Project Report of Master of Engineering

Early Defects Detection using Test Cases Prioritization in Iterative Software Development Process

반복적인 소프트웨어 개발 공정에서의 시험 사례 우선 순위화 기법을 이용한 조기 결함 탐지

February 2023

Graduate School of Engineering Practice
Seoul National University
Department of Engineering Practice - Data Science Track

Jaesung HWANG

Early Defects Detection using Test Cases Prioritization in Iterative Software Development Process

반복적인 소프트웨어 개발 공정에서의 시험 사례 우선 순위화 기법을 이용한 조기 결함 탐지

Professor Sung-Joon Cho, Seong-Woo Kim

Submitting a Master's Project Report

February 2023

Graduate School of Engineering Practice

Seoul National University

Department of Engineering Practice - Data Science Track

Jaesung HWANG

Confirming the master's Project Report written by

Jaesung HWANG

February 2023

Chair	Yoon-Mo Koo	(Seal)
Examiner	Sung-Joon Cho	(Seal)
Examiner	Seong-Woo Kim	(Seal)

Abstract

Large-scale continuous integration environments are increasingly applied to software development in modern industries. Although this environment guarantees high development productivity, it is greatly dependent on the amount of simultaneous testing. Though this environment is suitable for performing a test cycle as quickly as possible and giving feedback to the developer, the overall software development process can become slow and inefficient as the number of test cases increase.

To improve the execution efficiency of software testing, machine learning techniques based on test execution records accumulated through repeated execution were used to derive the test cases that are highly likely to fail in the next test execution In this study, Shift Left, a test method that advances execution through finding and preventing defects early in the software delivery process, was used to increase the efficiency of the software development. Specifically, test case prioritization method was applied to detect defects early and report back to the developers with quicker feedback on the failed test results. The success of the test case prioritization was confirmed by measuring the APFD value.

In this study, experiments were executed using test cases that are currently used in the actual industry. As a result of the Test Case priority of machine learning introduced in this study measured to be 1.7 times more effective compared to the test performance, of traditional method that uses

a heuristic method strategy.

Keywords : Defect detection, Test case prioritization, Regression testing, Machine Learning

Student Number : 2020-28114

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background of Research

The software development process is increasing the efficiency of development and verification through the continuous integration(CI) environment [1] [2]. When developing particular software such as an operating system(OS), verification process must be repeatedly performed not only on the part in current development, but also on the part that has been already deployed by repeatedly performing regression tests which eliminates defects continuously [3] to ensure that there are no residual issues in the previously deployed version.

In case of operating system software, as the size of modern software is growing rapidly while being continuously developed and distributed, the number of test cases that need to be repeatedly verified has to increase inevitably. [4] In order to efficiently perform software testing in the continuous software integration process, a method to increase efficiency by applying automation technology has been applied, but, currently, automation technology alone cannot bring about additional efficiency in a situation where test cases that need to be repeatedly performed are increasing [5].

Regression test is run to spot defects due to changed software configurations by repeatedly performing almost the same test cases that have been

accumulated. In this case, the list of test cases is already set before the actual test begins, and in many cases, the order of test cases for test execution gets continuously applied without changing. In this case, it is impossible to predict when a defect will be discovered while executing all test cases [6] which makes it difficult to give feedback on the defect to the developer until all tests are completed.

To make up for such disadvantage, a strategy of making case with higher probability of fail a priority in the test suites gets often applied. By doing so, it is possible to detect fail of test cases in the early stage of test execution, deliver quick feedback to developers, and improve defects quickly.

This research is to propose an algorithm that derives test case priorities through machine learning with data collected from test execution history records of webOS operating system software which are under development in the field, and is to measure the performance when the corresponding test case ranking is applied to test prioritization with APFD.

1.2 Scope of Research

There have been studies which presented adjusting of test case prioritization using test history data accumulated through regression tests in a continuous integration environment. However, such studies have been determined priorities based on the algorithm created based on the knowledge and experience of experts or testers who have been rich experience with the test for a long time. Although the results of research with this approach can show a significant increase in the APFD value, the specific condition that re-

quires professionals with expertise and experience in the field makes it hard to be universally applied. Moreover, since the test history data gets changed every time a regression test is performed, expert intervention will be able to be applied every time.

This Research takes an approach that does not use heuristic logical methods and instead extract features from test history data accumulated over a long period of time, and apply machine learning or artificial neural networks [7] to them to infer the result. In aiming this, the features of the collected data are selected, and the effectiveness of test case prioritization is compared and evaluated with APFD values using various models, and then the results get analyzed.

1.3 Composition of Paper

This paper as the following structure: chapter 1 describes the background and purpose of this research and then the scope of the research. chapter 2 examines previous studies related to the theoretical content of the technology for determining test case prioritization, which is necessary to understand this research. chapter 3 shows the field test performance data to be reviewed in this research, and explains the method and evaluation of the proposed test case priority prediction method using machine learning/artificial neural network through learning. In chapter 4, the feature values extracted from field test history data and the selected learning model are used to describe the test case prioritization experiment environment and discuss the comparative analysis of various experiment results. In chapter 5, finally, the

conclusion of this research report is stated, and the results and contents of the research described are written together with the remaining tasks for the future.

Chapter 2

Related Works

This chapter provides a basic overview of test case prioritization, the background of this paper, and introduces previous research. In section 1, firstly, test case prioritization and the data used in machine learning to determine it are presented. Section 2, subsequently introduces related works of test case priority prediction.

2.1 Test Case Prioritization

By adjusting the test case priority so that the test case with the highest probability of fail gets executed first, testers can detect defects at an early stage and can quickly deliver feedback on the results of failed test cases to developers.

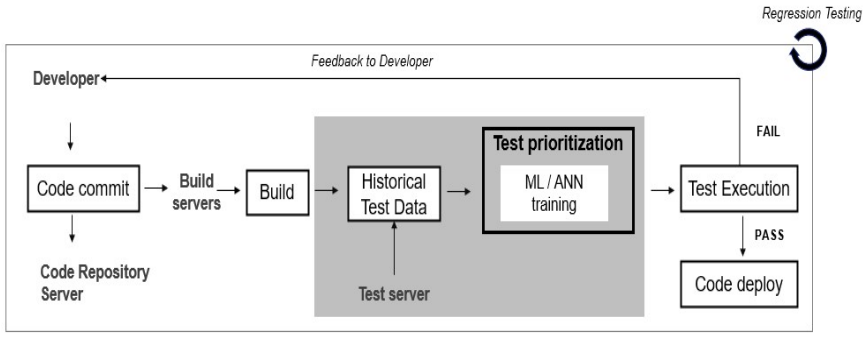


Figure 1: Regression test in a CI environment

2.2 Continuous Integration of Software

Continuous integration (CI) refers to the execution of a process where quality control is continuously applies in software engineering. In a continuous integration environment, developers' integrating their work several times a day into a single source code repository, and managing quality in conjunction with automated testing tools [8] are becoming software engineering method. The benefit of continuous integration is that defects can be detected in the early stage of software life-cycle, from development to production [9].

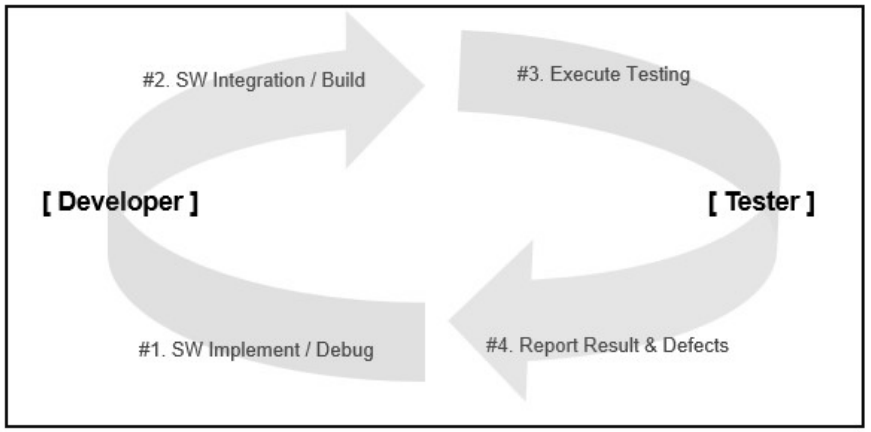


Figure 2: Continuous Integration of Software

Zhao et al. [10] reported that repeated performance of software development projects has a positive effect on reducing defects in the project life-cycle. However, it is said that the positive effect gradually diminishes over time after continuous integration is applied, and the trend gets flattened. [11]

To develop software such as an operating system (OS) especially, per-

forming repetitive regression tests is a must because the functions of previously distributed versions must be maintained intact. However, as the cycle of repetitive test increase, the effect of early detection of defects gradually declines.

This Research aims to maintain the effect of early defect detection and the efficiency of test execution by performing the test case prioritization [12] adjustment at every test cycle.

2.3 Test data features of based ML

Various data and their features that can be used to determine test case priorities using machine learning. [13] Test case priorities can be determined by the complexity of the source code, the developer's code-related history, the code coverage through test execution, and user input information through machine learning as well. Data on the test execution history can be collected in quality control department where the researcher conducts the test in the workplace.

Table 1: Example Features in Test History Data

Execution history	Description
Duration	Last execution time (duration) of test cases
Fail cases	Number of failed tests in the current commit
Test case result	Test cases' verdict (PASS/FAIL)
Result History	Verdicts which refer to the history length
Change Status	Number of changes of verdicts by each test case
Test age	Number of test cycles

2.4 Related Works

Related Works for determining test case priorities using test execution records used an approach where a heuristic [14] [15] [16] algorithm was applied. This is a method of calculating a priority value by characterizing test case results and execution time data from previous test execution results. [17] The following two assumptions are made about applying the heuristic method.

- Recent test result has a bigger influence weight (ω) on test case priorities.
- In case given test cases have the same priority value, the one with the short duration gets performed first.

Based on this assumption, the priority value is calculated through the following formula. Here, test case (i) / test cycle (j) / Executed Status (ES) were expressed in recording of the execution.

$$P(t_i) = \sum_{j \in 1...m} \omega_j * max(ES(i,j), 0) \tag{2.1}$$

$$P(t_i) \dot{+} = \frac{max(duration)}{mean(duration_i)} \tag{2.2}$$

For instance, assuming that the results and duration of each test case up to the previous 10 test cycles for three test cases are presented as shown in Table 2, the priority value is calculated by applying Equation 2.12.2. The test results are classified by the color of each cell in the table, gray indicates fail/white indicates pass, and the number in the cell means the execution

duration time for each test case.

Table 2: Test case results and duration time by test cycle

Test Cycle	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
Test Case #1	60	64	60	61	84	60	60	64	61	60
Test Case #2	180	61	60	65	76	62	77	61	57	60
Test Case #3	62	88	66	62	40	62	72	64	61	60

The five previous histories are selected here, and a larger weight (ω) value is given to the latest one in cycle.

Table 3: The results and duration of the last five test cycles

Test Cycle	-1	-2	-3	-4	-5
Weight (ω)	0.50	0.20	0.15	0.10	0.005
Test Case #1	60	64	60	61	84
Test Case #2	180	61	60	65	76
Test Case #3	62	88	66	62	40

The results in Table 4 can be obtained by calculated using equation 2.1 based on the results of the last five test cycles data and the weight (ω) value.

Table 4: Results of based equation (2-1)

Test Cycle	-1	-2	-3	-4	-5	$P(t_i)$
Test Case #1	0	0.20	0	0	0.005	0.205
Test Case #2	0	0	0.15	0.10	0.005	0.255
Test Case #3	0.50	0.20	0	0	0.005	0.705

When the priority value is finally calculated, the result of Table 5 can

be obtained by calculation using Equation 2.2

Table 5: Results of based equation (2-2)

Test Cycle	-1	-2	-3	-4	-5	mean	$P(t_i)$	priority value
Test Case #1	60	64	60	61	84	65.8	2.74	2.95
Test Case #2	180	61	60	65	76	88.4	2.04	2.30
Test Case #3	62	88	66	62	40	63.6	2.83	3.54

The order of test execution was originally performed in the sequence of 'Test Case 1 $\rightarrow$ Test Case 2 $\rightarrow$ Test Case 3'. Based on the above procedures, if the test execution order is determined by the size of the priority value, it can be changed to 'Test Case 3 $\rightarrow$ Test Case 1 $\rightarrow$ Test Case 2'.

Chapter 3

Methodology

Problems defined in the field are defined and experimental plans are established to solve them in this chapter. In the research, test execution record data used in the field is collected and analyzed, a model that predicts test case priority using various machine learning and artificial neural networks is selected, and the optimal method of predicting the next test performance result is explored.

This paper will further describe the performance evaluation method using indicators that can evaluate test case priorities for experiment results.

3.1 Problem Definition

In order to detect defects early in a regression test that is repeatedly performed in a continuous integration environment such as a software operating system [18], a test case priority strategy should be applied so that tests are performed from test cases with a high probability of occurrence of defects.

Test case prioritization is defined as following in general [5]

- Assume that a test suite is given as T, a set of permutations of T as PT, and a objective function from PT to the real value as f
- The problem is to find $\forall T'' \in PT, T'' \neq T' : f(T') \geq f(T'')$ which is

like $T' \in PT$

Here, f is a function that's used to accomplish the goal of early defect detection and its high result value means that the priority of early detection of defects is good.

This f value is a function that returns the weighted average rate of defect detection(APFD) [19]in this research and this value is used as a priority evaluation index for early defect detection.

APFD can be calculated by the following equation.

$$APFD = 1 - \frac{TF_1 + \ldots + TF_m}{nm} + \frac{1}{2n} \tag{3.1}$$

The APFD value is a real number between 0 and 1 whose higher values indicate higher priority. n is the number of test cases, m is the number of defects found through testing, T is a set of test cases, and TF_i is the test case that identified the i-th defect in test case execution.

The definition of the problem put together , $\forall T'' \in PT, T'' \neq T' : APFD(T') \geq APFD(T'')$ which is like $T' \in PT$ is to find a test case priority $T'$the objective function can be defined as Equation 3.2.

$$g = maximize(APFD) \tag{3.2}$$

For instance, if APFD value of a test case set is 0.540 and is increased to 0.944 by applying the test case priority method, the test case priority gets

improved by 1.7 times when it is interpreted in terms of APFD value. Looking at the defect density graph, the effect can be seen more clearly.

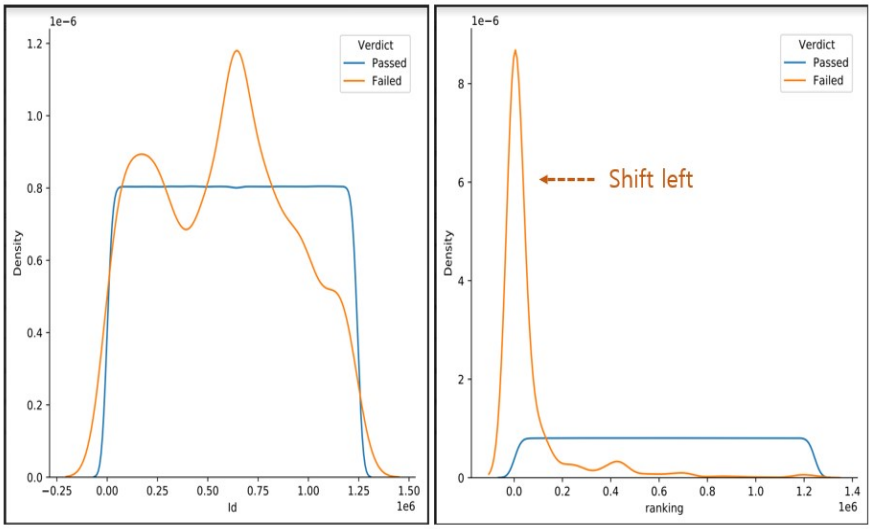


Figure 3: APFD = 0.540 , APFD = 0.944

The horizontal axis of figure 3 shows the order of test execution, and the vertical axis represents the density of test results success and failure. The blue line indicates the distribution of pass, and the red line shows the distribution of fail.

Especially, when checking the red line's density of fail, the higher APFD value, 0.944, the higher the defect density at the early stage of the test, and the effect of early detection (Shift-Left) can be confirmed.

3.2 Data Collection

For field test performance records for use in this research, API (Application Programming Interface) testing execution records were collected

among the regression tests of the continuous integration environment of webOS operating system development.

This research constructs a model that predicts results of tests that have not yet been performed by extracting 9 data feature values, including data from the previous the last five tests. How a model that predicts test case priorities through data feature values gets selected will be described in the next chapter.

3.3 Model Selection

This research selects a learning model that can predict test case priorities. As a result of confirming the features of the data, using 9 input variables, the following test results for each test case predict success/fail, which is categorical data, so rather a binary classification model can be considered..

On the other hand, the purpose of the research is to determine the ranking [20], which is an important goal, so Ordinal Regression [21] [22] model will be applied rather than the binary classification model. Ordinal regression, which is also known as ordinal classification in statistics, is a type of regression analysis used to predict variables whose values exist on an arbitrary scale, where only the relative order among categorical outcomes matters. Because of this characteristic, ordinal regression is also called rank learning [23].

That is, test cases are prioritized by ranking from the largest value to

Table 6: Selection of regression model

List of selected models
1) Logistic Regression [24]
2) Linear Regression
3) RandomForest Regression
4) XGBoost (eXtreme Gradient Boosting) Regression [25]
5) LGBM (Light Gradient Boosting Machines) Regression
6) ANN Regression

the smallest value, with the value between 0 and 1 as the relative order value that predicts the fail of the next test case result, using a regression model. Verification of the test case priorities determined in this way is determined by calculating APFD value.

In this research, among the following machine learning models, regression models and artificial neural network model are selected and experiments are performed in order to select the model that exhibits the best performance.

The performance evaluation index of the model is the MSE (Mean Squared Error) [26] which refers to the average of the squares of the difference between the actual value(y) and the predicted value$(\hat{y})$ gets calculated, and the formula is the following:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.3}$$

The main setting parameters of the selected models were applied to the experiment as follows. In case of a machine learning model, only changes made to the default settings are shown. Also, the weight values for the test

case results of the last 5 cycle test results with the heuristic method to be compared will be described as well.

Table 7: Information on setting parameters of the model

Model name	Hyper parameters	Value
0) Heuristic Method	Weights previous 5-cycle	[0.05,0.1,0.15,0.2,0.5]
1) Logistic Regression	Penalty	⌉2
	Solver	liblinear
	Tol	0.1
	Max iter	1000
	Class weight	balanced
2) Linear Regression	All	Default
3) RandomForest Regression	n estimators	1000
4) XGBoost Regression	n estimators	1000
	booster	gbtree
	max depth	7
5) LGBM Regression	All	Default
6) ANN	Initialization	Xavier (Default) [27]
	Activation Function	Mish [28]
	Hidden Layer 1	10 neurons
	Hidden Layer 2	20 neurons
	Hidden Layer 3	15 neurons
	Training Epochs	30
	Loss Function	Mean Sqaured Error [26]
	Optimizer	Adam [29]
	Learning Rate	0.001
	Total Parameters	651
	Trainable Parameters	651

Chapter 4

Experiment and Evaluation

4.1 Experiment Overview

Setting up of design and environment for the experiment before conducting an experiment is required. Test execution history data in the field were collected, and a data set was selected to be put in learning. The test case priorities extracted through the existing heuristic methodology are to be proved in this research and the priorities extracted through learning using machine learning and artificial neural network are compared in order to evaluate each performance. Subsequently, the evaluation is proceeded with APFD value, like the defined objective function. Then, additional experiments that require confirmation may be performed after the review of the experimental results.

4.2 Experiment Environment Configuration

The configuration of the experimental environment in this research entails a pre-processing process for learning models by extracting features from test history data collected in the field, and detailed setting information of each model must be confirmed as well. In terms of learning, the prediction of pass or fail of this test case is performed by ordinal regression,

and the test case prioritization is sorted in the order of the largest predicted value. Performance evaluation compares and evaluates test case priorities extracted by heuristic methodology and sequential regression of selected models based on APFD values, and selects the model with the best performance among given models.

Table 8: Overview of test history record data set

Data set Information	API Testing (Device-A)	API Testing (Device-B)
Test Cases	603	492
Test Executions	25,950	23,508
Failed Test Cases	418	97
Failed Test Executions(ratio)	0.016	0.004
Collection Period	22.04.11 - 05.18	22.07.25 - 09.13

There are many data features in the test execution history data, yet the following features were extracted and selected from the test record data stored in the field.

Table 9: List of experiment data features

Features	Description
1) Duration	Execution time for test case
2) Test Cycle	Cycle which test case executed
3) Execute Status (E5)	Result of previous 5-th cycle
4) Execute Status (E4)	Result of previous 4-th cycle
5) Execute Status (E3)	Result of previous 3-th cycle
6) Execute Status (E2)	Result of previous 2-th cycle
7) Execute Status (E1)	Result of previous 1-th cycle
8) Status Changes	Value has been changed up to the previous cycles
9) Last Run	Last test case execution time

In the Research, 9 data features are extracted and used for learning, including execution time, test cycle, record of the results of the previous 5 test cases, the number of test case changes during the previous cycles, and

the time when the last test case was executed. Such pre-processed data was trained to predict the test results of this round. That is, by learning the result data of the previous 5 test cycles, the priority of the test cases of this round of test is determined in the order of the highest predicted fail value in the current test.

The experiment proceeds with the following procedure.

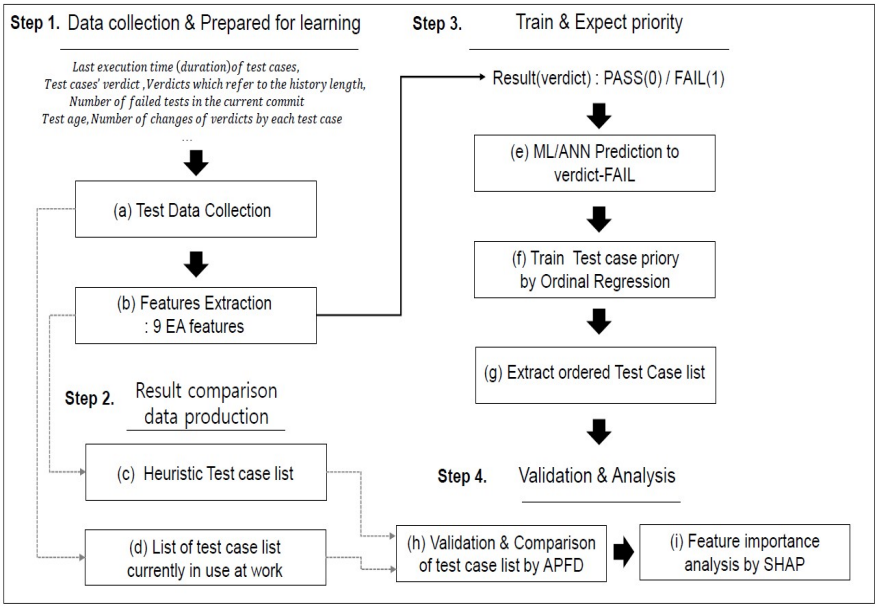


Figure 4: Experiment/Evaluation steps

This research collects test history data stored in the software continuous integration system(a) and extracts features from data to determine priorities(b). To compare the experimental results and priority performance results, the heuristic method(c) applied in previous studies and the procedure(d) currently being performed in the field are made ready. Machine

learning/artificial neural network models are selected(e), and the rank of modified test cases is extracted(g) by learning with the rank regression method(f). Test case priority performance results are compared by evaluating (c), (d), and (g) with APFD(h). In addition, we analyze which data features may affect the results using the SHAP (Shapley Additive Explanations) [30] method (i).

4.3 Experiment Result and Analysis

The research collected data sets which are applied to two devices of API testing in practice. Experiments were performed identically on both data sets. For each data set, the test case priorities established by ordinal regression were derived using the models, and the test case priorities currently being performed in the field were derived through the APFD value and the result derived using a heuristic method are compared and analyzed.

4.3.1 Device-A Experiment

The MSE values are compared to evaluate the performance of the model trained through the collected data, and then the APFD value results are compared to verify the performance of test case priorities calculated from each model. The performance change can be intuitively interpreted through the numerical results of the APFD value and the graph. Finally, SHAP analysis is checked to analyze the influence of data features on performance results.

4.3.1.1 MSE Result: Device-A

Comparing the performance of prediction models through data learning with MSE values, excluding the test case rankings and heuristic methods applied to the current practice that cannot be measured, showed that the LGBM regression model with the lowest value predicted the best with slight difference with others.

Table 11: MSE Result: Device-A

Model name	MSE
Original Test case list	N/A
Heuristic Method	N/A
Logistic Regression	0.0343
Linear Regression	0.0079
RandomForest Regression	0.0072
XGBoost Regression	0.0065
LGBM Regression	0.0062
ANN	0.0079

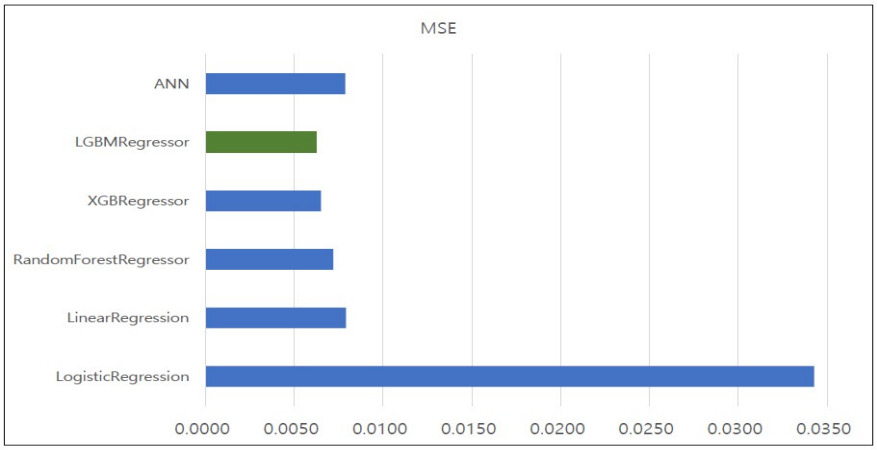


Figure 5: MSE Result: Device-A

4.3.1.2 APFD Result: Device-A

As a result of predicting by rank learning with each model, the LGBM Regression model shows the best result: it exhibits about 2.85 times higher than the result of the original test list used in the field, and 1.66 times higher than the existing heuristic methodology.

Table 12: APFD Result: Device-A

Model name	APFD (%)
Original Test case list	0.3367
Heuristic Method	0.5797
Logistic Regression	0.8891
Linear Regression	0.8585
RandomForest Regression	0.9570
XGBoost Regression	0.9389
LGBM Regression	0.9603
ANN	0.9124

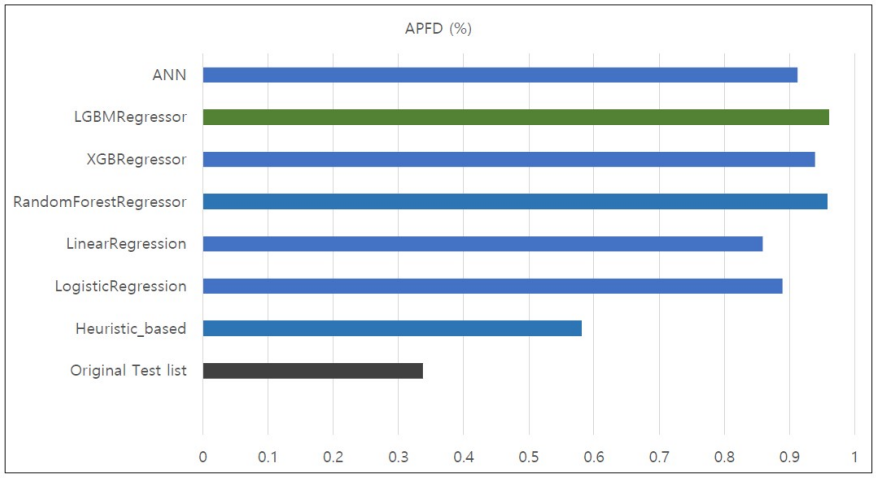


Figure 6: APFD Result: Device-A

4.3.1.3 APFD Result: Device-A Graph

In the graph of APFD result, the horizontal axis refers to the test execution order, and the vertical axis represents the density of test results pass and fail. Especially, the red line that shows the density of fail should be noted. It can be seen that as the APFD value increases, the fail density moves left, which means that the density of immediate defect detection according to the order of test execution is high, which can be interpreted as an early defect detection effect. These three models of machine learning (LGBM/RandomForest/XGBoost) show similarly good results in the graph.

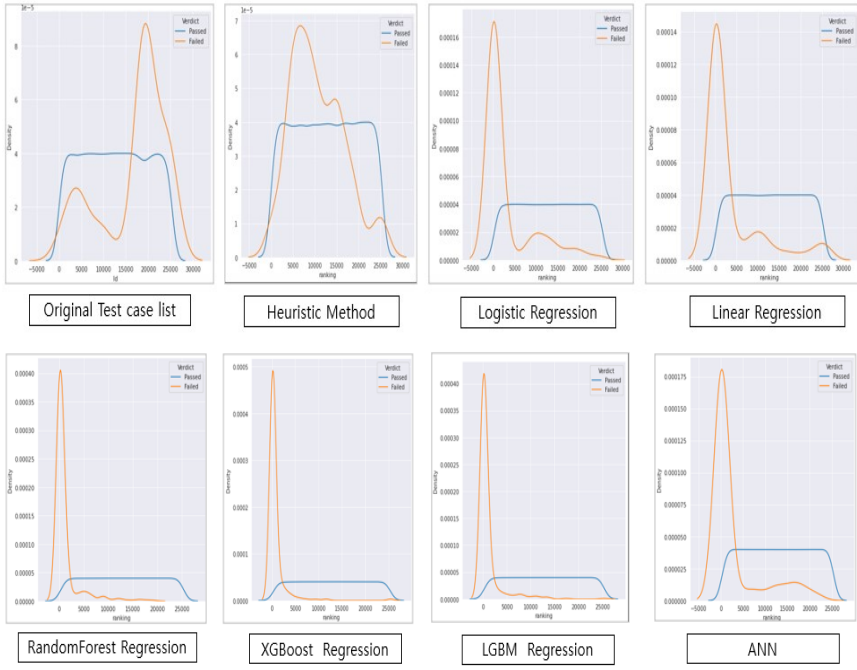


Figure 7: APFD Result: Device-A Graph

4.3.1.4 Feature Impact Analysis: Device-A

The SHAP [30] method is applied to check the impact of specific data features on the results predicted by the model. In the API Testing (Device-A) experiment, at two (RandomForest/LGBM) models with the best APFD values, it is confirmed that which data features affect the test results by using SHAP.

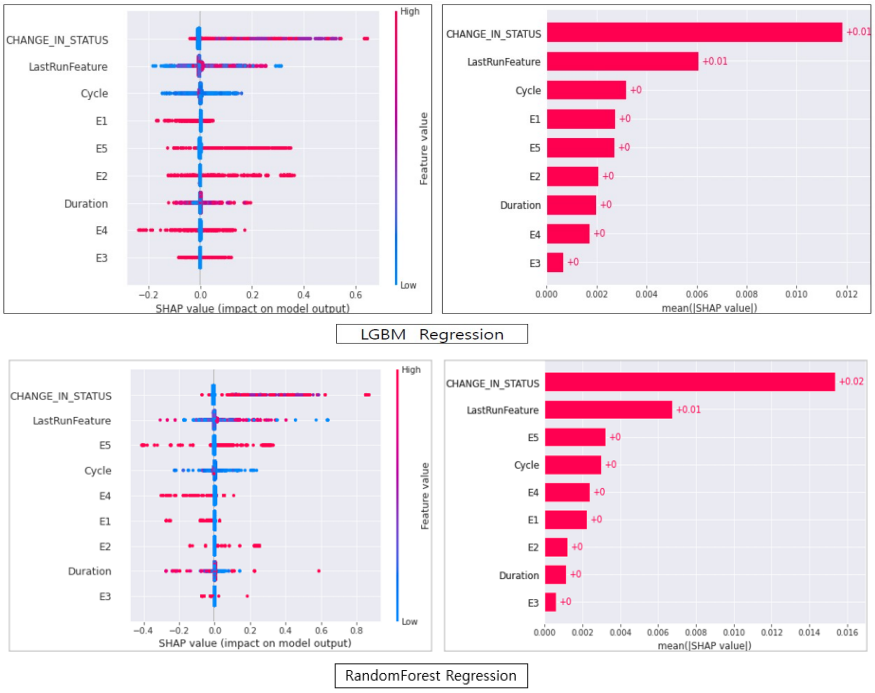


Figure 8: Feature Impact Analysis: Device-A graph

In the SHAP analysis of the two models, it was found that the data feature that has the most impact on the prediction result is 'Status Changes/Last Run'.

The calculation of SHAP is defined as the below equation. It is to determine the degree of impacts by calculating how much it contributes to the result when there is no specific variable $'i'$.

Shapley Value function $\varphi_i(f,x)$ = Shapley Value of a specific feature i.

$$\varphi_i(f,x) = \sum_{S \subseteq F|i} \frac{|S|!(F-|S|-1)!}{\mathrm{F!}} \left[f_{S \cup i}\right)\left(x_{S \cup i} - f_s\left(x_s\right)\right)] \tag{4.1}$$

- f : Prediction model,
- F : The entire set of variables,
- S : Subset without 'i'th variable,
- $f_S \cup_i (x_S \cup_i))$: Predicted value including 'i'th variable, and
- $f_S(x_S)$: Predicted value without 'i'th variable.

4.3.2 Device-B Experiment

As in the Device-A experiment, performance comparison evaluation is performed with MSE and APFD values, and the influence of data characteristics is analyzed using SHAP.

4.3.2.1 MSE Result: Device-B

In the same way as Device-A, it can be seen that the Random Forest Regression model with the lowest value predicted the result with slightly better performance than that of the other models.

Table 13: MSE Result: Device-B

Model name	MSE
Original Test case list	N/A
Heuristic Method	N/A
Logistic Regression	0.0043
Linear Regression	0.0010
RandomForest Regression	0.0007
XGBoost Regression	0.0009
LGBM Regression	0.0011
ANN	0.0021

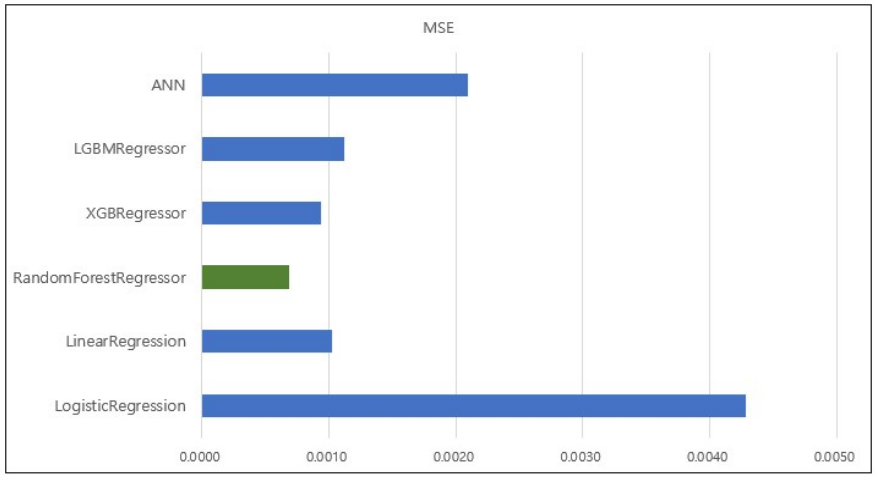


Figure 9: MSE Result: Device-B

4.3.2.2 APFD Result: Device-B

As per the Device-A results, the LGBM Regression model exhibits the best results. This is a value improved by about 28.85 times compared to that of the original test list used in practice, and a result improved by 1.80 times compared to the existing heuristic method.

Table 14: APFD Result: Device-B

Model name	APFD (%)
Original Test case list	0.0344
Heuristic Method	0.5528
Logistic Regression	0.9429
Linear Regression	0.9146
RandomForest Regression	0.9647
XGBoost Regression	0.9831
LGBM Regression	0.9924
ANN	0.9308

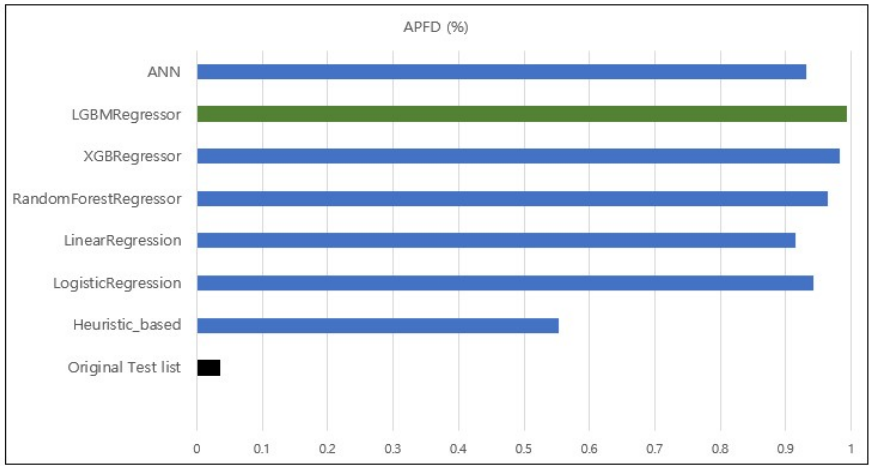


Figure 10: APFD Result: Device-B

4.3.2.3 APFD Result: Device-B graph

In case of Device-B, it is observed that the fail density of the red line increases rapidly when the test is finished in the original test case list result graph. From this, it can be confirmed that the APFD value is remarkably low.

The graph below shows similarly good results for the three models (LGBM/XGBoost/RandomForest) of this machine learning as in Device-A.

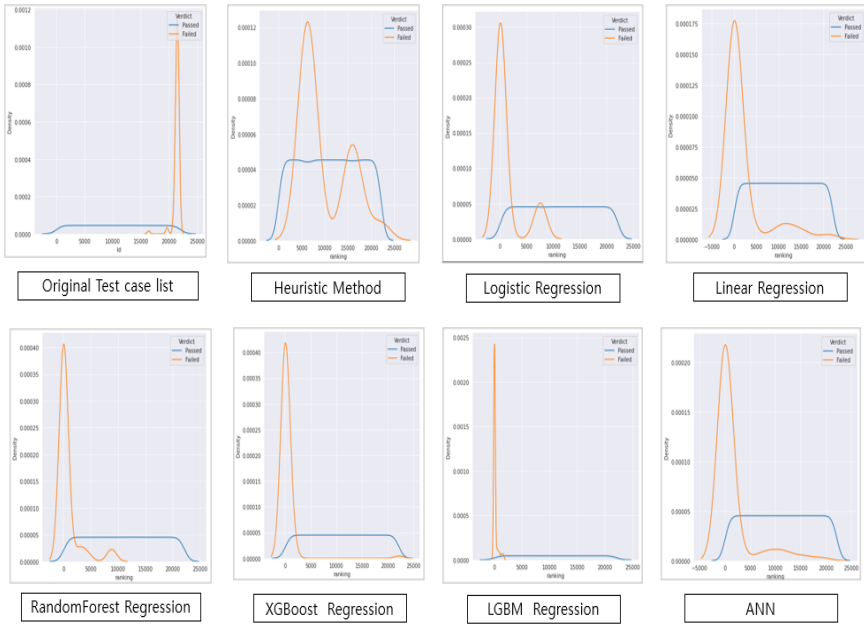


Figure 11: APFD Result: Device-B graph

4.3.2.4 Feature Impact Analysis: Device-B

As in the Device-A experiment, SHAP is used in the two (LGBM/XGBoost) models with the best APFD values to determine which values among the data features affect the results.

In the SHAP analysis of the two models, as the two data features 1 that have the most influence on the prediction results,

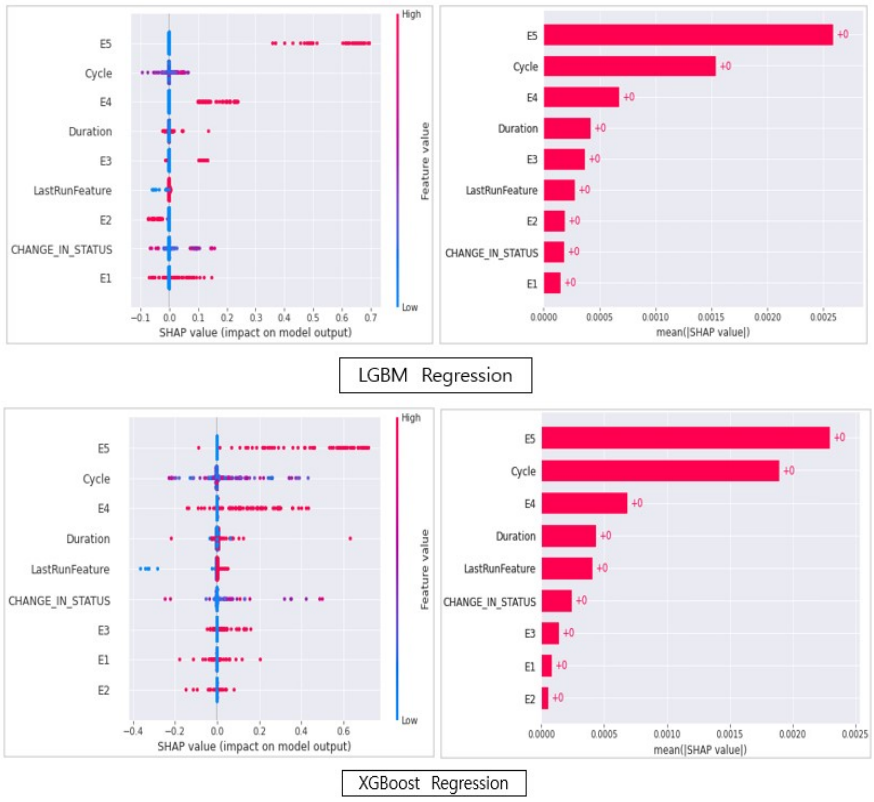


Figure 12: Feature Impact Analysis: Device-B graph

4.3.3 Experiment Result Analysis

It can be confirmed that the number of tests performed/number of failed test cases is reduced compared to the original amount of data when checking the data set applied to the experiment. For instance, it can be confirmed that the total number of tests executed by Device-A was 25,950, but the number applied to the experiment decreased to 25,387

Table 16 is the summary of the performance of models that extract test case priorities through experiments in this research. The performance

Table 15: Experiment data set

Data set Information	Device-A		Device-B	
	Original	Experiment	Original	Experiment
Test Cases	603	603	492	492
Test Executions	25,950	25,387	23,508	22,146
Failed Test Cases	418	391	97	97
Failed Test Executions(ratio)	0.016	0.015	0.004	0.004

of the model was confirmed with the MSE value, and the quality verification for the test case prioritization was calculated using the APFD value. In particular, in the process of predicting results in Device-A/Device-B, the very fact that the most important data feature is quite different as Status Changes/E5 shows that valid results are obtained from completely different features unlike the existing test methodology that applies weight to previous test records.

Table 16: Experiments result summary

Experiment Result	API Testing (Device-A)				API Testing (Device-B)			
	MSE	APFD	Improvement	Feature	MSE	APFD	Improvement	Feature
Original Test case list	N/A	0.3367	0	N/A	N/A	0.0344	0	N/A
Heuristic Method	N/A	0.5797	0.2431	N/A	N/A	0.5528	0.5184	N/A
Logistic Regression	0.0343	0.8892	0.5525	Cycle	0.0043	0.9429	0.9085	Last Run
Linear Regression	0.0079	0.8585	0.5219	Status Changes	0.0010	0.9146	0.8803	E5
ANN	0.0079	0.9124	0.5757	N/A	0.0021	0.9308	0.8964	N/A
XGBoost Regression	0.0065	0.9389	0.6022	Status Changes	0.0009	0.9831	0.9488	E5
RandomForest Regression	0.0072	0.9570	0.6204	Status Changes	0.0007	0.9647	0.9303	E5
LGBM Regression	0.0062	0.9604	0.6237	Status Changes	0.0011	0.9924	0.9581	E5

4.3.4 Further Analysis of Experimental Data

The test history data used in this research is a regression test performed repeatedly in a software continuous integration environment, and the quality level is stabilized. According to the test results, the number of failed cases (1.6%), compared to the number of successful cases (98.4%), are markedly less.

In this case, data imbalance [31] issue occurs as a result of the test case. Looking at the data set information of Device-A, the number of failed cases out of the total test results is at the level of 1.6%, which is significantly smaller than that of the successful results.

Analysis on whether there is an effect on the test results for data imbalance is further conducted. Firstly, in order to improve data imbalance, SMOTE (Synthetic Minority Oversampling Technique) [32] is applied to increase the number of fail results with a small number and reduce the number of pass results at random sampling and construct the synthesized data set. As a result, the number of pass and fail results is the same.

Table 17: Data Augmentation by SMOTE

Data set Information	API Testing (Device-A)	Synthetic Data Set
Test Cases	603	603
Test Executions	25,950	49,992
Failed Test Cases	418	24,996
Failed Test Executions(ratio)	0.016	0.5

After balancing the success and fail data, the MSE and APFD values are calculated through the same experiment and compared with the previous

experimental results.

4.3.4.1 MSE Result: SMOTE

In the same way as Device-A, the result shows that XGBoost Regression model with the lowest value shows slightly better prediction performance. However, the absolute value is about 10 times larger than the previous result, and this translates into the decreased the prediction accuracy; how this phenomenon affected the APFD value will be further examined.

Table 18: MSE Result: SMOTE

Model name	MSE
Logistic Regression	0.3682
Linear Regression	0.1395
RandomForest Regression	0.0862
XGBoost Regression	0.0773
LGBM Regression	0.0784
ANN	0.1088

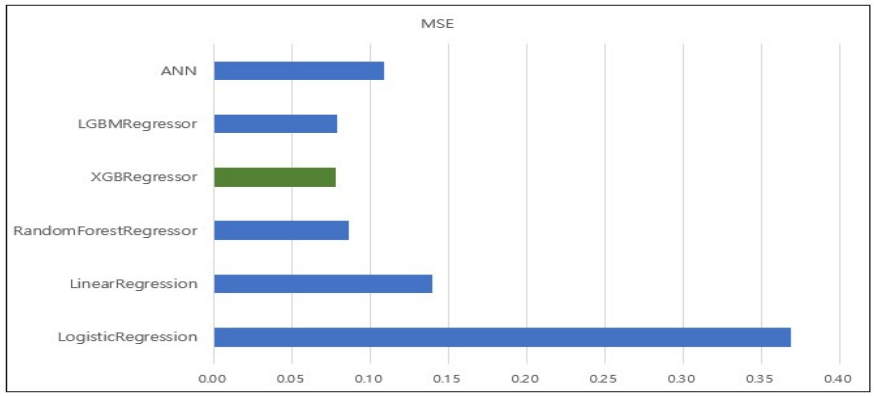


Figure 13: MSE Result: SMOTE

4.3.4.2 APFD Result: SMOTE

As for the APFD value, XGB Regression model shows the best results. This is about 2.88 times better than the results of the Original Test list and 1.23 times better than the heuristic method. Even if the methodology of this research balances it out, it can be inferred that the effectiveness is similar to that of the existing tests.

Table 19: APFD Result: SMOTE

Model name	APFD (%)
Original Test case list	0.2526
Heuristic Method	0.5896
Logistic Regression	0.6851
Linear Regression	0.6799
RandomForest Regression	0.7215
XGBoost Regression	0.7265
LGBM Regression	0.7254
ANN	0.7166

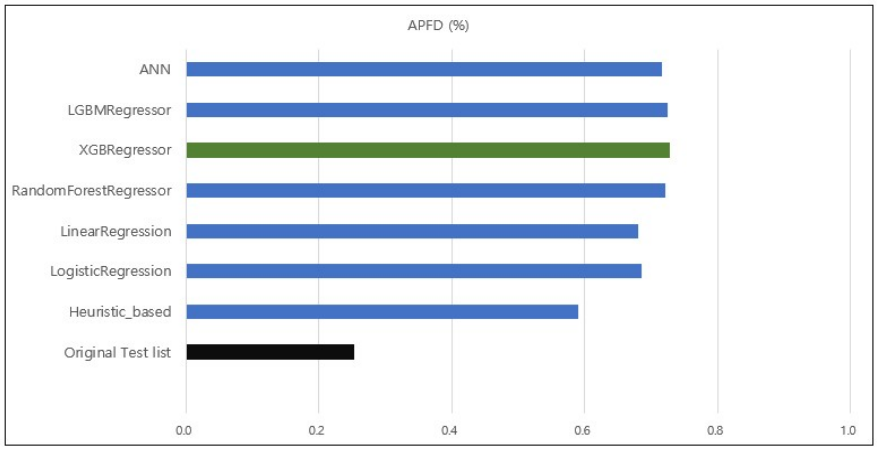
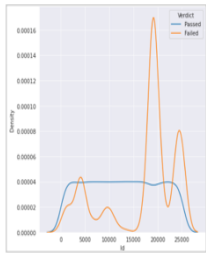
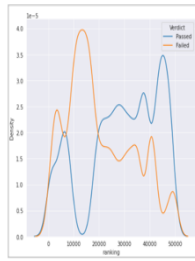


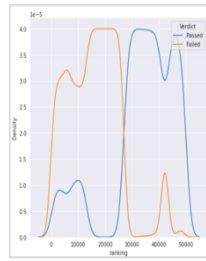
Figure 14: APFD Result: SMOTE

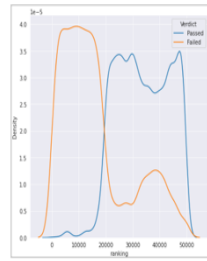
4.3.4.3 APFD Result: SMOTE graph

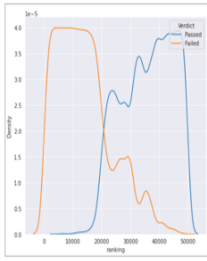
In a similar manner, in the result graph of APFD, the horizontal axis represents the test execution order, and the vertical axis represents the density of test results pass and fail. It also shows that as the APFD value increases, the density of fail shifts to the left (Shift-Left). Looking at the graph after balancing out the data, it is clear that the case of fail is performed first, and the case of pass is performed next, so it can be seen that the effect of early defect detection, which is the purpose of this research, has been applied equally. As in the experiment performed on the graph, the three machine learning models (LGBM/RandomForest/XGBoost) show similarly good results.

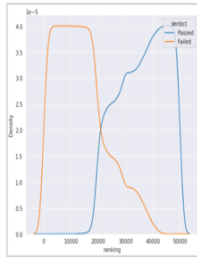


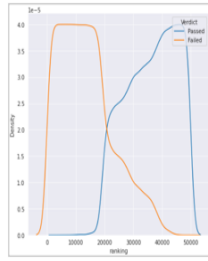


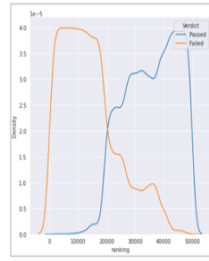












4.3.4.4 Feature Impact Analysis: SMOTE

As in the Device-A experiment, in the two (LGBM/XGBoost) models with the best APFD values, SHAP is applied to check which values of the data features affect the results.

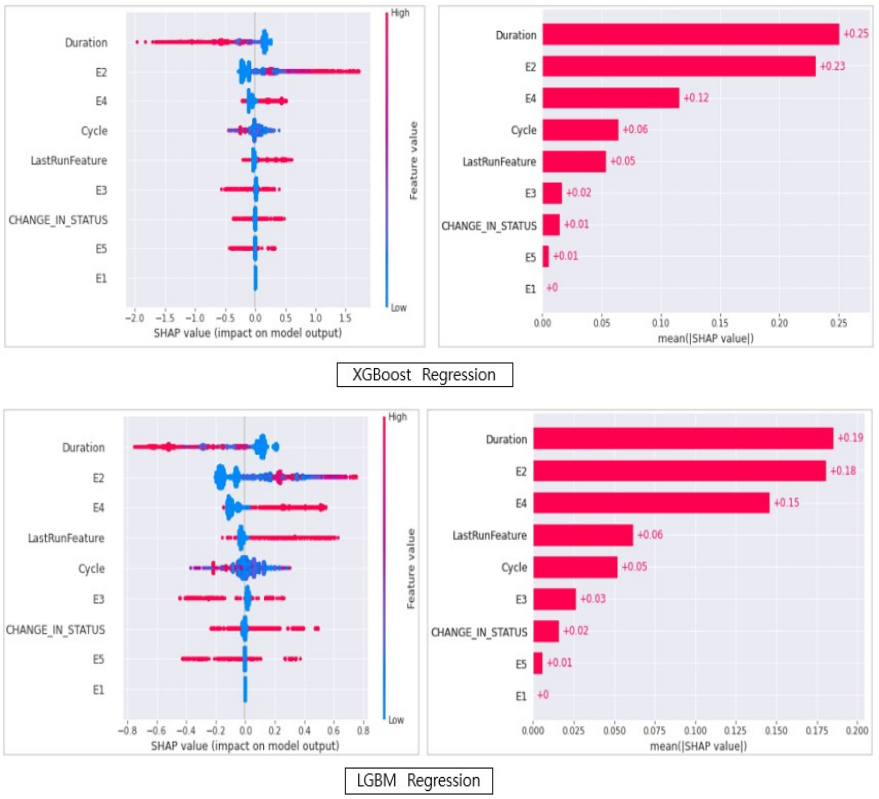


Figure 15: Feature Impact Analysis: SMOTE

In the SHAP analysis of the two models, 'duration' in Table1 acts greatly as the data that has the most influence on the prediction result, but in the left graph, there are many distributions below '0', indicating that it is hard to say it has a positive impact. Rather, the 3rd previous cycle record(E2)

can be interpreted as the most influential feature.

Through this additional experiment, the results of the experiment in the field data with severe data imbalance and the result after balancing the data by applying SMOTE seems to have the same tendency, so it can be seen that the applied method of the research is effective in test case priority.
On the other hand, in the sense that the application of regression test record data repeated for a long time in the software continuous integration environment to the research, in reality, the situation in which the number of failed results is significantly smaller than the number of pass results in existing experiments seems to be more closely related to the problem situation.

4.3.5 Further Analysis on Experimental History Length

Test record data used in this research took the history of the previous five cycles. The reason for proceeding with the length of the test history of five cycles was determined through the following experiment:

Using Device-A data, the length of test history was distinguished into 7 steps 2(H2), 3(H3), 5(H5), 7(H7), 10(H10), 13(H13), and 20(H20); then, experiments were performed with the same settings, and the results were confirmed as shown in Table 20 and Figure 16.

Table 20: Analysis on Experimental History Length

Model name	H2	H3	H5	H7	H10	H13	H20
Heuristic Method	0.5204	0.5843	0.5797	0.6224	0.5708	0.6472	0.5515
Logistic Regression	0.7916	0.8454	0.8892	0.8990	0.9212	0.9362	0.9441
Linear Regression	0.8265	0.8658	0.8585	0.8658	0.9220	0.9468	0.9600
RandomForest Regression	0.9032	0.9158	0.9570	0.9468	0.9586	0.9533	0.9560
XGBoost Regression	0.9077	0.9281	0.9389	0.9426	0.9595	0.9629	0.9625
LGBM Regression	0.9342	0.9473	0.9604	0.9651	0.9757	0.9807	0.9878
ANN	0.8047	0.8708	0.9194	0.9246	0.9337	0.9569	0.9659

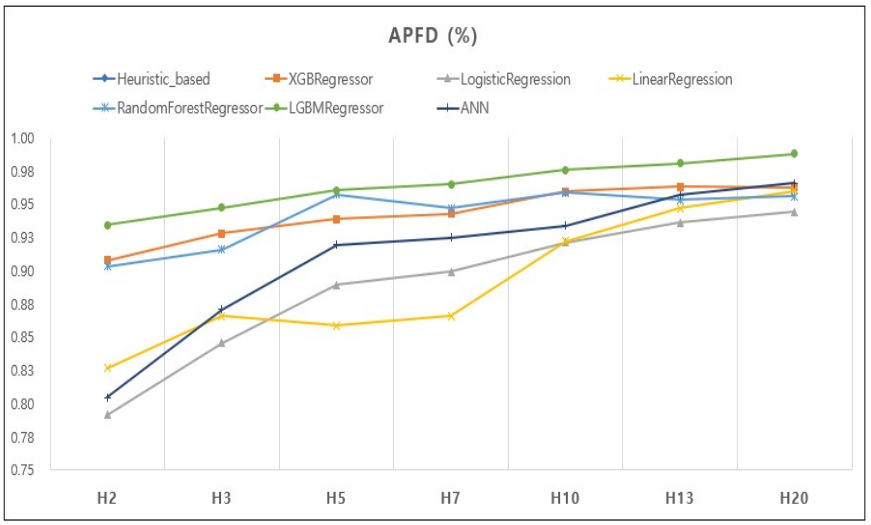


Figure 16: Analysis on Experimental History Length Graph

Looking at the results, it can be inferred that the longer the test history length is, the higher the APFD. In other words, it can be seen that the 20(H20) shows better performance than the 5(H5) length in this research. Looking at the APFD value, it can be concluded that the longer the test history length is, the better the performance.

However, it is necessary to review the following results together: it is the total number of test cases and the number of failed test cases according to the test history length. As can be seen in Tables 21 and 17, it can be seen that the quantity decreases rapidly after the 5(H5).

When considering performance, the number of total test cases and failed test cases can be deemed important because APFD is the variable used to calculate Equation 3.1. It was confirmed that as the number of two

test cases decreased, the number of test cases to re-prioritize decreased. The 5(H5), which is the history order before the section where the number of test cases rapidly decreases, was applied to this research.

Table 21: Number of Total Failed Test Cases according to Test History length

	H0	H2	H3	H5	H7	H10	H13	H20
Number of Total Test case	25,950	25,724	25,539	25,387	24,279	22,656	21,073	17,598
Number of Failed Test case	395	395	393	391	367	327	296	232

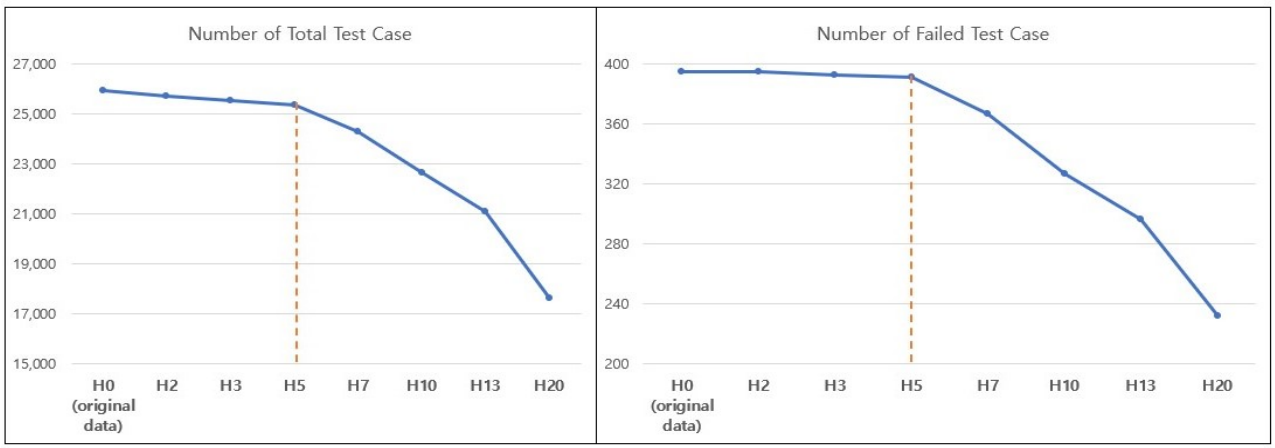


Figure 17: Number of Total Failed Test Cases graph

Chapter 5

Conclusion

This research defined the problem considering the situation in practice– how to detect defects early while conducting repeated regression tests in the software continuous integration environment. Also, features were extracted from previous studies using the accumulated test performance data, and test cases were ranked by rank-learning through machine learning and artificial neural network models, and results were compared through experiments on test case priorities. As a result of the experiment using two device (A/B) case data of API Testing performed in the existing collaboration, the APFD value compared to the priority used in the existing work was 2.85 times (Device-A) / 28.85 times (Device-B), compared to the related heuristic research methodology, 1.66 times (Device-A) / 1.80 times (Device-B) improved results were confirmed.

Through the experimental results of this research, when the LGBM regression model was applied while running machine learning, it was confirmed that the APFD value was higher than that of the existing test case ranking and heuristic methodology in both devices. This shows that defects can be detected early whenever repetitive tests are performed in practice by determining and applying the test record data through machine learning to prioritize each test case without human intervention.

The data, in terms of the software development process stage, is a history of the system test execution, which is the final stage before the product gets deployed. In determining test case priorities, there are many data with various features during the development life cycle other than test history data.

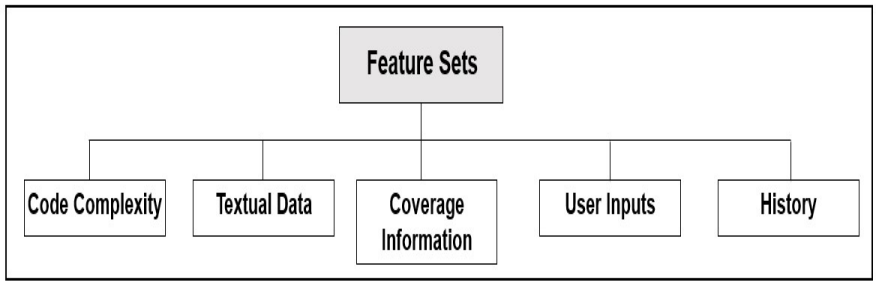


Figure 18: Data features for test case prioritization

On top of test history, although there are various records to be used as data features, such as software complexity, developer records, test coverage, and expert settings, these were not applied as in practice only test execution data would be available for the testing departments. This part will be expanded through future assignments to study improvement models that combine various data features.

The improvement my research brought was confirmed through experiments, yet there are various data features that require additional application as shown in Figure 18 in the field. If the access/usage rights to various data can be shared through among different parts in business organizations, it will be possible to construct a model for each data feature and continue

to search for improvements in the direction of evolution and development into an ensemble structure in the future. Moreover, in connection with test case prioritization, test cases that do not need to be executed are extracted through test case selection [33] which is intended to lead to research that continuously improves the efficiency of test execution time and resources.

Bibliography

[1] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[2] D. Marijan, A. Gotlieb, and S. Sen, "Test case prioritization for continuous regression testing: An industrial case study," in *2013 IEEE International Conference on Software Maintenance*, pp. 540–543, IEEE, 2013.

[3] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 235–245, 2014.

[4] J. Anderson, S. Salem, and H. Do, "Striving for failure: an industrial case study about test failure prediction," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 49–58, IEEE, 2015.

[5] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software testing, verification and reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[6] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th international conference on software engineering*, pp. 119–129, 2002.

[7] H. Jahan, Z. Feng, S. Mahmud, and P. Dong, "Version specific test case prioritization approach based on artificial neural network," *Journal of Intelligent & Fuzzy Systems*, vol. 36, no. 6, pp. 6181–6194, 2019.

[8] T. Avgerinos, A. Rebert, and D. Brumley, "Methods and systems for automatically testing software," Apr. 11 2017. US Patent 9,619,375.

[9] D. Marijan, M. Liaaen, and S. Sen, "Devops improvements for reduced cycle times with integrated test optimizations for continuous integration," in *2018 IEEE 42nd annual computer software and applications conference (COMPSAC)*, vol. 1, pp. 22–27, IEEE, 2018.

[10] A. Shi, P. Zhao, and D. Marinov, "Understanding and improving regression test selection in continuous integration," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 228–238, IEEE, 2019.

[11] C. Kaner, "Improving the maintainability of automated test suites," in *International Software Quality Week*, 1997.

[12] Y. Zhu, E. Shihab, and P. C. Rigby, "Test re-prioritization in continuous testing environments," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 69–79, IEEE, 2018.

[13] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, pp. 1–43, 2022.

[14] A. Haghighatkhah, M. Mäntylä, M. Oivo, and P. Kuvaja, "Test prioritization in continuous integration environments," *Journal of Systems and Software*, vol. 146, pp. 80–98, 2018.

[15] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on software engineering*, vol. 33, no. 4, pp. 225–237, 2007.

[16] S. Mirarab and L. Tahvildari, "An empirical study on bayesian network-based approach for test case prioritization," in *2008 1st International Conference on Software Testing, Verification, and Validation*, pp. 278–287, IEEE, 2008.

[17] A. Sharif, D. Marijan, and M. Liaaen, "Deeporder: Deep learning for test case prioritization in continuous integration testing,"

[18] J. Kim, H. Jeong, and E. Lee, "Failure history data-based test case prioritization for effective regression test," in *Proceedings of the Symposium on Applied Computing*, pp. 1409–1415, 2017.

[19] S. Omri, *Quality-Aware Learning to Prioritize Test Cases.* PhD thesis, Karlsruhe Institute of Technology, Germany, 2022.

[20] H. Li, "Learning to rank for information retrieval and natural language processing," *Synthesis lectures on human language technologies*, vol. 7, no. 3, pp. 1–121, 2014.

[21] P. A. Gutiérrez, M. Perez-Ortiz, J. Sanchez-Monedero, F. Fernandez-Navarro, and C. Hervas-Martinez, "Ordinal regression methods: survey and experimental study," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 127–146, 2015.

[22] P. McCullagh, "Regression models for ordinal data," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 42, no. 2, pp. 109–127, 1980.

[23] A. Shashua and A. Levin, "Ranking with large margin principle: Two approaches," *Advances in neural information processing systems*, vol. 15, 2002.

[24] T. G. Nick and K. M. Campbell, "Logistic regression," *Topics in biostatistics*, pp. 273–301, 2007.

[25] J. Chen, Y. Lou, L. Zhang, J. Zhou, X. Wang, D. Hao, and L. Zhang, "Optimizing test prioritization via test distribution analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, (New York, NY, USA), p. 656–667, Association for Computing Machinery, 2018.

[26] K. Das, J. Jiang, and J. Rao, "Mean squared error of empirical predictor," *The Annals of Statistics*, vol. 32, no. 2, pp. 818–840, 2004.

[27] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.

[28] D. Misra, "Mish: A self regularized non-monotonic neural activation function," *arXiv preprint arXiv:1908.08681*, vol. 4, no. 2, pp. 10–48550, 2019.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[31] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[32] L. Torgo, R. P. Ribeiro, B. Pfahringer, and P. Branco, "Smote for regression," in *Portuguese conference on artificial intelligence*, pp. 378–389, Springer, 2013.

[33] Á. Beszédes, T. Gergely, L. Schrettner, J. Jász, L. Langó, and T. Gyimóthy, "Code coverage-based regression test selection and prioritization in webkit," in *2012 28th IEEE international conference on software maintenance (ICSM)*, pp. 46–55, IEEE, 2012.

Abstract

Early Defects Detection using Test Cases Prioritization in Iterative Software Development Process

Jaesung HWANG
Graduate School of Practical Engineering
Seoul National University

현대의산업 현장에서소프트웨어를 개발하는 데에는 시간이 갈수 록 큰규모의 지속적 통합(Continuous Integration) 환경을 적용하고 있다. 이러한 환경은 높은 개발 생산성을 보장하는 것이지만, 동시에 테스트 수 행하는 양에 크게 의존적이다. 가능한한 빨리 시험 차수(Test Cycle)을 수행하여 개발자에게결과를피드백 해주기 위한 환경이지만, 점차적으 로 시험사례(Test Case)가 많아지면, 전체 소프트웨어 개발공정은느리고 비효율적이 된다.

소프트웨어 시험의 수행 효율성을 개선하기 위해서본 연구에서는 반복 수행되어 축적된 시험 수행 기록을 기반으로기계 학습(Machine Learning)기법을이용하여 학습하여 다음 시험 수행에서 실패할 가능성이 높은 시험사례를 도출한다. 우선 수행할 수 있는 시험사례우선순위화(Test Case prioritization)[1]방법을 적용하여, 조기에 결함을 탐지하여 개발자들에게개선해야 할 실패한 시험사례를 좀 더 빠른피드백으로 제공, 즉 공정 수행을 앞당기는 효과(Shift-Left)를 통해 소프트웨어 개발공정의 효

율성을 높히고자하였다. 시험사례우선순위화가 잘 적용되었는지는 실패한 시험사례의 가중 평균값(Average Percentages of Fail Detection, APFD) 값을측정하여 확인하였다.

현재 사용되는 산업 현장의 시험사례를 이용하여 실험을 진행하였다. 전통적으로 사용되었던 휴리스틱 기법의 시험사례우선순위 전략에 비해 본 연구에서 제시하는 시험수행결과기록을 학습한 기계학습의 시험사례우선순위를 APFD값으로 측정한 결과, 기존 기법에 비해 약 1.7배가 개선된 것을 확인하였다

Keywords : Fault detection, Test case prioritization, Regression testing, Machine Learning

Student Number : 2020-28114