International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:8, 2009

Software Maintenance Severity Prediction for Object Oriented Systems

Parvinder S. Sandhu, Roma Jaswal, Sandeep Khimta, and Shailendra Singh

***Abstract*—**As the majority of faults are found in a few of its modules so there is a need to investigate the modules that are affected severely as compared to other modules and proper maintenance need to be done in time especially for the critical applications. As, Neural networks, which have been already applied in software engineering applications to build reliability growth models predict the gross change or reusability metrics. Neural networks are non-linear sophisticated modeling techniques that are able to model complex functions. Neural network techniques are used when exact nature of input and outputs is not known. A key feature is that they learn the relationship between input and output through training. In this present work, various Neural Network Based techniques are explored and comparative analysis is performed for the prediction of level of need of maintenance by predicting level severity of faults present in NASA's public domain defect dataset. The comparison of different algorithms is made on the basis of Mean Absolute Error, Root Mean Square Error and Accuracy Values. It is concluded that Generalized Regression Networks is the best algorithm for classification of the software components into different level of severity of impact of the faults. The algorithm can be used to develop model that can be used for identifying modules that are heavily affected by the faults.

***Keywords*—**Neural Network, Software faults, Software Metric.

I. Introduction

STATISTICAL methods, machine learning and mixed techniques are widely used in literature to predict software faults [1-11]. Many researchers have carried out significant work in the area of fault prediction and very less work is performed for the software maintenance severity prediction. In [12], the author has used various machine learning techniques for an intelligent system for the software maintenance prediction and proposed the logistic model Trees (LMT) algorithms on the basis of Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Accuracy percentage.

The Neural Network (NN) is a network consisting of connected neurons. Information can propagate in NN by firing electric pulses through its connections. The connection (weights) change throughout the lifetime of a neuron and the amount of incoming pulses needed to activate a neuron also change. This behavior allows the NN to learn. We can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements. Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output.

The main aim of this work is to model the impact of faults in object oriented software modules. The main objectives are described as follows:

- To find the structural code and design attributes of software systems
- Find the best algorithms that can be used to model impact of faults in object oriented i.e. the predict the level of impact of the faults in the software system

This paper is organized as follows: Section two describes the Methodology part of work done, which shows the steps used in order to reach the objectives and carry out the results. In the section three, results of the implementation are discussed. In the last section, on the basis of the discussion various Conclusions are drawn and the future scope for the present work is discussed.

II. Proposed Methodology

The methodology consists of the following steps:

A. Find the Structural Code and Design Attributes

The first step is to find the structural code and design attributes of software systems i.e. software metrics. The real-time defect data sets are taken from the NASA's MDP (Metric Data Program) data repository. The dataset is related to the safety critical software systems being developed by NASA.

B. Collection of Metric Values

The suitable metrics like product module metrics out of these data sets are considered. 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. In most cases products will be synonymous with code related items such a functions and systems/sub-systems.

The metrics are as follows:

- LOC_BLANK
- BRANCH_COUNT
- CALL_PAIRS
- LOC_CODE_AND_COMMENT
- LOC_COMMENTS
- CONDITION_COUNT

Roma Jaswal is working with Govt. Shivalik College, Naya Nangal, Punjab, India.

Shailendra Singh is associated with Deptt. of Information Technology at Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India.

Parvinder S. Sandhu and Sandeep Khimta are associated with Rayat Bahra Institute of Engineering & Bio-Technology, Sahauran, Mohali (India).

- CYCLOMATIC_COMPLEXITY
- CYCLOMATIC_DENSITY
- DECISION_COUNT
- DESIGN_COMPLEXITY
- EDGE_COUNT
- ESSENTIAL_COMPLEXITY
- ESSENTIAL_DENSITY
- PARAMETER_COUNT
- LOC_EXECUTABLE
- HALSTEAD_CONTENT
- HALSTEAD_DIFFICULTY
- HALSTEAD_EFFORT
- HALSTEAD_ERROR_EST
- HALSTEAD_LENGTH
- HALSTEAD_LEVEL
- HALSTEAD_PROG_TIME
- HALSTEAD_VOLUME
- MAINTENANCE_SEVERITY
- MODIFIED_CONDITION_COUNT
- MULTIPLE_CONDITION_COUNT
- NODE_COUNT
- NORMALIZED_CYLOMATIC_COMPLEXIY
- NUM_OPERANDS
- NUM_OPERATORS
- NUM_UNIQUE_OPERANDS
- NUM_UNIQUE_OPERATORS
- NUMBER_OF_LINES
- PATHOLOGICAL_COMPLEXITY
- PERCENT_COMMENTS
- LOC_TOTAL.

C. Analyze and Refine Metrics the Metric Values

In the next step the metrics are analyzed, refined and normalized and then used for modeling of fault tolerance in software systems.

D. Explore Different Neural Network Techniques

It is very important to find the suitable algorithm for classification of software components into different levels of fault severity in software systems. In the implementation first the network is created and training is performed on the training data. Thereafter the trained network is tested by testing data in the testing phase. The results of the different algorithms are expressed in terms of *MAE*, *RMSE* and *Accuracy* values. The details of the different criteria used are in next step. The following steps will be followed to train a Neural Network:

- Load the data
- Divide data into Training, Validation and Test data
- Set number of hidden neurons
- Training is accomplished by sending a given set of inputs through the network and comparing the results with a set of target outputs.
- If there is a difference between the actual and target outputs, the weights are adjusted to produce a set of outputs closer to the target values.
- Network weights are determined by adding an error correction value to the old weight.
- The amount of correction is determined
- This Training procedure is repeated until the network performance no longer improves.
- If the network is successfully trained, it can then be given new sets of input and generally produce correct results on its own

E. Comparison of the 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 shown below:

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

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 [13]. It is just the square root of the mean square error as shown in equation given below:

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

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. 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.

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

III. Results & Discussion

The first step is to find the structural code and design attributes of software systems i.e. software metrics. The real-time defect data set used is taken from the NASA's MDP (Metric Data Program) data repository, the details of that dataset contains 293 Object Oriented modules with different values of impact of faults labeled as 1, 2, 3, 4 and 5. Details of the Type of Modules in the Dataset are shown in Table I in tabular form and Fig. 1 in graphical form.

TABLE I
DETAILS OF THE TYPE OF MODULES IN THE DATASET

Level	Count
1	48
2	207
3	28
4	8
5	2

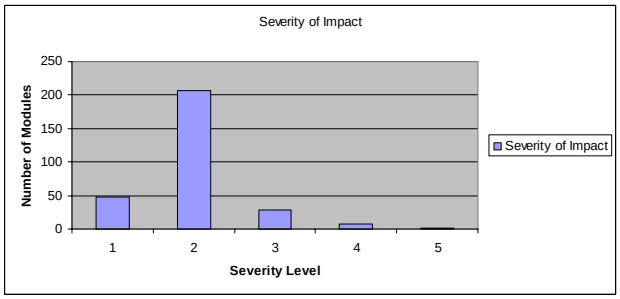

Fig. 1 Graphical Representation of Details of the Type of Modules in the Dataset

The algorithms are evaluated on the basis of the following criteria:

The developed software computes the mean absolute error, root mean squared error, relative absolute error and root relative squared error. However, the most commonly reported error is the mean absolute error and root mean squared error. The root mean squared error is more sensitive to outliers in the data than the mean absolute error. In order to minimize the effect of outliers, mean absolute error is chosen as the standard error. The prediction technique having least value of mean absolute error is chosen as the best prediction technique.

Mean absolute error, *MAE* is the average of the difference between predicted and actual value in all test cases. The root mean-squared error i.e. *RMSE* is simply the square root of the mean-squared-error. The root mean-squared error gives the error value as the same dimensionality as the actual and predicted values.

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.

The *MAE* and the *RMSE* can be used together to diagnose the variation in the errors in a set of forecasts. The *RMSE* will always be larger or equal to the *MAE*.

The greater difference between them, the greater the variance in the individual errors in the sample. If root mean squared error is equal to mean absolute error, then all the errors are of the same magnitude. Both root mean squared error and mean absolute error can range from 0 to ∞.

MAE and *RMSE* are negatively-oriented scores and lower values are better. So, algorithm with least value of mean absolute error is considered as the best algorithm.

In the present work the following Neural Network based algorithms experimented in Matlab 7.4 as listed in table:

TABLE II
PERFORMANCE RESULTS OF DIFFERENT NEURAL NETWORK ALGORITHMS

Sr. No.	Algorithm	*MAE*	*RMSE*	*Accuracy %*
1	Batch Gradient Descent without momentum	.3482	.6600	72
2	Batch Gradient Descent with momentum	.3453	.6615	72
3	Variable Learning Rate without momentum	.6948	.8666	33.333
4	Variable Learning Rate training with momentum	.3872	.6896	70
5	Resilient Backpropagation	.3667	.6658	70.666
6	Fletcher-Reeves version of the conjugate gradient algorithm	.9354	1.2182	52.666
7	Polak-Ribiére Update version of the conjugate gradient algorithm	.4808	.7132	62
8	Powell-Beale Restarts version of the conjugate gradient algorithm .	.3806	.6547	70
9	Scaled Conjugate Gradient version of the conjugate gradient algorithm	.3703	.6484	71
10	Quasi-Newton BFGS Algorithm	.3478	.6733	72
11	Quasi-Newton One Step Secant Algorithm	0.3484	0.6614	72
12	Levenberg-Marquardt Algorithm	.6530	1.8896	72.333
13	Generalized Regression Networks	0.0265	0.1056	97.666
14	Self Organizing Network	1.1229	1.3386	12.969

Table II shows the results of the evaluation of different Neural Network based machine learning algorithms for classification and modeling of software components into different levels of fault severity present in the software modules. From the Results we found, the best algorithm comes out to be Generalized Regression Algorithm with 97.6667, 0.0265 and 0.1056 as *Accuracy*, *MAE* and *RMSE* values respectively.

IV. Conclusion and Future Scope

Prediction of software maintenance severity in the software 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 severity prediction is used to improve software process control and achieve high software reliability. On comparing all the Neural Network based algorithms, it is observed that the results of the Generalized Regression Networks have outperformed all the other algorithms in terms of *MAE*, *RMSE* and *Accuracy* percentage values. The second best algorithm among the experimented algorithms is Batch Gradient Descent without momentum. The performance of the Batch Gradient Descent without momentum and Batch Gradient Descent with Momentum algorithms is almost same.

It is therefore, concluded that Generalized Regression Networks is the best algorithm for classification of the software components into different level of severity of impact of the faults. The algorithm can be used to develop model that can be used for identifying modules that are heavily affected by the faults and those can be debugged.

References

[1] Saida Benlarbi,Khaled El Emam, Nishith Geol (1999), "Issues in Validating Object-Oriented Metrics for Early Risk Prediction", by Cistel Technology 210 Colonnade Road Suite 204 Nepean, Ontario Canada K2E 7L5.

[2] 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, June 1995, pp. 12-19.

[3] Fenton, N. E. and Neil, M. (1999), "A Critique of Software Defect Prediction Models", Bellini, I. Bruno, P. Nesi, D. Rogai, University of Florence, IEEE Trans. Softw. Engineering, vol. 25, Issue no. 5, pp. 675-689.

[4] Giovanni Denaro (2000), "Estimating Software Fault-Proneness for Tuning Testing Activities" Proceedings of the 22nd International Conference on Software Engineering (ICSE2000), Limerick, Ireland, June 2000.

[5] Manasi Deodhar (2002), "Prediction Model and the Size Factor for Fault-proneness of Object Oriented Systems", MS Thesis, Michigan Tech. University, Dec. 2002.

[6] Bellini, P. (2005), "Comparing Fault-Proneness Estimation Models", 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), vol. 0, 2005, pp. 205-214.

[7] Khoshgoftaar, T.M., K. Gao and R. M. Szabo (2001), "An Application of Zero-Inflated Poisson Regression for Software Fault Prediction. Software Reliability Engineering", ISSRE 2001. Proceedings of 12th International Symposium on, 27-30 Nov. (2001), pp: 66 -73.

[8] Munson, J. and T. Khoshgoftaar, (1990) "Regression Modeling of Software Quality: An Empirical Investigation", Information and Software Technology, 32(2): 106 - 114.

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

[10] Menzies, T., K. Ammar, A. Nikora, and S. Stefano, (2003), "How Simple is Software Defect Prediction?", Journal of Empirical Software Engineering, October (2003).

[11] Eman, K., S. Benlarbi, N. Goel and S. Rai, (2001), "Comparing case-based reasoning classifiers for predicting high risk software components", Journal of Systems Software, 55(3): 301 – 310.

[12] Sandhu, Parvinder Singh, Sunil Kumar and Hardeep Singh, (2007), "Intelligence System for Software Maintenance Severity Prediction", Journal of Computer Science, Vol. 3 (5), pp. 281-288, 2007

[13] Challagulla, V.U.B. , Bastani, F.B. , I-Ling Yen , Paul, (2005) "Empirical assessment of machine learning based software defect prediction techniques", 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS 2005, 2-4 Feb 2005, pp. 263-270.