International Journal of Information Technology and Knowledge Management
July-December 2012, Volume 5, No. 2, pp. 453-457

PSO BASED NEURAL NETWORK APPROACHES FOR PREDICTION OF LEVEL OF SEVERITY OF FAULTS IN NASA'S PUBLIC DOMAIN DEFECT DATASET

Riju Kaushal[1] and Sunil Khullar[2]

Abstract: Faults in software systems continue to be a major problem. The possibility of early estimating the potential faultiness of software could help on planning, controlling and executing software development activities. Neural networks, which have been already applied in software engineering applications, to build reliability growth models predict the gross change or reusability metrics. PSO Trained Neural Network Algorithm can be used 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.

Keywords: Color reduction, Bacteria Foraging Optimization, CMYK, CMYK color difference.

1. INTRODUCTION

Faults in software systems continue to be a major problem. A software bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from behaving as intended (e.g., producing an incorrect result). A software fault is a defect that causes software failure in an executable product. In software engineering, the non-conformance of software to its requirements is commonly called a bug. Most bugs arise from mistakes and errors made by people in either a program's source code or its design, and a few are caused by compilers producing incorrect code. Knowing the causes of possible defects as well as identifying general software process areas that may need attention from the initialization of a project could save money, time and work. The possibility of early estimating the potential faultiness of software could help on planning, controlling and executing software development activities.

When a software system is developed, the majority of faults are found in a few of its modules. In most of the cases, 55 % of faults exist within 20 % of source code. It is, therefore, much of interest is to find out fault-prone software modules at early stage of a project. Using software complexity measures, the techniques build models, which classify components as likely to contain faults or not. Quality will be improved as more faults will be detected. Predicting faults early in the software life cycle can be used to improve software process control and achieve high software reliability. Timely predictions of faults in software modules can be used to direct cost-effective quality enhancement efforts to modules that are likely to have a high number of faults. Prediction models based on software metrics, can estimate number of faults in software modules.

[1,2] RIET, Ropar

Neural networks, which have been already applied in software engineering applications, to build reliability growth models predict the gross change or reusability metrics. A neural network is trained to reproduce a given set of correct classification examples, instead to produce formulas or rules. 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.

2. RELATED WORK

A variety of software fault predictions techniques have been proposed, but there is very less work on the prediction of Level of severity of faults present in the software system. As the major fault requires immediate attention or maintenance but the minor fault can be treated later on or the maintenance of the minor fault can be avoided depending on its level of severity (in case when there is very less time left for the final supply of the software). Therefore, still there is a need to find the best prediction technique for a given prediction dataset to calculate the level of severity of faults in the software systems.

Prediction of impact of faults in object oriented software systems:

- Supports software quality engineering through improved scheduling and project control.
- Can be a key step towards steering the software testing and improving the effectiveness of the whole process.
- Enables effective discovery and identification of defects.

- Enables the verification and validation activities focused on critical software components.
- Used to improve software process control and achieve high software reliability.
- Can be used to direct cost-effective quality enhancement efforts to modules.

3. METHODOLOGY

1. 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.

2. Select the suitable metric values as representation of statement.

3. 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.

4. Analyze and refine metrics the metric values.

In the next step table of module levels metrics PC4_product_module_metrics is joined with PC4_defect_product_relations and thereafter again the join operation of the resultant table is performed with PC4_static_defect_data. An Entity-Relationship diagram relates Modules to Defects and Defects to Severity of Defects is shown in figure 3.1.

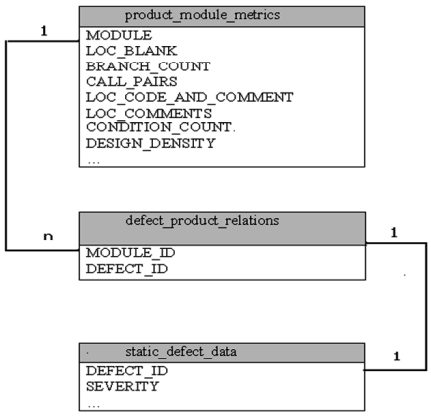

Figure 3.1: An Entity-Relationship Diagram Relates Modules to Defects and Defects to Severity of Defects

In the figure 3.1 the MODULE_ID is the unique numeric identifier of the module and DEFECT_ID is the unique numeric identifier of the associated defect. The SEVERITY field in the PC4_static_defect_data table shows the value that quantifies the impact of the defect on the overall environment in the range of 1 to 5. Where, 1 means most severe and 5 being least severe.

For example, severity 1 may imply that the defect caused a lossof functionality without a workaround where severity 5 may mean that the impact is superficial and did not cause any major disruptions to the system.

5. Explore the PSO trained Neural Network Approach for modeling.

Thereafter, the following are the steps for the hybrid PSO-Neural Network based Modeling system are to be performed:

- Designing of Neural Network and Perform Training : In this step the following three sub steps are there:
  - Calculate the minimum and maximum values in the attribute of input and setting the various parameters of feed-forward back-propagation network by like:
    - Size of the feed-forward back-propagation Neural Network
    - Type of Transfer function of each layer to be used
    - Type of Back-propagation network training function
    - Back-propagation weight/bias learning function
  - Generate the Neural Network
  - Perform the training of the Neural Network with PSO technique discussed after the testing phase using the training dataset
- Testing phase: In this step the PSO trained Neural Network is evaluated against the testing data on the different criteria is discussed in the next steps.

4. RESULTS

The first step is to find the structural code and design attributes of software systems i.e. software metrics. The final metrics are listed in Table 4.1.

Table 4.1
List of Metrics

Sr. No.	*List of Metrics*
1.	LOC_BLANK
2.	BRANCH_COUNT
3.	CALL_PAIRS
4.	LOC_CODE_AND_COMMENT
5.	LOC_COMMENTS
6.	CONDITION_COUNT
7.	CYCLOMATIC_COMPLEXITY
8.	CYCLOMATIC_DENSITY
9.	DECISION_COUNT
10.	DESIGN_COMPLEXITY
11.	EDGE_COUNT
12.	ESSENTIAL_COMPLEXITY
13.	ESSENTIAL_DENSITY
14.	PARAMETER_COUNT
15.	LOC_EXECUTABLE
16.	HALSTEAD_CONTENT
17.	HALSTEAD_DIFFICULTY
18.	HALSTEAD_EFFORT
19.	HALSTEAD_ERROR_EST
20.	HALSTEAD_LENGTH
21.	HALSTEAD_LEVEL
22.	HALSTEAD_PROG_TIME
23.	HALSTEAD_VOLUME
24.	MAINTENANCE_SEVERITY
25.	MODIFIED_CONDITION_COUNT
26.	MULTIPLE_CONDITION_COUNT
27.	NODE_COUNT
28.	NORMALIZED_CYLOMATIC_COMPLEXIY
29.	NUM_OPERANDS
30.	NUM_OPERATORS
31.	NUM_UNIQUE_OPERANDS
32.	NUM_UNIQUE_OPERATORS
33.	NUMBER_OF_LINES
34.	PATHOLOGICAL_COMPLEXITY
35.	PERCENT_COMMENTS
36.	LOC_TOTAL

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 178 modules of C Programming language with different values of software fault severity labeled present as 1, 2 and 3. The severity level 4 and 5 are not present in the PC4 dataset. So, the level 1 represents the highest severity, level 2 represents the medium and level 3 represents the minor fault that can be overlooked to save time. Details of the type of faults existing in different number of modules of the Dataset are shown in bar chart of figure 4.1 and numeric values are tabulated in table 4.2.

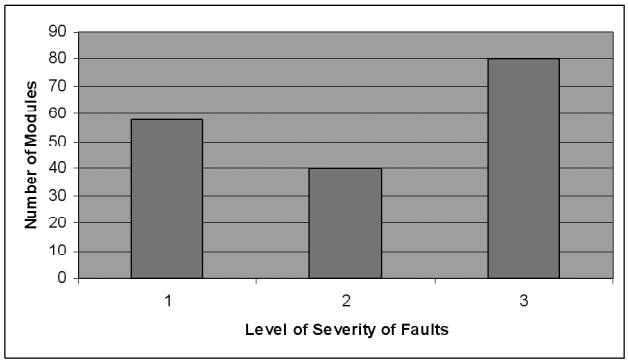

Figure 4.1: Graphical Representation of Different Severity of Faults in Modules

Table 4.2
Details of the Different Severity of Faults in Modules

Severity Level	*Count of Modules*
1	58
2	40
3	80

In the implementation of the PSO trained Neural Network particle swarm optimization toolbox for matlab is used. Size of the feed-forward back-propagation Neural Network is set as [21 21 1] means there are 21 neurons in the Input layer, 21 neurons in the hidden layer and one neuron in the output layer of the network. The Linear transfer function is the type of Transfer function used for the last layer and Hyperbolic tangent sigmoid transfer function is used for the rest of layer of the designed neural network.

Gradient descent with momentum weight and bias learning function is used as Backpropagation weight/bias learning function.

TRAINPSO type of Back-propagation network training function is used as TRAINPSO is a network training function that updates weight and bias values according to particle swarm optimization. The following are the additional parameters values used:

- maximum iterations: 2000
- population size: 25
- acceleration constants (for type = 0): [2,2]
- inertia weights (for type = 0): [0.9,0.4];
- minimum error gradient: 1e-9;

- Iterations at error grad value before exit: floor(0.2*trainParam.maxit)
- error goal: 0;
- Type of PSO: Trelea

After the training, the Architecture of Feed Forward Neural Network is shown in figure 4.2 where the Bright Green line shows more positive weight, bright red line shows more negative weight and dashed white line shows zero weight i.e. no connection between the neurons. In the testing phase MAE, RMSE and Accuracy values of the system are 0.7878, 0.8718 and 72.4719 respectively as shown in figure 4.3.

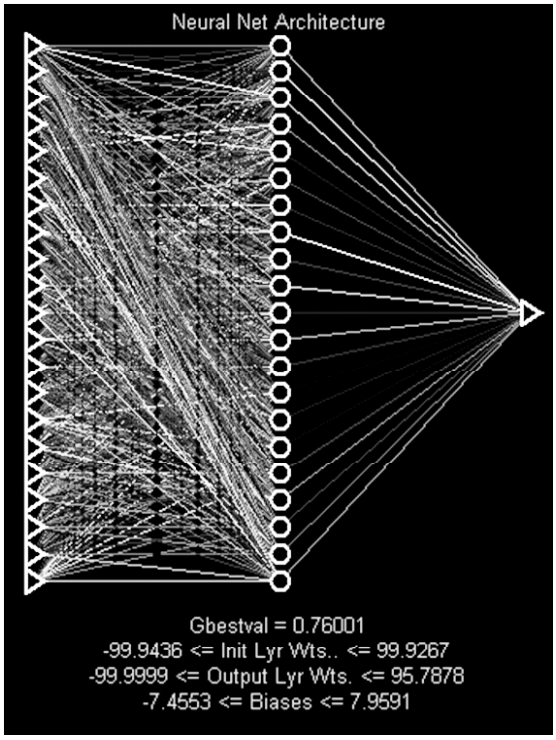

Figure 4.2: Architecture of Feed Forward Neural Network

```
--> Solution likely, GBest hasn't changed by at least 1e-009 for 400 epochs.
**********************************
*******************************

MAE =

    0.7878


RMSE =

    0.8718


Accuracy =

   72.4719
```

Figure 4.3: Snapshot of the Output

The plot of Global best value (Gbest) versus Iterations is shown in figure 4.4. As Gbest value is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. The Gbest value obtained is 0.63639.

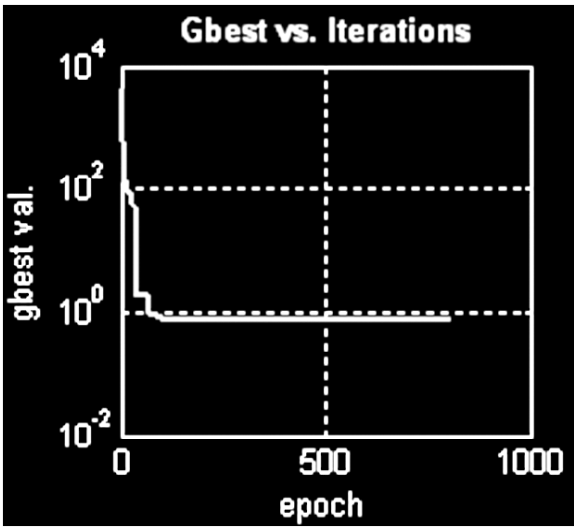

Figure 4.4: Graph Between Gbest Value and Iteration Index

5. CONCLUSIONS AND FUTURE SCOPE

It is concluded that PSO Trained Neural Network Algorithm can be used 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.

The future work can be extended in following directions:

- This work can be extended to other 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 tolerance can be found.
- Other dimensions of quality of software can be considered for mapping the relation of attributes and fault tolerance.

REFERENCES

[1] Aha, D. and Kibler, D. (1991). "Instance-based Learning Algorithms", *Machine Learning*, **6(1)**, January 1991, pp. 37-66.

[2] Atkeson, C., Moore, A. and Schaal, S. (1996). "Locally Weighted Learning", *AI Review*, **11**, Feb 1996, pp. 11-73.

[3] 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.

[4] Breiman L. (2001). "Random Forests", *Machine Learning*, **45(1)**, 2001, pp. 5-32.

[5] Breiman, L. (1996). "Bagging Predictors". *Technical Report*, 1996.

[6] Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (1984). "Classification and Regression Trees", *Wadsworth International Group*, Belmont, California, 1984.

[7] Buntine, W.L. (1996). "Operations for Learning with Graphical Models", *Journal of Artificial Intelligence Research*, **2**, 1994, pp. 159-225.

[8] Caruana, R., Niculescu, A., Crew, G., and Ksikes, A. (2004). "Ensemble Selection from Libraries of Models", *Proceedings International Conference on Machine Learning (ICML'04)*, 2004.

[9] 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.

[10] Cleary, J.G. and Trigg, L.E. (1995). "K*: An Instance-based Learner Using an Entropic Distance Measure", *Proceedings Tweleveth International Conference on Machine Learning*, 1995, pp. 108-114.

[11] Demiroz, G., and Guvenir, A. (1997). "Classification by Voting Feature Intervals", *Proceedings Ninth European Conference on Machine Learning*, 1997, pp. 85-92.

[12] Dong, L., Frank, E. and Kramer, S. (2005). "Ensembles of Balanced Nested Dichotomies for Multi-class Problems", Jorge, A., et al. (eds), *Proceedings Ninth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005)*, Springer-Verlag, Berlin, 2005, pp. 84-95.

[13] Eisenstein, J. and Davis, R. (2006). "Visual and Linguistic l Techniques ACM", **5**, pp. 130-150.

[14] Eric Rotenberg (1999). "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors", *Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, June 15-18, pp. 84-90.

[15] Evren Ceylan, F. Onur Kutlubay, Ayse B. Bener (2006). "Software Defect Identification Using Machine Learning Techniques", *Proceeding of 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, Bogaziçi University, Turkey, pp. 240-247.

[16] 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*, **25(5)**, pp. 675-689.

[17] Frank, E. and Witten, I. H. (1998), "Generating Accurate Rule Sets without Global Optimization", *Proceedings of Fifteenth International Conference on Machine Learning*, Morgan Kaufmann Publishers, 1998, pp. 144-151.

[18] Frank, E. and Hall, M. (2001). "A Simple Approach to Ordinal Classification", *Proceedings of Twelfth European Conference on Machine Learning*, 2001, pp. 145-156.

[19] Frank, E. and Kramer S. (2004). "Ensembles of Nested Dichotomies for Multi-class Problems", *Proceedings Twenty-first International Conference on Machine Learning*, 2004.

[20] Frank, E., Hall, M. and Pfahringer, B. (2003). "Locally Weighted Naive Bayes", *Proceedings Nineteenth Conference in Uncertainty in Artificial Intelligence*, 2003, pp. 249-256.

[21] Frank, E., Wang, Y., Inglis, S., Holmes, G. and Witten, I.H. (1998). "Using Model Trees for Classification", *Machine Learning*, **32(1)**, 1996, pp. 63-76.

[22] Freund, Y. and Schapire, R.E. (1996). "Experiments with a New Boosting Algorithm", *Proceedings Thirteenth International Conference on Machine Learning*, San Francisco, 1996, pp. 148-156.

[23] Friedman, J., Hastie, T. and Tibshirani, R. (1998). "Additive Logistic Regression: A Statistical View of Boosting", Stanford University, 1998.

[24] 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.

[25] Hastie, T., Tibshirani, R. (1998). "Classification by Pairwise Coupling", *Advances in Neural Information Processing Systems, Annals of Statistics*, **26(2)**, 1998, pp. 451-471.

[26] Ho, T.K. (1998). "The Random Subspace Method for Constructing Decision Forests", *Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20(8)**, pp. 832-844.

[27] John, G.H. and Langley, P. (1995). "Estimating Continuous Distributions in Bayesian Classifiers", *Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence*, San Mateo, 1995, pp. 338-345.

[28] Kaariainen, M. and Malinen, T. (2004). "Selective Rademacher Penalization and Reduced Error Pruning of Decision Trees", *Journal of Machine Learning*, **5**, pp. 1107-1126.