IJCSC International Journal of Computer Science and Communication Vol. 2, No. 2, July-December 2011, pp. 649-652

TO GENERATE RULE FOR SOFTWARE DEFECT PREDICTION ON QUANTITATIVE AND QUALITATIVE FACTORS USING ARTIFICIAL NEURAL NETWORK

Harish Kundra*, Neha Gautam[1], Sunil Khullar[2] and Parvinder. S. Sandhu[3]

[1]Student, Department of Computer Science, Rayat Institute of Engineering & Information Technology
[2]Senior Lecturer, Department of Computer Science, Rayat Institute of Engineering & Information Technology
E-mail: sunilkhullar222@yahoo.co.in
[3]Professor, Department of Computer Science & Engg., Rayat & Bahra Institute of Engineering & Biotechnology, Mohali, India

ABSTRACT

Fault-proneness of a software module is the probability that the module contains faults. A correlation exists between the fault-proneness of the software and the measurable attributes of the code (i.e. the static metrics) and of the testing (i.e. the dynamic metrics). Static code metrics such as Halstead complexity, Cyclomatic complexity, McCabe's complexity measure are inefficient to measure quality, The use of single features of software to predict faults is uninformative. Therefore Artifical Neural Network is used for software defect prediction. An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way a biological nervous system in human brain works. Large number of neurons present in the human brain forms the key element of the neural network paradigm and act as elementary processing elements. These neurons are highly interconnected and work in unison to solve complex problems. Likewise, an Artificial Neural Network can be configured to solve a number of difficult and complex problems. Neural Network Approaches such as Multilayer Perceptron & RBF are used for software Defect prediction on Quantative and Qualatative factors.

Keyword: Radial Basis Function (RBF), Artifical Neural Network, Multilayer Perceptron, Cyclomatic Complexity, Mc Cabe's Complexity, Halstead Complexity.

1. INTRODUCTION

Fault-proneness of a software module is the probability that the module contains faults. A correlation exists between the fault-proneness of the software and the measurable attributes of the code (i.e. the static metrics) and of the testing (i.e. the dynamic metrics). Early detection of fault-prone software components enables verification experts to concentrate their time and resources on the problem areas of the software system under development. Software quality models ensure the reliability of the delivered products. It has become important to develop and apply good software quality models early in the software development life cycle, especially for large-scale development efforts. Software quality prediction models seek to predict quality factors such as whether a component is fault prone or not. Faults in software systems continue to be a major problem. Many systems are delivered to users with excessive faults. This is despite a huge amount of development effort going into fault reduction in terms of quality control and testing. Despite this it is difficult to identify a reliable approach to identifying fault-prone software components. So, it is made attempt to make use of Multilayer Perceptron and RBF based Neural Network approaches to identify the relation between the various qualitative as well as quantitative factor of the modules with the number of faults present in the module that will be helpful for the prediction of faults.

1.1 Methodology

The methodology consists of the following steps:

I. Find the Qualitative and Quantitative Attributes of Software Systems

First of all, find the structural code and design attributes of software systems. Thereafter, select the suitable metric values as representation of statement. The following are the quantative metrics used:[1] Software size: the size, in KLoC of the developed code and the development language & Effortwhich is the development effort measured in person hours for the software development, from specification review to unit test. The Qualititative factors are grouped under five topics [2] will be Specification and Documentation process, New Functionality, Design and Development process, Testing and Rework & Project Management.

II. Select the Suitable Metric Values as Representation of Statement

The suitable metrics like product requirement metrics and

* A.P. Department of Computer Science, Rayat Institute of Engineering & Information Technology.

product module metrics out of these data sets are considered.[3] The term product is used referring to module level data. The term metrics data applies to any finite numeric values, which describe measured qualities and characteristics of a product. The term product refers to anything to which defect data and metrics data can be associated.

III. Analyze, Refine Metrics and Normalize the Metric Values and Explore Different Neural Network Techniques

It is very important to find the suitable algorithm for modeling of software components into different levels of fault severity in software systems.[10] The following two Neural Network algorithms are experimented:

- Multilayer Perceptron
- RBF based Neural Network Approaches

IV. Comparison of Algorithms

The comparisons are made on the basis of the more accuracy and least value of *MAE* and *RMSE* error values. *Accuracy value* of the prediction model is the major criteria used for comparison. The mean absolute error is chosen as the standard error. The technique having lower value of mean absolute error is chosen as the best fault prediction technique.

- **Mean Absolute Error**

Mean absolute error, MAE is the average of the difference between predicted and actual value in all test cases; it is the average prediction error [13]. The formula for calculating MAE is given in equation 1.

$$\frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (1)$$

Assuming that the actual output is a, expected output is c.

- **Root Mean-squared Error**

***RMSE* is frequently used measure of differences between values predicted by a model or estimator and the values actually observed from the thing being modeled or estimated [22]. It is just the square root of the mean square error as shown in equation 2.**

$$\sqrt{\frac{(a_1 - c_1)^2 + (a_2 - c_2)^2 + \ldots + (a_n - c_n)^2}{n}} \quad (2)$$

The mean-squared error is one of the most commonly used measures of success for numeric prediction. This value is computed by taking the average of the squared differences between each computed value and its corresponding correct value. [17] The root mean-squared error is simply the square root of the mean-squared-error. The root mean-squared error gives the error value the same dimensionality as the actual and predicted values.[23]

The mean absolute error and root mean squared error is calculated for each machine learning algorithm i.e. various algorithms for Neural Networks.

2. RESULT

The proposed Neural based methodology is implemented in WEKA environment is one such facility which lends a high performance language for technical computing.

The Classifier that uses backpropagation to classify instances. This network is created by the algorithm. The network can also be monitored and modified during training time. The nodes in this network are all sigmoid (except for when the class is numeric in which case the output nodes become unthresholded linear units).

The following parameters are used for running the multi perceptron based programme:

- *hiddenLayers:* This defines the hidden layers of the neural network. This is a list of positive whole numbers. 1 for each hidden layer. Comma seperated. To have no hidden layers put a single 0 here. This will only be used if autobuild is set. There are also wildcard values '*a*' = (attribs + classes) / 2, '*i*' = attribs, '*o*' = classes , '*t*' = attribs + classes. We have set value equal to '*a*'.
- *learningRate:* The amount the weights are updated. The value is set to 0.3.
- *momentum:* Momentum applied to the weights during updating. The value is set to 0.2.
- *nominalToBinaryFilter:* This will preprocess the instances with the filter. This could help improve performance if there are nominal attributes in the data. The value is set to true.
- *normalizeAttributes:* This will normalize the attributes. This could help improve performance of the network. This is not reliant on the class being numeric. This will also normalize nominal attributes as well (after they have been run through the nominal to binary filter if that is in use) so that the nominal values are between –1 and 1. The value is set to true.
- *normalizeNumericClass:* This will normalize the class if it's numeric. This could help improve performance of the network, It normalizes the class to be between –1 and 1. Note that this is only internally, the output will be scaled back to the original range. The value is set to true.
- *seed:* Seed used to initialise the random number generator. Random numbers are used for setting the initial weights of the connections betweem

nodes, and also for shuffling the training data. The value is set to 0.

- *trainingTime:* The number of epochs to train through. If the validation set is non-zero then it can terminate the network early. The value is set to 500.
- *validationSetSize:* The percentage size of the validation set.(The training will continue until it is observed that the error on the validation set has been consistently getting worse, or if the training time is reached). The value is set to 0. If This is set to zero no validation set will be used and instead the network will train for the specified number of epochs.
- *validationThreshold:* Used to terminate validation testing.The value here dictates how many times in a row the validation set error can get worse before training is terminated. The value is set to 20.

When Multi perceptron based neural network is applied, the results obtained after 10-fold cross-validation are:

- Mean absolute error 345.3643
- Root mean squared error 478.6951

In case of Radial basis function network, RBFNetwork, (Linear regression applied to *K*-means clusters as basis functions), the results obtained after 10-fold cross-validation are:

- Mean absolute 395.4943
- Root mean squared error 529.0774

Class that implements a normalized Gaussian radial basisbasis function network uses the *k*-means clustering algorithm to provide the basis functions and learns either a logistic regression (discrete class problems) or linear regression (numeric class problems) on top of that. Symmetric multivariate Gaussians are fit to the data from each cluster. If the class is nominal it uses the given number of clusters per class. It standardizes all numeric attributes to zero mean and unit variance.

The following parameters are used (as shown in figure 1):

- *clusteringSeed:* The random seed to pass on to *K*-means. It is set to value '1'.
- *maxIts:* Maximum number of iterations for the logistic regression to perform. Only applied to discrete class problems. It is set to value '–1'.
- *minStdDev:* Sets the minimum standard deviation for the clusters. It is set to value '0.1'.
- *numClusters:* The number of clusters for *K*-Means to generate. It is set to value '2' as we have only two classes required.
- *ridge:* Set the Ridge value for the logistic or linear regression. It is set to value '1.0 E -8'.

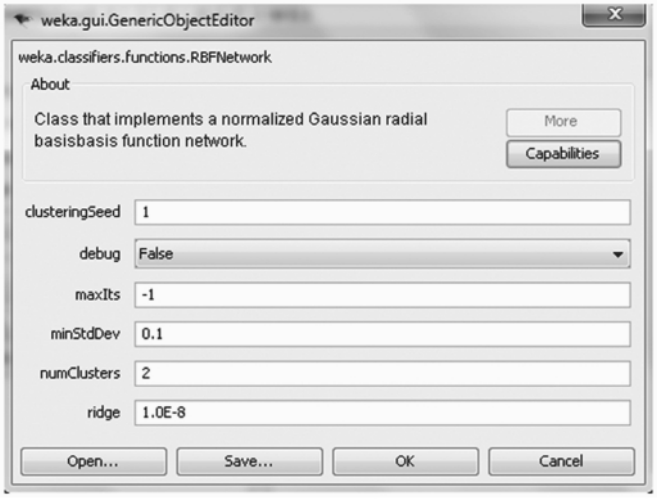

Figure 1: Snapshot of the Parameters used in the RBF Network Programme

In the Linear Regression Model the total defects TD can be calculated with help of following equation:

$$TD = 59.1858 * pCluster\_0\_0 + -59.1874 * pCluster\_0\_1 + 429.0416$$

3. CONCULUSION AND FUTURE SCOPE

Prediction of Level of faults in modules supports software quality engineering through improved scheduling and project control. It is a key step towards steering the software testing and improving the effectiveness of the whole process. Fault prediction is used to improve software process control and achieve high software reliability.

In this study, we investigate whether qualitative and quantitative factors can be used to identify level of number of faulty software modules. We compare the performance of Radial basis function network, where Linear regression applied to *K*-means clusters as basis functions and Multi Perceptron based neural network for the fault dataset. Multi Perceptron based neural network shows best results than RBF Network with lower values of MAE and RMSE calculated as error 345.3643 and 478.6951 respectively. It is therefore, concluded the Multi Perceptron based neural network model is implemented and the best algorithm for classification of the fault prone modules from the faultless modules of the software systems.

The future work can be extended in following directions:

- Most important attribute can be found for fault prediction and this work can be extended to further programming languages. More algorithms can be evaluated and then we can find the best algorithm.
- Further investigation can be done and the impact of attributes on the fault prediction can be found.

- Other dimensions of quality of software can be considered for mapping the relation of attributes and fault tolerance.

REFRENCES

[1] Seliya N., Khoshgoftaar T.M. and Zhong S. (2005), "Analyzing Software Quality with Limited Fault-proneness Defect Data", In *Proceedings of the Ninth IEEE international Symposium on High Assurance System Engineering*, Germany, pp. 89-98.

[2] Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski, Paul Krause, "Project Data Incorporating Qualitative Factors for Improved Software Defect", *Proceedings of the PROMISE Workshop*, Year: 2007.

[3] Jiang Y., Cukic B. and Menzies T. (2007), "Fault Prediction using Early Lifecycle Data", *ISSRE 2007, the 18th IEEE Symposium on Software Reliability Engineering*, IEEE Computer Society, Sweden, pp. 237-246.

[4] Bezdek J.C., Ehrlich R., and Full W. (1984), "FCM: Fuzzy c-means Algorithm", *Computers and Geoscience*, **10**, pp. 191-203.

[5] Khoshgoftaar T.M. and Munson J.C. (1990), "Predicting Software Development Errors using Complexity Metrics", *IEEE Journal on Selected Areas in Communications*, **8**, Issue: 2, pp. 253 -261.

[6] Pigoski M. and Nelson E. (1994), "Software Maintenance Metrics: A Case Study", *Proceedings of IEEE Conference on Software Maintenance*, Canada, pp. 392-401.

[7] Khoshgoftaar, T.M., Allen E.B., Ross F.D., Munikoti R., Goel N. and Nandi A. (1997), "Predicting Fault-prone Modules with case-based Reasoning", *ISSRE 1997, the Eighth International Symposium on Software Engineering*, Mexico, pp. 27-35.

[8] Menzies T., Ammar K., Nikora A., and Stefano S. (2003), "How Simple is Software Defect Prediction?" *Journal of Empirical Software Engineering*, **32**, Issue: 2, pp.1156-1161.

[9] Eman K., Benlarbi S., Goel N., and Rai S. (2001), "Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components", *Journal of Systems Software*, **55**, Issue: 3, pp. 301-310.

[10] Munson J.C. and Khoshgoftaar T.M. (1992), "The Detection of Fault-prone Programs", *IEEE Transactions on Software Engineering*, **18**, Issue: 5, pp. 423-433.

[11] Yaun X., Khoshgoftaar M. and Allen B. (2000), "An Application of Fuzzy Clustering to Software Quality Prediction", *Information Sciences: An International Journal*, **179**, Issue: 8, pp. 1040-1058.

[12] Seliya N. and Khoshgoftaar M. (2007), "Software Quality Analysis of Unlabeled Program Modules with Semi supervised Clustering", *Software Quality Journal*, **37**, Issue: 2, pp. 201-211.

[13] Challagula, Bastani B. and Yen (2006), "A Unified Framework for Defect Data Analysis using the MBR Technique", *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, Washington, pp. 39-46.

[14] Dav'e N. and Krishnapuram R. (1997), "Robust Clustering Methods: A Unified View", *IEEE Transactions on Fuzzy Systems*, **5**, Issue: 2, pp. 270-293.

[15] Saux B. and Boujemaa N. (2002), "Unsupervised Robust Clustering for Image Database Categorization", In *Proceedings of the IEEE-IAPR International Conference on Pattern Recognition (ICPR'2002)*, Turkey, pp. 259-262.

[16] Stark E. (1996), "Measurements for Managing Software Maintenance", *International Conference on Software Maintenance*, USA, pp. 152-161.

[17] Bellini P. (2005), "Comparing Fault-Proneness Estimation Models", *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, China, pp. 205-214.

[18] Lanubile F., Lonigro A., and Visaggio G. (1995), "Comparing Models for Identifying Fault-Prone Software Components", *Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering*, USA, pp. 12-19.

[19] Fenton N.E. and Neil M. (1999), "A Critique of Software Defect Prediction Models", *IEEE Transactions on Software Engineering*, **25**, Issue: 5, pp. 675-689.

[20] Runeson, Wohlin C. and Ohlsson M.C. (2001), "A Proposal for Comparison of Models for Identification of Fault-Proneness", *Journal of System and Software*, **56**, Issue: 3, pp. 301–320.

[21] Deodhar M. (2002), "Prediction Model and the Size Factor for Fault-proneness of Object Oriented Systems", *Journal of System and Software*, **56**, Issue: 3, pp. 157-162.

[22] Ma Y. and Guo L. (2006), "A Statistical Framework for the Prediction of Fault-Proneness", *Product Focused Process Improvement*, Edition: First, Publisher: Springer Berlin/Heidelberg, pp. 204-214.

[23] Challagulla V.U.B., Bastani F.B., Yen I. L. and Paul (2005), "Empirical Assessment of Machine Learning Based Software Defect Prediction Techniques", *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, USA, pp. 263-270.