Comparison of Back Propagation Training Algorithms for Software Defect Prediction

Ishani Arora[1], Anju Saha[2]
University School of Information and Communication Technology
Guru Gobind Singh Indraprastha University
Dwarka, Delhi - 110078, India
[1]ishaniarora06@gmail.com, [2]anju_kochhar@yahoo.com

***Abstract*— The cost of deleting a software bug increases ten times as it is floated onto the next phase of software development lifecycle (SDLC). This makes the task of the project managers difficult and also degrades the quality of the output software product. Software defect prediction (SDP) was proposed as a solution to the problem which could anticipate the defective modules and hence, deal with them in an efficient and effective manner in advance. The adequacy of artificial neural networks (ANNs) to handle the complex nonlinear relationships between the software metrics and the defect data demonstrates their suitability to build the defect prediction models. In this paper, multilayer feed forward back propagation based neural networks were constructed using seven defect datasets from the PROMISE repository. An empirical comparison of Levenberg-Marquardt (LM), Resilient back propagation (RP) and Bayesian Regularization (BR) back propagation training algorithms was performed using statistical measures such as MSE and R^2 values and the parameters computed from the confusion matrix. Bayesian based back propagation training method performed better than the LM and RP techniques in terms of minimizing mean square error and type II error and maximizing accuracy, sensitivity and R^2 value. An accuracy of more than 90 percent was achieved by BR on all the seven datasets and the best data fit during the regression analysis was shown with a R^2 value of 0.96. Overall, it is the context and the criticality of the software project which will aid the project managers to prioritize the performance measures and hence, decide upon the training algorithm to be applied, according to the goals and resources available.**

Keywords— artificial neural network, bayesian regularization, back propagation, Levenberg-Marquardt, resilient back propagation, software defect, software metrics

I. INTRODUCTION

The software development lifecycle incorporates activities such as testing, maintenance, or refactoring, which are equally costly as the software coding stage. This is because all the tasks need to be performed on all the modules of a large software project. Hence, the total cost of the software process touches the peak of the project's budget. This problem is addressed by software defect prediction (SDP), which aids in determining only those components of the project which require testing or refactoring. The project managers are therefore, made aware of the modules which require attention and hence, handle the tasks carefully. The SDP benefits in a way that it is able to anticipate the defective modules during the early stages of software development process and hence, reduce the construction cost and time, thereby improving the software quality [1].

Different researchers have proved the relation between the software metrics and the probability of occurrence of defect [2, 3]. Software defect prediction is, therefore treated as a binary classification problem in which the modules are classified either defective or non-defective. The defect prediction models are developed using the software metrics or features along with the defect data. The labeled or the historical data about a software project help to construct a prediction model which forecasts the membership of the module under development to either of the classes.

The inability of the prior statistical SDP approaches to achieve the desired performance has introduced a variety of machine learning techniques such as Naive Bayes [4], fuzzy logic [5], data mining [6] and artificial neural networks (ANNs) [7]. The linear models could not explore the complex relationships between the metrics and defect proneness, which led to the development of nonlinear SDP models, thereby providing a better performance than their counterparts.

Neural networks are adaptive computational models consisting of densely interconnected processing units called neurons. The basic architecture of an ANN is shown in Figure 1. Each input is multiplied by a weight value which acts as an input for each neuron. The neuron uses a transfer function which produces the output for each corresponding layer. The final output is produced in the output layer. The layers between the input and output layers are called hidden layers. The fundamental characteristic of these parallel operating architectures is the ability to self learn and discover knowledge. The following properties are inherently possessed by a neural network:

- Ability to approximate the complex nonlinear relationships
- Simple structure and the ability to expand
- Self learning capability and discovering knowledge

Artificial Neural Networks (ANNs) were introduced in the early 1950s as a major component of artificial intelligence. Today, due to its ability to solve a large number of complex real world problems in data mining and machine learning domain, it has gained momentum. The major class of problems solved by ANNs includes the regression and classification problems [8, 9]. ANNs hold the advantage of learning from the observed data and approximate any arbitrary function and hence, can fix a nonlinear complex function.

978-1-5090-5256-1/16/$31.00 ©2016 IEEE

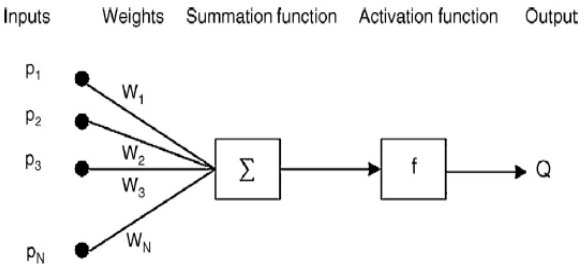


Fig. 1. General Architecture of a Neural Network

ANNs are a widely accepted and applied technique for solving various problems in the field of software, such as software effort estimation [10], cost estimation [11], optimization problems [12] and software defect prediction [13]. The application of artificial neural networks (ANNs) for building defect prediction models in SDP has been the focus of research in the recent times [14, 15]. These research studies have shown that ANNs perform better as compared to the traditional SDP models. Also, ANN meets the requirements of the SDP models in a way that these are built using software metrics and do not demand any expert knowledge, which is another advantage of ANNs.

This research study is carried out with an objective to determine the learning algorithm which performs the best among the three variants of the back propagation training algorithms, namely, Levenberg-Marquardt (LM), Resilient Back Propagation (RP) and Bayesian Regularization (BR), for software defect prediction. The paper is organized as follows: Section II provides a brief overview of the application of ANNs for defect prediction by the various researchers and practitioners. Section III describes the back propagation algorithm and its three forms studied in this work. Section IV presents the experiments conducted, which details the data collection process, data characteristics and the performance evaluation measures. The results and discussion are presented in Section V and Section VI, respectively. Conclusions are described in Section VII.

II. Background Study

The choice of a training algorithm for an artificial neural network is a major factor in determining its classification performance. One major class of training functions applied by the researchers and practitioners since the earliest times is the back propagation (BP) algorithms.

Lanubile et al. [16] performed an empirical comparison of six classification approaches: discriminant analysis, principal component analysis, logistic regression, layered neural networks, holographic networks and logical classification models. The experiment was performed on 27 academic projects. Neural networks were trained using back propagation learning algorithm. Predictive validity, misclassification rate, achieved quality and verification cost were used as performance measures. However, none of the models could achieve the desired classification performance criterion.

Neumann [17] presented a hybrid model based on principal component analysis (PCA) and ANNs in order to identify the high risk software components. PCA was employed for feature extraction while the task of risk classification was performed by ANNs. A 2 layer feed forward back propagation ANN architecture was used and compared with a single perceptron and a single layer Adaline. The combined approach proved to be better than either of the methods by itself in terms of the mean number of errors. The best performance resulted when a two layer back propagation ANN network with four hidden layers was employed.

Kanmani et al. [18] empirically compared two architectures of neural networks – back propagation NN (BPNN) and probabilistic NN (PNN) for estimating the defect proneness of the C++ classes. The data were collected from an academic project and the results were compared using two statistical approaches: discriminant analysis and logistic regression. Five quality parameters were used as performance evaluation measures and it was concluded that the PNN outperformed back propagation training algorithm in terms of software defect prediction.

Zheng [13] employed back propagation learning algorithm based three cost-sensitive boosting algorithms for software defect prediction. The three neural networks included one based on threshold and the other two based on weight updating architectures. The experiment was conducted using four NASA public datasets using normalized expected cost of misclassification as the evaluation measure. The results suggested that the threshold based back propagation feed forward neural network performed better than the other counterparts, especially for the object oriented software projects.

Different practitioners have applied back propagation learning algorithm in its variant forms, which performs well in some situations and degrades in others. This encourages us to perform an empirical comparison of the three most used forms of the BP training algorithms: Levenberg-Marquardt (LM), Resilient Back Propagation (RP) and Bayesian Regularization (BR), and hence, determine which among these three is the best in the context of software defect prediction.

III. Backpropagation Algorithm

Back propagation (BP) is the earliest error based approach of training a multi-layer feed forward artificial neural network [19] using supervised learning. The method works towards the minimization of the error between the actual output and the desired output and achieving an optimized set of synaptic connection weights in order to attain the required system's performance. The neural network training continues until the maximum number of iterations (epochs) or the minimum acceptable error is reached. After a network has been trained on a set of inputs, it can be tested on another set of inputs, to determine how well it generalizes on the set of untrained inputs. The performance is usually evaluated in terms of mean squared error (MSE) or root mean square error (RMSE) as follows:

$$MSE = \frac{\sum(t_i - o_i)^2}{n} \quad (1)$$

$$RMSE = \sqrt{MSE} \quad (2)$$

where, t_i is the required output, o_i is the actual output and n is the total number of instances in the data set. The three mostly used learning functions of BP algorithms are discussed below:

- Levenberg-Marqaurdt (LM) Algorithm: LM is used as a solution to most of the nonlinear least square problems. It was introduced as an intermediate between the Gauss Newton method and the method of gradient descent [20]. The Newton method approaches the second order expression during the NN training while the LM approximates the error in the first order expression. It uses the following approximation to achieve the optimized set of weights during the training phase:

$$x_{k+1} = x_k - [J^T J + \mu I]^{-1} J^T e \quad (3)$$

where, J is the Jacobian matrix consisting of the first derivatives of the network errors, e is the error vector, I is the identity matrix and μ is the network gain parameter. Another reason of LM algorithm's adequacy over Newton's method is the simple computation of the Jacobian matrix than the complex Hessian matrix in the latter form. Another advantage of LM is that it holds the fastest approach for training medium sized multi-layer feed forward networks. However, Jacobian matrix assumes that the evaluation measure is a sum or mean square errors. Hence, the networks must use either MSE or SSE as the performance function.

- Resilient Back Propagation (RP) Algorithm: Similar to LM, RP [21] is also a first order optimization algorithm for supervised learning. The motivation behind the RP algorithm was to remove the disadvantage of multilayer network when trained with steepest descent and sigmoid activation function. This method uses only the sign of the first order derivative to decide upon the direction of the weight update while the magnitude has no impact on the weight update. The update value is increased by a factor if the derivative of the performance function has the same sign for two consecutive iterations. On the other hand, the value is decreased by a factor whenever the derivative changes its sign from the previous iteration. However, it remains the same if the derivative is zero. RP algorithm is faster than the steepest descent training approach and does not demand much on the memory requirements. The algorithm is also not very sensitive to the training parameters.

- Bayesian Regularization (BR) Algorithm: BR approach [22] works similar to the LM optimization, in a sense that it minimises the squared errors and the weights and finds the optimal combination so that the network performs well. Bayesian based NN training is more robust than the standard back propagation nets. BR is applied as a solution for the optimization of a network architecture, validation effort, choice of validation set and checking the robustness of a model. The advantage of it being difficult to over train and over fit proves BR learning algorithms to be more adequate than the other training functions.

These three BP training algorithms were tuned using different parameters in order to achieve the required classification performance. Table I describes these parameters and their values used during the conduct of the experiments.

TABLE I. PERFORMANCE PARAMETERS USED

S. No.	Parameter	Value
LM Algorithm		
1	Max. no. of epochs	1000
2	Performance goal	0
3	Training gain	0.001
4	Epochs between displays	25
5	Max. time to train in seconds	Inf
6	Max. validation failures	6
7	Min. performance gradient	1-e7
RP Algorithm		
1	Max. no. of epochs	1000
2	Performance goal	0
3	Max. time to train	Inf
4	Min. performance gradient	1-e5
5	Max. validation failures	6
6	Epochs between displays	25
7	Learning rate	0.01
BR Algorithm		
1	Max. no. of epochs	1000
2	Performance goal	0
3	Marquardt parameter	0.005
4	Min. performance gradient	1-e7
5	Max. value of training gain	1e10
6	Epochs between displays	25
7	Max. validation failures	6
8	Max. time to train	Inf

IV. EXPERIMENTS

A. Data Collection

The study was conducted using seven data sets: PC1, PC2, PC3, PC4, PC5, KC2 and KC3 from the PROMISE repository [23]. The properties of the defect data sets are described in Table II. Table III lists the Halstead metrics [24], McCabe metrics [25] and Lines of Code (LOC) static code metrics used to predict the defect proneness of the modules. A module is treated as the smallest indivisible unit of functionality. These attributes were used as independent variables for determining the probability of the occurrence of a defect which was treated as the dependent variable.

TABLE II. DATA SETS INVOLVED

Data set	Metric	Samples	Defect
PC1	38	565	7.61
PC2	37	745	2.15
PC3	38	1077	12.44
PC4	38	1458	12.21
PC5	39	17186	3.00
KC2	22	522	20.50
KC3	40	194	18.56

A. Performance Evaluation Measures

The defect prediction models developed using levenberg-marquardt, resilient back propagation and bayesian

regularization training algorithms were empirically compared using the following performance evaluation measures for all the 7 datasets involved.

TABLE III. COMMON SOFTWARE METRICS

McCabe	Cyclomatic Complexity
	Design Complexity
	Essential Complexity
	Normalized Cyclomatic Complexity
Halstead	Num Operands
	Num Operators
	Num Unique Operands
	Num Unique Operators
	Error Estimate
	Length
	Level
	Program time
	Effort
	Difficulty
	Volume
	Content
Lines of Code (LOC)	Blank
	Code and Comments
	Executable
	Comments
	Total

1) Error based measures

The class of back propagation functions are error based optimization functions. The performance is therefore, compared in terms of MSE or RMSE as described by (1) and (2).

2) R value

The robustness of a modelling technique can be determined by the regression analysis. The R^2 value of the test data is computed to know how well the technique fits the data. $R^2 > 0.9$ is usually treated as a good fit [26].

3) Confusion matrix

		Actual/Target class	
		No	Yes
Predicted/Output class	No	TN	FN
	Yes	FP	TP

Fig. 2. Confusion matrix

where, TN= True negative=the no. of non-defective modules which are correctly predicted to be non-defective
FN=False negative=the no. of defective modules incorrectly classified as non-defective
FP =False positive=the no. of non-defective modules incorrectly classified as defective
TP=True positive=the no. of defective modules correctly predicted as defective

Figure 2 shows the confusion matrix which is usually used as a standard for determining how well a classification model performs. The different measures which are computed from a confusion matrix are shown as follows.

a) Accuracy

Accuracy is a measure of the number of modules which are correctly predicted, whether defective or non-defective, out of the total number of modules.

$$Accuracy = \frac{TP+TN}{TP+TN+FN+FP} \quad (4)$$

b) Sensitivity

Recall is another indicator of the number of predicted defective modules out of the total number of actually defective modules. Recall is sometimes also called as the sensitivity or probability of detection or true positive rate (TPR).

$$Sensitivity = \frac{TP}{TP+FN} \quad (5)$$

c) Specificity

True negative rate (TNR) or the specificity determines the number of predicted non-defective modules out of the total number of actually non-defective modules and is calculated as:

$$Specificity = \frac{TN}{FP+TN} \quad (6)$$

V. RESULTS

The experiments were conducted in the MATLAB environment 2015. MATLAB implementations of the LM, BR and RP training algorithms are shown in Table IV. Multilayer feed forward neural network architectures with a single hidden layer having default 10 neurons were used. The input layer has the number of neurons equal to the number of attributes in the data set and there are two neurons in the final layer belonging to the class of non-defective and defective modules. Each input data set was divided randomly into three parts: training set, validation set and test set. The obtained results are presented in Table V-XI, in terms of the performance indicators as described in Section IV.

TABLE IV. MATLAB IMPLEMENTATION FUNCTIONS

S. No.	Algorithm	Matlab Implementation
1	Resilient back propagation	trainlm
2	Resilient back propagation	trainrp
3	Bayesian Regularization	trainbr

TABLE V. ACCURACY

Data set	LM	RP	BR
PC1	93.63	92.38	96.28
PC2	97.85	97.85	97.58
PC3	87.93	87.09	93.50
PC4	91.08	87.65	92.66
PC5	97.31	97.24	98.53
KC2	83.33	81.99	91.18
KC3	84.53	81.44	90.72

TABLE VI. R SQAURE VALUE OF THE TEST SET

Data set	LM	RP	BR
PC1	0.84	0.84	0.51
PC2	0.94	0.95	0.96
PC3	0.77	0.76	0.50
PC4	0.82	0.79	0.31
PC5	0.96	0.95	0.89
KC2	0.70	0.73	0.39
KC3	0.78	0.60	0.34

TABLE VII. SENSITIVITY

Data set	LM	RP	BR
PC1	34.88	2.3	81.39
PC2	0.00	0.00	0.00
PC3	7.46	2.23	70.89
PC4	36.51	11.23	74.16
PC5	19.76	16.86	63.37
KC2	29.91	20.56	69.16
KC3	19.44	0.00	80.55

TABLE VIII. SPECIFICITY

Data set	LM	RP	BR
PC1	98.46	99.81	97.51
PC2	100.00	100.00	99.72
PC3	99.36	99.15	96.71
PC4	98.67	98.28	95.23
PC5	99.71	99.73	99.62
KC2	97.11	97.83	96.87
KC3	99.36	100.00	93.04

TABLE IX. MSE AND RMSE

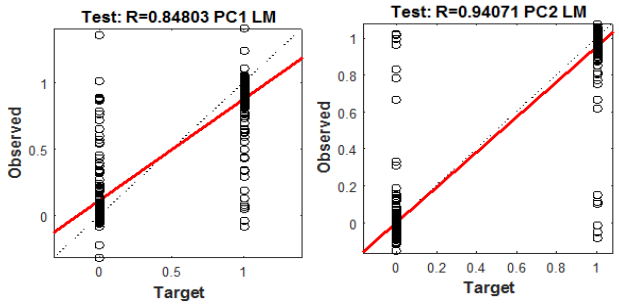
Data set	LM		RP		BR	
	MSE	RMSE	MSE	RMSE	MSE	RMSE
PC1	0.07	0.26	0.05	0.22	1.11e-10	0.00
PC2	0.03	0.17	0.02	0.14	0.03	0.17
PC3	0.11	0.33	0.09	0.3	0.02	0.14
PC4	0.06	0.24	0.08	0.28	0.01	0.1
PC5	0.02	0.14	0.02	0.14	0.01	0.1
KC2	0.10	0.31	0.15	0.38	0.03	0.17
KC3	0.22	0.47	0.18	0.42	4.91e-16	2.21e-8

TABLE X. FALE NEGATIVE RATE (TYPE II ERROR)

Data set	LM	RP	BR
PC1	65.11	97.67	18.60
PC2	100.00	100.00	100.00
PC3	92.54	97.76	29.10
PC4	63.48	88.76	25.84
PC5	80.23	83.14	36.62
KC2	70.09	79.44	30.84
KC3	80.55	100.00	19.44

TABLE XI. FALSE POSITIVE RATE (TYPE I ERROR)

Data set	LM	RP	BR
PC1	1.53	0.19	2.49
PC2	0.00	0.00	0.27
PC3	0.64	0.84	3.28
PC4	1.33	1.71	4.76
PC5	0.28	0.27	0.37
KC2	2.89	2.16	3.13
KC3	0.63	0.00	6.96



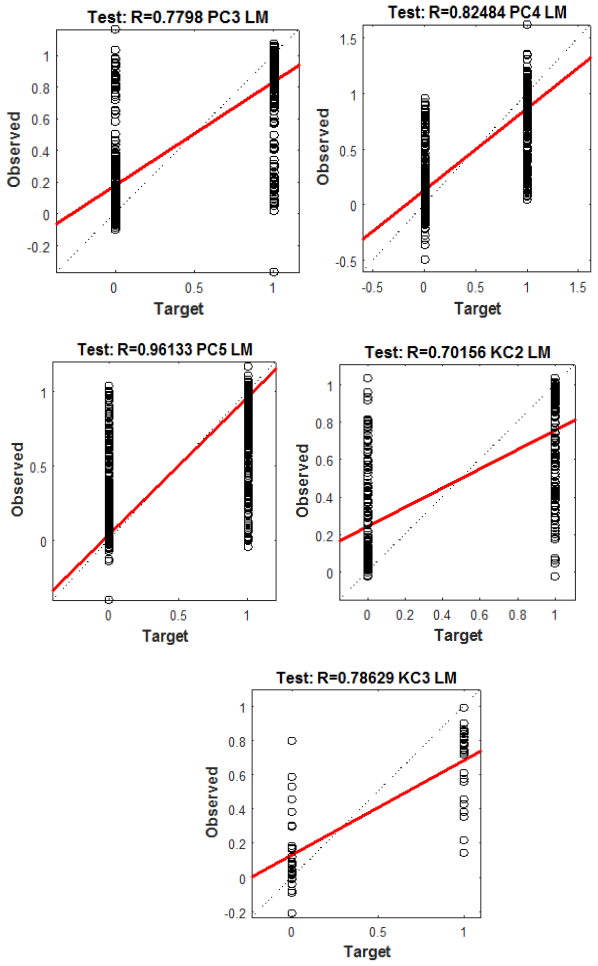
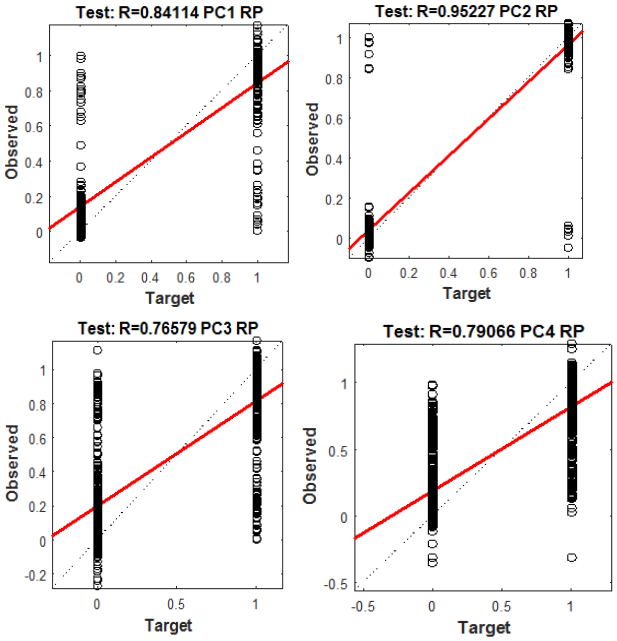


Fig. 3. Regression plots of observed vs. target plots of LM Algorithm for the seven datasets.



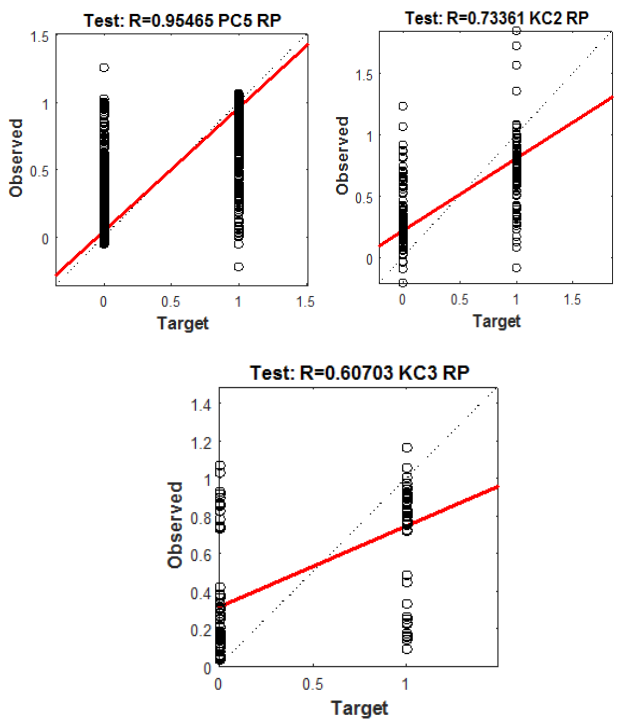
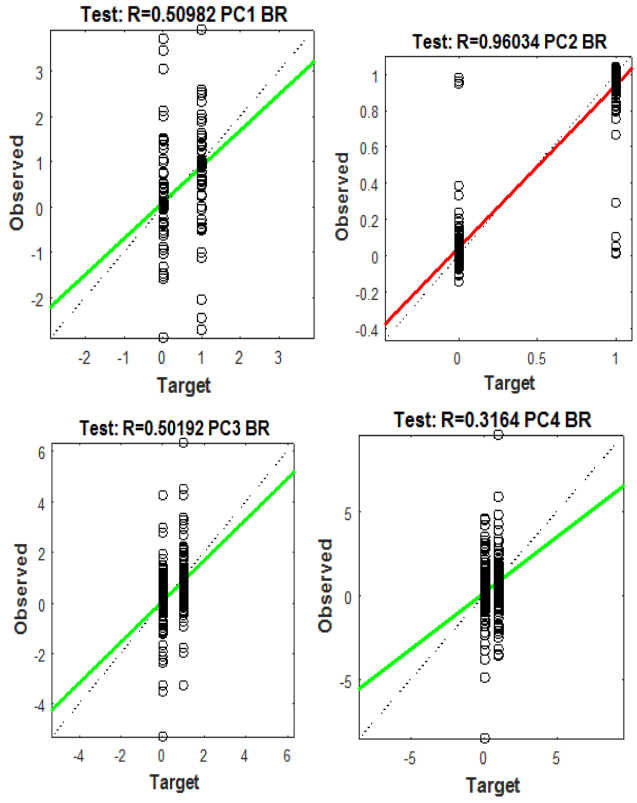


Fig. 4. Regression plots of observed vs. target values of RP algorithm for the seven datasets



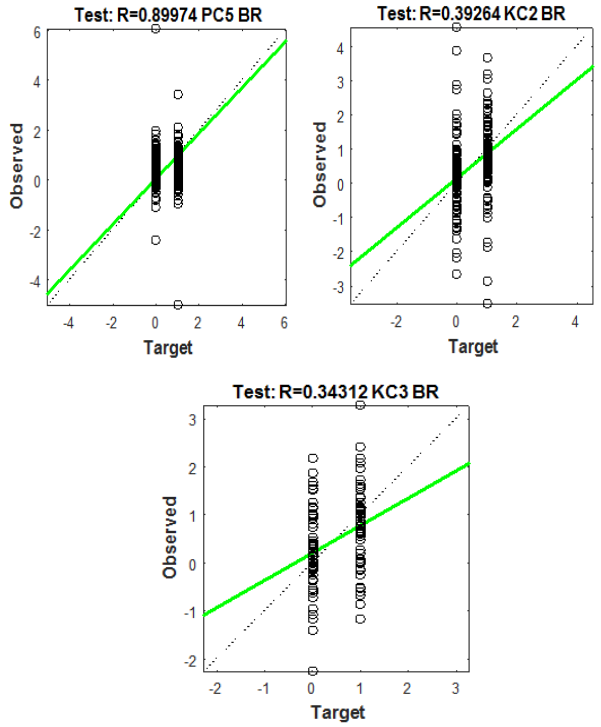


Fig. 5. Regression plots of observed vs. target plots of BR algorithm for the seven datasets

VI. Discussion

This section discusses the results obtained in section V. Table IX clearly shows Bayesian regularization (BR) algorithm as the winner on six out of seven datasets with minimum MSE of 4.91e-16 for KC3 and resilient propagation (RP) outperformed only for PC2 with MSE of 0.02. Levenberg-Marquardt (LM) method could not perform for any of the datasets.

However, the R^2 values are totally contradictory. The higher the R^2 result, the better the fit. LM and RP performed equally well only for PC1 with 84 percent data fit. Overall, in four out of seven data sets, LM proved to be better than the other two counterparts. Usually, $R^2 > 0.9$ is usually treated as a good fit [26]. According to this criterion, LM and BR proved to be robust modeling techniques for PC5 and PC2 defect datasets, respectively.

Similar to the error criterion, Bayesian based training function surpassed the LM and RP methods on the accuracy and sensitivity parameters with more than 90 percent accuracy on all the datasets as shown in Table V and Table VII, respectively. Also, BR achieved the maximum accuracy on six out of seven cases. Accuracy, however, can often give a false performance impression due to the defect datasets being unbalanced. Although LM and RP outperformed BR in terms of specificity, Table VIII, but specificity is not an important performance indicator as sensitivity is, because it is essential to classify the defective modules correctly rather than the non-defective modules.

Type II error is the type of error which occurs when the defective modules are predicted to be non-defective and hence, they are ignored. Type II error is also called as false negative. On the other hand, the type I error or the false positive occurs when the non-defective modules are predicted to be defective. Type II error proves to be more costly than the type I error because it is crucial to predict the defective modules correctly over the non-defective modules. The empirical comparison of Table X and Table XI clearly shows that Bayesian regularization based back propagation training algorithm outperforms the resilient and Levenberg-Marquardt based methods in terms of software defect classification.

VII. Conclusion

This paper performs an empirical study for the performance comparison of three standard back propagation based training algorithms, i.e. Laverberg-Marquardt (LM), Resilient back propagation (RP) and Bayesian Regularization (BR) in the context of software defect prediction. Multilayer feed forward artificial neural network was built using the MATLAB command line interface. Experiments were conducted using seven defective data sets from the PROMISE repository. The classification models were compared in terms of parameters calculated from confusion matrix and statistical measures such as MSE, RMSE and R^2 value. An overall comparison has shown that BR executed better than LM and RP in terms of MSE, R^2 value, accuracy, recall and false negative rate. The results have shown that it is the context and the criticality of the software project which will aid the project managers to prioritize the performance measures and hence, decide upon the training algorithm to be applied, according to the goals and resources available.

The back propagation training algorithms belong to the class of optimization algorithms where the weights are required to be optimized in order to achieve the best performance. Research has also shown the application of nature-inspired search based optimization algorithms in the field of software engineering. The future work may include an empirical study of the back propagation learning functions and the search based techniques in the context of software defect prediction.

References

[1] Y. Singh, A. Kaur and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models", Software Quality Journal, vol. 18, pp. 3-35, 2010.

[2] S. Lessmann, B. Baesens, C. Mues and S. Pietsch, "Benchmarking classification models for software defect prediction: a proposed framework and novel findings", IEEE Transactions on Software Engineering, vol. 34, pp. 485-496, 2008.

[3] E. Erturk and E. A. Sezer, "A comparison of some soft computing methods for software fault prediction", Expert Systems with Applications, vol. 42, pp. 1872-1879, 2015.

[4] C. Cagatay, S. Ugur and D. Banu, "Practical development of an eclipse-based software fault prediction tool using Naive Bayes algorithm", Expert Systems with Applications, vol. 38, pp. 2347-2353, 2011.

[5] G. Abaei and A. Selamat, "Software fault prediction based on improved fuzzy clustering", in 11th International Conference on Distributed Computing and Artificial Intelligence, Salamanca, Spain, pp. 165-172, 2014.

[6] G. Czibula, Z. Marian and I. G. Czibula, "Software defect prediction using relational association rule mining", Information Sciences, vol. 264, pp. 260-278, 2014.

[7] M. M. T. Thwin and T. S. Quah, “Application of neural networks for software quality prediction using object-oriented metrics”, Journal of Systems and Software, vol. 76, pp. 147-156, 2005.
[8] B. M. Ozyildirim and M. Avci, “Generalized classifier neural network”, Neural Networks, vol. 39, pp. 18-26, 2013.
[9] S. Chakraverty and S. Mall, “Regression-based weight generation algorithm in neural network for solution of initial and boundary value problems”, Neural Computing and Applications, vol. 25, pp. 585-594, 2014.
[10] G. R. Finnie and G. E. Wittig, “A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models”, Journal of Systems and Software, vol. 39, pp. 281-289, 1997.
[11] A R Venkatachalam, “Software cost estimation using artificial neural networks”, in Proceedings of International Joint Conference on Neural Networks, Nagoya, Japan, pp. 987-990, 1993.
[12] J-R. Zhang, J. Zhang, T-M. Lok and M. R. Lyu, “A hybrid particle swarm optimization - back-propagation algorithm for feedforward neural network training”, Applied Mathematics and Computation, vol. 185, pp. 1026-1037, 2007.
[13] J. Zheng, “Cost-sensitive boosting neural networks for software defect prediction”, Expert Systems with Applications, vol. 37, pp. 4537-4543, 2010.
[14] C. Jin and S-W. Jin, “Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization”, Applied Soft Computing, vol. 35, pp. 717-725, 2015.
[15] B-J. Park, S-K. Oh and W. Pedrycz, “The design of polynomial function-based neural network predictors for detection of software defects”, Information Sciences, vol. 229, pp. 40-57, 2013.
[16] F. Lanubile, A. Lonigro and G. Visaggio, “Comparing models for identifying fault-prone software components”, in Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering, Rockville, Maryland, USA, pp. 312-319, 1995.
[17] D. E. Neumann, “An enhanced neural network technique for software risk analysis”, IEEE Transactions on Software Engineering, vol. 28, pp. 904-912, 2002.
[18] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan and P. Thambidurai, “Object-oriented software fault prediction using neural networks”, Information and Software Technology, vol. 49, pp. 483-492, 2007.
[19] D. E. Rumelhart, G. E. Hinton and R. J. Williams, “Learning representations by back-propagation errors”, Nature, vol. 323, pp. 533-536, 1986.
[20] J. J. More, “The Levenberg-Marquardt algorithm: implementation and theory”, in Lecture Notes in Mathematics, Numerical Analysis, Springer Berlin Heidelberg, Berlin, pp. 105-116, 2006.
[21] M. Riedmiller and H. Braun, “Rprop – A fast adaptive learning algorithm”, in Proceedings of the 7th International Symposium on Computer and Information Science, 1992.
[22] D. J. C. MacKay, “A practical Bayesian framework for back propagation networks”, Computation and Neural Systems, vol. 139, pp. 448-472, 1992.
[23] T. Menzies, R. Krishna and D. Pryor, “The promise repository of empirical software engineering data”, http://openscience.us/repo. North Carolina State University, Department of Computer Science, 2016.
[24] T. M. Khoshgoftaar, K. Gao and R. M. Szabo, “An application of zero inflated poisson regression for software fault prediction”, in ISSRE 2001: Proceedings of 12th International Symposium on Software Reliability Engineering, Kowloon, Hong Kong, pp. 66-73, 2001.
[25] T. J. McCabe and C. W. Butler, “Design complexity measurement and testing’, Communications of the ACM, vol. 32, pp. 1415-1425, 1989.
[26] J. Bourquin, H. Schmidli, P. V. Hoogevest and H. Leuenberger, “Comparison of artificial neural networks (ANN) with classical modeling techniques using different experimental designs and data from a galenical study on a solid dosage form”, European Journal of Pharmaceutical Sciences, vol. 6, pp. 287–300, 1998.