International Journal of Soft Computing 12 (4): 199-203, 2017
ISSN: 1816-9503

A Comparative Analysis of Hybrid Learning over Back Propagation for Identifying Defective Prone Modules

[1]Satya Srinivas Maddipati, [2]A. Yesubabu and [3]G. Pradeepini
[1]K L University, Vijayawada, India
[2]Department of CSE, CRR College of Engineering, Eluru, Andhra Pradesh, India
[3]Department of CSE, K L University, Vijayawada, India

Abstract: The quality of software is improved by identifying defective prone modules which is influenced by various characteristics of software module like lines of code, Halstead metrics and cyclometric complexity values. There are various prediction models for identifying defective prone modules from these characteristics. In this study we are comparing neural networks and Adaptive Neuro Fuzzy Inference System (ANFIS) for software defect prediction. We applied gradient descendent learning for Neural Networks and Hybrid Leaning for ANFIS. The performance of the models are evaluated by using the metric Area under ROC curve (AuC) values. In these experiments, we considered Software Defect Prediction Datasets download from NASA repositories. The results of ANFIS are found satisfactory compared to Neural Networks.

Key words: Hybrid learning, back propagation, adaptive neuro fuzzy inference system, neural networks, software defect prediction, gradient descendent learning

INTRODUCTION

Defect density, number of defects per unit volume of code is one of the measure for the quality of software. Low defect density rate causes high quality for software that means to produce high quality software the number of defects should be lowered. Predicting defects in a software module in advance greatly reduces the cost of software development. During 1970's Akiyama derived an equation between number of defects and lines of code, No. of defects:

$$(N) = 4.86+0.018\times LOC$$

Later Mc Cube found cyclometric complexity metrics for defining complexity of software module. In 1977, Halstead introduced Halsted metrics for defining software complexity. Laterm, there are various prediction models for best fitting prediction of software defects from these metrics (Chamoli *et al*., 2015).

There are various methodologies for identifying defective prone modules using data mining. The method include neural networks, support vector machines, adaboost and random forest. But none of the method shows best performance on all the datasets downloaded from NASA data repositories which includes thousands of modules.

Literature review: Jun Zheng applied cost sensitive boosting neural networks for software defect prediction (Zheng, 2010). Catal *et al.* (2009) proposed a new technique, X-means clustering for faults of software (Catal *et al*., 2009). This technique is fully automated no need to specify number of clusters but applicable to only un labelled program modules. Meenakshi *et al.* (2010) applied Expectation Maximization (EM) algorithm and Quad Tree algorithm for identification of fault modules. Pandey and Goyal (2010) proposed fuzzy inference system from the information gained using Decision Tree (ID3). Akalya Devi proposed a hybrid feature selection method that predicts software faults with better accuracy. Shanthini and Chandrasekaran (2012) used support vector machines for best prediction performance. They used method level metrics and class level metrics. Okutan and Yildiz (2014) used Bayesian networks to determine the probabilistic influential relationships among software metrics and defect proneness. Goyal used Euclidean distance probability to find the probability of defect.

MATERIALS AND METHODS

Back Propagation algorithm: Back propagation is a supervised learning method for multi layer Artificial Neural Networks. The Back Propagation Neural Network (BPNN) learns by calculating errors of output layer to find

Corresponding Author: Satya Srinivas Maddipati, K L University, Vijayawada, India

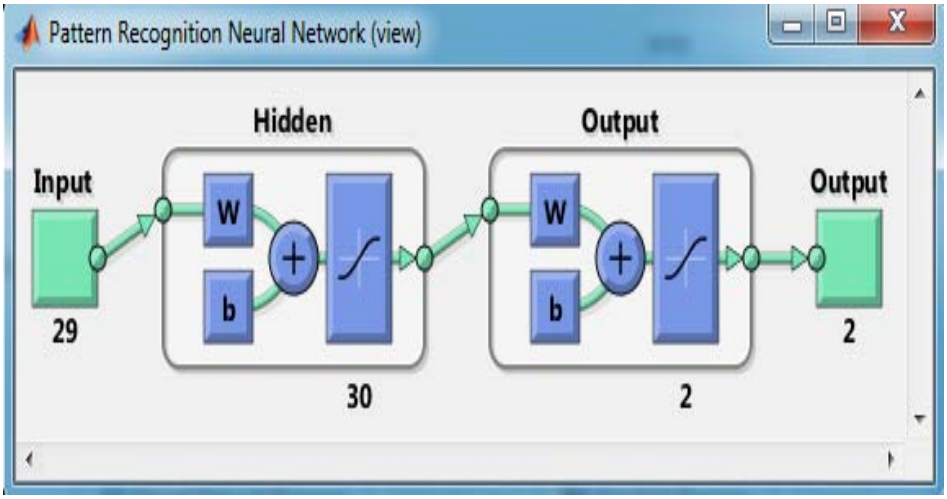

Fig. 1: Neural networks model structure

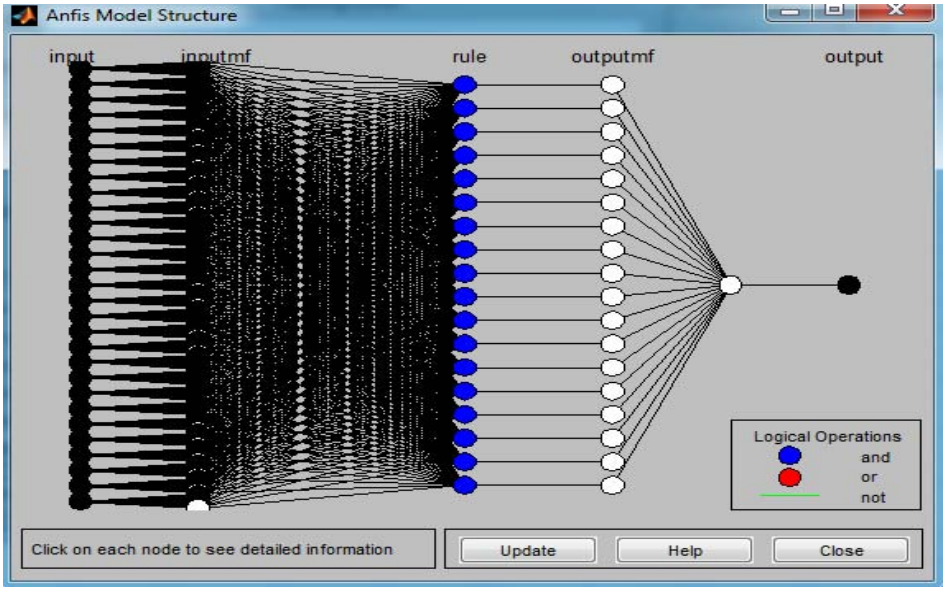

Fig. 2: ANFIS model structure

the errors in hidden layers. In BPNN, there are three layers input layer, hidden layer and output layer. The input patterns are presented to input layer and propagated towards output layer. The actual outputs are compared with target outputs and error values are measured. This is called as forward pass. These error values are back propagated until input layer and weights are adjusted in each layer by gradient descendent method. This is called backward pass (YegnaNarayana, 2005) (Fig. 1).

Adaptive neuro fuzzy inference system: Adaptive neuro fuzzy inference system is a five layered architecture, derives sugeno fuzzy inference system with hybrid learning rule. The first layer is an adaptive layer which computes the membership values of input variables. The second layer is composed with fixed nodes which determines the firing strength of the rule. Third layer consists of fixed node(s) for normalizing weight of the firing rule. Fourth layer is an adaptive layer with consequent parameters used for training dataset. Fifth layer is the output layer which sums all the inputs from previous layers (Fig. 2).

ANFIS uses a hybrid learning algorithm to identify the membership function parameters of single-output, sugeno type Fuzzy Inference Systems (FIS). Sugeno fuzzy inference system. A fuzzy rule in a sugeno fuzzy

model has the following form if x is in A and y is in B then z = f(x, y) where A, B are fuzzy sets in the antecedent and z = f(x, y) is a crisp function.

A combination of least-squares and back propagation gradient descent methods are used for training FIS membership function parameters to model a given set of input/output data.

Hybrid learning rule: In hybrid learning rule, there are two passes, forward pass, backward pass. In forward pass, the premises parameters are best estimated by method of least squares. In backward pass the consequent parameters are updated by method of gradient descendent.

Least square estimation: In general least square problem, the output 'y' of linear model is given by the expression:

$$Y = A\Theta \quad (1)$$

Where:

A = The matrix of values of independent variable

$\ominus$ = Th eset of coefficients that best fits the model

For best estimation of $\ominus$ least square estimation was applied. To identify unknown vector $\ominus$ from the above equation:

$$\Theta = A^{-1}Y \quad (2)$$

As the data might be contaminated by noise the model should be incorporated by noise vector and hence the expression is given by:

$$Y = A\Theta+\varepsilon \quad (3)$$

where, ε is the error vector. To minimize the error (Eq. 3) should be derivate w.r.t. ε and assign to zero. Now the equation becomes:

$$\Theta = \left(A^{T}A\right)^{-1}A^{T}Y \quad (4)$$

Gradient descendent method: Gradient descendent is a derivative based optimization method for minimizing a function defined on multi dimensional input space. This method is mostly used in non linear optimization problems. According to this method the parameter estimation for next iteration:

$$\Theta_{next} = \Theta_{old} - \eta g \quad (5)$$

Where:

η = The learning rate parameter

g = The amount of descendent in a particular direction

Table 1: Acc values of back propagation and hybrid learning on SDP datasets

Dataset\Algorithm	Back propagation	Hybrid learning
Cm_1	0.60	0.25
Kc_1	0.60	0.30
Kc_2	0.35	0.30
Pc_1	0.34	0.30

Performance metrics: The simple metrics for describing classifier performance are accuracy and error rate:

$$\text{Accuracy} = \frac{T_p+T_n}{P_c+N_c}, \quad \text{Error rate} = 1\text{-Accuracy}$$

Where:

T_p = True positives

T_n = True negatives

P_c = Number of positives

N_c = Number of negatives

But software defect prediction datasets are highly imbalanced in nature, so these simple metrics does not provide adequate information on classifier's functionality (Paramshetti and Phalke, 2014). In order to overcome such issues, ROC assessment makes use of true positive rate and false positive rates:

$$\text{TP rate} = \frac{T_p}{P_c}, \quad \text{FP rate} = \frac{F_p}{N_c}$$

ROC graph is formed by plotting TP rate over FP rate. Any classifier that appears in the upper right triangle of ROC space perform better than random guessing.

RESULTS AND DISCUSSION

Matrix laboratory introduced neuro fuzzy tool box for experimentation on adaptive neuro fuzzy inference system. It contains five modules, load dataset, generate initial FIS, train FIS using ANFIS, test FIS. Initial fuzzy inference system was generated in two ways, grid partitioning, subtractive clustering. Grid partitioning generates a rule for each possible value of attributes but subtractive clustering will perform clustering on the values of the attribute. Grid partitioning is applicable only for discrete attributes and Subtractive method will be applicable for numerical and continuous attributes. Table 1 shows error rate for back propagation and hybrid learning applied over software defect prediction datasets downloaded from NASA repositories. Figure 3 and 4 show ROC curves generated by back propagation and hybrid learning over software defect prediction.

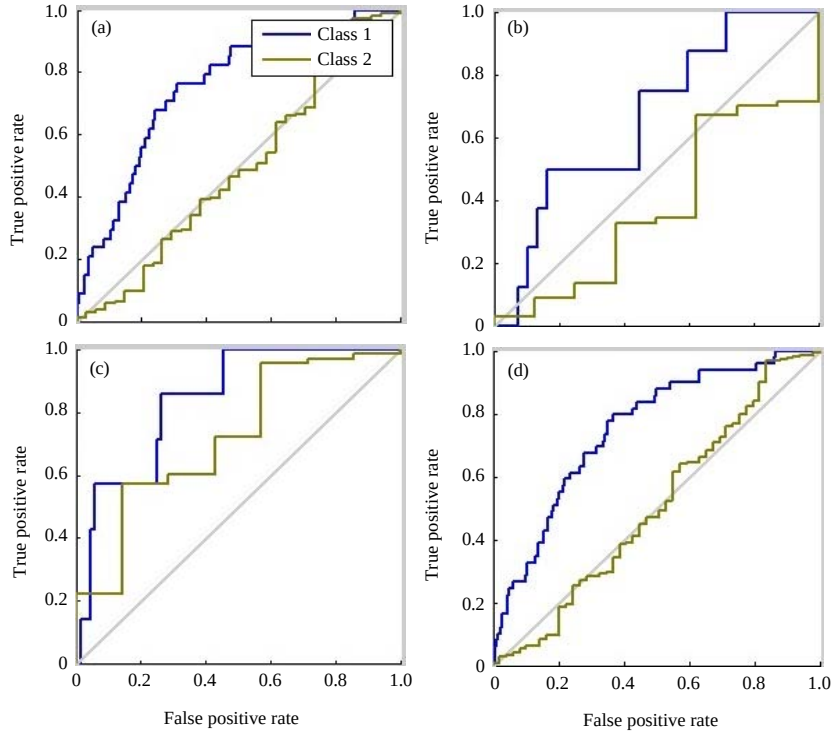

Fig. 3: ROC curves generated by back propagation: a) Training ROS; b) Validation ROC; c) Test ROC and d) All ROC

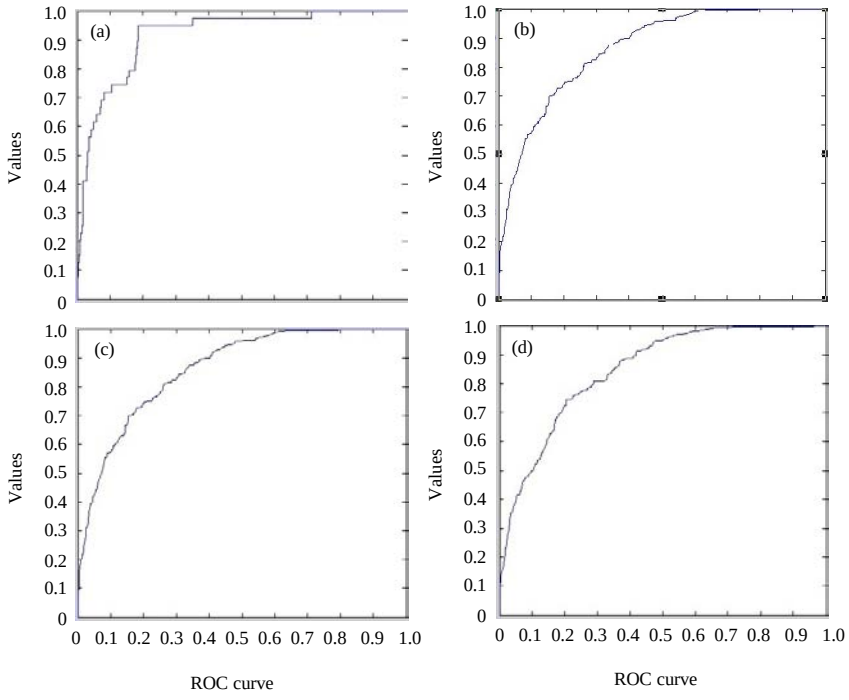

Fig. 4: ROC curves generated by ANFIS: a) ROC cueve cm_1; b) ROC cueve kc_1; c) ROC cueve kc_2 and d) ROC cueve pc_1

CONCLUSION

There are various prediction models for software defects like decision trees, random forests, support vector machines and neural networks. We applied neural networks and adaptive neuro fuzzy inference system on software defect prediction. Neural networks uses back propagation learning and ANFIS uses hybrid learning rule. We compared the results in terms of area under ROC curves. The performance of ANFIS is high compared to back propagation neural networks.

REFERENCES

Catal, C., U. Sevim and B. Diri, 2009. Clustering and metrics thresholds based software fault prediction of unlabeled program modules. Proceedings of the 6th International Conference on Information Technology: New Generations. Las Vegas, Nevada, April 27-29, pp: 199-204.

Chamoli, S., G. Tenne and S. Bhatia, 2015. Analysing software metrics for accurate dynamic defect prediction models. Indian J. Sci. Technol., 8: 96-100.

Meenakshi, P.C., S. Meenu, M. Mithra and P.L. Rani, 2010. Fault prediction using quad tree and expectation maximization algorithm. Intl. J. Appl. Inf. Syst., 2: 36-40.

Okutan, A. and O.T. Yildiz, 2014. Software defect prediction using Bayesian networks. Empirical Software Eng., 19: 154-181.

Pandey, A.K. and N.K. Goyal, 2010. Predicting fault-prone software module using data mining technique and fuzzy logic. Intl. J. Comput. Commun. Technol., 2: 56-63.

Paramshetti, P. and D.A. Phalke, 2014. Survey on software defect prediction using machine learning techniques. Intl. J. Sci. Res., 3: 1394-1397.

Shanthini, A. and R.M. Chandrasekaran, 2012. Applying machine learning for fault prediction using software metrics. Intl. J. Adv. Res. Comput. Sci. Software Eng., 2: 274-277.

YegnaNarayana, B., 2005. Artificial Neural Network. Prentice Hall, Upper Saddle River, New Jersey, USA., pp: 88-141.

Zheng, J., 2010. Cost-sensitive boosting neural networks for software defect prediction. Expert Syst. Appl., 37: 4537-4543.