DOI: http://dx.doi.org/10.26483/ijarcs.v9i1.5273

ISSN No. 0976-5697

Volume 9, No. 1, January-February 2018

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A MACHINE LEARNING BASED APPROACH FOR DETECTING NON-DETERMINISTIC TESTS AND ITS ANALYSIS IN MOBILE APPLICATION TESTING

Mr. Rajkumar Joghee Bhojan,
Principal Consultant, Wipro Technologies,
Bangalore, India

Dr. D. Ramyachitra,
Assistant Professor, Department of Computer Science,
Bharathiar University, Coimbatore, India

Dr. K.Vivekanandan,
Director, BSMED, Bharathiar University,
Coimbatore, India

Dr. Subramaniam Ganesan,
Professor, Oakland University, Rochester,
MI 48309, USA

Abstract: Hundreds of different mobile devices are on the market, produced by different vendors, and with different software features and hardware components. Mobile applications, while running on different devices, may behave differently due to variations in the hardware or O.S. components. Since mobile applications are expected to be deployed and executed on diverse mobile platforms, they must be validated on different mobile platforms and devices. Due to the peculiarities of mobile application development, there is a need for a quality assurance approach that focuses on its challenges. Moreover, mobile test executions take long time because all the tests were executed on different environments and developers had to create complex tear down procedures. Such procedures were lengthy and far from perfect, leading to unpredictable failures. Regression testing is a crucial part of Mobile app development and it checks that software changes do not break existing functionality. In every regression test execution, the final results are expected either always pass or always fail for the same code. But, in real time project release cycles, some of the tests will be non-deterministic, in other words called flaky tests. It reduces the importance of regression testing cycle and it is very difficult to trust on these results. These results significantly reduced the trust in the tests and thus undermined the whole mobile app test automation effort. We trained machine learning classifiers separately on each test result dataset and compared performance across datasets. The proposed model predicts result types as Non-Deterministic orDeterministic tests from the regression suite results executed in various release cycles.

Keywords: Mobile App Testing, Regression Testing, Non-Deterministic Tests, Machine Learning Algorithms, Decision Tree, Random Forest Algorithms.

1. INTRODUCTION

Software Testing represents an important part of a software development Life Cycle process. It is a stage where the errors and issues still presented in the system should be discovered and fixed. It is also a very costly process. Resources — time and people — are spent to prepare the test case scenarios and to execute them. Its cost is estimated to be between 40% and 80% of the total cost of development [1]. Mobile devices such as smartphones and tablets have become the de facto computers in our daily lives [2].As authors, Rajkumar J Bhojan, et.al.,described in [3] the mobile applications and mobile users are growing rapidly, it is indeed for researchers and testing experts to come up with effective verification techniques to ensure reliability of these mobile applications. Authors also mentioned that mobile applications are becoming increasingly sophisticated, they significantly increase the requirement for functional and non-functional tests. [3]
The rest of this paper is organized as follows. In Section II, we review related work in identifying Non-Deterministic tests using different methods. In Section III, we proposed Data Collection and Random Forest algorithm for test results. In Section IV, we evaluated the performance of our algorithms over large set of rows of simulated data, followed by discussions and concluding remarks in Section V.

2. RELATED WORK

Authors Alex Gyori, et.al., in "Reliable Testing: Detecting State-Polluting Tests to prevent Test Dependency” proposed a technique, called PolDet, for finding tests that pollute the shared state. Their findings in a nutshell, PolDet finds tests that modify some location on the heap shared across tests or on the file system; a subsequent test could fail if it assumes the shared location to have the initial value before the state was modified. They also stressed to aid in inspecting the pollutions, PolDet provides an access path through the heap that leads to the polluted value or the name of the file that was modified. Finally they implemented a prototype PolDet tool for Java and evaluated it on 26 projects, with a total of 6105 tests. PolDet reported 324 polluting tests, and our inspection found that 194 are relevant pollutions that can easily affect other tests. [4]
Authors ArashVahabzadeh, et.al., did a thorough research work on Non-Deterministic test. In their research paper "An Empirical study of Bugs in Test Code", they presented the first empirical study of bugs in test code to characterize their

prevalence and root cause categories. They mined the bug repositories and version control systems of 211 Apache Software Foundation (ASF) projects and found 5,556 test-related bug reports. They also compared properties of test bug with production bugs, such as active time and fixing effort needed and qualitatively study 443 randomly sampled test bug reports in detail and categorize them based on their impact and root causes. (1) Compare properties of test bugs with production bugs, such as active time and fixing effort needed, and (2) qualitatively study 443 randomly sampled test bug reports in detail and categorize them based on their impact and root causes.

Their results are listed below

- Around half of all the projects had bugs in their test code;
- The majority of test bugs are false alarms, i.e., test fails while the production code is correct, while a minority of these bugs result in silent horrors, i.e., test passes while the production code is incorrect;
- Incorrect and missing assertions are the dominant root cause of silent horror bugs;
- Semantic (25%), Non-Deterministic (21%), environment related(18%) bugs are the dominant root cause categories of false alarms;
- The majority of false alarm bugs happen in the exercise portion of the tests, and
- Developers contribute more actively to fixing test bugs and test bugs are fixed sooner compared to production bugs. [5]

In another research work “Detecting Assumptions on Deterministic Implementations of Non-deterministic Specifications”, authors August Shi, et.al.found a technique called NONDEX for detecting Non-Deterministic tests due to ADINS (Assumes a Deterministic Implementation of a method with a Non-Deterministic Specification) code. They implemented NONDEX for Java and found 31 methods with non-deterministic specification in the Java Standard Library, manually built non-deterministic models for these methods, and used a modified Java Virtual Machine for these methods, and used a modified JVM to explore various non-deterministic choices. They also evaluated NONDEX on 195 open-source projects for GitHub and 72 student submissions from a programming homework assignment. NonDEX detected 60 Non-Deterministic test in 21 open-source projects and 110 Non-Deterministic tests in 34 student submissions. [6]

3. MACHINE LEARNING

Authors in [7] describe that Machine learning is an exciting area that is seen significant advances lately in both theories and practices. As the machine learning tools and techniques getting more mature, they are increasingly applied to many different field for both research and business needs. However, the amount of data that is generated in today’s world has being grown exponentially. In machine learning algorithms, variable and feature selection is nothing but trivial. A set of carefully chosen features can make the prediction more accurate; make the calculating process faster; and can lead to better understanding between the process and the data it generated. In addition, if only a subset of the variables are useful to construct learning features, it can reduce storage requirements and simply data visualization results. Traditionally, finding the dominating sets of variables relies on the experts’ domain knowledge of the system. However, as pointed in the survey study [8], a variable that is deemed useless by itself can provide substantial performance improvement if it is paired with other variables. Or put it differently, two non-contributing variables can be useful if they are both included in the features of a machine learning algorithm. Likewise, it is possible to gain better performance if a presumably redundant variable is included in the learning process. In this research paper, we propose Decision Tree Algorithm and Random Forest Algorithm for identifying Non-Deterministic Tests from pool of Results/Reports files. As noted by authors [9], in real-world data are often represented on multiple manifolds. Examples of this can be usually found in machine learning, signal and image processing, pattern recognition, computer vision and information retrieval.

In our research work, Decision Tree Algorithm will identifyNon-Deterministic tests in Mobile application testing results based on DT algorithm prediction. Decision tree learning is a type of algorithm which maps observations about an item to conclusions about the items' target value. Every node is denoted asa Non-Deterministic Test and trimming method is used for each node in the tree. The trimming method is otherwise known as pruning technique which reduces the complexity of the classifier and boosts the predictive accuracy. This algorithm stops removing Non-Deterministic tests when no further selection can be made.

As there are some issues like “overfitting” with Decision tree algorithm, we further extended our finding with Random Forest Algorithm. Random Forest (RF) [10, 11, 12] is an ensemble classification and regression trees (CARTs) that are grown in a random subspace of data. Each tree in the forest is grown using a bootstrap sample of instances from the data. One third of samples are left in out-of-bag (OOB) to estimate the prediction error. The attributes in the tree are chosen using random feature selection. A binary split is represented as a node, which terminates at a leaf. The data is recursively split into distinct subsets using appropriate splitting rules. Each subset of child nodes is purer than the corresponding parent node. Finally, the subjects are classified by majority votes over all trees in the forest. RF outperforms the prediction when the trees do not exhibit a correlation with each other. It can estimate the importance (Gini) of each attribute in the training data. It can also determine the pairwise proximity among the samples. These features of RF uncover the interactions between genes in the absence of main effect. The algorithm is implemented in a number of open source software packages, such as R [13], Rapid miner [14], Weka [15] and Willows packages [16]. RF can be very suitable for handling large p-value problems.

As shown in the Table -1, the non-deterministic dataset has been created for training after cleaning the dataset. The training data set has both True (1) and False labels (0) and it has five attributes. The first step will be to find the partition with the least entropy. Then, the minimum-entropy partition for the whole data set was found.

Based on our previous experience, we found the following reasons for non-deterministic test failures as shown in the below list.

[a] Not having a framework [b] Hardcoded Test data [c] Using X,Y Coordinates or Xpath for element recognition [d] Using Shared Test Environments [e] Having test that are dependent on one another [f] Test not starting in a known state [g] Test no managing their own test data[h] Failure to use proper synchronization [i] Badly written test scripts [j]Network [k]Time [l] Floating point Operations [m]Async Wait [n] Resource Leak and [o] Unordered Collections

From the above list, important attributes / features are filtered for the training the model. In order to find the subset of correct attributes from the dataset, feature selection process is used. Similarly, all possible combinations of attributes will be selected from the dataset. The good combination will enhance the performance over selecting all attributes. The important benefits of feature selection are as follows:

- Overfitting Problem will be resolved
- Improves model accuracy
- Training time will be reduced, so that training will be faster

Table –1 Feature Selection

Test Results	Test Data	TE_Shared	Test Dependent	Sync Issue	Test Scripts	Long Time	Floating Point Operation	Selection
Yes	No	Yes	Yes	No	No	Yes	No	1
No	Yes	No	No	Yes	No	Yes	Yes	0
Yes	No	No	Yes	Yes	Yes	No	No	1
No	Yes	Yes	No	Yes	No	Yes	Yes	0
No	No	Yes	No	No	Yes	No	No	0
Yes	No	Yes	Yes	No	No	Yes	No	1
Yes	Yes	No	No	Yes	No	Yes	Yes	1
Yes	Yes	No	Yes	Yes	Yes	No	No	0
Yes	Yes	Yes	No	Yes	No	Yes	Yes	1
Yes	No	Yes	Yes	No	No	Yes	No	1

The above table (Table-1) shows the sample training set data for constructing a model.

4. IMPLEMENTATION

As shown in the Figure -1, a model has been implemented to detect Non-Deterministic tests from mobile testing results in our research POC. As a first step in our implementation, we collected all the mobile testing results from different devices, work stations, online results and stored them in a separate repository.

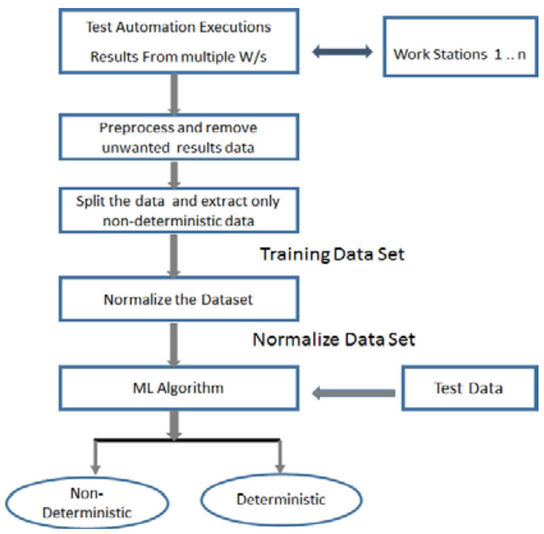
Test Automation Executions Results From multiple W/s
Work Stations 1 .. n
Preprocess and remove unwanted results data
Split the data and extract only non-deterministic data
Training Data Set
Normalize the Dataset
Normalize Data Set
ML Algorithm
Test Data
Non-Deterministic
Deterministic

Fig – 1 Flow chart for the Model

In preprocess step, all other data like test steps, test descriptions, expected results, etc., are removed from the results. The next step is to extract only non-deterministic failed / passed results for training dataset for the model. Then dataset normalization is done before Machine Learning algorithm is used for mining the data and detecting Non-Deterministic tests. In this research POC, ten sets of sample results were taken from five different iterations. All the results were generated as 'Pass' or 'Fail' and it was named "1" for Pass and "0" for Fail for the calculations purposes. Similarly, the attributes are divided based on its presence.

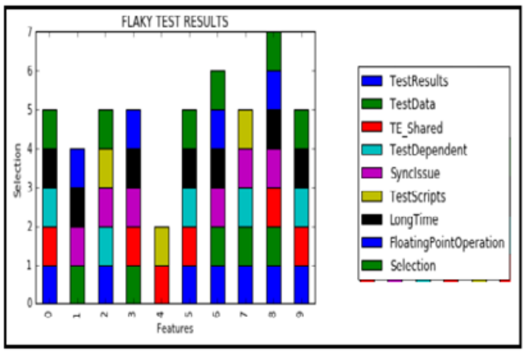

Fig. 2. Graph for different Features

Though there are more number of algorithms available in machine learning area for predictions, Decision Tree and

Random Forest Algorithms are selected in this research paper. Decision tree and Random Forest algorithms stopremoving devices when no further selection can be made. Based on our Proof Of Concept (POC) data, the following steps are derived.

- If the list of attributes like "No Framework", "Test Data", "XY coordinates", etc., is empty, i.e., there are no more possible questions to ask, then create a leaf node that predicts the most common label and then stop.
- Otherwise, try partitioning the data by each of the attributes
- Choose the partition with the lowest partition entropy
- Add a decision node based on the chosen attribute
- Recur on each partitioned subset using the remaining attributes.
- Decision Tree Algorithm is used to split dataset into subset,
- Are they all pure?
- If it is "Yes" stop or if all "No" stop;
- Else repeat.
- The lowest entropy comes from splitting on first attribute "No Framework", so it will need to make a subtree for each possible No Framework value. When attribute comes pure "Yes" or "No", it stops further classifications. As part of research, we used 100 records for decision tree as shown in the Fig.3. In this way, Non-Deterministic test selection will be narrowed down and a logical decision is arrived based on attributes which test set has to be selected for testing and which one should not be selected.

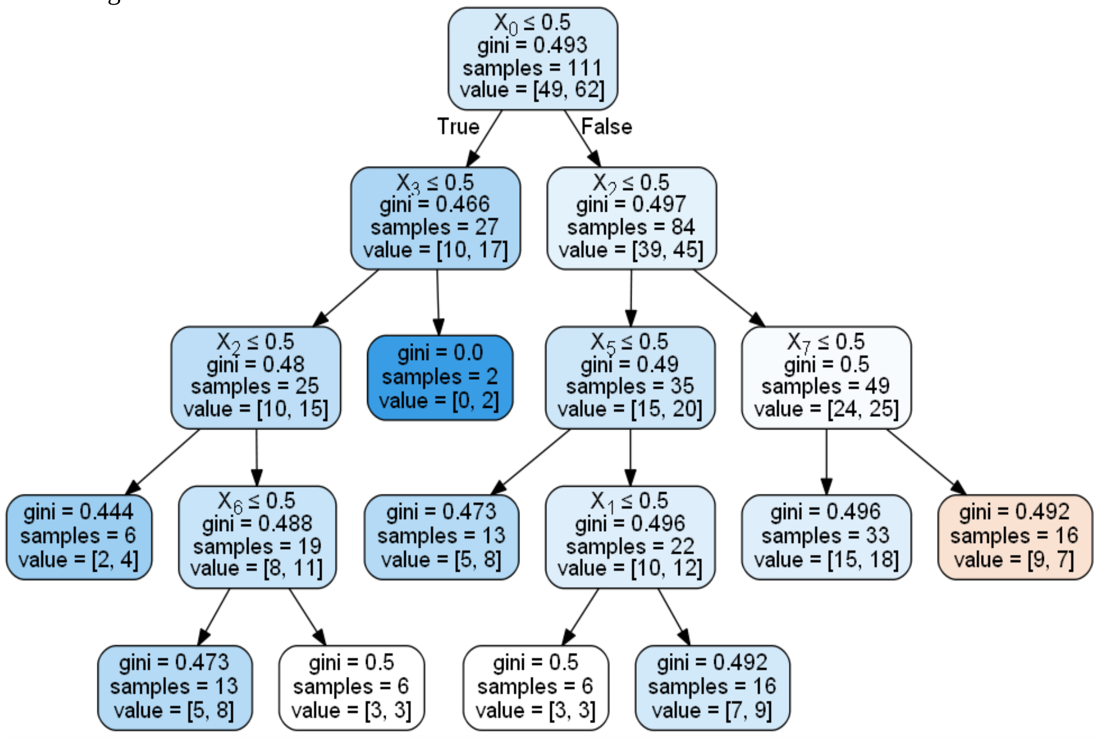
$X_0 \leq 0.5$ gini = 0.493 samples = 111 value = [49, 62]
True
False
$X_3 \leq 0.5$ gini = 0.466 samples = 27 value = [10, 17]
$X_2 \leq 0.5$ gini = 0.497 samples = 84 value = [39, 45]
$X_2 \leq 0.5$ gini = 0.48 samples = 25 value = [10, 15]
gini = 0.0 samples = 2 value = [0, 2]
$X_5 \leq 0.5$ gini = 0.49 samples = 35 value = [15, 20]
$X_7 \leq 0.5$ gini = 0.5 samples = 49 value = [24, 25]
gini = 0.444 samples = 6 value = [2, 4]
$X_6 \leq 0.5$ gini = 0.488 samples = 19 value = [8, 11]
gini = 0.473 samples = 13 value = [5, 8]
$X_1 \leq 0.5$ gini = 0.496 samples = 22 value = [10, 12]
gini = 0.496 samples = 33 value = [15, 18]
gini = 0.492 samples = 16 value = [9, 7]
gini = 0.473 samples = 13 value = [5, 8]
gini = 0.5 samples = 6 value = [3, 3]
gini = 0.5 samples = 6 value = [3, 3]
gini = 0.492 samples = 16 value = [7, 9]

Fig-3. DT Classification Result

A. Prediction Results:

The following tables Table2 & 3 display prediction results derived from Decision Tree Algorithm and Random Forest Algorithm. As shown in the Table -2, the model will predict whether the given data is Non-Deterministic or not. If it is a Non-Deterministic or Flaky test, the same test will be sent back for either correction or rerunning the execution.

Table –2 Prediction Results

Inputs for Prediction	Predicted Result	Description
clf.predict([[1,0,1,1,0,0,1,0]])	[1]	Non-Deterministic
clf.predict([[1,0,1,1,0,0,1,0]])	[1]	Non-Deterministic
clf.predict([[1,0,1,1,0,0,1,0]])	[1]	Non-Deterministic

clf.predict([[1,0,1,1,0,0,1,0]])	[0]	Deterministic
clf.predict([[1,0,1,1,0,0,1,0]])	[0]	Deterministic

Random forests is currently one of the most used machine learning algorithms in the non-streaming (batch) setting. This preference is attributable to its high learning performance and low demands with respect to input preparation and hyper-parameter tuning.[17] The Random forest algorithm is an ensemble learning method that takes average results of several decision trees to classify its samples. As the name denotes, each decision tree is trained on a random training data subset, perhaps using random features as well. Random forest is a collection of many decision trees. If we use many trees in the forest, many or all of the features will be included. This inclusion of many features will help limit our error due to bias and error due to variance. If features weren't chosen randomly, trees in the forest could become highly correlated. This is because a few features could be particularly predictive and thus, the same features would be chosen in many of the trees. The following table shows predicted results found by random forest classifier.

Table – 3 Random Forest Classifier

Different Inputs for Prediction	**Predicted Result**	**Description**
clf.predict([[1,0,1,1,0,0,1,0]])	[1]	Non-Determi-nistic
clf.predict([[1,0,1,1,0,0,1,0]])	[1]	Non-Determi-nistic
clf.predict([[1,0,0,1,1,1,0,0]])	[0]	Deterministic
clf.predict([[0,0,1,0,0,1,0,0]])	[1]	Non-Determi-nistic
clf.predict([[1,1,0,0,1,0,1,1]])	[1]	Non-Determi-nistic

5. CONCLUSION

We have proposed a model to classify Non-Deterministic test based on several results derived from multiple executions. We found that feature set are sufficientto classify Non-Deterministic tests. We also did in-depth analysis on the performance of two Machine Learning algorithms with respect to identifying Non-Deterministic tests. The future works will include performance enhancement by examining and comparing the classification accuracy of Non-Deterministic Tests with other classification algorithm using deep learning.

6. ACKNOWLEDGEMENT

Authors would like to thank project stakeholders for their valuable support in collecting the data. . I am immensely thankful to Dr. D. Ramyachitra, who provided insight and expertise that greatly assisted this research paper.My sincere thanks toDr.K.Vivekanandan and Dr.Subramaniam Ganesan for theirconstant encouragement and support throughout this research work.

REFERENCES

[1] S. Eldh, H. Hansson, S. Punnekkat, A. Pettersson, and D. Sundmark, "A framework for comparing efficiency, effectiveness and applicability of software testing techniques," in Testing: Academic and Industrial Conference - Practice And Research Techniques, 2006. TAIC PART 2006. Proceedings, Aug 2006, pp. 159–170.

[2] S. Allen, V. Graupera, and L. Lundrigan, "The smartphone is the new PC," in Pro Smartphone Cross-Platform Development. Apress, 2010, pp. 1–14.

[3] Rajkumar J. Bhojan, K.Vivekanandan and Subramaniam Ganesan, "Mobile Test Automation Framework for Automotive HMI", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 3, Issue 1, January 2014

[4] Alex Gyori, et.al Reliable Testing: Detecting State-Polluting Tests to prevent Test Dependency", ISSTA -2015, Baltimore, MD, USA DOI - 10.1145 /2771783.2771793

[5] ArashVahabzadeh, Amin MilaniFard, Ali Mesbah, "An Empirical Study of Bugs in Test Code", ICSME 2015, Bremen, Germany, DOI:978-1-4673-7532-0/15, IEEE-2015

[6] August Shi, Alex Gyori, OwolabiLegunsen and Dark Marinov, "Detecting Assumption on Deterministic Implementations of Non-Deterministic Specifications", IEEE International Conference on Software Testing, Verification and Validation, 2016. DOI:10.1109/ICST.2016.40

[7] Baijian Yang, Tonglin Zhang, "A Scalable Feature Selection and Model Updating Approach for Big Data Machine Learning", IEEE International Conference on Smart Cloud, 2016, DOI 10.1109/SmartCloud.2016.32.

[8] I. Guyon and A. Elisseeff, An introduction to variable and feature selection, The Journal of Machine Learning Research, 3, 11571182, 2003.

[9] Lili Li, JianchengLv,ZhangYim, "A non-negative representation learning algorithm for selecting neighbors", Springer Link, Machine Learning, Feb-2016, volume 102, issue 2, pp 133-153

[10] J. Han, M. Kamber, and J. Pei, Data mining: concepts and techniques: Morgan kaufmann, 2006.

[11] L. Breiman, "Random forests," Machine learning, vol. 45, pp. 5-32, 2001.

[12] Y. Qi, "Random Forest for Bioinformatics," in Ensemble Machine Learning, ed: Springer, 2012, pp. 307-323.

[13] A. Liaw and M. Wiener, "Classification and regression by random Forest," R news, vol. 2, pp. 18-22, 2002.

[14] X. Liu, K. Tang, J. R. Buhrman, and H. Cheng, "An agent-based framework for collaborative data mining optimization," in Collaborative Technologies and Systems (CTS), 2010, International Symposium on, 2010, pp. 295-301.

[15] E. Frank, M. Hall, L. Trigg, G. Holmes, and I. H. Witten, "Data mining in bioinformatics using Weka," Bioinformatics, vol. 20, pp. 2479-2481, 2004.

[16] H. Zhang, M. Wang, and X. Chen, "Willows: a memory efficient tree and forest construction package," BMC bioinformatics, vol. 10, p. 130, 2009.

[17] Gomes, H.M., Bifet, A., Read, J. et al., "Adaptive random forests for evolving data stream classification", Mach Learn (2017) 106: 1469.https://doi.org/10.1007/s10994-017-5642-8