International Journal of Engineering & Technology, 7 (1) (2018) 53-56

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i1.7870
Research paper

Fault detection technique for test cases in software engineering

Jagatjot Singh [1]*, Sumit Sharma [2]

[1] *P.G. Student, Department of Computer Science and Engineering, Chandigarh University, Chandigarh, India*
[2] *Assistant Professor, Department of Computer Science and Engineering, Chandigarh University, Chandigarh, India*
**Corresponding author E-mail: jagatjotsingh22@gmail.com*

Abstract

The processing of software and performing various operations on it is known as a software engineering process. The application of test cases for detecting the faults within the software is done through the testing process. There are various types of faults that occur within a software or test case which are to be identified and preventive approaches are to be applied to prevent them. In this paper, the Learn-to-rank algorithm is utilized which helps in detecting the faults from the software. The Back-Propagation technique is included with the LRA approach for enhancing its performance and improving the detection of fault rate. 10 test cases of different types are used for running various experiments and the MATLAB tool is utilized for performing various simulations. It is seen through the various simulation results that the fault detection rate is increased as well as the execution time is minimized with the help of this approach.

Keywords: *Faults; Test Cases; Neural Networks; Back Propagation; Learn-to-Rank.*

1. Introduction

Once any defect occurs within the software it can result in providing some completely different outcomes. There is a need of designing a prediction model for predicting the defective files using the predictors. These predictors are gathered from one project or variety of other projects. For the purpose of building prediction models for each project, a universal defect prediction model is designed from variety of projects. The software metrics is used for predicting the amount of defect present within the software as well as its distribution within it. On the basis of the program properties of the previous software versions, the software defect classification prediction is designed [10]. The defects that can occur can also be predicted with the help of such studies. The software defect prediction model is partitioned into three different parts. They are the software metrics, the classifier and the evaluation of the classifier.

Whenever a software fault is identified within the system it results in causing a defect within it. Any difference between the achieved performance of the system and the assumed performance is known as the error occurring within it. When there is a change in the outcome achieved or it behaves differently as compared to the requirements proposed by the user, the software failure occurs. The identification of a problem within the system without knowing its cause is known as the fault detection process. There are various quantitative as well as qualitative techniques that will help in detecting the faults within the system. There are varieties of multi-variable model-based techniques within this method. The identification of at least one of the main causes which results in causing defect within the system is known as fault diagnosis. This is done so that various preventive measures can be taken to prevent it. It is not necessary that there is a complete failure of the system due to the occurrence of a fault or any other problem. There are various root causes for non-optimal operation of the various failures occurring at the hardware. There can be variety of reasons that can result in causing failure within the system such as the operating targets or some error caused by human.

There are various software-based Fault Detection Techniques, some of which are listed below:

a) Control Flow Checking (CFC): An application program is divided into important blocks or the parts of code that do not have branch parts, in the CFC method. For each block a deterministic signature is provided. Comparisons are made with the run-time signature and the pre-computed signature for identifying the faults within the system. The matching of test granularity which is to be utilized within the system is very difficult to be done within the CFC methods.
b) Fingerprinting: The fingerprinting method differentiates the execution over a dual modular redundant (DMR) processor pair. A processor's execution history is seen with the help of a hash-based signature. Through the comparisons made across the fingerprints achieved, the differences amongst two mirrored processors are depicted.
c) Reconfiguration: The reconfiguration method attempts to eliminate the failed modules from the system. Whenever, there is any failure identified within a module, the segments that are isolated from the rest of system are also affected. For the purpose of replacing a failed module, various functional modules are utilized which are also switched automatically.
d) Algorithm Based Fault Tolerance (ABFT): With the help of a specific software procedure, the detection, location and correction of faults are done within the ABFT technique. The structure of numerical operations can be exploited through this technique and is not much generally available even though it is effective enough. The applications which involve regular types of structures can use this method.
e) Procedure Duplication (PD): Most of the important procedures here are duplicated by the programmer within this system. The achieved results are compared when these procedures are executed on two different processors. The procedures that are to be duplicated are chosen by the programmer.

The results achieved here are provided a legitimate checking as well. There is a manual modification done for the codes, and various errors might be identified within the system.

f) Error Detection by Duplicated Instructions (EDDI): Before the computed outcomes achieved from master and shadow instructions are written within the memory, they undergo various comparisons. The program restarts in case where a mismatch occurs upon which the program also jumps to an error handler.
g) Replication: This technique has higher cost with respect to the hardware and the runtime. However, the reliability within the system is ensured here. The main objective here is to achieve a majority vote on the calculation that is repeated numerous numbers of times. Each processor is made to run N copies related to the surrounding computations through the software solution.
h) Restore Architecture: Within the Restore architecture, the transient errors as well as soft errors are recognized with the help of time redundancy. The method uses the transient error symptoms.
i) Periodic Memory Scrubbing: On the basis of periodic reloading of the code on an immutable memory, the periodic memory scrubbing method is proposed. Due to the repetitions occurring for memory reading, the performance penalty occurs.
j) Assertions: There are various assertions or logic statements present at different points within the program. They help in depicting various relationships among the variables that are mentioned within the program. There are various issues that are also prompted within this method as the assertions are not easily understood by the programmer.

2. Literature survey

Xiaoxing Yang et.al (2015) proposed in this paper [1], the description of construction of previous work and new study that is beneficial for the construction of software defect prediction model. There are two aspects to be considered in this paper. First is the new application of learning-to-rank technique to real-world data sets which will help in predicting the software defect. The second is the comprehensive evaluation of the procedure. It is seen through the empirical studies that the performance measures of the learning-to-rank approach are very effective as compared to the already existing approaches.

Shaik Nafeez Umar et.al (2013) proposed in this paper [2] the method through which the statistical model predicts the defects for the newly designed software projects. Here, the earlier released 20 data points, and 5 parameters are utilized for designing a model. The descriptive statistics, correlation as well as multiple linear regression models are also applied with various confidence intervals (CIs). The R-square value within this multiple regression model is 0.91 as well as the standard error is 5.90%. Through the simulation results achieved it is seen that there is a precision of 90.76% between the actual defects and the predicted defects.

Muhammad Dhiauddin et.al (2012) in this paper [3] they described that an initial effort of building a prediction model for defects in system testing carried out by an independent testing team. The motivation to have such defect prediction model is to serve as early quality indicator of the software entering system testing and assist the testing team to manage and control test execution activities. Mathematical equation that has p-value of less than 0.05 with Rsquared and R-squared (adjusted) more than 90% is selected as the desired prediction model for system testing defects. This model is verified using new projects to confirm that it is fit for actual implementation.

Mrinal Singh Rawat, Sanjay Kumar Dubey (2012) in this paper [4] they proposed that software defects may lead to degradation of the quality which might be the underlying cause of failure. In today's cutting edge competition it is necessary to make conscious efforts to control and minimize defects in software engineering. However, these efforts cost money, time and resources. This paper identifies 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.

Christopher Henard, (2013) in this paper [5], they explained that mass customization and economics force to design software product line. Due to large size of products present within the software, product line is a challenging. In this paper, existing technique based on the feature model of the product line by selecting limited set of products. In this paper test suites are used to detect such errors. In particular, two mutation operators are proposed to derive erroneous feature models (mutants) from an original feature model and assess the capability of the generated original test suite to kill the mutants. Experimental results demonstrate that dissimilar tests suites have higher mutant detection ability than similar ones, thus validating the relevance of similarity-driven product line testing.

Jan Peleska, (2013) in this paper [6], they explained that model based testing is one of the leading technologies. The key factors are essential for industrial scale application of MBT. Both are identified from the feature extraction. With former view they had described techniques for automated test cases, test data and test procedure generation for concurrent reactive real times system which enables for MBT. Their experience introduced MBT approaches in MBT for testing teams. There are many scientific problems to improve the acceptance and effectiveness of MBT.

3. Learn to rank algorithm

The various machine learning techniques that are designed for training the models present within a ranking task are known as learn-to-rank techniques. The performance of the model can also be calculated through this approach. For the purpose of optimizing the ranking performance directly the LTR linear model is utilized. As compared to other existing approaches the LTR method is widely used which is also comparable with other already existing non-linear models. The utilization of trained data is mainly done here and this method can be used on variety of data sets as well. For various applications related to information retrieval, natural language processing and data mining LTR is utilized. The ranking performance can be optimized directly which further results in providing a linear model within the LTR method. There are varieties of models present and the LTR method can work with almost all such variety of models. For the purpose of evaluating ranking of software defects, various types of date sets can be utilized within this method. There is a need to provide comparisons amongst various types of techniques for providing a proper evaluation on the outcomes achieved. The LTR method is very efficient in terms of various parameters [17]. There are three different categories present within this approach. They are explained below:

a) The pointwise approach: The feature vector of each of the individual document is present within the input space of the pointwise approach. Within the output space the relevance degree of each of the document is present. The functions which take the feature vector of a document are taken as input which further predicts the relevance degree of that particular document.
b) The pairwise approach: There is a pair of documents present within the input space of the pairwise approach. The pair is depicted as feature vectors.
c) The listwise approach: There is a complete group of documents which are connected with query q with the input space of the listwise approach. The output space is of two types as per the relevance degrees present within the documents.

4. Boltzmann learning

The connection of symmetrically connected units which depend on stochastic decisions to be on or off are known as Boltzmann machines. The complex distributions related to the observed data are

identified with the help of the simple learning algorithms provided by the Boltzmann machines. There are various scientific tasks that are used here for learning. There is a settlement of weights on connections and thresholds for solving the inference issues. A cost function is also represented here through these methods. There are various advancement issues that are solved by using the Boltzmann machines as a tool within those inferences. The accuracy of the correlations that are estimated with the help of mean field strategy is enhanced with the utilization of linear response approximation (LRA). The empirical moments are matched here with the help of various approximation methods within the learning systems. The inexact learning method is similar to the pseudo-moment matching whether it involves the LRA method or not. There are various studies being proposed related to the pseudo-moment matching issues within the Boltzmann machines. The BPA technique when combined with the LRA method is tested in terms of accuracy. It is seen that the LRA technique enhances the estimation of correlations which is mainly done due to the impacts of loops on a specific system which is absent in case of 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, $s_i \in \{0,1\}$, 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. Proposed methodology

The fault prediction is the technique which is applied to predict the percentage of faults in the test cases. This work is 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 Construct Network Layers ()
Initialize Weights Network, test cases)
For (i=0; i= test cases; i++)
 Select Input Pattern (Input fault values)
 Forward Propagate (p)
 Backward Propagate Error (P)
 Update Weights (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
```

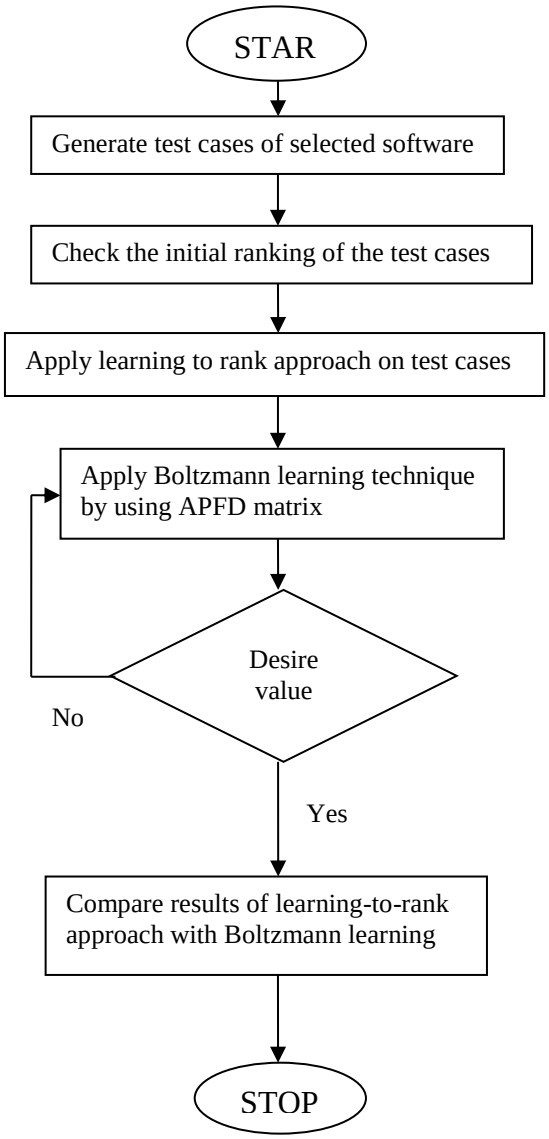

Fig. 1: Proposed Flowchart.

6. Simulation results

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

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

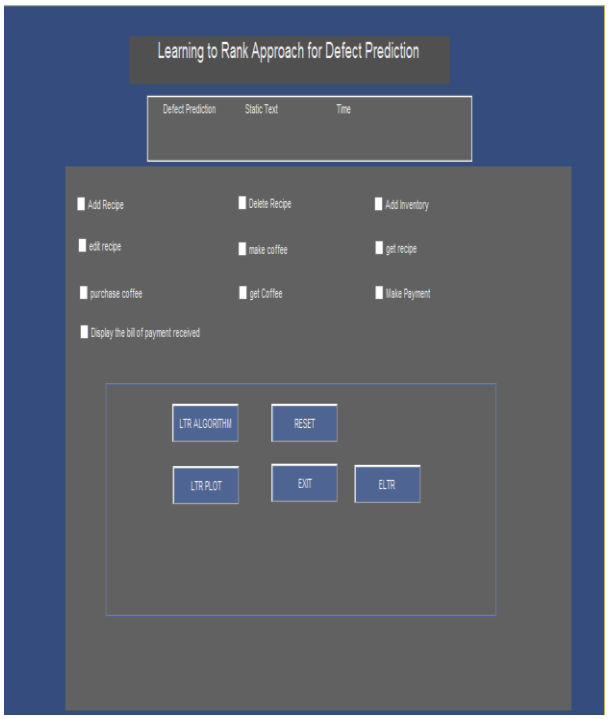

Fig. 1: Interface Is Designed for Implementation.

As shown in figure 1, the interface is designed for the implementation of Learn-to-rank and improved rank-learn algorithm. Ten test cases are shown within the interface. Here the existing and proposed algorithms are executed. The result is analyzed in terms of the parameter of fault detection rate.

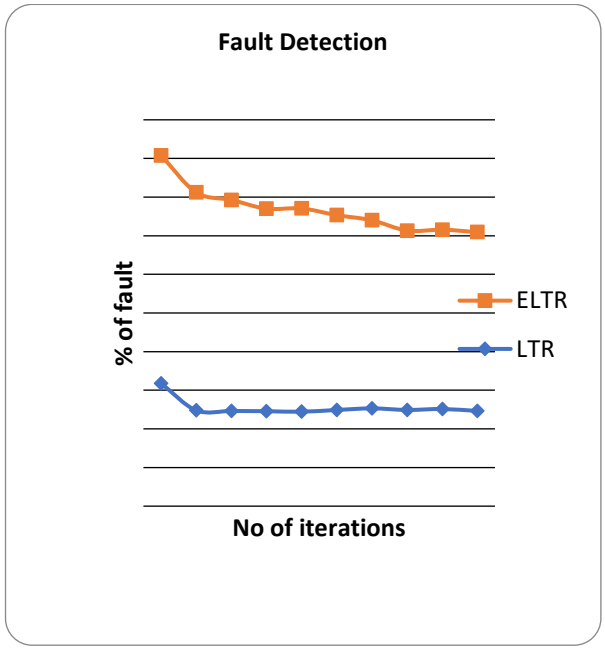

Fig. 2: Comparison Graph.

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

7. Conclusion

For detecting the faults from the software or input test cases the fault detection technique is utilized. For this purpose, the Learn-to-rank method is utilized which helps in identifying the various faults present within the particular software. The fault detection rate is minimized here by selection the population on random basis through this algorithm. The method known as Back Propagation method is utilized here that involves a detailed evaluation of the already existing techniques and helps in deriving new values. The method helps in enhancing the fault detection rate and minimizing the execution time. A bio-inspired technique will be proposed in the future work for detection the fault 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. https://doi.org/10.1109/32.859533.

[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. https://doi.org/10.1109/TSE.2005.49.

[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. https://doi.org/10.1109/PROMISE.2007.10.

[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. https://doi.org/10.1007/s10664-008-9079-3.

[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. https://doi.org/10.1109/TSE.2008.35.

[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. https://doi.org/10.1145/1368088.1368114.

[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. https://doi.org/10.1145/1540438.1540448.

[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. https://doi.org/10.1016/j.jss.2009.06.055.

[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. https://doi.org/10.1007/s10664-009-9111-2.

[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. https://doi.org/10.1142/S0218194012400013.

[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. https://doi.org/10.1109/TEVC.2010.2087271.

[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.