the reduced size train dataset are expanded by a basis function to m number of nodes, where m is functionally expanded nodes. In software cost estimation domain Chebyshev polynomial basis function is the most effective function for functional expansion [9, 10, 19]. The basis function maps the input space to higher dimension.

5.2 Algorithm-3

Let us assume, D is the dataset, ND is the normalized Dataset. TrainDataset and TestDataset are the training and testing parts respectively, P is the data point of reduced size, L and H are lower and higher dimensions respectively, and TC is the termination criteria, O is the output layer, W is the weighted sum, E is the error, current best fitness value is CF. Fitness value is F, GB represents global best, PV is the particle velocity.

Step 1: For each $Dataset\ D[1 \dots N, 1 \dots M]$
    Step 1.1: $ND[1 \dots N, 1 \dots M] = Normalization(D[1 \dots N, 1 \dots M])$
Step 2: Divide ND 2/3rd parts into TrainDataset and 1/3rd part into TestDataset.
Step 3: For each TrainDataset
    Step 3.1: Apply $Quick(TrainDataset[1 \dots N-1, 1 \dots M])$
Step 4: For each P
    Step 4.1: Map from L to H.
Step 5: For each particle initialize with small values from [-1, -1].
Step 6: While(!TC)

```
{
Apply PSO(Y, Ŷ, c1, c2, r1r2, velocity, position, ω, pbest, gbest)
{
        For each swarm
                {
                For each particle in the swarm
                        {
                        For each sample in the training sample
                                {
                                Calculate W, and send it as input to O.
                                Calculate E.
                                }
                        Assign E to F.
                                If (F is better than CF)
                                        {
                                        Assign F to CF
                                        }
                        Assign CF to GB
                        }
                For each particle
                        {
                        Call Reduced () and Find PV.
                        Update Particle Position
                        }
                }
}
}
```

The overall architechture of the FAS model with training and testing process is shown in Fig. 2.

5.3 Performance Evaluation Metrics

Five evaluation criteria were used to assess the degree of accuracy to which the estimated effort matches actual effort. These evaluation criteria have been chosen as they are widely accepted benchmark metrics for performance evaluation in the software cost estimation literature. They are as follows: Mean Magnitude of Relative Error (MMRE), Median Magnitude of relative error (MdMRE), and PRED (0.25) [5, 8, 20], Standardized Accuracy (SA) and Delta [17].

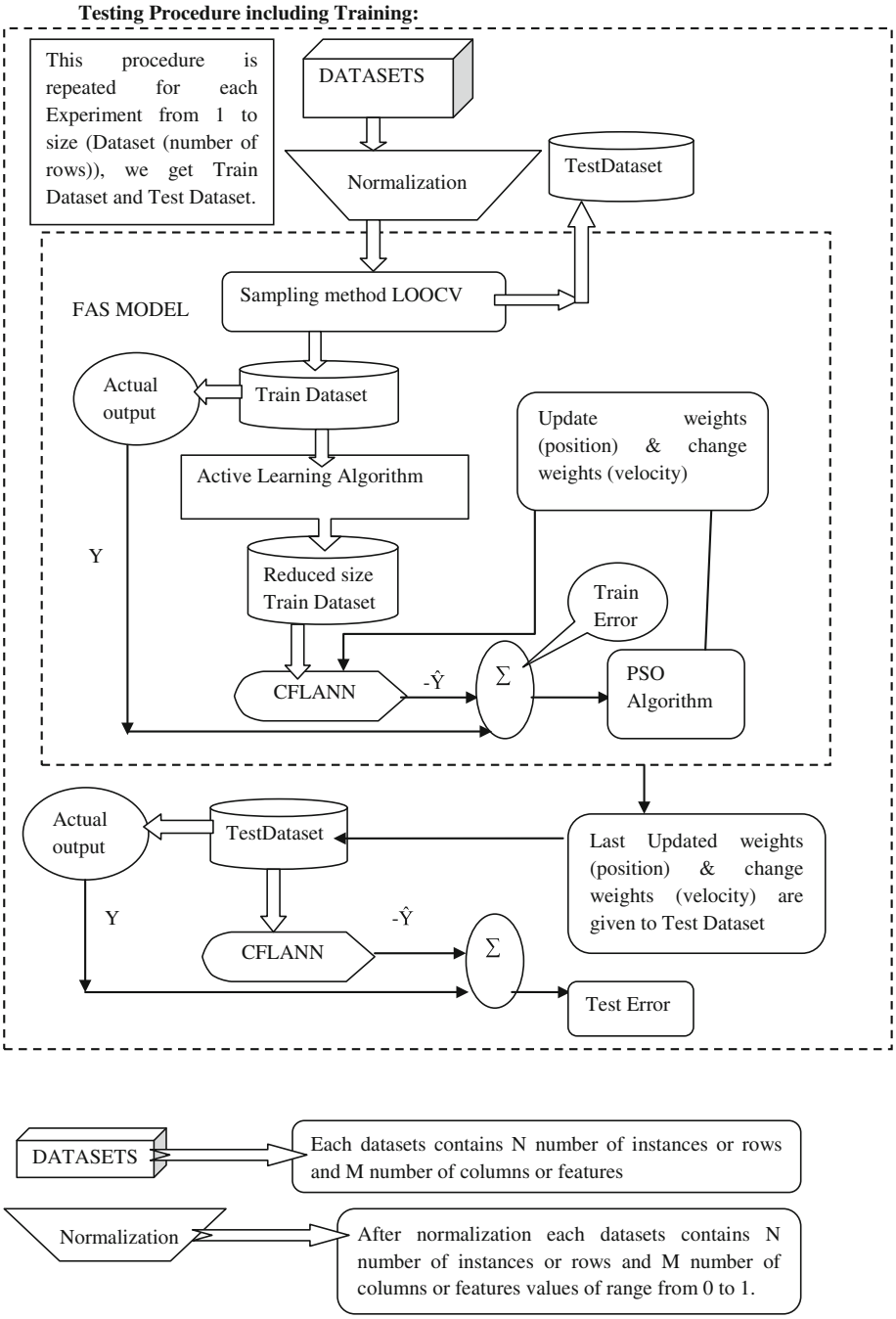

Fig. 2. Overall proposed methodology

The software effort estimation literature says a good prediction model should have MMRE and MdMRE less than 0.25, and that value of PRED (0.25) should not be less than 0.75.

MMRE is defined as the average of all MRE's (Magnitude of relative errors).

$$MMRE = \frac{\sum MRE}{n}, \tag{1}$$

where, $MRE = mod\left(\frac{C_i - C_i'}{C_i}\right)$,

n is the total number of projects,
C_i is the original value and
C_i' is the new estimated value.

MdMRE is defined as median of all MRE's. It is more likely in proper evaluation than MMRE as in case of extreme values (outliers) of MREs, MMRE predicts a model incorrectly due to the fact that while calculating the arithmetic average of MREs in case of MMRE the outliers will influence the result and wrongly assess the model.

$$MdMRE = median(MRE's), \tag{2}$$

PRED (0.25) is the percentage of estimations that are within 25 % of original value.

$$PRED = \frac{k}{n}, \tag{3}$$

where k is the number of MRE values which are less than 0.25 and n represents the number of projects.

The paper [5, 17] shows that measures based on MRE are potentially inappropriate due to their asymmetry, biasing towards prediction models that under-estimate. It includes the very popular performance measure based on MRE in the SEE literature: MMRE. Another measure, MAE, does not present asymmetry problems and is not biased. However, it is difficult to interpret, since the residuals are not standardized. So, measures such as MMRE have kept being widely used by most researchers in the area. However, Shepperd and MacDonell [18] very recently proposed a new measure called Standardized Accuracy (SA). To judge the effect size we use Delta. These are defined as follows:

SA can be interpreted as the ratio of how much better P_i is the random guessing, giving a very good idea of how well the approach does.

$$SA = 1 - \frac{MAE_{P_i}}{\overline{MAE_{P_0}}}, \tag{4}$$

Where MAE_{P_i} is the mean absolute error of the prediction model P_i and $\overline{MAE_{P_0}}$ is the mean value of large number, typically 1000, runs of random guessing. This is defined as predicting $\widehat{y}$ for the example t by random sampling over the remaining n − 1 examples and taking $\widehat{y}_t = y_r$, where r is drawn randomly with equal probability from $1 \ldots n \bigwedge r \neq t$.

To judge the effect size, he following measure is suggested:

$$\Delta = \frac{\mathrm{MAE_{P_i}} - \overline{\mathrm{MAE_{P_0}}}}{\mathrm{s_{P_0}}}, \tag{5}$$

Where s_{P_0} is the sample standard deviation of the random guessing strategy. The values of Δ can be interpreted in terms of the categories proposed by Cohen [16] of small (≈ 0.2), medium (≈ 0.5) and large (≈ 0.8).

6 Experiments, and Results

In this section, studies were carried out on three PROMISE repository datasets, namely, Cocomo81 (Coc81), Nasa93, Maxwell [7] to investigate the performance of our method. These datasets are available at promisedata.googlecode.com.

6.1 Dataset Preparation

Before the experiments, all input variables (a. k. a features) are normalized using min-max normalization in order to eliminate the possibility of unequal influences. The three datasets shown in Table 1 are subjected to Leave-one-out cross-validation; in each iteration all the data points except for the single observation are used for training and the model is tested on that single observation. Table 1 provides main features of the data sets, including the number of features, size, the skewness, minimum, mean and maximum of effort and size in KLOCS.

Table 1. Descriptive statistics for public datasets [22]

Dataset	Nasa93	Maxwell	Coc81
Features	17	27	17
Size	93	62	63
Units	Months	Hours	Months
Minimum	8	583	6
Effort			
Median	252	5189.5	98
Mean	624	8223.2	683
Maximum effort	8211	63694	11400
Skew	4.2	3.26	4.4

6.2 Cost Estimation Models

This study explored the feasibility of using different models based on FLANN. The functional expansion, namely, Chebyshev polynomial based FLANN (C-FLANN) model is included in our experiments. Hereafter, C-FLANN will be annotated as FLANN. The proposed models using Particle Swarm Optimization and Active Learning is:FAS. For a comprehensive evaluation of the proposed models, for comparison, other popular estimation models including, Functional Link Artificial Neural Networks (FLANN) [9, 10], PSO-FLANN, and Artificial Neural Networks (ANN), are also included in the experiments.

Table 2. Nasa93 dataset

Method	MMRE		MdMRE		PRED (0.25)		SA		DELTA	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
PSO-FLANN	0.0012	9.8824e-07	4.2531e-04	1.1360e-04	1	0.0108	0.9999	1	0	1.3140
FAS	0.0022	0.0049	6.3687e-04	9.0165e-04	1	1	0.9918	0.9694	0	3.5012
FLANN	0.0013	0.0021	7.8309e-04	7.9131e-04	1	1	0.9999	0.9999	0	0.0761
ANN	0.0022	0.0053	0.0018	0.0045	1	1	0.9999	0.9998	0	0.8594

Table 3. Maxwell dataset

Method	MMRE		MdMRE		PRED (0.25)		SA		DELTA	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
PSO-FLANN	5.8207e-05	2.3489e-07	3.2986e-05	2.3190e-05	1	0.0161	0.9999	1	0	3.7282
FAS	8.0178e-05	1.4138e-04	3.8868e-05	4.7363e-05	1	1	0.9928	0.9752	0	3.4267
FLANN	6.6870e-05	3.4855e-06	5.3921e-05	1.9447e-06	1	1	0.9999	0.9999	0	0.2741
ANN	8.1571e-05	5.7393e-06	7.4071e-05	5.3091e-06	1	1	0.9999	0.9998	0	1.0522

Table 4. Coc81 dataset

Method	MMRE		MdMRE		PRED (0.25)		SA		DELTA	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
PSO-FLANN	0.0023	1.3378e-06	9.0977e-04	1.0478e-04	1	0.0159	0.9999	1	0	2.4339
FAS	0.0034	0.0049	0.0011	7.8331e-04	1	1	0.9953	0.9840	0	3.4247
FLANN	0.0022	0.0041	6.4478e-04	4.8037e-04	1	1	1.0000	0.9999	0	0.3207
ANN	0.0050	0.0158	0.0047	0.0121	1	1	0.9998	0.9998	0	1.1774

6.3 Experimental Procedure

For the purpose of validation, we adopt Leave-one-out cross validation to evaluate the generalization error of the methods. In this scheme for each dataset of n data points and given m candidate models, each model is trained with n-1 data points and then it is tested on the sample that was left out. This process is repeated n times until every data point in the dataset have been used as cross-validation instance. Then the average training error and testing error across all three trails are computed. The advantage of this scheme is that it does not matter how the data is split since each data point is assigned in a test set, a training set and a validation set respectively once. At first, the performances of PSO-FLANN, FLANN, FAS, and ANN are investigated. The best variants on training set are selected as the candidates for comparison. Next, the optimizations of parameters of the machine learning methods are performed on the training dataset by searching through their solution space. Thirdly, the training and testing results of the best variants of all estimation methods are summarized and compared. The experimental results and the analysis are presented in the next section [20].

6.4 Experimental Results

Tables 2, 3 and 4 present a summary of all the methods applied on three PROMISE repository datasets given in Table 1. The second column, third column and fourth column in each table shows the performance of various methods with respect to performance metrics MMRE, MdMRE and PRED (0.25) respectively. Similarly, the fourth and fifth column represents SA and DELTA respectively.With these values it can be interpreted that the testing results in the proposed methods outperform the testing results of FLANN, ANN. The FAS model has comparable results with PSO-FLANN in terms of MMRE. MdMRE and PRED (0.25) and performs well in terms of DELTA in NASA93 and Coc81compared to PSO-FLANN.

Our experiments suggest that a hybrid combination of ACTIVE-FLANN-PSO (FAS) improves the accuracy very effectively when compared to FLANN, ANN and has comparable results with respect to PSO-FLANN.

7 Conclusion and Future Work

Software cost estimation by hybrid system using Active learning, PSO and FLANN-an improvement of PSO-FLANN, has been presented in this work. We have evaluated the performance of FAS. The experimental results show that our method gives improved performance as compared to conventional FLANN, ANN and outperforms the competitive techniques such as PSO-FLANN with respect to performance measure DELTA. As a future note, our best effort towards the development of a good effort estimation technique will be based on meta-heuristic techniques like artificial bee colony (ABC) algorithm, and stochastic technique like differential evaluation (DE), and simulated annealing.

References

1. Bakr, A., Turhan, B., Bener, A.: A comparative study of estimating software development effort intervals. Softw. Qual. J. **19**, 537–552 (2010)
2. de Araujo, R.A., Oliveria, A.L.I., Soares, S.: A shift-invariant morphological system for software development cost estimation. Expert Syst. Appl. **38**, 4162–4168 (2011)
3. Braga, P.L., Oliveria, A.L.I., Ribeiro, G.H.T., Meria, S.R.L.: Software effort estimation using machine learning techniques with robust confidence intervals. In: Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI), pp. 181–185 (2007)
4. Kocaguneli, E., Menzies, T., Keung, J., Cok, D., Madachy, R.: Active learning and effort estimation: Finding the essential content of software effort estimation data. IEEE Trans. Softw. Eng. **39**(8), 1040–1053 (2013)
5. Foss, T., Stenrud, E., Kitchenham, B., Myrtveit, I.: A Simulation study of the model evaluation criterion MMRE. IEEE Trans. Softw. Eng. **29**(11), 985–995 (2003)
6. Keung, J.W.: Theoretical maximum prediction accuracy for analogy-based software cost estimation. In: Proceedings of 15th Asia-Pacific Software Engineering Conference, pp. 495–502 (2008)
7. Menzies, T., Caglayan, B., Kocaguneli, E., Krall, J., Peters, F., Turhan, B.: The PROMISE Repository of Empirical Software Engineering Data. Department of Computer Science, West Virginia Universit (2012). http://promisedata.googlecode.com
8. Stensrud, E., Foss, T., Kitchenham, B.A., Myrtveit, I.: An empirical validation of the relationship between the magnitude of relative error and project size. In: Proceedings of the IEEE 8th Metrics Symposium, pp. 3–12 (2002)
9. Tirimula Rao, B., Sameet, B., Kiran Swathi, G., Vikram Gupta, K., Raviteja, Ch., Sumana, S.: A novel neural network approach for software cost estimation using functional link artificial neural networks. Int. J. Comput. Sci. Netw. Secur. (IJCSNS) **9**(6), 126–131 (2009)
10. Tirimula Rao, B., Dehuri, S., Mall, R.: Functional link artificial neural networks for software cost estimation. Int. J. Appl. Evol. Comput. (IJAEC) **3**(2), 62–82 (2012)
11. Tirimula Rao, B., Chinnababu, K., Mall, R., Dehuri, S.: Particle swarm optimized functional link artificial neural networks (PSO-FLANN) n 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)
12. Dehuri, S., Cho, S.-B.: Evolutionarily optimized features in functional link neural network for classification. Expert Syst. Appl. **37**(6), 4379–4391 (2010)
13. Dehrui, S., Cho, S.-B.: A comprehensive survey on functional link neural networks and an adaptive PSO-BP learning for CFLNN. Neural Comput. Appl. **19**(2), 187–205 (2010)
14. Tirimula Rao, B., Dehuri, S., Mall, R.: Computational intelligence in software cost estimation: an emerging paradigm. ACM SIGSOFT Softw. Eng. Notes **37**(3), 1–7 (2012)
18. Chakravarty, S., Dash, P.L., Pandi, V.R., Panigrahi, B.K.: An evolutionary functional link neural fuzzy model for financial time series forecasting. Int. J. Appl. Evol. Comput. **2**(3), 27–38 (2011)
16. Cohen, J.: Quantitative methods in psychology: a power primer. Psychol. Bull. **112**(1), 155–159 (1992)
17. Minku, Leondro L., Yao, Xin: Ensembles and locality: insight on improving software effort estimation. Inf. Softw. Technol. **55**(8), 1512–1528 (2013)
18. Shepperd, M., MacDonell, S.: Evaluating prediction systems in software project estimation. Inf. Softw. Technol. **54**(8), 820–827 (2012)
19. Abutheraa, M.A., Lester, D.: Computable function representations using effective Chebyshev polynomial. Int. J. Math. Comput. Phys. Quantum Eng. **1**(7), 294–300 (2007)

20. Tirimula Rao B., Mall, R., Dehuri, S., ChinnaBabu, K.: Software effort prediction using unsupervised learning (clustering) and functional link artificial neural networks. In: Proceedings of the IEEE World Congress on Information and Communication Technologies, India, pp. 115–120 (2012)
21. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942–1948 (1995)