International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:2, 2009

Software Maintenance Severity Prediction with Soft Computing Approach

Ebru Ardil, Erdem Uçar, and Parvinder S. Sandhu

***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 on time especially for the critical applications. In this paper, we have explored the different predictor models to NASA's public domain defect dataset coded in Perl programming language. Different machine learning algorithms belonging to the different learner categories of the WEKA project including Mamdani Based Fuzzy Inference System and Neuro-fuzzy based system have been evaluated for the modeling of maintenance severity or impact of fault severity. The results are recorded in terms of *Accuracy*, Mean Absolute Error (*MAE*) and Root Mean Squared Error (*RMSE*). The results show that Neuro-fuzzy based model provides relatively better prediction accuracy as compared to other models and hence, can be used for the maintenance severity prediction of the software.

***Keywords*—**Software Metrics, Fuzzy, Neuro-Fuzzy, Software Faults, Accuracy, MAE, RMSE.

I. Introduction

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 [1]. 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 the impact of the 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.

Prediction of severity of faults:

- 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.

In the literature [1] [2], [3], [4], [5], [6] made prediction of fault prone modules in software development process and mostly used the metric based approach with machine learning techniques to model the fault prediction in the software modules. Khoshgoftaar [7] used zero-inflated Poisson regression to predict the fault-proneness of software systems with a large number of zero response variables. Munson and Khoshgoftaar [8, 9] also investigated the application of multivariate analysis to regression and showed that reducing the number of "independent" factors (attribute set) does not significantly affect the Accuracy of software quality prediction. Menzies, Ammar, Nikora, and Stefano [10] compared decision trees, naïve Bayes, and 1-rule classifier on the NASA software defect data. Emam, Benlarbi, Goel, and Rai [11] compared different case-based reasoning classifiers and concluded that there is no added advantage in varying the combination of parameters (including varying nearest neighbor and using different weight functions) of the classifier to make the prediction Accuracy better. Many modeling techniques have been developed and applied for software quality prediction [12], [13], [14], [15]. The software quality may be analyzed with limited fault proneness data [16].

In [17], the author has used various machine learning techniques for an intelligent system for the software maintenance prediction and proposed the logistic model Trees (LMT) and Complimentary Naïve Bayes (CNB) algorithms on the basis of Mean Absolute Error (*MAE*), Root Mean Square Error (*RMSE*) and *Accuracy* percentage.

Soft-Computing algorithms have proven to be of great practical value in a variety of application domains. Not surprisingly, the field of software engineering turns out to be a fertile ground where many software development and maintenance tasks could be formulated as learning problems and approached in terms of learning algorithms.

In this present work, various machine learning algorithms including Fuzzy and Neuro-Fuzzy Based techniques are explored and comparative analysis is performed for the prediction of level of impact of faults in the software modules.

In this paper, Section two describes the Methodology part

Ebru Ardil and Erdem Uçar are with Department of Computer Engineering, Trakya University, Edirne, Turkey.

Parvinder S. Sandhu is with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 India (phone: +91-98555-32004; e-mail: parvinder.sandhu@gmail.com).

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

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. Select the Suitable Metric Values as Representation of Statement

The suitable metrics like product module metrics out of these data sets are considered. The term product is used referring to module level data.

C. Analyze and Refine Metrics

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

D. Explore the Different Machine Learning Algorithms including Fuzzy and Neuro-Fuzzy Inference System

In this step aim is to find the best algorithm for classification of software components into different levels of impact of fault. Forty Six Machine learning algorithms are used for modeling of the data.

As per Abraham in [18], A Mamdani Neuro-Fuzzy system uses a supervised learning technique (backpropagation learning) to learn the parameters of the membership functions [19]. Architecture of Mamdani Neuro-Fuzzy system is illustrated in Fig. 1. The detailed function of each layer is as follows:

Layer-1 (Input Layer): No computation is done in this layer. Each node in this layer, which corresponds to one input variable, only transmits input values to the next layer directly. The link weight in layer 1 is unity.

Layer-2 (Fuzzification Layer): Each node in this layer corresponds to one linguistic label (excellent, good, etc.) to one of the input variables in layer 1. In other words, the output link represents the membership value, which specifies the degree to which an input value belongs to a fuzzy set, is calculated in layer 2. A clustering algorithm will decide the initial number and type of membership functions to be allocated to each of the input variable. The final shapes of the MFs will be fine tuned during network learning.

Layer-3 (Rule Antecedent Layer): A node in this layer represents the antecedent part of a rule. Usually a T-norm operator is used in this node. The output of a layer 3 node represents the ring strength of the corresponding fuzzy rule.

Layer-4 (Rule Consequent Layer): This node basically has two tasks. To combine the incoming rule antecedents and determine the degree to which they belong to the output linguistic label (high, medium, low, etc.). The number of nodes in this layer will be equal to the number of rules.

Layer-5 (Combination and Defuzzification layer): This node does the combination of all the rules consequents using a T-conorm operator and finally computes the crisp.

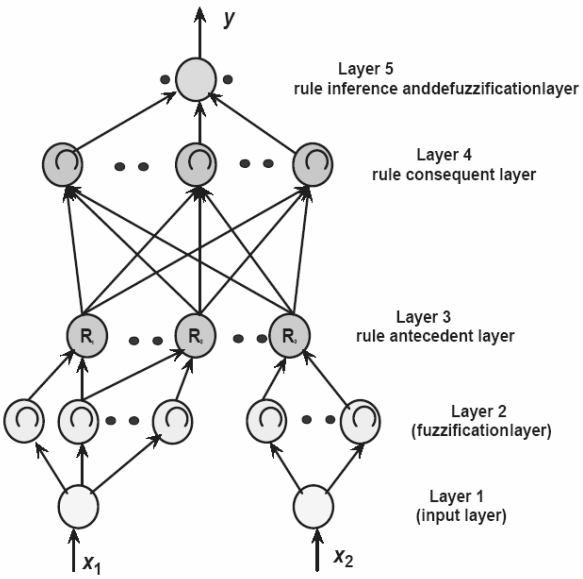

Fig. 1 Mamdani Neuro-Fuzzy System Structure [18]

According to [20], a fuzzy system can be considered to be a parameterized nonlinear map, called *f*, which can be expressed as (1):

$$f(x)=\frac{\sum_{l=1}^{m} y^{l}\left(\prod_{i=1}^{n}\mu_{A_i^{l}}(x_i)\right)}{\sum_{l=1}^{m}\left(\prod_{i=1}^{n}\mu_{A_i^{l}}(x_i)\right)} \quad (1)$$

Where y^l is a place of output singleton if Mamdani reasoning is applied or a constant if Sugeno reasoning is applied. The membership function $\mu_{Ai}{}^{l}(x_i)$ corresponds to the input x=[x_1, x_2, x_3,… x_m] of the rule l . The "*and*" connective in the premise is carried out by a product and defuzzification by the center-of-gravity method. Consider a Sugeno type of fuzzy system having the rule base

Rule1: If x is A1 and y is B1, then f1= $p_1x + q_1y + 1$

Rule2: If x is A2 and y is B2, then f2= $p_2x + q_2y + r2$

Let the membership functions of fuzzy sets Ai, Bi, i=1,2, be , μ_{Ai} , μ_{Bi} .

-Evaluating the rule premises results in $w_i = \mu_{Ai}(x) * \mu_{Bi}(y)$ where i = 1,2 for the rule rules stated above.

-Evaluating the implication and the rule consequences gives (2).

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \quad (2)$$

Let

$$\overline{w_i} = \frac{w_i}{w_1 + w_2} \quad (3)$$

Then f can be written as (4).

$$f = \overline{w_1} f_1 + \overline{w_2} f_2 \quad (4)$$

E. Comparison Criteria

The comparisons of machine learning algorithms are made on the basis of the least value of *MAE* and *RMSE* values. *Accuracy* value of the prediction model is also used for the comparison. The best algorithm is picked up after the 10 fold cross validation results and tested for the testing dataset. The *Accuracy* of the model is compared with the results of Mamdani based FIS and Neuro-Fuzzy based systems. The details of the MAE and RMSE are:

- *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 [21]. The formula for calculating *MAE* is given in equation shown below:

$$\frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (5)$$

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

$$\sqrt{\frac{(a_1 - c_1)^2 + (a_2 - c_2)^2 + \ldots + (a_n - c_n)^2}{n}} \quad (6)$$

F. Conclusions Drawn

The conclusions are made on the basis of the comparison made in the previous section.

III. Results & Discussion

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 60 modules of Perl Programming language with different values of software fault severity labeled as 1, 2, 3, 4 and 5. Details of the Type of Modules in the Dataset are shown in Fig. 2.

The first step is to find the structural code and design attributes of software systems i.e. software metrics. As most of the values of the other metrics are zero or metrics are redundant in nature. So, selected five metrics representing input attributes are:

- Branch_Count
- Cyclometric_Complexity
- Design_Complexity
- Essential_Complexity
- Number_Of_Lines

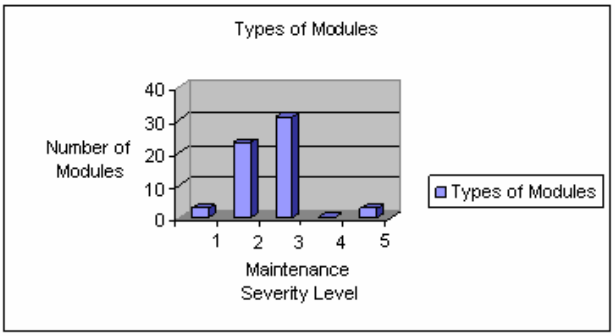

Fig. 2 Graphical Representation of Details of the Type of Modules in the Dataset

The algorithms which are explored are already built java classes in WEKA project [22]. For this a variety of many machine learning algorithms and neural network techniques are analyzed.

When analyzing performance of all the algorithms, Logistic Model Trees (LMT) and Simple Logistic algorithms have outperformed all the other algorithms used in the comparative study with *Accuracy*, *MAE* and *RMSE* values as 65, 0.2145 and 0.3285 respectively when the 10 fold cross validation is performed.

When Logistic Model Trees (LMT) and Simple Logistic algorithms are tested for the fifteen exemplar inputs 86.66% accuracy is obtained.

In the Mamdani based fuzzy inference system model five metrics are considered as input attributes and one attribute named as "software maintenance severity level" is used as output attribute as shown in Fig. 3.

Each input and output attribute is represented with fifteen fuzzy sets and the membership function value of the each attribute is shown in Fig. 4. Different membership function values that are used to convert the crisp values into the fuzzy values and that process is called fuzzification. Once you have got the fuzzy values then you can use the values in the fuzzy rule evaluation which is the next step in the Fuzzy Inference system. In Fig. 5, fifteen rules used for the inference of the Mamdani based FIS are shown.

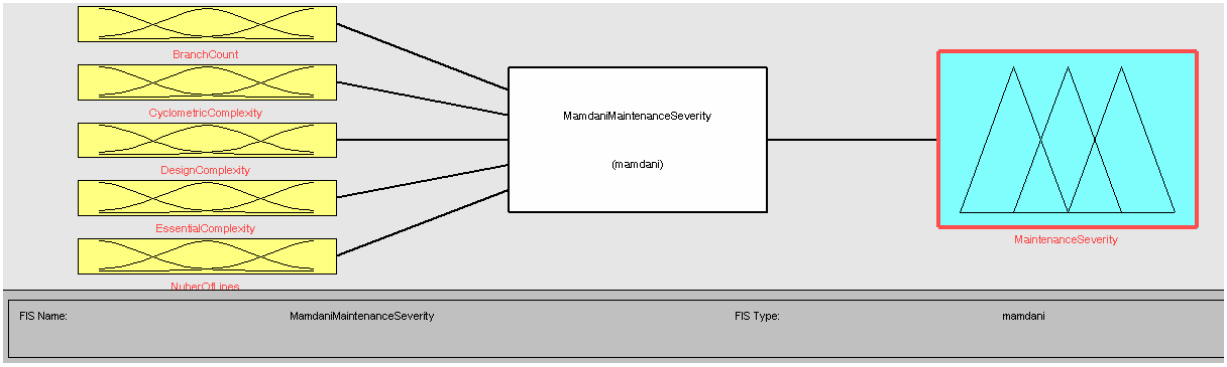

Fig. 3 Mamdani Based FIS Inference System

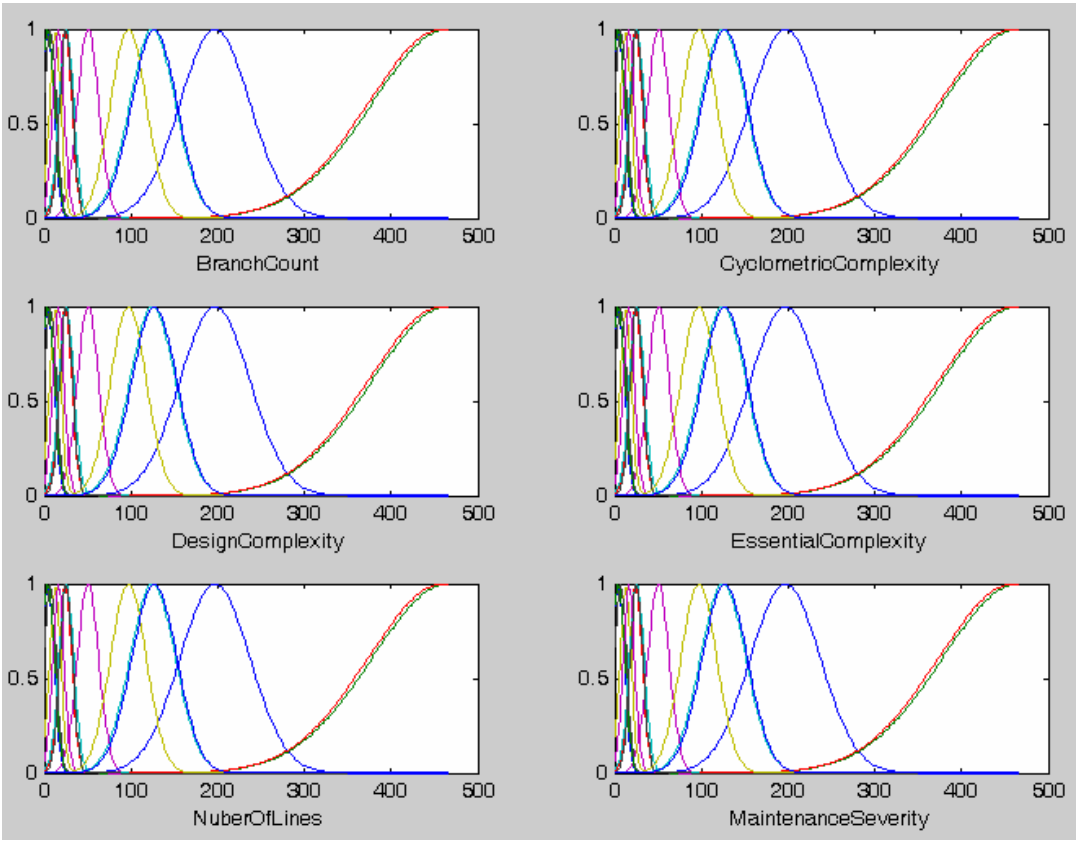

Fig. 4 Membership Functions of the Input and Output Attributes

1. If (BranchCount is in1cluster1) and (CyclometricComplexity is in2cluster1) and (DesignComplexity is in3cluster1) and (EssentialComplexity is in4cluster1) and (NuberOfLines is in5cluster1) then (MaintenanceSeverity is out1cluster1) (1)
2. If (BranchCount is in1cluster2) and (CyclometricComplexity is in2cluster2) and (DesignComplexity is in3cluster2) and (EssentialComplexity is in4cluster2) and (NuberOfLines is in5cluster2) then (MaintenanceSeverity is out1cluster2) (1)
3. If (BranchCount is in1cluster3) and (CyclometricComplexity is in2cluster3) and (DesignComplexity is in3cluster3) and (EssentialComplexity is in4cluster3) and (NuberOfLines is in5cluster3) then (MaintenanceSeverity is out1cluster3) (1)
4. If (BranchCount is in1cluster4) and (CyclometricComplexity is in2cluster4) and (DesignComplexity is in3cluster4) and (EssentialComplexity is in4cluster4) and (NuberOfLines is in5cluster4) then (MaintenanceSeverity is out1cluster4) (1)
5. If (BranchCount is in1cluster5) and (CyclometricComplexity is in2cluster5) and (DesignComplexity is in3cluster5) and (EssentialComplexity is in4cluster5) and (NuberOfLines is in5cluster5) then (MaintenanceSeverity is out1cluster5) (1)
6. If (BranchCount is in1cluster6) and (CyclometricComplexity is in2cluster6) and (DesignComplexity is in3cluster6) and (EssentialComplexity is in4cluster6) and (NuberOfLines is in5cluster6) then (MaintenanceSeverity is out1cluster6) (1)
7. If (BranchCount is in1cluster7) and (CyclometricComplexity is in2cluster7) and (DesignComplexity is in3cluster7) and (EssentialComplexity is in4cluster7) and (NuberOfLines is in5cluster7) then (MaintenanceSeverity is out1cluster7) (1)
8. If (BranchCount is in1cluster8) and (CyclometricComplexity is in2cluster8) and (DesignComplexity is in3cluster8) and (EssentialComplexity is in4cluster8) and (NuberOfLines is in5cluster8) then (MaintenanceSeverity is out1cluster8) (1)
9. If (BranchCount is in1cluster9) and (CyclometricComplexity is in2cluster9) and (DesignComplexity is in3cluster9) and (EssentialComplexity is in4cluster9) and (NuberOfLines is in5cluster9) then (MaintenanceSeverity is out1cluster9) (1)
10. If (BranchCount is in1cluster10) and (CyclometricComplexity is in2cluster10) and (DesignComplexity is in3cluster10) and (EssentialComplexity is in4cluster10) and (NuberOfLines is in5cluster10) then (MaintenanceSeverity is out1cluster10) (1)
11. If (BranchCount is in1cluster11) and (CyclometricComplexity is in2cluster11) and (DesignComplexity is in3cluster11) and (EssentialComplexity is in4cluster11) and (NuberOfLines is in5cluster11) then (MaintenanceSeverity is out1cluster11) (1)
12. If (BranchCount is in1cluster12) and (CyclometricComplexity is in2cluster12) and (DesignComplexity is in3cluster12) and (EssentialComplexity is in4cluster12) and (NuberOfLines is in5cluster12) then (MaintenanceSeverity is out1cluster12) (1)
13. If (BranchCount is in1cluster13) and (CyclometricComplexity is in2cluster13) and (DesignComplexity is in3cluster13) and (EssentialComplexity is in4cluster13) and (NuberOfLines is in5cluster13) then (MaintenanceSeverity is out1cluster13) (1)
14. If (BranchCount is in1cluster14) and (CyclometricComplexity is in2cluster14) and (DesignComplexity is in3cluster14) and (EssentialComplexity is in4cluster14) and (NuberOfLines is in5cluster14) then (MaintenanceSeverity is out1cluster14) (1)
15. If (BranchCount is in1cluster15) and (CyclometricComplexity is in2cluster15) and (DesignComplexity is in3cluster15) and (EssentialComplexity is in4cluster15) and (NuberOfLines is in5cluster15) then (MaintenanceSeverity is out1cluster15) (1)

Fig. 5 Fifteen Rules of the Mamdani Based FIS

During the testing phase of the Mamdani Based Fuzzy Inference System fifteen inputs are used and it shows 0.2183, 0.3066 and 80 percentage as *MAE*, *RMSE* and *Accuracy* values.

As performance of Adaptive Neuro Fuzzy Inference System is found to be the best out of all the hybrid NF systems [23] and the extra complexity in structure and computation of Mamdami based Adaptive NF Inference system with max-min composition does not necessarily imply better learning capability or approximation power [24]. Hence, in MATLAB 7.4, the Sugeno based Adaptive Neuro-fuzzy Inference System is used for modeling of software maintenance severity. The ideal inference system for the evaluation of software components should be less complex and more precision. The inference system, which is already trained, will get the metric values from the earlier stages and estimate the software maintenance severity value of the software components or modules.

The following is the information regarding the structure of the adaptive Neuro-fuzzy Based Inference system and pictorially represented in Fig. 6:

- Number of nodes: 32
- Number of linear parameters: 12
- Number of nonlinear parameters: 20
- Total number of parameters: 32
- Number of training data pairs: 60
- Number of checking data pairs: 0
- Number of fuzzy rules: 2

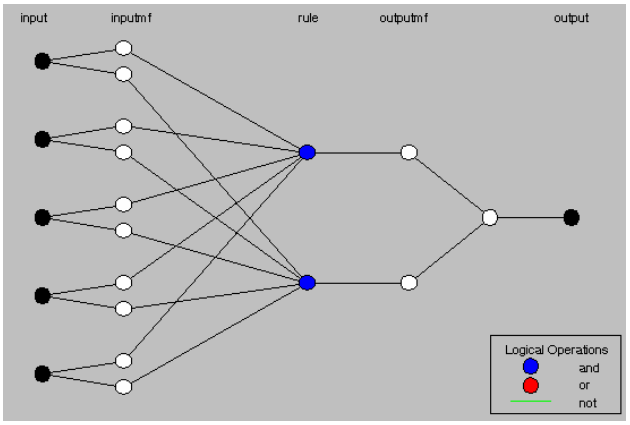

Fig. 6 Structure of Adaptive Neuro-Fuzzy Inference System

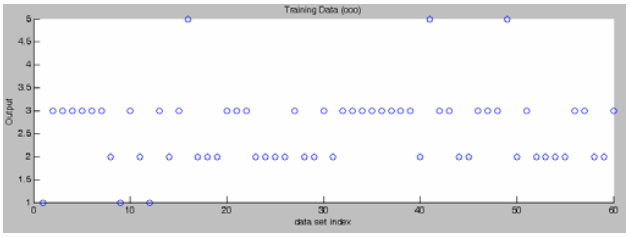

Fig. 7 Training Data for the Neuro-fuzzy system

The graphical representation of the input exemplars for the NF system is shown in Fig. 7.

The NF system is trained using a hybrid learning algorithm using both least squares method and backpropagation. In the forward pass the consequent parameters are identified using least squares and in the backward pass the premise parameters are identified using backpropagation. The trained NF system is then tested for the fifteen inputs and it shows 0.1571, 0.2140 and 93.3333 as *MAE*, *RMSE* and *Accuracy* values respectively.

The plot of the expected and the output of the NF system for the different inputs are shown in Fig. 8. Fig. 8 shows the plot of the result of accuracy of the system that is developed. The red star is the expected vale and the blue dot is the value calculated by our model. Means the overall accuracy picture is shown with help of that chart.

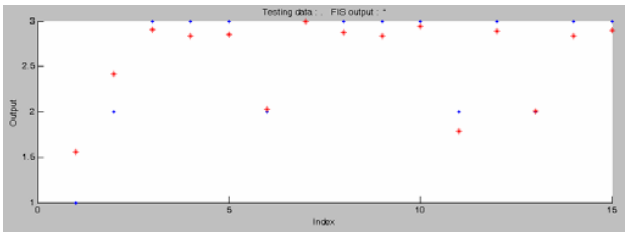

Fig. 8 Plot of the Testing Data V/S FIS Output

IV. Conclusion

On comparing all the classes of WEKA's machine learning algorithms, it is observed that Logistic Model Trees and Simple Logistic algorithms are better techniques as compared with other classes of machine learning algorithms with the 65% Accuracy in prediction of fault tolerance. In both the algorithms of the WEKA project the classification algorithm is the same i.e. logistic classifier. Both the algorithms have least Mean Absolute Error and Root Mean Square Error values: 0.2145 and 0.3285. During the testing phase LMT and Simple Logistic algorithm has shown 86.66% *Accuracy*.

The results of the Mamdani based fuzzy inference system are comparatively equivalent for the testing data as that of the Logistic Model Trees and Simple Logistic algorithm with 0.2183, 0.3066 and 80 as Mean Absolute Error, Root Mean Square Error and *Accuracy* values.

The Neuro-fuzzy based Modeling technique has outperformed the other technique on the basis of the testing data with 0.1571, 0.2140 and 93.3333 as Mean Absolute Error, Root Mean Square Error and *Accuracy* values.

It is therefore, concluded the model is implemented and the best algorithm for classification of the software components into different level of severity of impact of the fault is found to be Neuro-Fuzzy based technique. 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] Hudepohl, J. P., S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. E. Mayrand, (1996), "Software Metrics and Models on the Desktop", IEEE Software, 13(5): 56-60.

[13] Khoshgoftaar, T. M., E. B. Allen, K. S. Kalaichelvan, and N. Goel, (1996), "Early quality prediction: a case study in telecommunications", IEEE Software (1996), 13(1): 65-71.

[14] Khoshgoftaar, T. M. and N. Seliya, (2002), "Tree-based software quality estimation models for fault prediction", METRICS 2002, the Eighth IIIE Symposium on Software Metrics, pp: 203-214.

[15] Seliya N., T. M. Khoshgoftaar, S. Zhong, (2005), "Analyzing software quality with limited fault-proneness defect data", Ninth IEEE international Symposium, Oct 12-14, (2005).

[16] Munson, J. C. and T. M. Khoshgoftaar, (1992), "The detection of fault-prone programs", IEEE Transactions on Software Engineering, 18(5): 423-433.

[17] 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

[18] Abraham A., (2005), "Hybrid Intelligent Systems: Evolving Intelligence in Hierarchical Layers", Studies in Fuzziness and Soft Computing, vol. 173, 2005, pp. 159–179.

[19] Yen J. and Langari R. (2003), "Fuzzy Logic: Intelligence, Control, and Information" Pearson Education.

[20] J-S. R. Jang and C.T. Sun, (1995), "Neuro-fuzzy Modeling and Control", Proceeding of the IEEE, March 1995.

[21] 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.

[22] www.cs.waikato.ac.nz/~ml/weka/.

[23] Abraham, Ajith, (2001), "Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques, Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence", Mira Jose, Prieto Alberto ed., Lecture Notes in Computer Science, vol. 2084. Germany: Springer-Verlag; 2001, pp. 269-276.

[24] Jang, J.-S. R., Sun, C.-T. and Mizutani, E., (2004), "Neuro-Fuzzy and Soft Computing- A Computational Approach to Learning and Machine Intelligence", Pearson Education (Singapore) Pvt. Ltd., 1st Edition, 2004.