International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
IJISAE
ISSN:2147-6799 www.ijisae.org Original Research Paper

Automating Software Testing with Multi-Layer Perceptron (MLP): Leveraging Historical Data for Efficient Test Case Generation and Execution

D. Manikkannan[1], Dr. S. Babu [2]

Submitted:20/03/2023 **Revised:**24/05/2023 **Accepted:**10/06/2023

Abstract— Software testing is an essential step in the software development process. Defects in software are mostly caused by newer technology, a lack of version control, and the complexity of systems. Because of these issues, the cost of software maintenance rises, as do its consequences. Manual testing necessitates the use of human labour to seek for and analyse data. As software systems get more complicated, automated software testing approaches are becoming increasingly important. Machine Learning approaches have proven extremely beneficial in automating this procedure. Machine learning is also utilised to find essential software testing variables that aid in forecasting software testing cost and time. Predicting testing effort, tracking process expenses, and measuring results all contribute to improve software testing efficiency. Previously, classification trees were used to identify key properties of software testing, and regression approaches were employed to categorise defective data sets. Our framework is useful for automating the software testing process.

Keywords: *Software testing, automation, efficient testing, test case generation, ML in software testing*

Introduction

A. Regression Testing

Verifying if a change in the code of the software has any impact on the functionality that exists is called as regression testing. This type of testing is useful in ensuring the continuous working of the software even after a bug fix or change. The result of the change are verified by repeatedly executing the already existing test cases. So, this type of testing is considered to be similar to a verification methodology. As we need to repeatedly execute the existing test cases, manual methodology is proved to be taking more time. Thus, the test cases for a regression testing is usually automated.

High power software that need a large amount of resources and also capital are tested using regression testing. Two different versions of the software are taken as input for regression test so as to check if the new modification of the system doesn't disturb the existing features. The correct information regarding the changes made are usually available except in a few cases.

Regression testing can sometimes be a complex process due to the latest technology change in the software development process. Let us consider the following example: A software development process that is dependent on its components often leads to usage of a lot of components which come under the category of black box components. So, any modifications in those components might affect the entire system, although performing regression test is difficult as the components' internals are available to the users.

To overcome the difficulties mentioned above, regression testing uses a wide range of techniques. Some of them are:

- Regression Test Selection
- Test Case Prioritization
- Hybrid
- Retest all

B. Test case Prioritization

Test case prioritization is an important aspect in the software testing process which helps to prioritize the test cases based on different factors that is dependent on the software need. The priorities are then allotted to the test cases which is then useful in further proceedings. The main use of this test case prioritization is that the developers/testers can decrease the cost and time that is spent in the testing phase of the software development process. Also, the quality is ensured due to this process. The goal is categorizing the high priority test cases and executing them in order to enhance the faults detected in the code.

[1]*AP, Department of Computer Science and Engineering, SRMIST, Vadpalani Campus, Chennai, manikkad@srmist.edu.in*
[2]*Assoc. Prof, Department of Computing Technologies, SRMIST, Kattankualatur, babus@srmist.edu.in*

C. Coverage based Test case Prioritization

The coverage based TCP is solely dependent on the coverage of the code. The types of coverage based test case prioritization are as follows:

- Total Branch Coverage Prioritization – The factor that is used for prioritizing the test cases is total branch coverage i.e., every condition's possible outcome's coverage.
- Total Statement Coverage Prioritization – The factor that is used for prioritizing the test cases is total statement coverage i.e., coverage of total count of statements.
- Total Fault Exposing Potential Prioritization – The factor that is used for prioritizing the test cases is total fault exposing potential i.e., test case's capability to expose its fault. This factor is not considered in statement and branch coverage methodologies.
- Additional Branch Coverage Prioritization – Test cases that have the maximum branch coverage is selected first and then the test cases are selected repeatedly to cover those which were not covered earlier.
- Additional Statement Coverage Prioritization – Test cases that have maximum statement coverage are selected in an iterative manner and then the test cases are selected repeatedly to cover those which were not covered earlier.

D. Defect Prediction

From frequent observations we can say that throughout the software development process few areas in the code are highly defect prone. Various features like difficulties faced during the implementation, agitated code and improper designs are reasons for frequent fault proneness. Important information about software failures are collectively stored in error trackers and defect databases. These data are used by fault prediction techniques to analyze and recognize defect prone areas in the code. Fault prediction uses ML to analyze the history of defects in the software and create an efficient fault predicting model. These fault prediction technique usually follows the steps below.

- Extraction of Features: Development process, Code metrics and other information are extracted from each segment of the code and used as feature vectors. Also defect history of each segments are extracted and saved separately.
- Creation of Prediction Model: Data extracted from the previous step is given as input to ML algorithm to create a defect prediction model. To create the defect prediction model code metrics is used as feature vector and defect history of each segment is the target function.
- Validation using Prediction Model: This defect predicting model that is created can be used for allotting each segment of code a fault proneness score. Some set of data are usually not involved in the learning stage of the model. Validation set is used to validate the predicting strength of the created defect prediction model.

E. Multilayer Perceptron

Multilayer perceptrons [1] are used to map two vectors (one being the input and its corresponding output) in a non-linear fashion. Many areas of study involve this kind of mapping in a non-dynamic environment. Daily-life problems can be structured using this kind of a static model. One such model is character recognition. However most of the practical problems like speech, vision and time series models are dynamic in nature as the outcome of the current situation relies completely on the input and output of the previous situation. Many attempts have been made to apply this MLP structure with this kind of problems.

F. Traditional methods

In our work, we are working with a modification of existing system. We analyze these traditional methods to contrast them with their modified methods.

- Random strategy: In this strategy the test cases are sorted randomly. The average percentage of faults detected of this method is 50 percentage and is used as reference for comparing with other methods for evaluation.
- Total strategy: In this strategy, the sorting of test cases is done according to their total coverage so that the first test case will have the maximum total coverage. This strategy is simple and efficient when compared to other existing methods.
- Additional strategy: This method starts by calculating the overall coverage of each and every test cases and then using greedy approach for prioritizing required test cases in a sequence so that the test case with the maximum coverage over the exposed code is selected and added to the ordered list of test cases then marked so that it is not selected for further steps. The part of the code covered by the selected test case is marked as covered part.

Literature Survey

M. Khatibsyarbini et al. [2] proposed Test suite minimiza- tion in Regression Testing. The system examines and classifies the Test cases based on articulated research papers. This approach was helpful in choosing the best available resources. The metric used for comparison of algorithms is Average Percentage of Faults Detected. The problem in this method was that it included manual decision making and the estimation was needed to be automated to reduce error.

Kalyani, Naveen et al. [3] proposed Requirements Cluster- ing and Mapping for Test case selection. The system takes modules of ATM application as input. This system helped in improving the productivity to 80% by using requirements information in test cases prioritization. The metric used for comparison of algorithms is Average Percentage of Faults Detected. The accuracy of the system was not as expected.

Arnaud Gotlieb, Morten et al. [4] proposed Reinforcement Learning (RL) algorithm to perform Test case prioritization. The system compares different variants of RL agents for automated test case selection problems.

The aim is to find the best performing agents from the given input. The metric used for comparison of algorithms is Normalized Average Percentage of Faults Detected. The drawback was that only small networks were applied.

Lachmann and Remo [5] compared Support Vector Machine, K-Nearest Neighbor, Neural Networks and Logistic regression to find the best suitable algorithm for Test case prioritization. The aim was to identify important test cases with a likelihood to find failures. The best performance was achieved when logistic regression was used. The metric used for comparison of algorithms is Average Percentage of Faults Detected. The drawback was the link between failures and test cases were not always provided.

Go¨kc¸e, Fevzi et al. [6] used Clustering using an unsupervised Neural network and fuzzy C-means clustering to rank the test cases in accordance with their preference degrees. The clustering of events was based on 13 attributes. The metric used for comparison of algorithms is Average Percentage of Faults Detected. The drawback was only behavioral sequence- based faults are revealed. Also, the logical and calculation errors are ignored.

Nida Go¨kc¸e and Mu¨bariz Emily [7] proposed Model based test case prioritization using neural networks. Test cases of previous papers which cannot run due to cost and time constraints were taken as input for this system. The cost and time were taken as metrics for comparison purposes. The classification results were found to have a high classification accuracy of 96% in test case prioritization. The problem with this system was that addition of new test case to the test suite made this approach less effective.

Methodology

For proposing our approach we have two motivations: First, some works related to defect prediction suggest that defect prediction strategies like test case prioritization are focused on automation tasks. Second, software developers mostly test the faulty parts of the program first but the existing test case prioritization methods do not consider the developers tendency. By considering the fault proneness score, which is evaluated by the defect prediction model in the test case prioritization methods we can address our mentioned motivations.

A. Modifying the coverage

Modifying the coverage so that we have more weightage for the test cases with higher probability of faults so that more priority is given to these test cases is our first motivation. Be- fore calculating coverage, it is important to find a probability function such that it completely ignores the sections of code which are not fault prone.

B. Proposed Defect Prediction Model

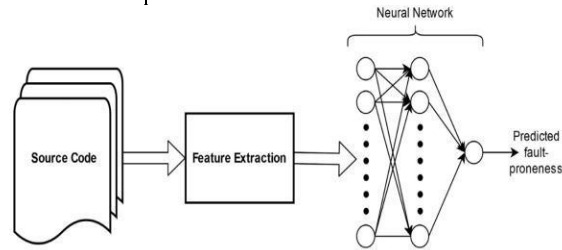

Fig. 1. Defect prediction model

An important metric used in this method is the fault prone- ness score of the system's classes. Our fault prediction model is shown in "Fig 1". This method starts with marking source code with bugs as buggy. Then features are extracted on the source code's classes that results in feature vectors. Then the prediction model is learnt based on the feature vectors and fault- proneness using a neural network.

The neural network used consists of two layers (total of three components):

- Input layer: Takes features obtained from extraction as input. Number of neurons in this layer = 104 (total number of features).
- Hidden layer: Consists of two components, the sigmoid activation function and 300 neurons.
- Output neuron: Consists of the sigmoid activation function.

The output of this neural network training is a classification with 2 classes (binary): Fault prone(1), Not fault prone(0). The output is in the range between 0 and 1 which is considered to be the fault-proneness score.

C. Proposed Algorithm

The Test case prioritization algorithm proposed is shown in "Fig 2". We start our algorithm by training the classifier of defect prediction. The information related to the source code's existing bugs is used in training the model. Further, this is used in predicting the fault-proneness i.e., the classes that are buggy are marked positive and otherwise negative.

The fault proneness score is then allotted to the code segment. Also, the coverage of code in the previous execution of code are extracted. Now, the coverage of test and the fault- proneness score are used to achieve an order of priority for test cases. We use the average percentage of faults detected metric for evaluation of performance in proposed test case prioritisation algorithm.

D. Modified Additional Strategy

Before executing the modified additional strategy, learning of the neural network with the help of the discovered bugs and the earlier forms of code must be done.

The modified additional strategy has 3 steps:

- Running defect prediction model to produce the array of fault-proneness score.
- Coverage based on fault is determined for every test case.
- Ordering of test cases with the use of a greedy approach. This approach finds test case with the

maximum coverage based on fault and chosen for execution in the next place in each iteration.

The following details are taken into account while performing this strategy:

- If the additional coverage is same for many test cases, then total coverage is taken into consideration and the test case with highest total coverage is picked for execution.
- Furthermore, if there is similarity in the total coverage as well, then the order of appearance in the suite is considered.
- Firstly, test cases' additional coverage is calculated and then the coverage of that particular test case is decreased from all of the available units.

E. Modified Total Strategy

This strategy is mostly related to the modified total strategy in the first two process. The final step in modified total strategy is sorting the test cases based on the value of coverage that is based on the faults.

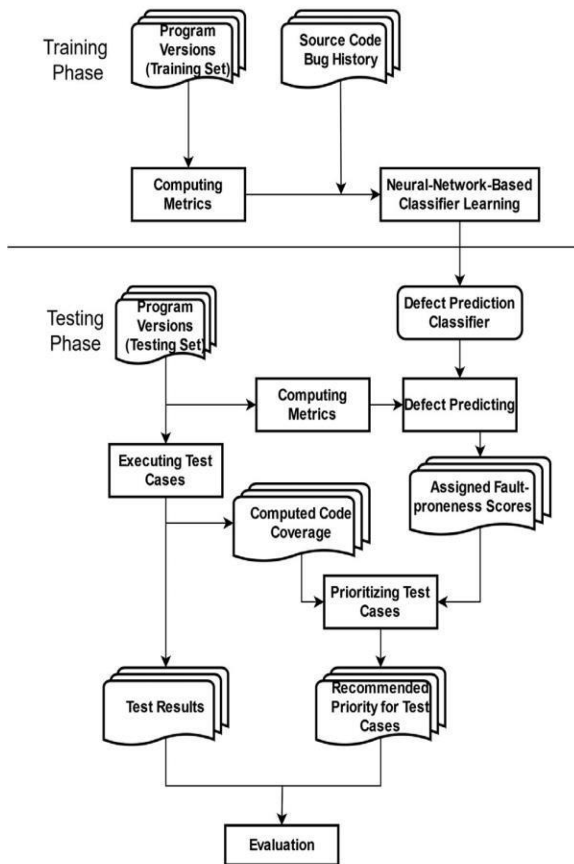

Fig 2. Proposed algorithm

F. Time Complexity Comparison

There is a need to compare the proposed modified strategies. One thing to be noted here is that the count of the features that are used in our methodology is constant. Thus, the time complexity of these algorithms is approximately equal to their traditional methodologies. Also, the difference between the time taken for execution of the modified and traditional algorithm is very minimal.

TABLE I TIME COMPLEXITY OF DIFFERENT METHODS

Method	Traditional method	Modified method
Additional method	O (n2m)	O (n2m + mf) = O (n2m)
Total method	O (mn + nlogn)	O (mn + nlogn + mf) = O (n + nlong)

TABLE II FEATURES OF DEFECT PREDICTION

Feature Type	Feature Category	Feature Definition	Feature Count
Input	Metrics of source code	Used to quantify different program code traits	52
Input	Clone metrics	Used to perceive the number of clones with same syntax but different variable names	8
Input	Coding rule violations	Used for counting coding violation by regulations	42
Input	Git metrics	Used for counting the number of committers and commits for every file	2
Output	Defect label	Label that indicates whether the file is faulty in this version of the project or not	1

Results And Conclusion

We have proposed a new approach in Test case prioritization so that we take into consideration the fault-proneness score of each code unit in the TCP method that is based on the coverage. We have studied our methodology with the help of the average percentage of faults detected metric and observed that all modified versions of strategies are more efficient. In addition, the comparison that we have performed with the performance metric shows that the difference in performance between the traditional and modified techniques is quite significant.

Future Work

The results of the execution of test cases can be considered to improve results in the future. Also, the defect prediction model can be used cross project so as to obtain an improved analysis of source code's fault proneness. Investigating the possibility for modification of the previously available TCP methods based on the code coverage. Also, we can test the proposed methodology on bigger datasets and wide range of programming languages.

References

[1] P. Gupta and N. Sinha, "Neural networks for identification of nonlinear systems: An overview," in Soft Computing and Intelligent Systems, 2020.

[2] Khatibsyarbini, M. A. Isa et al., "Test case prioritization approaches in regression testing: A systematic literature review," in Information and Software Technology 93, January 2018, pp. 74–93.

[3] Kalyani, Rayapureddy et al., "Test case prioritization using requirements clustering," in International Journal of Applied Engineering Research, vol. 13, no. 15, July 2018, pp. 11 776–11 780.

[4] Spieker, Helge et al., "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, July 2017.

[5] Lachmann and Remo, "Machine learning-driven test case prioritization approaches for black-box software testing," in The European Test and Telemetry Conference, Nuremberg, Germany, 2018.

[6] GO¨ KC¸ E, NIDA et al., "Model-based test case prioritization using cluster analysis: a soft-computing approach," in Turkish Journal of Electrical Engineering and Computer Sciences, vol. 23, no. 3, April 2015, pp. 623–640.

[7] Go¨kc¸e, Nida et al., "Model-based test case prioritization using neural network classification." in Computer Science and Engineering Computer Science and Engineering An International Journal, vol. 4, no. 1, February 2014, pp. 15–25.