International Journal of Modern Computer Science (IJMCS)
Volume 5, Issue 3, June, 2017
ISSN: 2320-7868 (Online)

Evaluate and Propose Fault Detection Technique from Test Cases in Software Testing

Pushpinder Kaur Sidhu
Department of CSE
RBIEBT, Sahuaran
Punjab,India.
E-mail: pushpindersidhu919@gmail.com

Dr. Neeraj Mohan
Department of CSE
RBIEBT, Sahuaran
Punjab,India.
E-mail:erneerajmohan@gmail.com

Abstract: The software engineering is the technology to process the software and perform various operations on that software. Testing is the important application of software engineering in which test cases are applied to detect faults from the software. In the recent times, it has been analyzed that faults may also arise in the test cases which are used for the fault detection. In this work, Rank-to-learn algorithm has been applied for the detection of faults from the software. To improve the performance of Rank-to-learn algorithm in terms of fault detection rate, the technique of back propagation has been applied. The system has been tested on 10 test cases and simulation has been performed in MATLAB. The simulation results show that the fault detection rate has been increased and execution time has been reduced.

Keywords*:* Faults, Test Cases, Neural Networks, Back- Propagation, Rank-to-learn.

I. INTRODUCTION

There can be certain undesired outputs achieved due to certain defects occurring within the software. A prediction model is built with the help of predictors which are collected from either the project itself or the numerous other projects. This model helps further in predicting the defective files present in the software. There is a need to build and design the prediction models for specific projects. For this purpose a universal defect prediction model is proposed which includes the parameters of numerous numbers of projects. The various software metrics are utilized for predicting the number as well as the distribution of software defects in the new software defects prediction. The program properties of the historical software versions are initially set as base for the research methods of software defects classification prediction. These further help in building the various prediction models and then forecasting the defects in the latest versions. There are three categories in which this technique can be divided. They are the software metrics, the classifier and the evaluation of the classifier [10].

There is a defect caused within the system due to the fault in the software. The irregularity seen in the observed performance of the system when compared to its desired performance is known as the error. When the final product is different from the current service and there is a sudden change in behavior as per the user requirements, a software failure occurs. A software failure is not caused by any software fault or error. The identification of any problem occurring in the software without knowing the cause is known as the fault detection. There are various qualitative as well as quantitative approaches through which the fault can be detected. There are also various multivariable, model-based approaches included here. The investigation of minimum one root causes of problems such that some measures can be taken against it is known as fault diagnosis also known as fault isolation. The software product failure is not completely caused due to the fault or a problem. The main reasons behind the non-optimal operations might be the hardware failures. The problems, however, might also arise due to the poor decisions related to operating targets, quality or other parameters [5].

Software Based Fault Detection Techniques [13]

1. Algorithm Based Fault Tolerance (ABFT): ABFT is used for detecting, locating, and correcting faults with a software procedure. The structure of numerical operations is exploited here. There is no generality within this technique even though it is effective. The organizations which use regular structures can use this technique. Therefore, only limited applications can use it.

2. Assertions: The invariant relationships amongst the variables of program are reflected by the assertions or logic statements present at various points within the programs. Assertions are not transparent to the programmer and their effectiveness depends on the way of application as well as on the capacity of the programmer. These problems can be faced during their regular prompting.

3. Control Flow Checking (CFC): The partitioning or dividing the program into essential blocks is the main objective of the CFC. For each block a different signature or number is assigned. The comparisons are made across the run-time signature and the pre-computed signature. This helps in detecting the faults. The test granularity which is utilized here is to be set as per the problems with the help of CFC strategies.

4. Procedure Duplication (PD): The most crucial procedures are duplicated here within this technique. The achieved results are compared with the procedures being executed on two different processors. The decision regarding which procedure

ISSN: 2320-7868 (Online)

to be selected here for duplicating is to be taken by the programmer. Legitimate checking on the results is also to be ensured here. There are manual modifications made here which can also cause errors in the system.

5. Error Detection by Duplicated Instructions (EDDI): Before writing the computation results achieved from the master and shadow instructions into the memory, certain comparisons are to be made. The program is restarted once any mismatch occurs which is caused after the program jumps to an error handler. Due to the introduction of time redundancy within the system, there is high error coverage as a result of performance penalty within the EDDI. There are more register spilling caused in EDDI as there are only general purpose registers being utilized as shadow registers. As a result the performance overhead is caused due to increase in the number of memory operations in case of additional spilling.

6. Software Implemented Error Detection and Correction (EDAC): Software Implemented EDAC approaches are effective in error detection. However, there is high time overhead caused in these systems. Hamming, BCH and RS codes have good mathematical structures. There are limited code lengths within these types of systems.

7. Periodic Memory Scrubbing: The periodic reloading of core on the fundamental memory from an immutable memory is known as the periodic memory scrubbing. The code segment of Operating system and application programs is protected using this technique. Due to the repetitive memory reading, the performance penalty is caused here.

8. Masking Redundancy: When faults are present within a system, the marking redundancy helps in running an application. There is a need to numerous processors for running the similar program and recognizing the error within any particular processor. From the application software, the errors can be masked. For fixing the errors there is no need for the software rollbacks.

9. Reconfiguration: The removal of failed modules within the system is done through the reconfiguration method. The various segments of the module that are separated are affected due to failure caused in any one part of the module. There are various functional modules utilized which can replace the failing module.

10. Replication: The reliability of the system can be ensured through the replication technique. However, this technique is expensive in terms of hardware and runtime cost. The calculation which is replicated N times is to be taken as a majority vote. N copies of the surrounding computations are to be made for providing the software solution for each processor. Further the result is to voted. The computation is minimized to a factor of N through this system.

11. Restore Architecture: Within the Restore architecture, the transient errors also known as soft error are recognized with the time redundancy. The utilization of transient error symptoms is done through the novelty of Restore architecture.

12. Fingerprinting: Over a dual modular redundant (DMR) processor pair, the recognition of differences in the execution is done through the fingerprinting mechanism. With the help of a hash-based signature the execution history of a processor is determined. By making comparisons with the fingerprints the differences between the two mirrored processors are recognized.

II. LITERATURE SURVEY

Gao K. et.al [2007] proposed that how count models based upon poisson regression model and negative binomial regression model can be used for software defect predictions. It evaluates the comparative hypothesis testing, model selection and performance evaluation for the count models [3]. Zimmermann T. et al. [2007] mapped defects from the bug database of Eclipse to source code locations. The resulting data set lists the number of pre and post release defects for every package and file in the Eclipse releases 2.0, 2.1, and 3.0 [4]. Lessmann S. et al. [2008] improved software quality and testing efficiency by constructing predictive classification models from code attributes to enable a timely identification of fault-prone modules. Several classification models have been evaluated for this task [5].

Moser R. et al. [2008] identified a comparative analysis of the predictive power of two different sets of metrics for defect prediction. It choose one set of product related and one set of process related software metrics and use them for classifying Java files of the Eclipse project as defective respective defect-free [6]. D'Ambros M. et al. [2011] described the performance of the approaches using different performance indicators: classification of entities as defect-prone or not, ranking of the entities, with and without taking into account the effort to review an entity [11].

Rawat S. et al. [2012] introduced causative factors which in turn suggest the remedies to improve software quality and productivity. The paper also showcases on how the various defect prediction models are implemented resulting in reduced magnitude of defects [13]. Yang X. et al. [2012] presented the ranking approach for allocating testing resources to software modules. In this paper predicting models can be used for predicting the defects. This paper only concerned with the construction of models, which include the ranking performance measure in the objective function, perform better in predicting defect-proneness rankings of multiple modules [16]. Yang X. et al. [2015] proposed a linear LTR approach.

In this paper, comparisons have been made amongst the LTR approach and other count models. There are better results shown by the LTR in comparison to those models. The performance of the software is enhanced with the help of this technique. Through direct optimization of the ranking performance, the LTR technique constructs the software defect prediction models. The comparison of this method with other techniques which are utilized for prediction of the order of

ISSN: 2320-7868 (Online)

software modules as per the predicted number of defects is shown here [16].

III. LEARN TO RANK LEARNING

The method in which the machine learning techniques are involved for training the model in a ranking task is known as the learning to rank method. The performance of the model is measured with the help of the LTR approach. The ranking performance of the system is optimized directly with the help of the LTR linear model. In comparison to the other models, the LTR model is utilized and can also be compared with the other non-linear models. There can be a utilization of trained data within the LTR technique and it is also applicable on various data sets. There are various applications in which LTR is involved such as the information retrieval, in NLP and also in data mining. A linear model is obtained here by optimizing the ranking performance directly within the LTR method. There are various models along with which the LTR technique can work. The various count models, data sets for ranking of software defects and many other models are used here. There are comparisons and evaluations being made amongst other techniques and the LTR technique for the purpose of constructing SDP models which will further rank the tasks [16].

There are various algorithms which can be suitable for the provided framework of LTR. A proper categorization is performed on these algorithms focusing on the main objective of providing enhancements in them [16].

a. The Point-wise Approach: The feature vector of every single document is present in the input space of the point-wise approach. The relevance degree of each single document is present in the output space. The functions which involve feature vector of a particular document as input and predict the relevance degree of the document are involved within the hypothesis space. The accurate prediction of the extreme truth label for each document is examined with the help of loss functions. Ranking is displayed as regression, classification and ordinal regression for various point-wise ranking algorithms.

b. The Pair-wise Approach: There is a pair of documents which are represented as feature vectors within the input space of the pair-wise approach. The output space contains the pair-wise preference (which takes values from $\{1,-1\}$) between every pair of documents. There are bi-variate functions h present in the hypothesis space which take the pair of documents are input and the relative request amongst them is given as output.

c. The List-wise Approach: The complete groups of documents that are connected with the query q are considered in the input space of the list-wise approach. Two different ways of output spaces are used as a part of the list-wise approach. There is relevance degree of few numbers of documents connected with a query within the output space in case of come list-wise ranking algorithms.

IV. BOLTZMANN LEARNING

The systems of symmetrically connected units which settle on stochastic decisions regarding being on or off are known as the Boltzmann machines. The complex distributions amongst the observed data are discovered through the simple learning algorithm. For various scientific tasks, the learning or inference in Boltzmann machines is very important. There is a proper settlement of the weights on connections and thresholds related to the inference problems. These are further used for representing the cost function. For being utilized as a tool in case of some advanced problems, the inference in Boltzmann machine is utilized more often. This also includes some combinatorial issues which are in accordance with the NP finish or the hard issue classes [5,13].

There is a need of expectation of one unit and correlation amongst two units for the learning in Boltzmann machines. It is also important to provide the precise estimation of the correlations. The accuracy of the correlations is improved by the Linear Response Approximation (LRA) with the help of mean field mechanism. The empirical moments are matched with the help of approximation methods within the learning systems in accordance with those which are achieved through inappropriate inference techniques. There is a concept of pseudo-moment matching involved in the inexact learning algorithms with or without the LRA. There are various studies proposed for the pseudo-moment matching issues which also include the learning algorithm of Boltzmann machines [5,13].

The BP algorithm is combined with the LRA for calculating the accuracy of probabilistic inference systems. It is seen that the LRA enhances the estimation of correlations by involving the impacts of loops within a specific system to the BP algorithm.

The global energy, E, in a Boltzmann machine is identical

$$E = -\left(\sum_{i,j} w_{i,j} s_i s_j + \sum_i \theta_i s_i\right)$$

Where, w_{ij} is the connection strength between unit j and unit i.
s_i is the state, s_i $\{0,1\}$, of unit i.
Θ_i is the bias of unit i in the global energy function.
Often the weights are represented in matrix form with a symmetric matrix W, with zeros along the diagonal [5,13].

V. PROPOSED METHODOLOGY

The fault prediction is the technique which is applied to predict the percentage of faults in the test cases. This work is

ISSN: 2320-7868 (Online)

based on to detect faults from the test cases using learn-to-rank algorithm. The learn-to-rank algorithm is based on three steps. The first step is selection of population. The second step is calculation of mutation value. The last step is calculation of fitness value. The calculation of fitness value depends upon the initial population value which is selected randomly. In this work, Back Propagation technique is applied in which system learns from the experience values and derives new values. The selection of population value is not random. It depends upon the system condition which is derived using back propagation algorithm.

5.1 Proposed Algorithm

```
    Init population P (t)

    evaluate P (t);

    t := 0;

Network  ConstructNetworkLayers()

 InitializeWeights(Network, test cases)

For ( i=0;i=testcases;i++)

    SelectInputPattern(Input fault values)

    ForwardPropagate(p)

    BackwardPropagateError(P)

    UpdateWeights(P )

 End

Return (P)
     while not done do
           t := t + 1;
         P' := test case P (t);
                 Recombine P' (t);

mutate P' (t);

         evaluate P' (t);

         P := survive P,P' (t);

     end
```

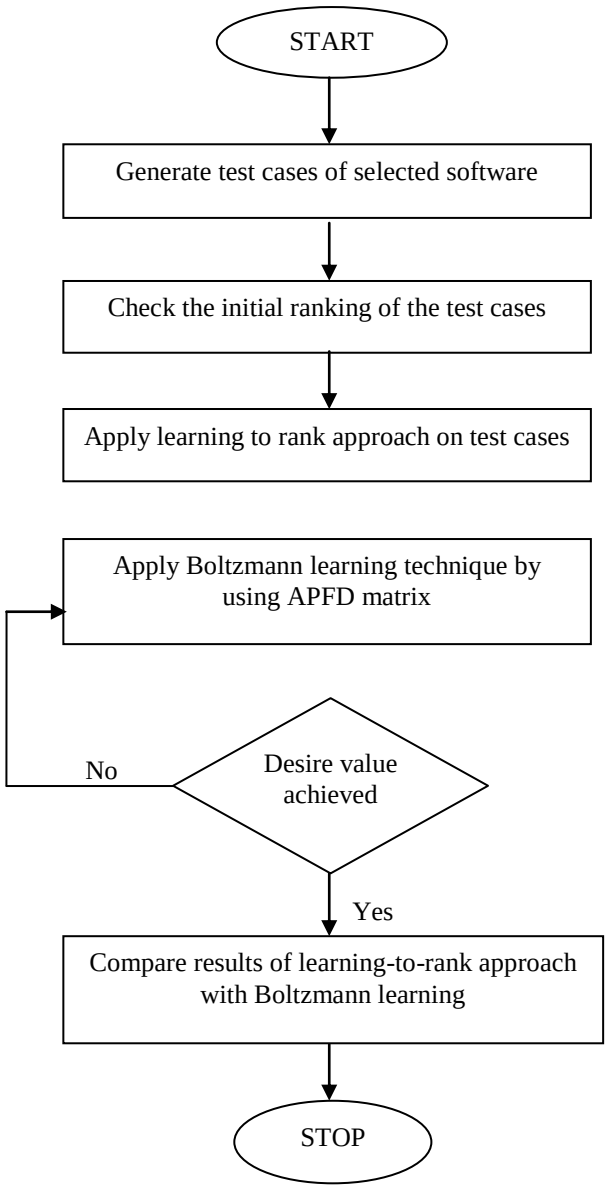

Figure 1: Proposed Flowchart

VI. SIMULATION RESULTS

The Rank-to-rank and improved Rank-to-learn algorithms are implemented in MATLAB. The dataset is considered for the implementation which is described in table 1.

ISSN: 2320-7868 (Online)

Table 1: Properties of dataset

Attributes	Values
Number test cases	10
Repeated Test cases	No
Fault in the Test cases	Yes
Number of applications	1

The proposed algorithm is implemented and interface is designed for the implementation which is described in the figures shown below:

Figure 2: Interface is designed for implementation

As shown in figure 1, the interface is designed for the implementation of rank-to-learn and improved rank-learn algorithm. In the interface ten test cases are shown in which is executes existing and proposed algorithm. The result is analyzed in terms of fault detection rate.

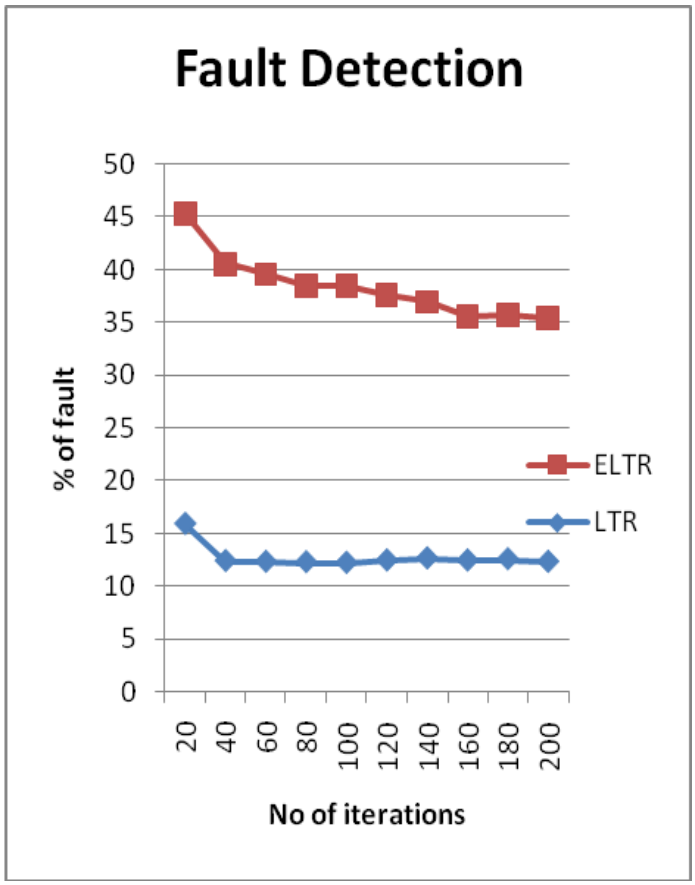

Figure 3: Comparison Graph

As illustrated in fig. 2, the comparison graph is drawn between proposed and exiting algorithm. The existing algorithm is Rank-to-learn algorithm and proposed algorithm is improved Rank-to-learn algorithm. When the back propagation algorithm is implemented with Rank-to-learn algorithm the fault detection rate is improved as shown the graph.

VII. CONCLUSION

The fault detection is the testing technique which is applied to detect faults from the software or from the input test cases. The Rank-to-learn is the algorithm which is applied for the faults in the software. This algorithm selects population randomly which reduce fault detection rate. In this work, technique of back propagation is applied in which system learns from the previous experiences and drive new values. This leads to improve fault detection rate and reduce execution time . In future technique will be proposed which is based on bio-inspired techniques for the fault detection rate

REFERENCES

[1] Graves T. L. , Karr A. F. , Marron J. S. , and Siy H. , "Predicting fault incidence using software change history," in Proc. IEEE Trans. Softw. Eng., Vol.26, no. 7, pp. 653–661, 2000.

 ISSN: 2320-7868 (Online)

[2] Ostrand T. J., Weyuker E. J., and Bell R. M., "Predicting the location and number of faults in large software systems," IEEE Trans. Softw. Eng., Vol. 31, no. 4, pp. 340–355, 2005.

[3] Gao K. and Khoshgoftaar T.M. , "A comprehensive empirical study of count models for software defect prediction," in Proc. IEEE 28th Int. Conf. Trans. Rel., Vol. 56, no. 2, pp. 223–236, June. 2007.

[4] Zimmermann T. , Premraj R. , and Zeller A. , "Predicting defects for eclipse," in Proc. IEEE Int. Workshop Predictor Models in Software Engineering(PROMISE'07), pp. 9–15, 2007.

[5] Jiang Y. , Cukic B. , and Ma Y. , "Techniques for evaluating fault prediction models," in Proc. Empiric. Softw. Eng., Vol. 13, no. 5, pp. 561–595, 2008.

[6] Lessmann S. , Baesens B. , Mues C. , and Pietsch S. , "Benchmarking classification models for software defect prediction: A proposed frame work and novel findings," in Proc. IEEE Trans. Software Engineering., Vol. 34, no. 4, pp. 485–496, 2008.

[7] Moser R. , Pedrycz W. , and Succi G. , "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in Proc. ACM/IEEE 30th Int. Conf. Software Engineering, pp. 181–190 , Dec.2008.

[8] Mende T. , and Koschke R. , "Revisiting the evaluation of defect prediction models," in Proc. 5th Int. Conf. Predictor Models in Software Engineering, 2009, pp. 1–10.

[9] Arisholm E. , Briand L. C. , and Johannessen E. B., "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," in Proc. J. Syst. Softw., Vol. 83, no. 1, pp. 2–17, 2010.

[10] Weyuker E.G. , Ostrand T. J. and Bell R. M. , "Comparing the effectiveness of several modeling methods for fault prediction," in Proc. IEEE Int. J. Empiric. Softw. Eng., Vol. 15, no. 3, pp. 277–295, 2010.

[11] D'Ambros M. , Lanza M. , and. Robbes R. , "Evaluating defect prediction approaches:A benchmark and an extensive comparison," in Proc. IEEE Conf. Softw. Eng., pp. 1–47, 2011.

[12] Wang H. , Khoshgoftaar T. M. , and Seliya N. , "How many software metrics should be selected for defect prediction," in Proc. 24th Int. Florida Artificial Intelligence Research Society Conf., pp. 69–74, 2011.

[13] Khoshgoftaar T. M. , Gao K. , and Napolitano A. , "An empirical study of feature ranking techniques for software quality prediction," in Proc. IEEE Int. J. Softw. Eng. Knowl. Eng., Vol. 22, no. 2, pp. 161–183, 2012.

[14] Wang Y. , Cai Z. , and Zhang Q. , "Differential evolution with composite trial vector generation strategies and control parameters," in Proc. IEEE Trans. Evol. Computat., Vol. 15, no. 1, pp. 55–66, 2011.

[15] Rawat S. M, Dubey K .S, "Software Defect Prediction Models for Quality Improvement: A Literature Study", in Proc. International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2, September 2012 ISSN (Online): 1694-0814.

[16] Yang X. , Tang K. , and Yao X. , "A effective algorithm for constructing defect prediction models," Int. J. in Intelligent Data Engineering and Automated Learning-IDEAL, pp. 167–175, 2012.

BIOGRAPHY

Pushpinder kaur Sidhu is a Research Scholar in the Computer Science and Engineering Department, University School of Engineering and Technology, Rayat and Bahra University, Mohali. She has received Bachelor of Technology degree in 2014 from SBSSTC, Ferozpur, Punjab, India. She has varied interests in the fields of Computing, Management