A Hybrid Neuro Fuzzy Approach for Bug Prediction using Software Metrics

Aditi Thakur[1], Dr. Ajay Goel[2]

[1]*Research Scholar & Professor*
[1,2]*Department of Computer Science, Baddi University of Emerging Sciences and Technology*
Baddi, Solan.H.P-173205, India

Abstract— *Software quality is an important factor since software systems are playing a key role in today's world. There are several perspectives within the field on software quality measurement. This measurement is frequently used so many defects which can cause crashes, failures, or security breaches encountered in the software. Testing the software for such defect is essential to enhance the quality. However, due to the increase in intricacy of software manual testing was becoming extremely time consuming task and some automatic supporting tools have been developed. One such supporting tool is defect prediction models. Some defect prediction models can be found in the literature and most of them share a common procedure to develop the models. In general, the models' development procedure indirectly assumes that underlying data distribution of software systems is relatively stable over time. But, this assumption is not necessarily true and consequently, the reliability of those models is doubtful at some points in time.*

Keywords *- feature selection, ANFIS, LDA, parameters, approaches.*

I. INTRODUCTION

This chapter discusses the background knowledge about the thesis topic and the motivation behind the bug prediction. It also discusses the categories of bug prediction approaches and bug prediction approaches.

1.1 Background

Bug prediction generated widespread interest for a considerable period of time, leading to more than a hundred publications in the last ten years. The driving scenario is resource allocation: Time and manpower being finite resources, it makes sense to assign personnel and resources to software components that are likely to generate bugs. Researchers proposed code metrics, process metrics (*e.g.*, number of changes, recent activity) or previous defects. The jury is still out on the relative performance of these approaches. Most of them were evaluated in isolation, or were compared to only few other approaches.
Limitations of Previous Models:

In 2000s, there had been existed several limitations for defect prediction:

The first limitation was the prediction model could be usable before the product release for the purpose of quality assurance. However, it would be more helpful if we can predict defects whenever we change the source code.

The second limitation is that it was not possible or difficult to build a prediction model for new 5 projects or projects having less historical data. As use of process metrics was getting popular, this limitation became the one of the most difficult problems in software defect prediction studies.

The third limitation was from the question, "are the defect prediction models really helpful in industry?" In this direction, several studies such as case study and proposing practical applications have been conducted.

1.2 Motivations

We surveyed approaches to defect prediction, the kind of data they require and the various datasets on which they were validated. Here we recall the main bug prediction families and observe why techniques are difficult to compare. Defect prediction techniques may be divided into 3 categories:

1. ***SCM approaches*** use information extracted from versioning systems, assuming that recently or frequently changed files are the most probable source of future bugs.

2. ***Single-version approaches***. These technique do not need the history of the system, but analyze its current state in more detail, using a variety of metrics.
.
3. ***Other approaches*** exploit different types of data such as network metrics computed on developer-artifact networks or graphs of binary dependencies, cohesion measurements based on information retrieval techniques, call structure metrics, *etc.*

1.3 Overview of Software Bug Prediction Process

Fig 1 shows the common process of software defect prediction based on machine learning models. Most software defect prediction studies have utilized machine learning techniques. Each instance can represent a system, a software component, a source

code file, a class, a function, and a code change according to prediction granularity.

The prediction model can predict whether a new instance has a bug or not. The prediction for Bug-proneness (buggy or clean) of an instance stands for binary classification, while that for the number of bugs in an instance stands for regression.

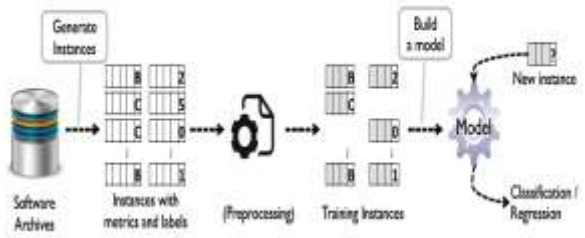

Fig 1 Common Process of Software Defect Prediction

Table 1: Categories of bug prediction approaches

Type	Rational	Used By
Change metrics	Bugs are caused by changes.	
Previous defects	Past defects predict future defects.	
Source code metrics	Complex components are harder to change, and hence error-prone.	
Entropy of changes	Complex changes are more error prone than simpler ones.	
Churn (source code metrics)	Source code metrics are a better approximation of code churn.	
Entropy (source code metrics)	Source code metrics better describe the entropy of changes.	

1.5 Bug Prediction Approaches

A. Code Churn

Code Churn approach mainly based on the lines modified in the software system. In this, source code files purely assumed as text files. It computes lines modified in a source code file, like number of added lines, number of lines deleted from source code and lines changed per revision.

B. Fine Grained Source code Changes (SCC)

H1:- SCC posses a stronger correlation with the number of bugs then lines modified.
H2:- SCC gives better results to differentiate source files into buggy and not bug prone files than LM.
H3:- SCC gives better results than LM while predicting the number of bugs in source files.

C. Fine Grained Bug Severity Prediction

In this approach previous bug reports are automatically analyzed. These are analyzed on the basis of assigned severity labels. These labels assigned on the basis of how much that bug effect the system and when it is necessary to detect that bug and remove it. In this approach five severity labels are assigned named: blocker, critical, major, minor, and trivial. In this approach for comparison with the previous data, instead of using NASA data set, the files that stored in the bugzilla are used for comparison. The limitation of this approach is that we make assumption of the complexion of the bug on the basis of its previous occurrence but there may be that bug can create more critical then the time when it occurred in other system.

D. Reducing Features to improve Code Changes Based Bug Prediction

In this approach machine learning based classifiers are trained on the selected features of history data. After that these classifiers are used on the new systems to predict the bugs that are produced due to changes in the system.

E. Sampling-based approach to software defect prediction

In this approach some modules are taken as samples from all modules of source code. A classification model is constructed based on sample to check the quality of sampled module. Then this classification model is used on the un-sampled modules to predict the defects.

1.6 Applications on Defect Prediction

One of major goals of defect prediction models is effective resource allocation for inspecting and testing software products. However, the case studies using defect prediction models in industry is few. In this reason, many studies consider cost-effectiveness. A recent case study conducted in Google by based on the number of closed bugs found that developers preferred Rahman's algorithm.

II. LITERATURE SURVEY

This chapter contains the literature survey which describes the related work previously done by the researchers on the bug prediction.

A. E. Hassan [1] proposed a new approach to predict the bugs in the software on the basis of how much complex the code change process is. This approach not only focuses on how much code is changed but also focuses on when the code is changed. This approach is based on the assumption that complex change process of code creates more defaults in the software system. In this paper he give three models to measure the code change process's complexity.

Emanuel Giger et. al. [2] compares the two bug prediction approaches. They compare fine grained-source code changes approach and code churn approach using Eclipse platform's dataset with

machine learning algorithm. They compare both approaches bases on three hypotheses. In first hypothesis they assume that there is good correlation of SCC with the number of bugs as compare to lines modified.

Youn tian et. al. [3] purposed a new approach for bug prediction. In this approach previous bug reports are analyzed on the basis of severity labels of bugs automatically and new severity labels are assigned on the basis of information of bugs that is collected from the past bug reports. This information report contains the information about bug whether the bug is textual or not contains any text.

Shivkumar Shivaji et. al. [4] proposed multiple feature selection approaches. This proposed method is applied to bug prediction approaches based on classification. The proposed approach removes irrelevant features until optimal performance is reached. The total number of features used for training is substantially reduced, often to less than 10 percent of the original.

Phiradet Bangcharoensap et. al. [5] presented an approach to rapidly place the buggy file. The text mining method categorized the files depending on the similarity among source code and bug report. The code mining method utilizes source code product metrics. The performance of proposed method indicates gain around 20%in top 1 prediction.

Marco D'Ambros and Michele Lanza [6] proposed a new visual technique to reveal the relationship among software and its bugs. Development of software artifacts may be characterized by putting 2 aspects nearer to each other.They validate the approach on 3 very large open source software systems. The approach is based on the application of Discrete Time Figs at any level of granularity.

Alberto Bacchelli et. al. [7] in this paper a metrics has been produced which measures the source code artifacts. Moreover, accuracy of other approaches for prediction of defect may be developed using popular metrics. This approach not only focuses on how much code is changed but also focuses on when the code is changed Also they discovered the data collected in email archives is associated to the defect found in the system.

K. E. Emam et. al. [8] studied an object-oriented design metrics. These metrics helps to make several predictions models. This information report contains the information about bug whether the bug is textual. In this, data is used which is collected from the one version of java application for building prediction model. This model has higher accuracy.

A. E. Hassan and R. C. [9] presented a new approach to assist managers in determining which subsystems to focus their limited resources on. By using this approach faults are located in a timely manner and fix them as soon as possible. The approach uses ideas that have been researched in the literature of web and file systems.

A. Marcus et. al. [10] defines the conceptual cohesion of classes, which captures new and complementary dimensions of cohesion compared to a host of existing structural metrics. PCA of measurement results on three open source software systems statistically supports this fact.

III. METHODOLOGY

The thesis proposes a novel idea of bug prediction using Adaptive Neuro-Fuzzy Inference System aided by Linear Discriminant Analysis for dimensionality reduction. This thesis aims to develop a reliable model for bug prediction. The major inspiration behind this is the fact that more time and manpower should be devoted to those segments of the software which is more prone to bugs. The problem with any bug prediction algorithm is testing its validity with respect to previous approaches due to lack of common platform. Our approach to solve the bug prediction problem is a combination of various segments.

The major step of our methodology is explained below:

1. Data set is first extracted from the Promise Repository of NASA and converted to a readable format.
2. The dataset contains several numbers of columns which needs to be modified.

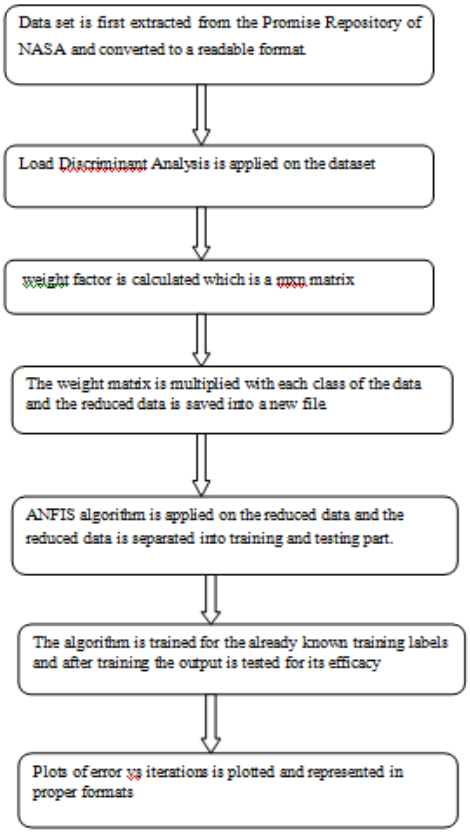

Fig 2. Flow chart of Methodology

3. Load Discriminant Analysis is applied on the dataset and a weight factor is calculated which is

a mxn matrix where m is the number of classes and n is the number of attributes.

4. The weight matrix is multiplied with each class of the data and the reduced data is saved into a new file.

5. ANFIS algorithm is applied on the reduced data and the reduced data is separated into training and testing part.

3.1 Hardware /Software Requirements

Hardware Requirements

• Processor: Intel core 2 Duo with CPU clock rate 2.10 GHZ.

• RAM : Memory two DDR2 with 3 GB

• HDD: SATA with capacity of 500 GB.

3.2 Software Requirements

• Operating System : Microsoft 7 x86
• MATLAB R2013a
• MS Word: MS Word is used for documentation purpose.

This chapter introduces the Adaptive Neuro Fuzzy Inference System and Linear Discriminent Analysis algorithms proposed in methodology.

Adaptive Neuro Fuzzy Inference System (ANFIS): This algorithm was first introduced by Jang. Both the fuzzy logic and neural network makes possible to design the Adaptive Neuro Fuzzy Inference System. ANFIS is basically a *graphical* network representation of Sugeno-type fuzzy systems endowed with the neural learning capabilities.ANFIS are a class of adaptive networks that are functionally equivalent to fuzzy inference systems. The network consist nodes with specific functions collected in layers. ANFIS is able to construct a network realization of IF / THEN rules.

Neuro-fuzzy modeling is a method where the fusion of neural networks and fuzzy logic find their strengths. This fuzzy modelling applied heuristic learning scheme which is derived from the neural network. It is required to completely map the neural network knowledge to fuzzy logic.

3.3 Fuzzy Inference System

Fuzzy inference system performs the following operations:

- Fuzzification of the input variables.
- Determination of membership functions for the parameters.
- Application of the fuzzy operator in the antecedent.
- Implication from the antecedent to the consequent.
- Defuzzification.

3.4 Train Network

In this work we use an Adaptive Neuro Fuzzy Inference System which allows to train and tune a Fuzzy Inference System. It includes a fuzzy system which finds the relationship between quality parameters and the overall QoS(input/output relationship). The system is trained using learning techniques in order to optimize the membership functions based on a collection of input/output dataset.

3.5 Structure of the ANFIS network

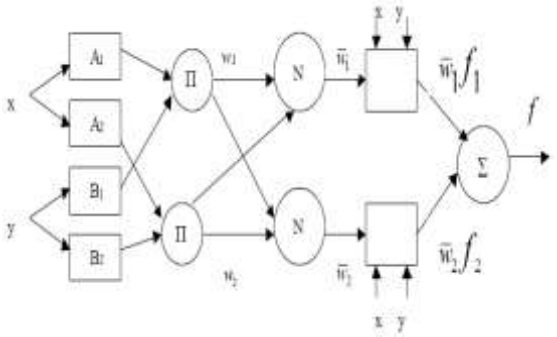

Fig 3 Equivalent ANFIS

A two inputs (*x* and *y*) and one output (*z*) ANFIS

Rule 1: IF *x* is A_1 and *y* is B_1, then $f_1 = p_1x + q_1y + r_1$

Rule 2: IF *x* is A_2 and *y* is B_2, then $f_2 = p_2x + q_2y + r_2$

3.6 ANFIS architecture

Layer 1(Input nodes): Nodes produces membership grades of the crisp inputs that belong to each of convenient fuzzy sets by using the membership functions.

$$O_{1,i} = \mu_{A_i}(x) \quad i = 1,\ 2$$

$$O_{1,i} = \mu_{B_{i-2}}(x) \quad i = 3,\ 4$$

$\mu_{A_i}(x)$ and $\mu_{B_i}(x)$: any appropriate parameterized membership functions

$$\mu_{A_i}(x) = \frac{1}{1 + [\frac{(x - c_i)^2}{a_i^2}]^{b_i}}$$

$\{a_i, b_i, c_i\}$ ➔ premise parameters

Layer 2: fixed nodes with function of multiplication

$$O_{2,i} = w_i = \mu_{A_i}(x) \times \mu_{B_i}(x) \quad i = 1,\ 2$$

(Firing strength of a rule)

Layer 3: fixed nodes with function of normalization

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2} \qquad i = 1,\ 2$$

(Normalized firing strength)

Layer 4: adaptive nodes

$$O_{4,i} = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i)$$

$\{p_i, q_i, t_i\}$ ➔Con sequent parameters

Layer 5: a fixed node with function of summation

$$O_{5,1} = overall\ output = \sum_i \overline{w}_i f_i$$

The computation time is reduced by using a dimensionality reduction technique known as Linear Discriminant Analysis which is an improved algorithm over the traditional Principal Component Analysis.

Linear Discriminant Analysis (LDA) is popularly used as dimensionality reduction approach in the pre-processing step for pattern-classification and machine learning applications.

LDA is closely related to analysis of variance and regression analysis, which also attempt to express one dependent variable as a linear combination of other features or measurements.

Linear discriminant analysis has continuous independent variables and a categorical dependent variable.

3.7 LDA Algorithm:

Input: data matrix A.
Output: transformation matrix G.
1: Calculate S and Sb.
2: Perform the singular value decomposition of Sw as Sw = URVT, where U = V because Sw is symmetric.
3: Let V = [v1,. . .,vq,vq+1,. . .,vm] and Q = [vq+1,. . .,vm], q = rank(Sw).
4: Compute e Sb, where e Sb ¼ QQTSb.

5: Calculate the eigenvectors corresponding to the set of the largest eigenvalues of e Sb and use them to form the transformation matrix G.

Applications of Linear Discriminant Analysis:

- Bankruptcy prediction
- Face recognition
- Marketing
- Biomedical studies
- Earth Science

3.8 Data Reduction using LDA

After the LDA is applied on the data, a weight for data manipulation is found is transformation of data is done by multiplying the weights with each data point. The output of the LDA is a weight matrix which is of size mxn where 'n' is the total number of attributes and n is the total number of classes. The weights of row are multiplied with the data of all points which belongs to first class. Similarly, it is extended.

$$\mathbf{D_{OUT} = W_{IN} * D_{IN}}$$

For class i=1,2,3,4

IV. RESULTS

To perform bug prediction, we are using standard bug prediction data set. Bug prediction data set is a large collection of metrics and models of software systems and their previous versions. The main target of such datasets is to allow people to compare different bug prediction approaches and to evaluate whether a new technique is an improvement over existing ones. The dataset is especially designed to perform and evaluate bug prediction at the class level. The bug prediction dataset comprises of data about the following software systems :- Eclipse PDE UI, Equinox Framework, Lucene, Mylyn, Eclipse JDT Core.

For each system the dataset includes the following pieces of information:

1. Biweekly versions of the systems parsed (with the inFusion tool) into object-oriented models, provided as mse files.
2. Historical information extracted from the cvs change log, including reconstructed transaction and links from transactions to model classes.
3. Value of 15 metrics computed from cvs change log data, for each class of the systems.
4. Values of 17 source code metrics, for each version of each class.
5. Categorized (with severity and priority) post-release defect **c**ounts for each class.

Fig 4: Data Set

This Fig represents the weight outputs of the LDA when LDA is applied on the data set. As it can be observed, the weights are a function of LDA parameters and varies accordingly as the dataset is varied. The weights for certain parameters are quite high as compared to others. These are those parameters which have high importance.

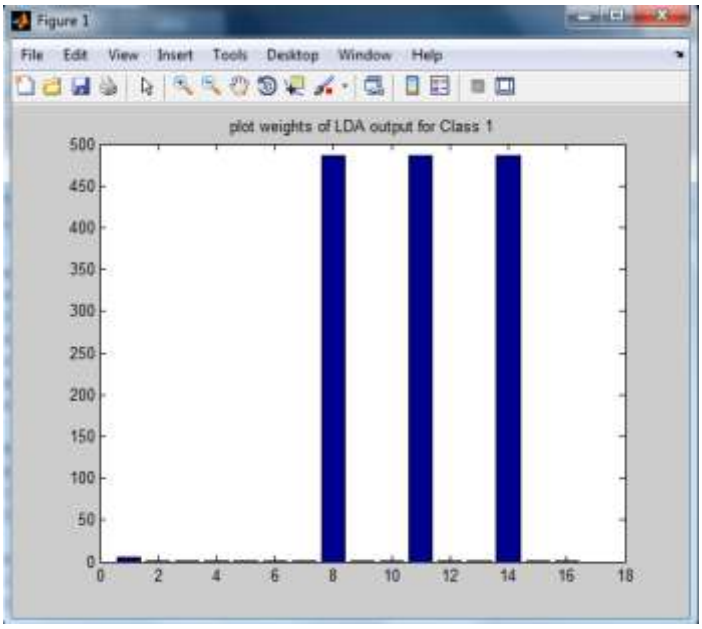

Fig 5 Shows plot weights of LDA output for class 1

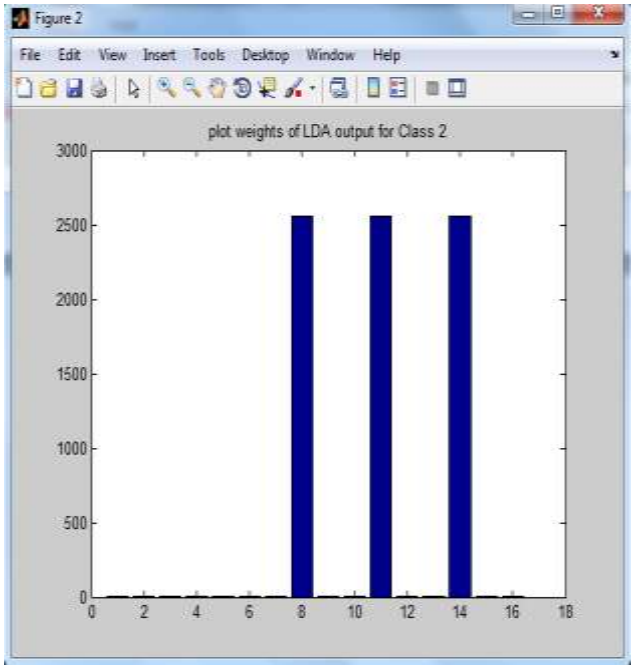

Fig 6 Shows plot weights of LDA output for class 2.

The above Fig represents the weight output for class 2 after it is passed through LDA. As it can be observed, the weights are high for some parameters and these parameters also depends on the class for which they are predicted.

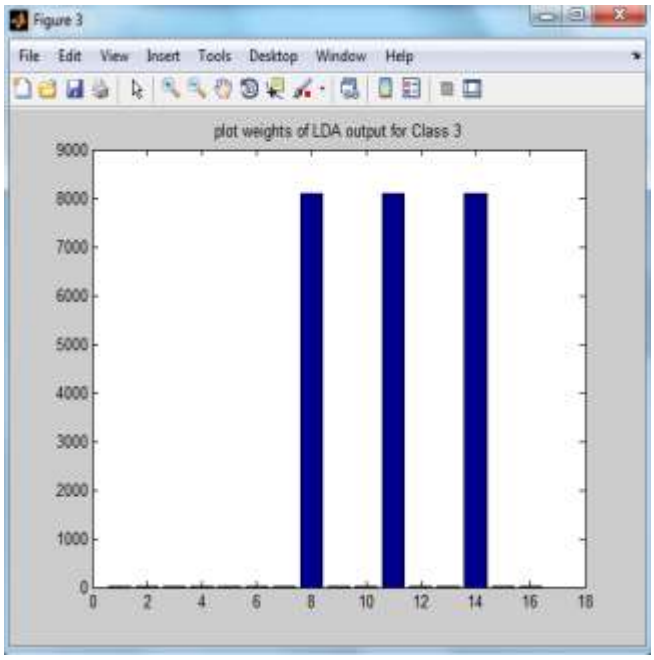

Fig 7 Shows plot weights of LDA output for class 3

The above Fig 7 represents the weight outputs after being passed through LDA for class 3. The weights are found to be quite high for 8th, 10th and 14th attribute.

The Fig 8 represents the weights value of LDA output. The weights are varied and it forms a 4x16 matrix which is utilized for different classes. The values are multiplied with the values for different classes values and the results are obtained.

Fig 9 represents the error in training the ANFIS algorithm for bug prediction and as the algorithm is trained for bug prediction the error is decreasing with time and after certain iterations, it becomes saturated and the error cannot decrease beyond a certain limit.

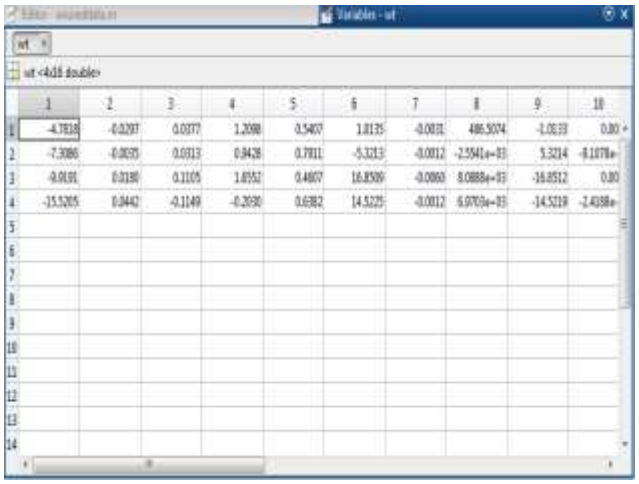

Fig 8 Assigned Weights

The algorithm has also been executed without the aid of LDA. The ANFIS algorithm is applied on the complete data without feature selection and the output of the performance is shown below.

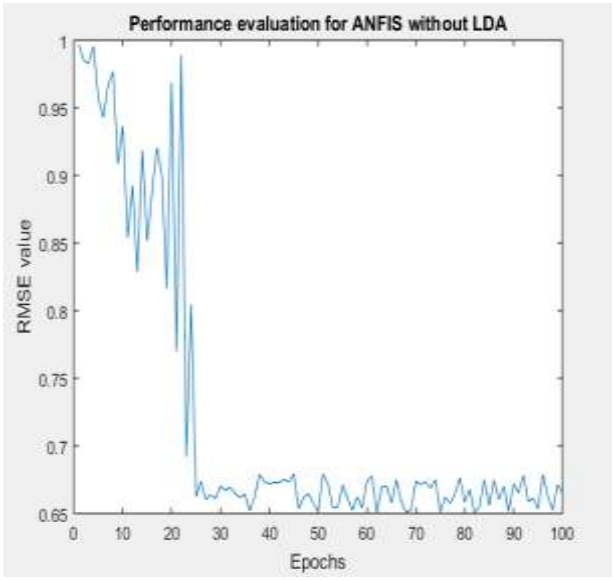

Fig 9 Showing error decreasing with time

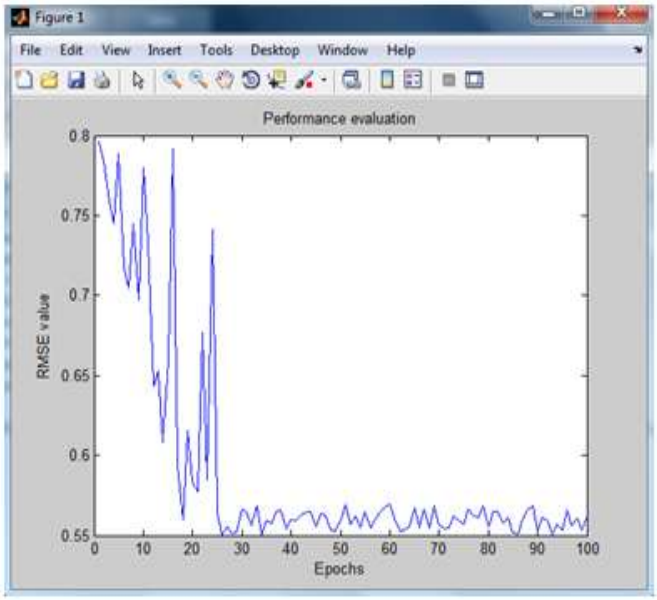

Fig 10 Showing error decreasing with time without LDA

The performances are both compared and are shown in Fig 11 as shown below.

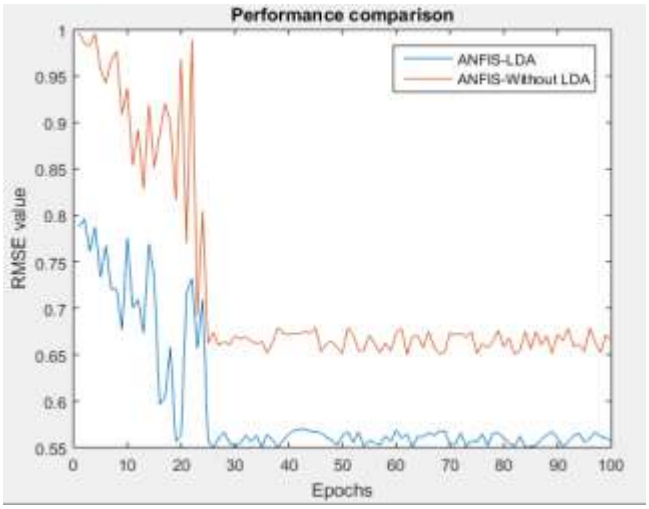

Fig 11 Showing Comparison

V. CONCLUSION AND FUTURE SCOPE

Bug prediction shows the resource allocation problem: Having an accurate estimate of the distribution of bugs across components helps project managers to optimize the available resources by focusing on the problematic system parts. Many Different approaches have been proposed for predicting future defects to be occurred in software systems, which vary in the data sources they use and in the systems they were validated. We have introduced a benchmark to allow for common comparison, which provides all the data needed to apply several prediction techniques proposed in the literature. In this thesis we develop a hybrid model for

bug prediction using machine learning techniques. The information contained in the dataset is reduced in dimension for better computation using Linear discriminate analysis. When the data set is analyzed by LDA then weights are assigned to those features that produce more bugs in the system. Reduced dataset is then trained and tested by the using Adaptive Neuro Fuzzy Inference System. The results are found to be satisfactory and the error in bug prediction decreases as the iteration count increases. Thus, there our proposed algorithm is proved to be well within acceptable limits in terms of convergence and accuracy and also in terms of computation time as dimension of the problem is reduced using Linear Discriminant Analysis.

In future, the correlation of the features can be utilised as parameters of LDA and also other feature reduction techniques can be applied on the dataset for better feature selection. Other algorithms for classification like K-NN, K-means can be applied and the results can be compared. Hybrid algorithms can be developed for classification and New fuzzy models can be utilised for the same.

REFERENCES

[1] A. E. Hassan, "Predicting faults using the complexity of code changes," in Proceedings of ICSE 2009, 2009, pp.78–88, 2009.

[2] Emanuel Giger, Martin Pinzger, Harald C. Gall, "Comparing Fine-Grained Source Code Changes And Code Churn For Bug Prediction", 8th IEEE Working Conference on Mining Software Repositories, ISBN: 978-1-4503-0574-7, May 2011.

[3] Yuan Tian, David Lo, and Chengnian Sun, "Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction", 19th IEEE Working Conference on Reverse Engineering, ISSN: 10951350, 2012.

[4] Shivkumar Shivaji, E. James Whitehead, Ram Akella, Sunghun Kim, "Reducing Features to Improve Code Change-Based Bug Prediction", *IEEE Transaction on Software Engineering*, VOL. 39, NO. 4, APRIL 2013.

[5] Phiradet Bangcharoensap, Akinori Ihara, Yasutaka Kamei, Ken-ichi Matsumoto, "Locating Source Code to be Fixed based on Initial Bug Reports", 4th IEEE International Workshop on Empirical Software Engineering, pp. 978-0-7695-4866-1, 2012.

[6] Marco D'Ambros and Michele Lanza, "Software Bugs and Evolution: A Visual Approach to Uncover Their Relationship", Proceedings of the 10th European Conference on Software Maintenance and Reengineering, 2006, pp. 238, ISSN: 1534-5351, March 2006.

[7] Alberto Bacchelli, Marco D'Ambros and Michele Lanza, "Are Popular Classes More Defect Prone? Springer, Vol 6013, pp 59-73, 2010.

[8] K. E. Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," Journal of Systems and Software, vol. 56, no. 1, pp. 63–75, 2001.

[9] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in Proceedings of ICSM 2005, 2005, pp. 263–272.

[10] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in objectoriented systems," IEEE Trans. Software Eng., vol. 34, no. 2, pp. 287–300, 2008.