International Journal of Physical Sciences Vol. 5 (2), pp. 074-085, February, 2010
Available online at http://www.academicjournals.org/IJPS
ISSN 1992 - 1950 © 2010 Academic Journals

Full Length Research Paper

A soft computing approach for modeling of severity of faults in software systems

Ebru Ardil[1] and Parvinder S. Sandhu[2]*

[1]Department of Electrical and Electronics Engineering, Fatih University, Istanbul, Turkey.
[2]Head of Computer Science and Engineering and Information Technology Department, Rayat and Bahra Institute of Engineering and Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 India.

Accepted 25 January 2010

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. In this present work, hybrid fuzzy-Genetic Algorithm and Particle Swarm Optimization trained Neural Network techniques are empirically evaluated and earlier published results of the Mamdani Based Fuzzy Inference System and Neuro-Fuzzy Based techniques are also discussed for the comparative analysis in order to predict level of impact of faults in NASA's public domain defect dataset coded in Perl programming language. The results are recorded in terms of accuracy, mean absolute error (MAE) and root mean squared error (RMSE). The results of Neuro-Fuzzy model are also convincing but Fuzzy-GA based hybrid model provide relatively better prediction accuracy as compared to other models and hence, it is proposed for the maintenance severity prediction of the software systems.

Key words: Fuzzy, neuro-fuzzy, genetic algorithm, particle swarm optimization (PSO), accuracy, MAE, RMSE.

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 to find out fault-prone software modules at early stage of a project (Benlarbi et al., 1999). 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:

(1) Supports software quality engineering through improved scheduling and project control.
(2) Can be a key step towards steering the software testing and improving the effectiveness of the whole process.
(3) Enables effective discovery and identification of defects.
(4) Enables the verification and validation activities focused on critical software components.
(5) Used to improve software process control and achieve high software reliability.
(6) Can be used to direct cost-effective quality enhancement efforts to modules.

In the literature (Benlarbi et al., 1999; Lanubile et al., 1995; Fenton et al., 1999; Denaro, 2000; Deodhar, 2002; Bellini, 2005) 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 et al. (2001) used zero-inflated Poisson regression to predict the fault-proneness of software

*Corresponding author. E-mail: parvinder.sandhu@gmail.com. Tel: +91-98555-32004.

systems with a large number of zero response variables. Munson et al. (1990) and Khoshgoftaar et al. (1990) 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 et al. (2003) compared decision trees, naïve Bayes and 1-rule classifier on the NASA software defect data. Eman et al. (2001) 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 (Hudepohl et al., 1996; Khoshgoftaar et al., 1996; Khoshgoftaar et al., 2002; Seliya et al., 2005). The software quality may be analyzed with limited fault proneness data (Munson et al., 1992).

In (Sandhu et al., 2007), 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. The Soft Computing (SC) paradigm also known as Computational Intelligence differs from conventional computing in that, techniques belonging to it can be tolerant of imprecise, incomplete or corrupt input data. Some of them can solve problems without requiring the solution steps or reasoning process to be explicitly stated. Some soft computing systems develop the capability to solve problems through repeated observation and adaptation. Some arrive at a solution through a process similar to evolution in nature.

In more than one ways, the human mind is the role model for soft computing techniques - for example, the ability to solve problems expressed in vague terms, or solving problems without making use of explicit solution steps. Arriving at a solution through an evolutionary process is commonplace in nature.

The predominant SC methodologies found in current intelligent systems are:

(1) Artificial Neural Networks (ANN).
(2) Fuzzy Systems.
(3) Genetic Algorithms (GA).

Particle Swarm Optimization (PSO) shares many similarities with evolutionary computation techniques such as genetic algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles.

This paper is the extension of the earlier work (Ardil et al., 2009) in which various machine learning algorithms including Fuzzy and Neuro-Fuzzy based techniques are experimented for the prediction of level of impact of faults in the software modules. In this present work, hybrid fuzzy-GA and PSO trained neural network techniques are evaluated and results of the Fuzzy and Neuro-Fuzzy based techniques are also discussed for the comparative analysis in order to predict level of impact of faults in the software modules.

In this paper, 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.

Fuzzy inference systems

A fuzzy inference system (FIS) is a way of mapping an input space to an output space using fuzzy logic. A FIS tries to formalize the reasoning process of human language by means of fuzzy logic (that is, by building fuzzy IF-THEN rules).

On wide categorization, following are the basic types of fuzzy inference systems:

(1) Mamdani fuzzy inference system.
(2) Takagi-sugeno fuzzy inference system.
(3)Adaptive neuro-fuzzy inference system (ANFIS).

Mamdani fuzzy inference system

Mamdani's fuzzy inference method is the most commonly seen fuzzy methodology. Mamdani's method was among the first control systems built using fuzzy set theory. It was proposed in 1975 by Mamdani et al. (1975) as an attempt to control a steam engine and boiler combination by synthesizing a set of linguistic control rules obtained from experienced human operators. Mamdani's effort was based on Lotfi Zadeh's 1973 paper on fuzzy algorithms for complex systems and decision processes (Zadeh, 1973). Although the inference process described in the next few sections differs somewhat from the methods described in the original paper, the basic idea is much the same.

Mamdani-type inference expects the output member-

ship functions to be fuzzy sets. After the aggregation process, there is a fuzzy set for each output variable that needs defuzzification. It is possible and in many cases much more efficient, to use a single spike as the output membership functions rather than a distributed fuzzy set. This type of output is sometimes known as a singleton output membership function and it can be thought of as a pre-defuzzified fuzzy set. It enhances the efficiency of the defuzzification process because it greatly simplifies the computation required by the more general Mamdani method, which finds the centroid of a two-dimensional function. Rather than integrating across the two-dimensional function to find the centroid, we use the weighted average of a few data points. Sugeno-type systems support this type of model. In general, Sugeno-type systems can be used to model any inference system in which the output membership functions are either linear or constant.

According to Abraham (2005) NF computing is a popular framework for solving complex problems. If one has knowledge expressed in linguistic rules, one can build a fuzzy inference system (FIS) and if one has data, or can learn from a simulation (training) then one can use artificial neural networks (ANNs). For building a FIS, one has to specify the fuzzy sets, fuzzy operators and the knowledge base. Similarly, for constructing an ANN for an application the user needs to specify the architecture and learning algorithm. An analysis reveals that the drawbacks pertaining to these approaches seem complementary and therefore, it is natural to consider building an integrated system combining the concepts. While the learning capability is an advantage from the viewpoint of FIS, the formation of linguistic rule base will be advantage from the viewpoint of ANN.

In the simplest way, a cooperative model can be considered as a preprocessor wherein ANN learning mechanism determines the FIS membership functions or fuzzy rules from the training data. Once the FIS parameters are determined, ANN goes to the background (Jang et al., 1995). The rule based is usually determined by a clustering approach (self organizing maps) or fuzzy clustering algorithms. Membership functions (MF) are usually approximated by neural network from the training data.

In a concurrent model, ANN assists the FIS continuously to determine the required parameters especially if the input variables of the controller cannot be measured directly. In some cases the FIS outputs might not be directly applicable to the process. In that case ANN can act as a postprocessor of FIS outputs (Abraham, 2005).

In fused NF architecture, ANN learning algorithms are used to determine the parameters of FIS. Fused NF systems share data structures and knowledge representations. A common way to apply a learning algorithm to a fuzzy system is to represent it in a special ANN like architecture. However, the conventional ANN learning algorithms (gradient descent) cannot be applied directly

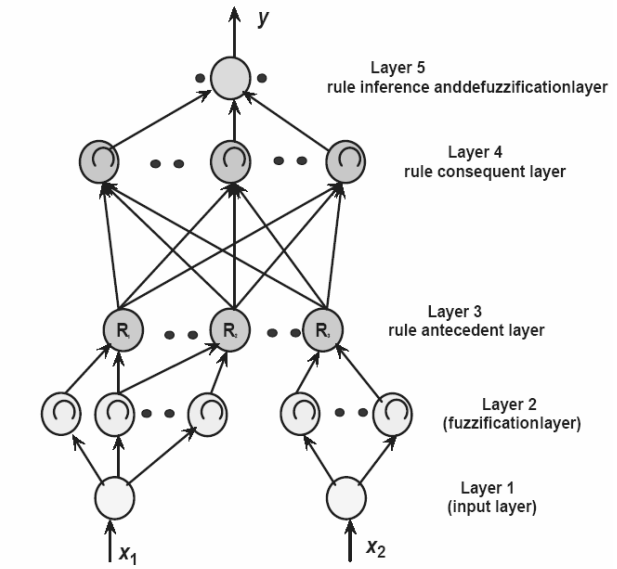

Figure 1. Mamdani fuzzy inference system structure (Abraham, 2005).

to such a system as the functions used in the inference process are usually non differentiable. This problem can be tackled by using differentiable functions in the inference system or by not using the standard neural learning algorithm. Some of the major woks in this area are GARIC (Bherenji et al., 1992), FALCON (Lin et al., 1991), ANFIS (Jang, 1992), NEFCON, FUN (Sulzberger et al., 1993), SONFIN (Feng 1998), FINEST, EFuNN (Kasabov et al., 1999), dmEFuNN (Kasabov et al., 1999), evolutionary design of Neuro-Fuzzy systems and many others.

Architecture of Mamdani fuzzy inference system is illustrated in Figure 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.

Takagi-Sugeno fuzzy inference system

Abraham (2005) discussed Takagi-Sugeno fuzzy inference systems make use of a mixture of back propagation to learn the membership functions and least mean square estimation to determine the coefficients of the linear combinations in the rule's conclusions and that makes it Takagi-Sugeno neuro-fuzzy system. A step in the learning procedure got two parts: In the first part, the input patterns are propagated and the optimal conclusion parameters are estimated by an iterative least mean square procedure, while the antecedent parameters (membership functions) are assumed to be fixed for the current cycle through the training set. In the second part, the patterns are propagated again and in this epoch, back propagation is used to modify the antecedent parameters, while the conclusion parameters remain fixed (Jang et al., 2004). This procedure is then iterated. Architecture of Takagi-Sugeno neuro-fuzzy system is illustrated in Figure 2.

The detailed functioning (Abraham, 2005) of each layer is as follows:

Layers 1, 2 and 3: These layers functions the same way as Mamdani FIS.

Layer 4 (Rule strength normalization): Every node in this layer calculates the ratio of the i^{th} rule's firing strength to the sum of all rules firing strength.

$$\bar{\omega}_i = \frac{\omega_i}{\omega_1 + \omega_2}, i = 1, 2... \quad (1)$$

Layer-5 (Rule consequent layer): Every node *i* in this layer is with a node function

$$\bar{\omega}_i f_i = \bar{\omega}_i (p_i x_i + q_i x_2 + r_i) \quad (2)$$

Where; ω_i is the output of layer 4 and $\{f_i; q_i; r_i\}$ is the parameter set. A well established way is to determine the consequent parameters using the least means squares algorithm.

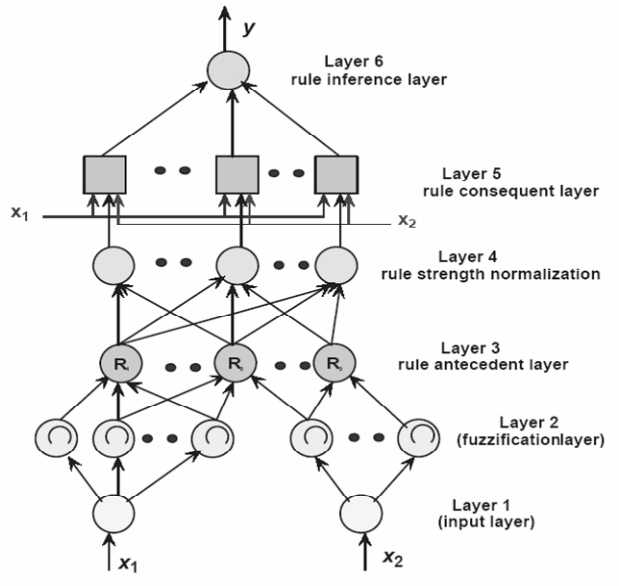

Figure 2. Tagaki-Sugeno neuro-fuzzy system structure (Ardil et al., 2009).

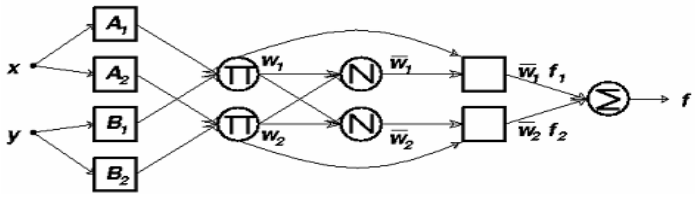

Figure 3. Architecture of ANFIS implementing Tsukamoto fuzzy inference system (Jang, 1992).

Layer-6 (Rule inference layer): The single node in this layer computes the overall output as the summation of all incoming signals Overall output = X_i

$$OverallOut\,put = \sum_i \varpi_i f_i = \frac{\sum_i \omega_i f_i}{\sum_i \omega_i} \quad (3)$$

Adaptive network based fuzzy inference system (ANFIS)

ANFIS proposed by Jang (1992), is perhaps the first integrated hybrid neuro-fuzzy model and the architecture is very similar to Figure 2, a modified version of ANFIS which is shown in Figure 3 is capable of implementing the Tsukamoto fuzzy inference system as depicted in Figure 4. Hence, ANFIS have non-linear antecedent parameters and linear consequent parameters. These consequent and antecedent parameters are learned using hybrid learning algorithm. More specifically, in the forward pass

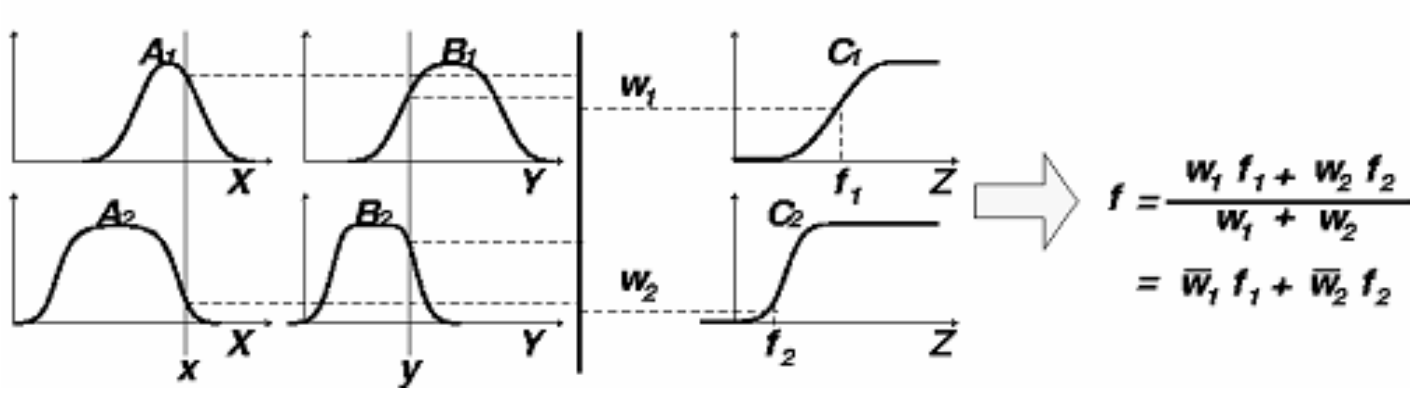

Figure 4. Tsukamoto fuzzy reasoning (Jang, 1992).

of the hybrid learning algorithm, functional signals go forward till layer 4 and consequent parameters are identified by the least square estimates. In the backward pass, the error rates propagate backward and the premise parameters are updated by the gradient descent. In the Tsukamoto FIS, the overall output is the weighted average of each rules crisp output induced by the rules firing strength (the product or minimum of the degrees of match with the premise part) and output membership functions. The output membership functions used in this scheme must be monotonically non-decreasing. The first hidden layer is for fuzzification of the input variables and T-norm operators are deployed in the second hidden layer to compute the rule antecedent part. The third hidden layer normalizes the rule strengths followed by the fourth hidden layer where the consequent parameters of the rule are determined. Output layer computes the overall input as the summation of all in coming signals. In ANFIS, the adaptation (learning) process is only concerned with parameter level adaptation with in fixed structures. The structure of ANFIS ensures that each linguistic term is represented by only one fuzzy set.

In most fuzzy systems, fuzzy rules were obtained from the human expert. However, every expert does not want to share his knowledge and there is no standard method that exists to utilize expert knowledge. As a result, ANNs were incorporated into fuzzy systems to be able to acquire knowledge automatically by learning algorithms. The learning capability of the NNs was used for automatic fuzzy if-then rules generation (Czogala et al., 2000).

PROPOSE METHODOLOGY

Find the structural code and design attributes

The first step is to find the structural code and design attributes of software systems that is, 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.

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.

Analyze and refine metric values

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

Empirical evaluation of different machine learning algorithms

In this step, the aim is to find the best algorithm for classification of software components into different levels of impact of fault. In order to model the polished dataset of the previous step, Hybrid Fuzzy-GA and PSO-trained neural network techniques are explored along with the earlier experimented fuzzy and neuro-fuzzy techniques as discussed in (Sandhu et al., 2007).

Fuzzy and neuro-fuzzy techniques

According to (Jang et al., 1995), a fuzzy system can be considered to be a parameterized nonlinear map, called *f*, which can be expressed as (4):

$$f(x) = \frac{\sum_{l=1}^{m} y^{l} \left(\prod_{i=1}^{n} \mu_{A_i^{l}}(x_i) \right)}{\sum_{l=1}^{m} \left(\prod_{i=1}^{n} \mu_{A_i^{l}}(x_i) \right)} \quad (4)$$

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 $x = [x_1, x_2, x_3, \ldots x_m]$ 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 = p1x + 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 (5).

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \quad (5)$$

Let

$$\overline{w_i} = \frac{w_i}{w_1 + w_2} \quad (6)$$

Then *f* can be written as (7).

$$f = \overline{w_1} f_1 + \overline{w_2} f_2 \quad (7)$$

The steps for designing of adaptive neuro-fuzzy system are already discussed in section II of the paper.

PSO trained neural network system

The following are the steps for the hybrid PSO-neural network based modeling system:

(1) Designing of neural network and perform training: In this step the following three sub steps are there:

(a) Calculate the minimum and maximum values in the attribute of input and setting the various parameters of feed-forward back-propagation network by like:

(i) Size of the feed-forward back-propagation neural network.
(ii) Type of transfer function of each layer to be used.
(iii) Type of back-propagation network training function.
(iv) Back-propagation weight/bias learning function.

(b)Generate the neural network.
(c) Perform the training of the neural network with PSO technique discussed after the testing phase using the training dataset.
(2) 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.

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by the following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest (Web URL: http://www.swarmintelligence.org/tutorials.php).

After finding the two best values, the particle updates its velocity and positions with following equations (8) and (9).

$$v[\,] = v[\,] + c_1 * rand(\,) * (pbest[\,] - present[\,]) + c_2 * rand(\,) * (gbest[\,] - present[\,]) \quad (8)$$

$$present[\,] = persent[\,] + v[\,] \quad (9)$$

where; v[ ] is the particle velocity, persent[ ] is the current particle (solution), pbest[ ] and gbest[ ] are defined as stated before, rand ( ) is a random number between (0,1) ans c_1, c_2 are learning factors usually $c_1 = c_2 = 2$.

As mentioned in (Web URL: http://www.swarmintelligence.org/tutorials.php), the pseudo code of the procedure is as follows:

For each particle
Initialize particle

END

Do
For each particle
Calculate fitness value.
If the fitness value is better than the best fitness value (pBest) in history. set current value as the new pBest.

End

Choose the particle with the best fitness value of all the particles as the gBest.
For each particle,
Calculate particle velocity according equation (8).
Update particle position according equation (9).

End

While maximum iterations or minimum error criteria is not attained.

Particles' velocities on each dimension are clamped to a maximum velocity V_{max}. If the sum of accelerations would cause the velocity on that dimension to exceed V_{max}, which is a parameter specified by the user, then the velocity on that dimension is limited to V_{max}.

Hybrid fuzzy-GA based approach

Genetic algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. As you can guess, genetic algorithms are inspired by Darwin's theory of evolution. Simply said, problems are solved by an evolutionary process resulting in a best (fittest) solution (survivor) - in other words, the solution is evolved.

Rosenberg introduced evolutionary computing in the 1960s in his work "Evolution strategies" (Evolutions strategies in original). Other researchers then developed his idea. Genetic algorithms (GAs) were invented by John Holland and his students and colleagues. These lead to Holland's book" Adaption in Natural and Artificial Systems" published in 1975. In 1992, John Koza used genetic algorithms to evolve programs to perform certain tasks. He called his method "genetic programming" (GP).
The steps of the hybrid fuzzy-GA algorithm are:

(1) Read the input as the metric values.
(2) Find the nearest match with example data using Euclidean Distance.
(3) Calculate the output of the fuzzy inference system corresponding to the Input set.

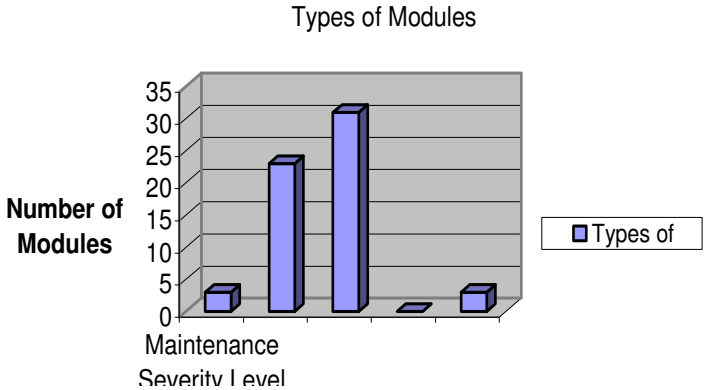

Figure 5. Graphical representation of details of the type of modules in the dataset.

(4) Treat FIS value and the nearest match value as chromosome and convert the values into binary after multiplying the values with 100.
(5) Perform the single point cross over at randomly generated point in the selected parents.
(6) Get the offsprings from the previous step and generate the output by dividing the offspring with 100.
(7) Repeat the process up to 100 generations or the error reduces to certain minimum level.
(8) The model generated is evaluated against the testing data on the different criteria is discussed in the next steps.

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 (Bherenji et al., 1992). The formula for calculating MAE is given in equation shown below:

$$\frac{|a_1-c_1|+|a_2-c_2|+\ldots+|a_n-c_n|}{n} \quad (10)$$

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 (Challagulla et al., 2005). It is just the square root of the mean square error as shown in equation given below:

$$\sqrt{\frac{(a_1-c_1)^2+(a_2-c_2)^2+\ldots+(a_n-c_n)^2}{n}} \quad (11)$$

Conclusions drawn

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

RESULTS AND 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. The fault severity level-1 means the fault is having highest severity level and that need urgent attention to be removed as it falls in the category of major faults. The fault severity level-2 means the fault is having high severity level and that need less urgent attention as compared to level-1 faults. The fault severity level-3 means the fault is having medium severity level and that require less attention as compared to level-2 faults. The fault severity level-4 means the fault is less severe and that need low attention to be removed. The faults of this level belongs to category of minor faults and fault severity level-5 means the fault is least severe means having negligible effect of the performance of system and belongs to the category of the purely minor faults or no faults category. Graphical details of the type of modules in the dataset are shown in Figure 5. The details of the modules present in the dataset are shown in Table 1.

The first step is to find the structural code and design attributes of software systems that is, 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.

Table 1. Details of the type of modules in the dataset.

Label	Count
1	3
2	23
3	31
4	0
5	3

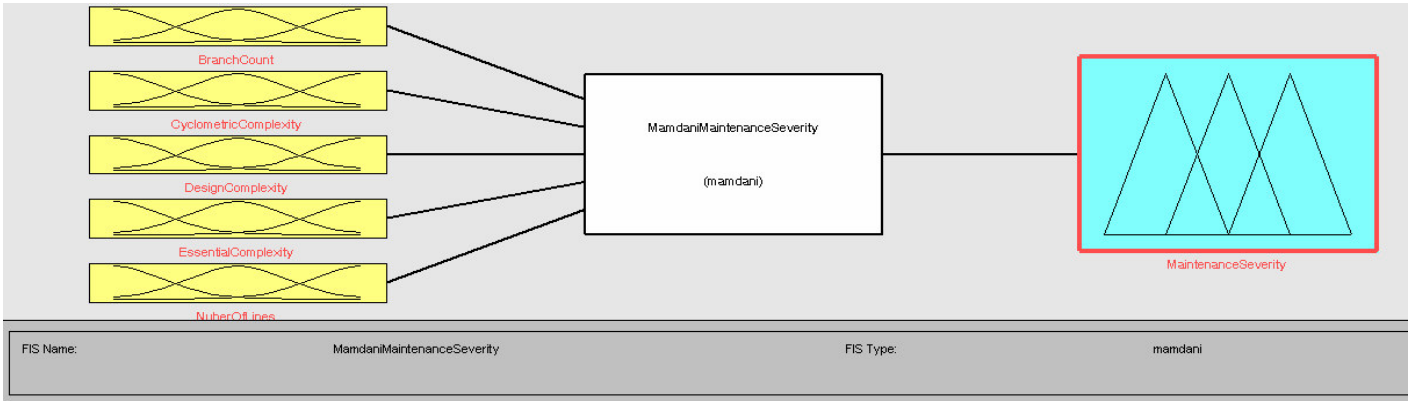

Figure 6. Mamdani based FIS inference system.

- Essential_Complexity.
- Number_Of_Lines.

When analyzing performance of all the WEKA project (Web URL WEKA: www.cs.waikato.ac.nz/~ml/weka/) 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 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 (Mamdani et al, 1975) five metrics are considered as input attributes and one attribute named as “software maintenance severity level” is used as output attribute as shown in Figure 6.

Each input and output attribute is represented with fifteen fuzzy sets and the membership function value of the each attribute is shown in Figure 7. In Figure 8, fifteen rules used for the inference of the Mamdani based FIS are shown.

During the testing phase of the Mamdani based fuzzy inference system (Mamdani et al., 1975), fifteen inputs are used and it shows 0.2183, 0.3066 and 80% as MAE, RMSE and Accuracy values respectively.

As performance of adaptive neuro-fuzzy inference system is found to be the best out of all the hybrid NF systems (Abraham, 2001) 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 (Jang et al., 2004). 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 Figure 9:

- 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.

The graphical representation of the input exemplars for the NF system is shown in Figure 10.

The NF system is trained using a hybrid learning algorithm using both least squares method and back-propagation. In the forward pass, the consequent para-

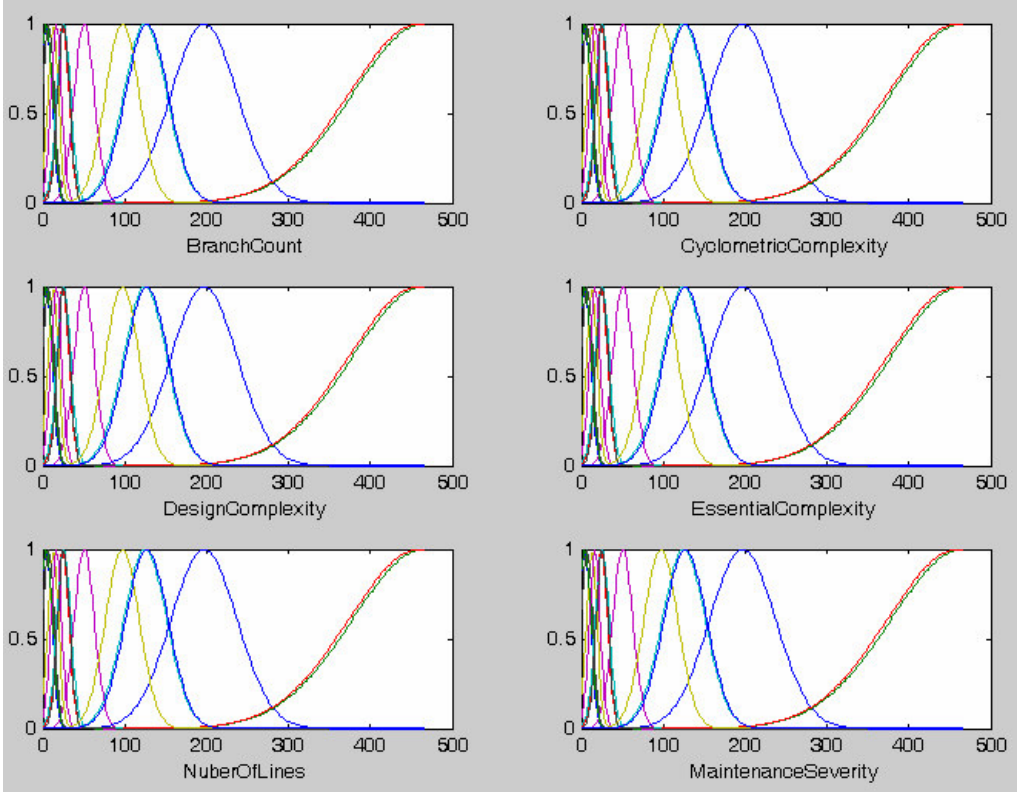

Figure 7. Membership functions of the input and output attributes.

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)

Figure 8. Fifteen rules of the Mamdani based FIS.

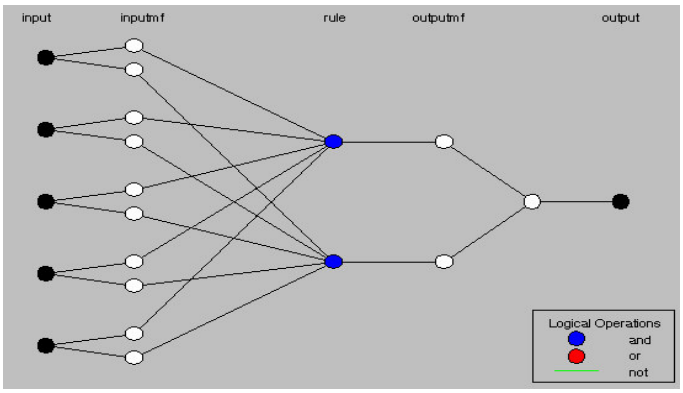

Figure 9. Structure of adaptive neuro-fuzzy inference system.

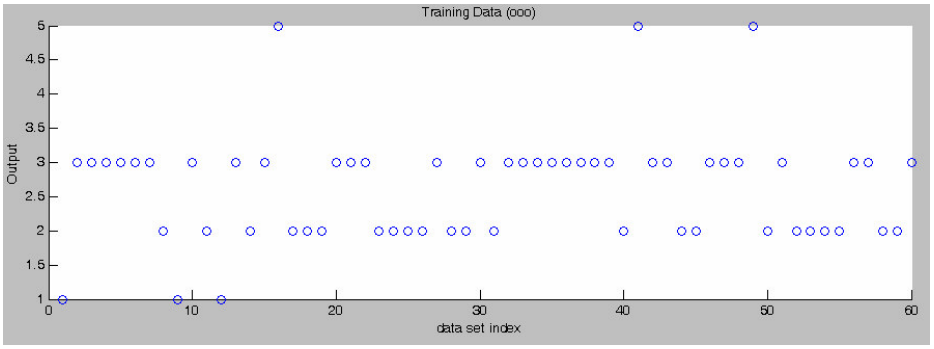

Figure 10. Training data for the neuro-fuzzy system.

meters are identified using least squares and in the back-ward pass the premise parameters are identified using back-propagation. 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.

PSO-neural network results

In the implementation of the PSO trained neural network particle swarm optimization toolbox for matlab (Birge, 2003) is used. Size of the feed-forward back-propagation neural network is set as [5 5 1] means there are 5 neurons in the input layer, 5 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 neutral 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:

(1) Maximum iterations: 2000;
(2) Population size: 25;
(3) Acceleration constants (for type = 0): [2,2];
(4) Inertia weights (for type = 0): [0.9,0.4];
(5) Minimum error gradient: 1e-9;
(6) Iterations at error grad value before exit: floor (0.2*trainParam.maxit);
(7) Error goal: 0;
(8) Type of PSO: Trelea.

After the training, the architecture of feed forward neural network is shown in Figure 11 where the Bright Green line shows more positive weight, bright red line shows more negative weight and dashed white line shows zero weight

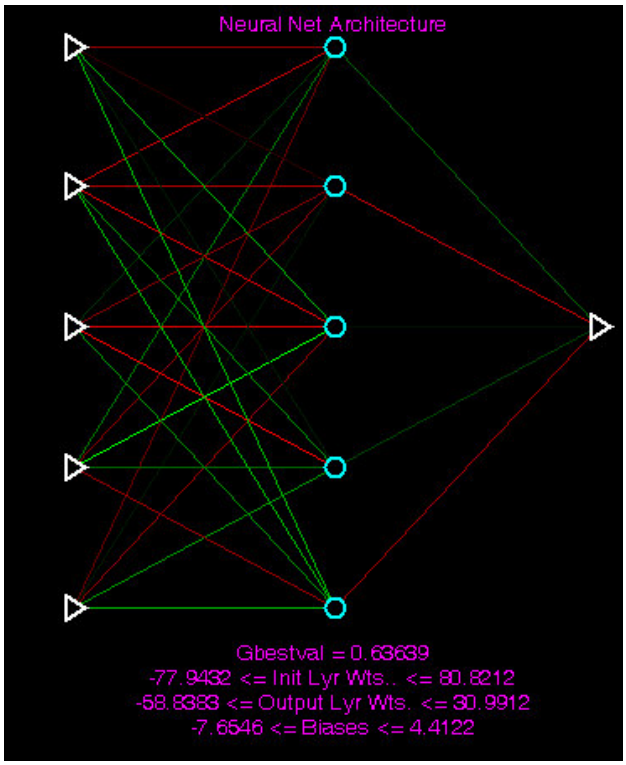

Figure 11. Architecture of feed forward neural network.

that is, no connection between the neurons. In the testing phase MAE, RMSE and Accuracy values of the system are 0.5278, 0.6112 and 66.6667 respectively as shown in Table 2. The plot of Global best value (Gbest) versus Iterations is shown in figure 12. 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.

The fuzzy-GA hybrid system is also implemented in Matlab 7.4. During the testing of the developed system, 0.1220, 0.1587 and 100 are calculated as MAE, RMSE and Accuracy values for the testing dataset as shown in

Table 2. Results of the proposed systems for the fault severity prediction.

Performance Criteria	Prediction model			
	Mamdani based fuzzy inference system	Neuro-fuzzy system	PSO trained neural network	Fuzzy-GA system
MAE	0.2183	0.1571	0.5278	0.1220
RMSE	0.3066	0.2140	0.6112	0.1587
Accuracy	80	93.3333	66.6667	100

Figure 12. Graph between gbest value and Iteration index.

Table 2.

Conclusion and future direction

On comparing various WEKA's machine learning algorithms as mentioned in (Ardil et al., 2009), it is observed that logistic model trees and simple logistic algorithms are better techniques in prediction of fault severity level for NASA's public domain defect dataset coded in Perl programming language. Both the algorithms use same classification algorithm that is, logistic classifier and have shown least mean absolute error and root mean square error values: 0.2145 and 0.3285 among other algorithms listed in WEKA's project. During the testing phase, LMT and simple logistic algorithm has shown 86.66% Accuracy.

When experimented with Mamdani based fuzzy inference system (Mamdani et al., 1975), the testing phase results are comparatively equivalent 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 performance of feed forward neural network trained with particle swarm optimization algorithm is not promising as the testing phase MAE, RMSE and Accuracy values of the developed system are 0.5278, 0.6112 and 66.6667% respectively. The bad performance of this technique could be due to the incapability of the PSO to train the neural network designed. The prediction Accuracy percentage is the lowest among Mamdani based fuzzy inference system, logistic model trees and simple logistic algorithms Accuracy values.

Fuzzy –GA hybrid algorithm is proved to be best as compared to the other algorithms considered in this work with 0.1220, 0.1587 and 100 as MAE, RMSE and Accuracy values respectively for the testing dataset. In such data search application the design and developed fuzzy GA code has shown its superiority because it includes the advantages of fuzzy as well as genetic algorithms. Fuzzy provides a robust inference mechanism with no learning and adaptability while on the other hand, the genetic algorithms provide an efficient data modification in the wake of optimization objectives of given application. Neuro -fuzzy algorithm is definitely superior to fuzzy algorithm as it inherits adaptability and learning. The performance of neuro -fuzzy algorithm is somewhat satisfactory and better than fuzzy system. From the simulation and the result obtained, it has been shown that the percentage average error is least in the case of fuzzy-GA algorithms and maximum in the case of PSO trained neural network algorithms. Neuro-fuzzy algorithm has yielded accuracy lying between the accuracy levels as in the case of fuzzy and fuzzy-GA algorithms.

It is therefore, the best algorithm for classification of the software components into different level of severity of impact of the fault is found to be fuzzy-GA 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 modules can be debugged timely. Hence, for non linear and complex engineering applications involving control, inference and analysis by and large fuzzy-GA is an efficient technique.
The future work can be extended in following directions:

i)The performance of the PSO based technique can further be investigated by increasing the number of hidden layers and changing the population size.
ii)This work can be extended to other programming language datasets.
iii) More algorithms can be evaluated and then we can

find the best algorithm.
iv) Further investigation can be done and the impact of attributes on the fault tolerance can be found.
v) Other dimensions of quality of software can be considered for mapping the relation of attributes and fault tolerance.

REFERENCES

Abraham A (2001). Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques, Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence. Lecture Notes in Computer Science. Springer-Verlag Germany. Jose Mira and Alberto Prieto (Eds.). 2084: 269-276.

Abraham A (2005). Hybrid Intelligent Systems: Evolving Intelligence in Hierarchical Layers. Stud. Fuzziness Soft Comput. 173:159–179.

Ardil E, Ucer E, Sandhu PS (2009). Software Maintenance Severity Prediction with Soft Computing Approach. International Conference on Computer, Electrical, and Systems Science and Engineering. Penang (Malaysia). 38: 139-144.

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

Benlarbi S, Emam Khaled El, Geol N (1999). Issues in Validating Object-Oriented Metrics for Early Risk Prediction. 10th Int. Symp. on Software Reliability Engineering, ISSRE'99, Boca. pp. 17-18.

Bherenji HR, Khedkar P (1992). Learning and Tuning Fuzzy Logic Controllers through Reinforcements. IEEE Trans. Neural Networks. 3: 724-740.

Birge B (2003). PSOt - a particle swarm optimization toolbox for use with Matlab. IEEE Proceedings of Swarm Intelligence Symposium SIS '03. pp. 182-86.

Challagulla VUB, Bastani FB, Yen IL, Paul RA (2005). Empirical assessment of machine learning based software defect prediction techniques. 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. WORDS 2005. pp. 263-270.

Czogala E, Leski J (2000). Fuzzy and Neuro-Fuzzy Intelligent Systems. Physica-Verlag Heidelberg. New York.

Denaro G (2000). Estimating Software Fault-Proneness for Tuning Testing Activities. Proceedings of the 22nd International Conference on Software Engineering (Limerick, Ireland). pp. 704-706.

Deodhar M (2002). Prediction Model and the Size Factor for Fault-proneness of Object Oriented Systems. MS Thesis Michigan Tech. University.

Eman K, Benlarbi S, Goel N, Rai S (2001). Comparing case-based reasoning classifiers for predicting high risk software components. J. Syst. Software 55(3): 301 – 310.

Feng JC, Teng LC (1998), An Online Self Constructing Neural Fuzzy Inference Network and its Applications. IEEE Trans. Fuzzy Systems. 6(1): 12-32.

Fenton NE, Neil M (1999). A Critique of Software Defect Prediction Models. IEEE Trans. Software Eng. arch. 25(5): 675 - 689.

Hudepohl JP, Aud SJ, Khoshgoftaar TM, Allen EB, Mayrand JE (1996). Software Metrics and Models on the Desktop. IEEE Software. 13(5): 56-60.

Jang JSR, Sun CT (1995). Neuro-Fuzzy Modeling and Control. Proc. IEEE. 83(3): 378-406.

Jang JSR, Sun CT, Mizutani E (2004). Neuro-Fuzzy and Soft Computing - A computational approach to learning and machine intelligence. Pearson Education. Singapore. Indian edition. Delhi.

Jang R (1992). Neuro-Fuzzy Modeling: Architectures, Analyses and Applications. Ph.D. Thesis. University of California(Berkeley).

Kasabov N, Song Q (1999). Dynamic Evolving Fuzzy Neural Networks with 'm-out-of-n' Activation Nodes for On-line Adaptive Systems. Technical Report TR99/04. Department of information science, University of Otago.

Khoshgoftaar TM, Allen EB, Kalaichelvan KS, Goel N. (1996). Early quality prediction: a case study in telecommunications. IEEE Software 13(1): 65-71.

Khoshgoftaar TM, Gao K, Szabo RM (2001). An Application of Zero-Inflated Poisson Regression for Software Fault Prediction. Proceedings of 12th International Symposium on Software Reliability Engineering. pp: 66 -73.

Khoshgoftaar TM, Munson JC (1990). Predicting Software Development Errors using Complexity Metrics. IEEE J. Selected Areas Commun. 8(2): 253 -261.

Khoshgoftaar TM, Seliya N (2002). Tree-based software quality estimation models for fault prediction. METRICS 2002. 8th IIIE Symposium on Software Metrics. pp: 203-214.

Lanubile F, Lonigro A, Visaggio G (1995). Comparing Models for Identifying Fault-Prone Software Components. Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering. pp. 312-19.

Lin CT, Lee CSG (1991). Neural Network based Fuzzy Logic Control and Decision System. IEEE Trans. Comput. 40(12): 1320-1336.

Mamdani EH, Assilian S (1975). An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Man-Machine Studies, Vol. 7(1): 1-13.

Menzies T, Ammar K, Nikora A, Stefano S (2003). How Simple is Software Defect Prediction? J. Empirical Software Eng. Oct.

Munson J, Khoshgoftaar T (1990). Regression Modeling of Software Quality: An Empirical Investigation. J. Info. Software Technol. 32(2): 106 - 114.

Munson JC, Khoshgoftaar TM (1992). The detection of fault-prone programs. IEEE Trans. Software Eng. 18(5): 423-433.

Sandhu PS, Kumar S, Singh H (2007). Intelligence System for Software Maintenance Severity Prediction. J. Comput. Sci. 3(5): 281-288.

Seliya N, Khoshgoftaar TM, Zhong S (2005). Analyzing software quality with limited fault-proneness defect data. Ninth IEEE international Symposium. pp. 89-98.

Sulzberger SM, Tschicholg-Gurman NN, Vestli SJ (1993). FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks. Proceedings of IEEE Conference on Neural Networks. San Francisco. pp 312-316.

Web URL WEKA: www.cs.waikato.ac.nz/~ml/weka/.

Web URL: http://www.swarmintelligence.org/tutorials.php

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

Zadeh LA (1973). Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans. Syst. Man Cybernetics 3(1): 28-44.