CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Implementation and Evaluation of Automatic Prioritization for Continuous Integration Test Cases

Master's thesis in Software Engineering

Jiayu Hu

Yiqun Li

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Master's thesis 2016:06

Implementation and Evaluation of Automatic Prioritization for Continuous Integration Test Cases

JIAYU HU

YIQUN LI

Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden 2016

Implementation and Evaluation of Automatic Prioritization for Continuous Integration Test Cases
JIAYU HU
YIQUN LI

Academic Supervisor: Eric Knauss, Computer Science and Engineering
Industrial Supervisor: Gustaf Johansson, Ericsson AB
Examiner: Regina Hebig, Computer Science and Engineering

Master's Thesis 2016:06
Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
SE-412 96 Gothenburg | SE-405 30 Gothenburg
Telephone +46 31 772 1000 | +46 31 786 0000

Implementation and Evaluation of Automatic Prioritization for Continuous Integration Test Cases
JIAYU HU
YIQUN LI
Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg

Abstract

Nowadays, continuous integration is widely accepted and implemented by most industrial companies due to the benefits of adding new functions and accelerating the delivery of new products. But at the same time, frequent integration brings challenges, especially to large software companies. One challenge in the case company is to prioritize the test cases that are most likely to fail first in order to minimize the feedback loop from change to test result. In order to guarantee the product quality, a significant number of tests shall be executed after every integration, which has high demands on resources and thus has are associated with a high cost and time consumption. This thesis proposes a method based on mining correlations between historical data of test results and modified files. The Matthews Correlation Coefficient (MCC) and a scoring curves refers to Average Percentage of Faults Detected (APFD) are used to evaluate results and determine correlations. We aim to create an automatic way to provide a list of test cases that prioritized by their probability to fail, and further help the case company to shorten their feedback loop and time to market.

Keywords: continuous integration, prioritization, test cases, changed files, data miming, failure probability

Acknowledgements

We wish to thank our academic supervisor Eric Knauss for his valuable suggestions and enthusiastic feedback. We would also like to thank Ericsson, especially to our industry supervisor Gustaf Johansson, for giving us the opportunity to conduct this study and generously provided historical data sets. Special thanks to our examiner Regina Hebig for constructive feedback and active cooperation.

Jiayu Hu & Yiqun Li, Gothenburg, June 2016

Contents

1
Introduction

In recent years, software companies are putting more and more efforts towards continuous integration in order to deliver high quality software to their customers as fast as possible. Continuous integration (CI), as one of the practices of agile development, demands that members of the development team integrate their work frequently [1] [2]. This practice supports the advocation of agile development: effectively respond to change [3].

However, when companies try to adopt CI, many of them are struck by the time limit. For instance, companies find that the feedback loop from regression tests are too long [4]. To be more specific, if the developers integrate their work every day, while not receiving feedback response until two days later. In this case, the whole developing process would be slowed down. For increasing the efficiency, it is of great significance for developers to find a method that solves above mentioned issue while not decreasing the quality of software testing.

One of the most effective ways is prioritizing critical test suits [30]. The prioritization helps developers to find out the most critical test cases from test suits, which dramatically optimizes the feedback time from the selected test suites. The study of Felderer had also confirmed the feasibility of this approach. He did several case studies to prove that the process could be used in real world for risk-based testing and used a cache to monitor fault-prone files and recommends test cases to rerun to cover updated files [5] [6].

In this study, we aim to create and evaluate a methodology that automatically prioritizes test cases based on mining the historical data of test results and modified files. One academic and one industrial supervisor supervised this thesis.

Section 2 provides the background of this study and a review of related literature; Section 3 describes the purpose and approach of this study; Section 4 shows the study and analysis processes; Section 5 reports and explains results; Section 6 contains the discussion and Section 7 gives the final conclusion.

2
Background and Related Work

This section first provides the background of this study, and gives reviews of related literature to test case prioritization.

2.1 Background

The background of this study is firstly introduced in this subsection.

2.1.1 Case Company

In this study, the case company is Ericsson. It is a world leader in the rapidly changing environment of communications technology-providing equipment, software and services to enable transformation through mobility. Some 40 percent of global mobile traffic runs through networks they have supplied. More than one billion subscribers around the world rely every day on networks that they manage. With more than 39,000 granted patents, they have one of the industry's strongest intellectual property rights portfolios. Currently, the company is putting efforts on CI and trying to improve their processes. And we are working with one of their testing teams, gathering and analyzing the data of actual industrial test results.

2.1.2 Continuous Integration and Testing

Generally, Continuous Integration (CI) is a set of principles that apply to the daily workflow of development teams, which recommends that all code must be kept in a repository. In its life cycle, when a developer commits his or her changes to the repository, system would pick up changes and verify them. At the same time, change history is recorded for version control [1] [19]. Such frequent integration requires people to increase test coverage for each new change, while also keep the tests fast.

Testing aids and simplifies the detection of software failures and further discovers and corrects the defects. Testing does not work to guarantee the functions of a product under all conditions but finds the problems under specific conditions [21]. Functional testing is widely used in industrial development, which is a quality assurance (QA) process and usually describes what the system does [22]. In black

box testing, functions are tested by putting them as the input and examining the output, without considering internal program structure [23]. In the case company, they have test suits that can simulate different situations and give accurate feedback. Regression testing is typically the largest test effort in industrial software includes functional testing [24]. It checks numerous details from prior product features, and tests new designs of new software to ensure that prior functionality are still supported.

2.1.3 Data Mining

Data Mining, also popularly known as Knowledge Discovery in Databases (KDD), refers to the nontrivial extraction of implicit, previously unknown and potentially useful information from data in databases [8]. It is an automatic or more usually semiautomatic process [9]. There are four basic different styles of learning that appear in data mining applications. In classification learning, the learning scheme is presented with a set of classified examples from which it is expected to learn a way of classifying unseen examples. In association learning, any association among features is sought, not just ones that predict a particular class value. In clustering, groups of examples that belong together are sought. In numeric prediction, the outcome to be predicted is not a discrete class but a numeric quantity [10].

2.2 Related Work

Reviews of literature that related to test case prioritization are given in this subsection.

2.2.1 Mining Software Repositories (MSR)

In the continuous integration developments, a large amount of information about the change history and the results of executed test cases can be produced and needs to be stored. The case company Ericsson does have the repositories that stored these version control data. By extracting from that data, we are able to obtain the adequate materials that we need for our analysis.

The methodology of Mining Software Repositories (MSR) helps to analyze the big data from software repositories [17]. And Ahmed [18] also demonstrated that it is valuable to mining software repositories for assisting managers and developers in performing a variety of software development, maintenance, and management activities. The MSR processes contain two main phases, which start with preparing the data and followed by the analysis phase. We applied both two phases in this thesis work, and put the main focus on data analysis. The reason is that we did not have direct access to Ericsson's databases, instead, we asked the help of industrial supervisor to extract related historical data from repositories and processed some transformation which exclude noises before we performed analysis.

The original data sets provided were filtered CSV files, and we processed a further transformation by using our own scripts which transforms CSV files into ARFF files and make them usable in Weka for later analysis.

2.2.2 Test Case Prioritization

Saha et al [20] came up with a method that prioritizes test cases, so the higher priority test cases would have a higher likelihood of locating bugs. They combined the statistical methods with the scripts written in programming languages for addressing the problem of regression test prioritization. Their approaches bring us good inspiration and starting point.

At the same time, this study is based on the risk of test cases execution history, and we focused on prioritizing test cases. According to Felderer and Schieferdecker's [11] taxonomy of risk-based testing, our study is one of the risk-based testing specifics in test processes.

Wang [7] did a similar study in another case company and analyzed data sets which had relatively small sizes. In her study, she aimed at defining out fitness functions based on source code changes. She looked into changes of software artifacts and correlating those with test failures, and then took into account not only test quality but also hidden technical dependencies in the software. In this study, we extend her method and prioritize test cases based on changed files. The two algorithms that Wang applied are NaiveBayes and RandomForest, where the latter is a notion of the general technique of random decision forests [12] [13] and an ensemble learning method that can be applied to process classification, regression and other tasks.

The RandomForest is operated by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees [14]. This algorithm can provide many benefits. It can handle a large number of input variables, while producing highly accurate classifier for a variety of materials. And for incomplete or unbalanced classification data sets, it can balance errors and generate precise predictions. Both from the actual testing results and several useful literatures [25] [26] [27], we found out that the RandomForest is arguably be the most appropriate and efficient algorithm in this study. Therefore, even if it has a few shortcomings, such as high hardware requirements and time consumption, we decided to focus on using RandomForest for our analysis and discuss trade-offs with the company.

Based on the above, we would analyze received historical data by using RandomForest algorithm, and then prioritize test cases based on our analysis results. Generally, if the change of a file made some test cases to fail recurrently, those test cases would have higher priorities for executing when the same file was changed.

2.2.3 Average Percentage of Faults Detected (APFD)

Gregg et al [28] presented the methodology of Average Percentage of Faults Detected in their report. It is a metric designed for measuring the rate of fault detection for test suite execution. They described several techniques for using test execution information to prioritize test cases for regression testing, and reported their outcomes of different experiments. Wherein its calculation is a great reference for our scoring system in the late stage of study.

3
Research Methods

This section focuses on describing the research purpose, questions and methodologies of this study. Several contents of data analysis are also included to introduce scientific research methods and processes.

3.1 Research Purpose

The main purpose of this thesis is to study data mining methodology and analyze data sets of historical testing results from the case company. We aim to create an appropriate method that automatically prioritizes the test cases. In the later stages, the outcomes are going to be tested and evaluated by real industry data to guarantee their feasibility, applicability and practicality. For the case company, the final result shall be able to help shorten the time-to-market while also guaranteeing the software quality.

3.2 Research Questions

In order to achieve the above study purpose, we set up three research questions (RQ) that help us to complete this study step by step:

RQ1: To what extent is the available data at the case company suitable for prediction of test case failure?

With RQ1, we check if the data is suitable for automatic classification and whether we can compute the probability of failure for individual tests based on MCC measure (see Section 3.3).

RQ2: To what extent can APCIT be used to prioritize the test cases, are there any existing restrictions?

With RQ2, we check to what extent the failure probabilities of the individual tests can be used to prioritize functional test cases at the case company. For this, we create baselines and use ranking referring APFD metric to evaluate the prioritization.

RQ3: To what extent can APCIT bring benefits to the case company?

With RQ3, we enter a critical discussion on how the results for RQ1-2 can provide value to the case company, based on semi-structured interviews with related development groups in the case company.

3.3 Research Methodology

In Continuous Integration and automatic integration tests, the changes on one file or similar files could cause the same error and make the same test case fail. The repository of the case company can record file updates of each integration, as well as test results of different testing jobs into the historical data. By implementing data mining on the historical data, we aim to find out the correlation of changed files and test cases.

In order to achieve the research purpose, we combined the statistical methods with data mining knowledge to guide our study. We established and verified the results of prioritizing test cases based on historical data that contains file changes and results of automatic integration tests. For the final results, the method shall take a set of test results and a set of file changes as input. The method shall automatically prioritize test cases and give the highest prioritization to those tests that are the most likely to fail given one set of changed files.

3.3.1 Interviews

This study process started from shallow to deep, gradually progressing. In order to understand how automatic prioritization of functional integration tests could help the case company, we conducted semi-structured interviews with the case company and aimed to probe what is the use case of prioritization. To be more specific, we were trying to understand why the company needs prioritization and what kinds of results were desired. Interviews were handled during the weekly supervision meetings where both supervisors participated. And up to two more people from the same development department also joined several meetings. The presence of them helped to guide us in obtaining sufficient as well as accurate information that we needed for our thesis work.

Since those interviews are semi-structured, specific questions and contents were not written down nor recorded, instead, we had discussions and made notes. By comparing and summarizing notes of two researchers after every meeting, we made a maximum effort to ensure that there was no misunderstanding or loss of information.

After having direct dialogues with industry, we learned that all test cases shall be run eventually regardless of the prioritization results in the case company. For not running all test cases every iteration, our method is supposed to help to find

which test cases are most likely to fail for each job. By using our solution, developers would be able to gain a maximum guarantee of quality by selecting test cases from the sorted test cases when they were facing time-critical deliveries or needing fast feedback.

3.3.2 Literature Reviews

After the in-depth understanding of use cases, we started to study helpful and valuable literature that we found. Due to this study is experimental and focusing on actual operation, we did not intend to do systematic literature reviews.

The purpose of reading literature is to gain domain knowledge, as well as comparing different data mining methods and algorithms. The selection of papers was focused on literature that published in recent years and new reports given in related conferences.

3.4 Data Obtaining and Processing

After correct understanding of the use cases and learned necessary domain knowledge, we began to analyze data sets provided by the case company. The processing started with data sets that with rather small size, this helps us to understand data structure and then created our own data processing scripts. At the same time, this strategy also provides convenience for testing the scripts. In this study, we obtained and processed four kinds of data sets:

Table 3.1: Four kinds of data sets.

Name of data set(s)	Description
Initial data sets	Two sets, one lists details of test results based on each job and another based on test cases.
Small data set 16_90*pct*	Combined job-based and test-case-based historical data, contains 168 jobs and 299 test cases. These test results are from 2016, and only jobs with over 90% pass rate are included. It is a sub-set of large date set.
Large data set 14_90*pct*	Combined job-based and test-case-based historical data, contains 5540 jobs and 683 test cases. These test results are from 2014, and only jobs with over 90% pass rate are included.
Large data set 14_*all*	Combined job-based and test-case-based historical data, contains 5708 jobs and 983 test cases. These test results are from 2014, all jobs are included.

Processed results were discussed immediately and directly with the case company, we determine whether the final outputs meet the actual demands. After the case company recognized processed results of small data size, we also analyzed larger sets. In the end, we confirmed feasibility, applicability and practicality of our method by implementing and testing it to the data sets with different sizes and features.

3.4.1 Data Mining Tool

In this study, we used the Waikato Environment for Knowledge Analysis (Weka) as our analysis engine. Weka is a public data mining workbench that assembles a large number of machine learning algorithms that can take on the task of data mining. Weka also includes data preprocessing, classification, regression, clustering, association rules and a visualized interactive interface [31]. It is worth mentioning that Weka contains lots of useful algorithms including RandomForest analysis, which is beneficial to our case.

Like most of other spreadsheet or data analysis software, Weka can only deal with the data collections that are in two-dimensional forms. Weka format for analyzing and storing data is in ARFF file, which is an ASCII text file and is going to be introduced in Section 3.4.3.

3.4.2 Data Structure

The original data sets are CSV files that include timestamp, version, parent, build job, scope, executed test cases and corresponding test results.

The jobs are test events, where the timestamp indicates the operation time for each job. Several jobs could be run at the same timestamp; the scopes are applied test suits. Different scopes could be used at the same time overlap, and the same test cases might have different names. In our study, test cases with other names were treated as separate objects; the columns start with string of characters show test results specific cases, while the columns start with string “file” or “test” indicate the changed files.

An example of the data sets we received is shown in Figure 3.1.

timestamp	version	parent	build	job	scope	apr_12597	cdr_23986	clci_1421	clci_1421	file_1015	file_1015	file_101
Mon Feb 01 19:25:21	a0480fa	08730d8	B172160201	1480468	scope_B2	PASS	PASS	PASS	PASS			
Wed Feb 03 14:33:25	9928dfd	8ad39b0	B203160203	1487944	scope_B2	PASS	PASS	PASS	PASS			
Thu Jan 28 22:26:27	7cf26954	b384e7d	B133160128	1470541	scope_B2	PASS	PASS	PASS	PASS			
Wed Feb 03 03:18:46	7e8c8c4	822c0ac	B192160203	1485455	scope_B2	PASS	PASS	PASS	PASS		CHANGED	
Fri Jan 22 17:29:42	2ae3933	d357212	B62160122	1452067	scope_B2	PASS	PASS	PASS	PASS			
Fri Jan 29 08:21:26	55c3d17	6e058c5	B139160129	1471855	scope_B2	PASS	PASS	PASS	PASS			
Mon Feb 01 10:37:52	e31d09d	1dbabe0	B164160201	1478516	scope_B2	PASS	ERROR	PASS	PASS			
Mon Jan 25 12:33:32	9fe8af4	da279c9	B76160125	1456748	scope_B2	PASS	PASS	PASS	PASS			
Wed Jan 27 17:54:03	b4ba2ec	f4070f4	B113160127	1466046	scope_B2	PASS	PASS	PASS	PASS			
Fri Jan 29 17:34:18	d7199c8	6852704	B87616012	1473974	scope_B2	PASS	PASS	PASS	PASS			
Tue Jan 26 05:19:53	fffecff	284faa3	B86160126	1459368	scope_B2	PASS	PASS	PASS	PASS			
Wed Jan 27 11:56:08	108d4a5	799f26b	B108160127	1464730	scope_B2	PASS	PASS	PASS	PASS			
Fri Jan 29 01:18:31	be3ac13	9dd6a71	B134160129	1470784	scope_B2	PASS	PASS	PASS	PASS			
Wed Jan 27 12:33:51	6dbeb05	eebbd10	B109160127	1464834	scope_B2	PASS	PASS	PASS	PASS			
Wed Jan 27 09:53:19	98f97a0	1d76b48	B106160127	1464213	scope_B2	PASS	PASS	PASS	PASS		CHANGED	
Mon Feb 01 21:39:38	544049c	226ed96	B174160201	1480912	scope_B2	PASS	PASS	PASS	PASS			
Sun Jan 31 00:46:50	99a55e6	390abaf	B154160131	1475821	scope_B2	PASS	ERROR	PASS	PASS			
Thu Jan 28 03:46:29	aaa3434	f61ba24	B115160128	1466874	scope_B2	PASS	PASS	PASS	PASS			
Wed Jan 27 04:32:24	c49956c	ee04a65	B101160127	1463160	scope_B2	PASS	PASS	PASS	PASS			
Mon Jan 25 11:50:56	7d5ede6	24c6383	B75160125	1456629	scope_B2	PASS	FAIL	PASS	PASS			
Mon Feb 01 08:43:05	54f8965	53a8e98	B162160201	1478086	scope_B2	PASS	ERROR	PASS	PASS			
Mon Feb 01 02:36:07	83e33de	803d1a0	B158160201	1477202	scope_B2	PASS	ERROR	PASS	PASS			

Figure 3.1: Original CSV file.

In order to analyze the relevance of files and test cases, we created a java script that removes redundant data, unifies test results to three states (PASS, FAIL_or_ERROR, NONE) and transforms the original files so that the data mining tool can analyse them.

3.4.3 Data Transformation

The original data sets were transformed, which were in CSV format, into ARFF files. The Attribute-Relation File Format (ARFF) is ASCII text files, it includes the relationship declaration and the property declaration of attributes. The relationship declarations are defined from the first active line of ARFF file with the format: @relation <relation-name>, where the <Relation-name> is a string. If the string contains spaces, it must be quoted; the property declarations are defined after all relationships are declared and are represented by the declarations that start with “@attribute” statement. These statements define the names and types for attributes. As shown in Figure 3.2, the orders of the declaration statements are very important, because they define positions of attributes in the data sets. The recognition of ARFF files are branches, and thus arbitrary line breaks are not allowed. Blank lines (or all blank lines) and the lines start with "%" (as comments) will be ignored by Weka.

Figure 3.2 shows one of the ARFF files generated from the original data.

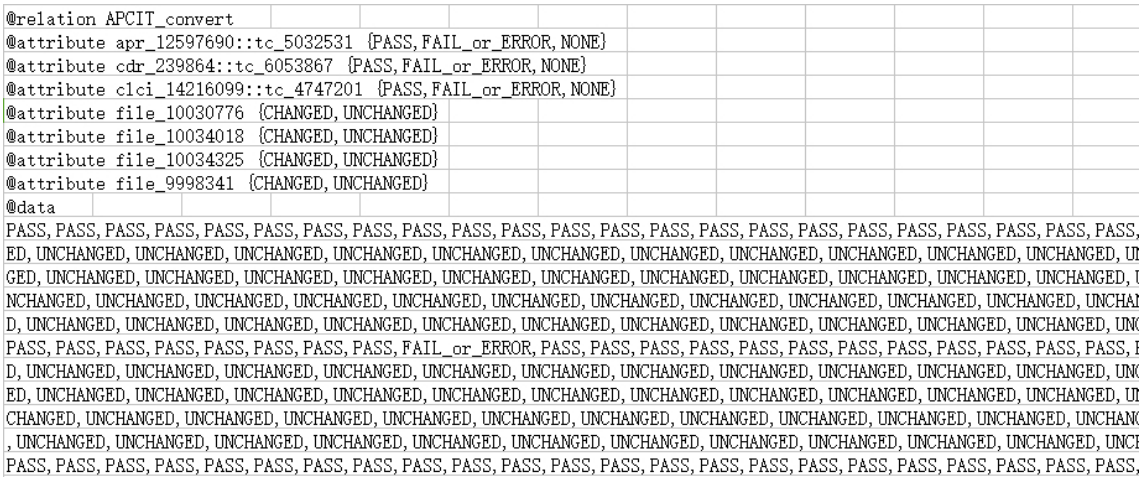

Figure 3.2: Transformed ARFF file.

3.5 Measurements

Models can be built by the data mining tool from the transformed files. In order to measure the quality of these models, the confusion matrix and some measurements need to be introduced.

3.5.1 Confusion Matrix

The confusion matrix contains the actual and the predicted classifications, and further help to judge right and wrong. Since we have three states in our data, there are three class classifiers in our confusion matrix (Figure 3.3). The columns represent the instances in a predicted class, and the rows represent the instances in an actual class.

		Predicted		
		a	b	c
Actual	a	aa	ab	ac
	b	ba	bb	bc
	c	ca	cb	cc

Figure 3.3: The Prioritization Results Example.

In this confusion matrix, the letter “a”, “b” and “c” represent “PASS”, “FAIL/ERROR” and “NONE” respectively. In more detail, "aa", "bb" and "cc" are the true

positive (TP) values; the false negative (FN) values are the sum of corresponding rows but exclude TP, for example, "ab + ac" is the FN of "a"; the true negative (TN) value of a class can be calculated by summing all columns and rows but exclude its own', in our case, "bb + bc + cb + cc" is the TN of "a"; and the false positive (FP) values are the sum of corresponding column but exclude TP, for instance, "ba + ca" is the FP of "a".

3.5.2 Measurement Definitions

We used four retrieval measures in our study, which are recall, precision, f-measure and Matthews correlation coefficient (MCC). However, the first three are just reflection of statistical characteristics of data, we decided to focus on MCC. The results of these measures are all based on four categories:

Table 3.2: The four categories.

True Positives (TP)	The test cases that are predicted to fail and actually failed according to the ground truth; it is the true correct rate.
False Positives (FP)	The test cases that are predicted to fail but actually not failed according to the ground truth; it is the false alarm rate.
True Negatives (TN)	The test cases that are not predicted to fail and actually not failed according to ground truth; it is the false correct rate.
False Negatives (FN)	The test cases that are not predicted to fail and actually failed according to the ground truth; it is the false negative rate.

MCC is a measurement that is used in machine learning as a measure of the quality of binary (two-class) classifications [15]. Since it includes true and false positives and negatives, it is actually calculating the correlation coefficients between the observed and predicted classifications. Thus it is a balanced measure that is used even if the classes are of very different sizes. In our case, we have several data sets with very different sizes, and thus MCC can give the most reliable guarantee to the correctness of our final results. After considering, we decided to focus on using MCC to verify our final results. The MCC can be calculated automatically by Weka or with the following equation:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The return value of MCC is between -1 and +1. Rumsey [16] determines the meaning of these coefficients as following:

Table 3.3: The meaning of different MCC coefficients.

MCC value	Description
+1.0	Indicates a perfect prediction, which tests that are recommended by the recommender system and failed according to the ground truth.
$+0.70 \; +0.99$	Indicates a very strong positive relationship.
$+0.40 \; +0.69$	Indicates strong positive relationship.
$+0.20 \; +0.39$	Indicates moderate positive relationship
$+0.19 \; -0.19$	Indicates no or negligible relationship.
$-0.20 \; -0.29$	Indicates weak negative relationship.
$-0.30 \; -0.39$	Indicates moderate negative relationship.
$-0.40 \; -0.69$	Indicates strong negative relationship.
$-0.70 \; -0.99$	Indicates very strong negative relationship.
−1.0	Indicates total disagreement between prediction and observation.

Generally, the higher the MCC values, the better the prediction. If a value is between -0.19 to +0.19, it means the result is random guessing and not reliable nor useful. Those MCC values reflect the quality of classification by using classifier.

4
Data Analysis

This section describes the details of data processing on different sets. The analysis processes of data were implemented step by step. In order to have a comprehensive analysis of the characteristics of data, we started with familiarizing us with the initial data sets (Section 4.1), while also creating our own analyzing scripts. The scripts were first tested with the data set with small sizes (Section 4.2) for refinement and validation, and then were applied to the large data set (Section 4.3) for obtaining the final results.

4.1 Familiarizing with Initial Data Sets

By discussing with the case company, we learned that the case company has started applying CI in its development process and it is trying to improve related workflow. In order to develop software with high quality, a large number of functional test cases are constantly added and modified during the feature development phase. Thus they are faced with the problem that running all test cases creates long feedback loops, which directly increases the time-to-market and reduces the flexibility when responding to frequently changed requirements.

In the case company, functional test cases are testing the system from a black box perspective where people supplies the inputs and validates the outputs, details of execution are hidden. These test cases simulate different environments and situations. Test cases are grouped into test suites and into different scopes.

As introduced in Section 3.4.2, the original data set contains historical test case results, and it is in CSV format. In the beginning, the data we acquired were initial data sets (introduced in Table 3.1). By parsing those data sets, we understood the data structure and captured the basic characteristics of the test results.

The Figure 4.1 visualizes and summaries the parsed job-based outcome in the initial data set.

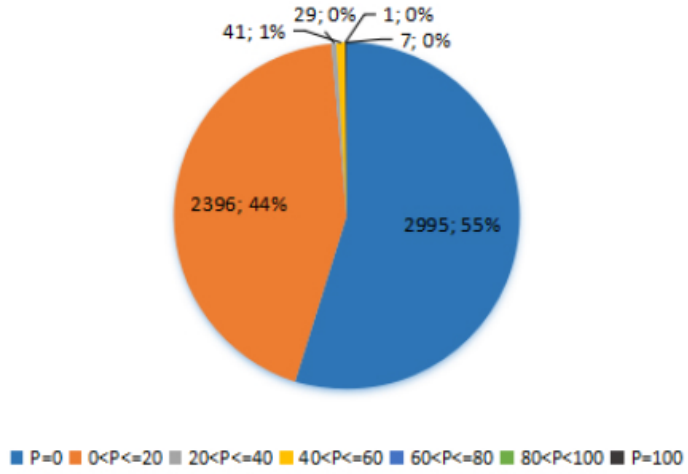

Figure 4.1: The percentage of failed test case distribution for each job in initial data set.

This pie chart shows the percentage of failure for each job. The letter "P" in the subscript represents percentage of failed test cases, and the numbers are percentages. For instance, in this data set, 55% jobs inclue zero failed test cases. More than half of jobs do not contain any failed test case, and the rest of jobs only have few failed tests.

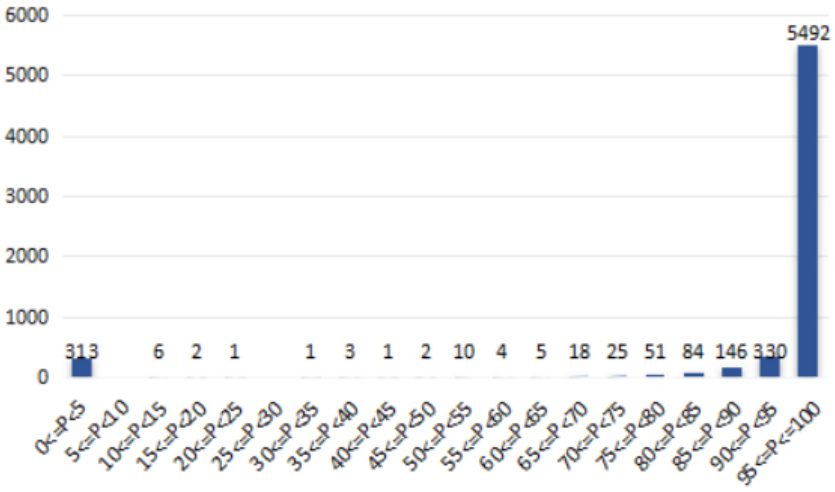

Figure 4.2: The pass rate distribution based on test cases in initial data set.

Similar characteristics can be found in Figure 4.2, which is the parsed test-case-based outcome in initial data set. Here, the x-axis shows the pass rate and the y-axis describes the number of test cases. Similar to Figure 4.1, the letter "P" in this subscript represents actual pass rate, and the numbers are percentages. The histogram not only shows that most of test cases never or just rarely failed, but also

indicates some test cases, 313 of them, actually failed very often. This is a finding that cannot be ignored. After discussing with the case company, we learned that one of the reasons could be those often-failed test cases themselves are problematic. Also, in some situations, test cases failed because of the inappropriate testing environments.

Due to the presence of above-mentioned situations, we recognize that those historical data sets are imbalanced and hard to analyze. Some algorithms cannot handle imbalanced data sets, but some can. As argued in Section 2.2.2, the algorithm of RandomForest is arguably be the best choice for our case. Currently, the present inputs seem excessively extreme where a few test cases have over 95% failure rate and most of test cases never fail.

4.2 Processing the Data Set with Small Size

After we had familiarized ourselves with the initial data sets, we received a new data set from the case company that combined job-based and test-case-based data - the small data set 16_90pct (introduced in Table 3.1). It helped us to analyze the correlation between test case results and changed files.

For analyzing the correlation between test cases and the failure rate, we decided to focus on studying two correspondences in the data: the number of test cases and failure rate based on each job, and the number of executions and failure rate based on each test case. By doing this, our objective was to understand better on how to do prioritization and evaluation according to the information we got.

In this study, heat maps are used to visualize the statistical results. All maps follow these principles:

a.Results are sorted according to the failure rate (including FAIL and ERROR) in descending order, thus the upper jobs or test cases are having higher failure rate.

b.The left column represents number of test cases executed in each job or the number of executions for each test case.

c.The right column shows corresponding failure rates of objects in the left column.

d.The colors in the rows indicate differences with the average value, the redder the higher than the average, and the bluer the lower the average.

e.The blue solid lines in the middle of both columns also represent differences with the average value, when it skewed to right means higher, while to left means lower.

f.The dotted lines in the middle of both columns represent average values.

The Figure 4.3 is the heat map for small data set 16_90pct, which shows the correlation between jobs and test cases. Several test cases were verified in same job, and the failure rates of jobs are indicated by colors.

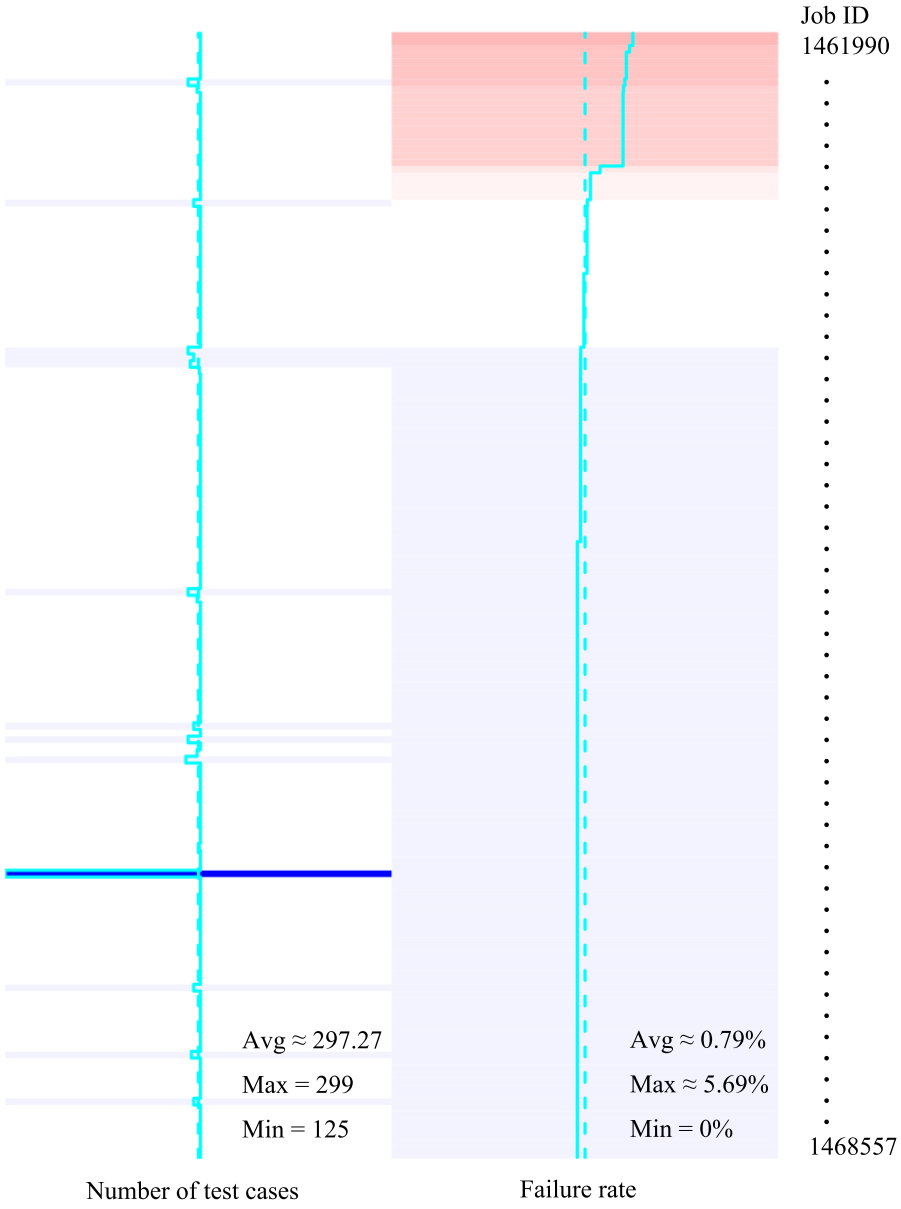

Figure 4.3: The number of test cases and failure rate based on each job in small data set 16_90pct.

From the above heat map we can see that solid line is relatively stable in the left column and the color does not change a lot, which means in this data set, there is a very even distribution of tests run per job. In the source data, 92 out of 168 jobs

do not contain failures, and it is shown in the picture that most of jobs have blue and white colors in the column of failure rate, where these colors means low or none differences with the average. Less than half of the jobs have high failure rates, which are indicated by color red. The same features are shown by lines as well, where in the column of failure rate, only part of solid line crosses the dotted line and goes right, while most of solid line stay in the left. Since the average value is only 0.79%, and minimum failure rate is 0%, we can say that most of jobs contains none or few failed test cases.

Besides observing the characteristics of jobs, it is very helpful to study the features of test cases. The Figure 4.4 shows the correspondence between number of execution for different test cases and their related failure rates for the same data set.

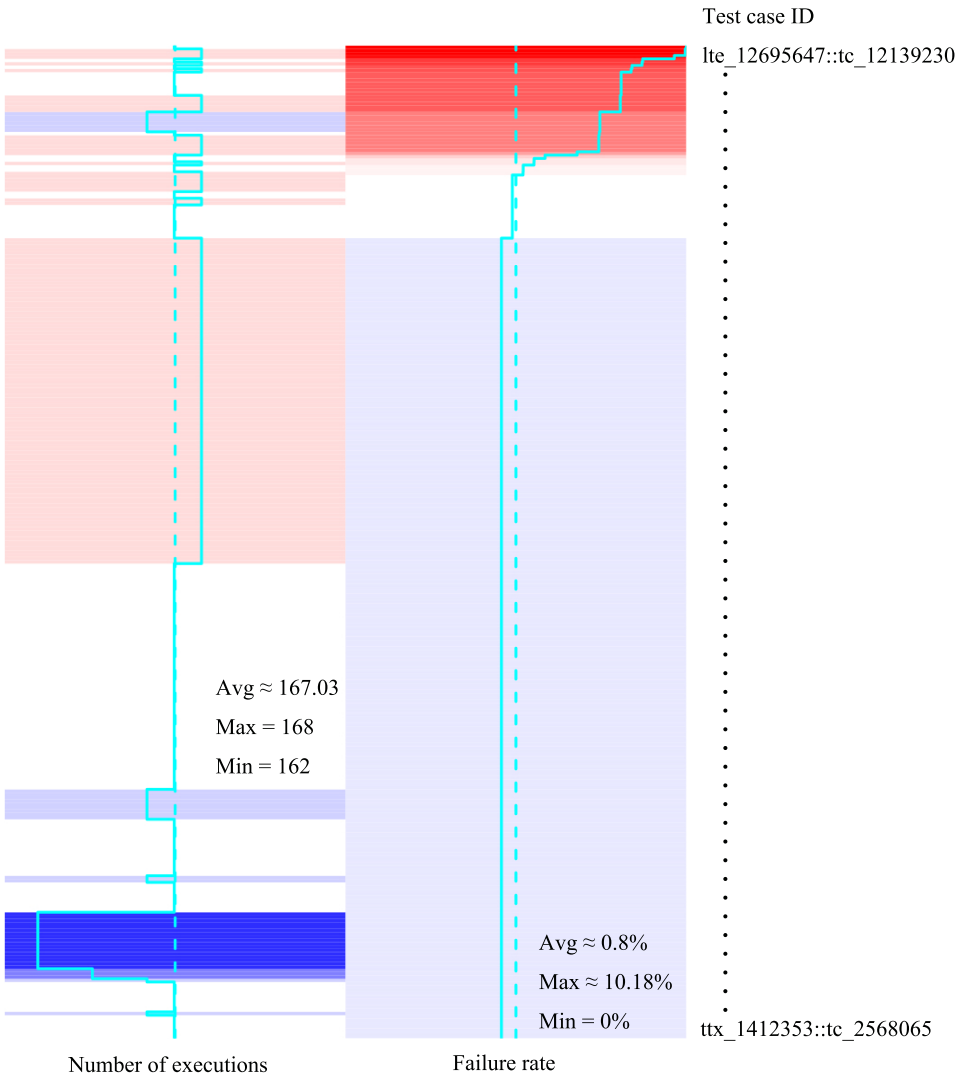

Figure 4.4: The number of executions and failure rate based on each test case in small data set 16_90pct.

As shown in above heat map, blue solid and dotted lines from the left column in-

dicates that test cases were executed multiple times. In the right column, most of these test cases have blue and white color for their failure rate, which means most of them rarely or never failed. Since the most of solid line stays in the left of dotted line, it means their failure rates are lower than average value, which is around 0.8%. According to the actual statistical results, around 20% of the test cases have failed records. A test case could have up to 10.18% failure rate. By comparing lines in both columns, we also found that some of the failed test cases are having relatively high number of executions.

After discussion, we agreed that if a test case had relatively high failure rate but its runs were far less than the average number, this test case would have low priority or be ignored. Such statistical results were not considered in this study, but they could be good references for company's test selection.

Besides creating the heat maps, we used Weka as the data mining tool and built up models with the classifier RandomForest, then successfully calculated MCC values. In order to create more reliable analysis results, 10-fold cross validation was applied to evaluate the models. Concretely, 90% of instances in the data set were used as training set while rest 10% were treated as test set. Those 10% were randomly chosen by the system, and after processing them, the program then will automatically alternate another 10% data as new test set until all instances were treated as test sets.

MCC value of three status were computed: MCC of Pass, MCC of Fail/Error and MCC of None. The analyzed outcomes are summarized and visualized in Figure 4.5.

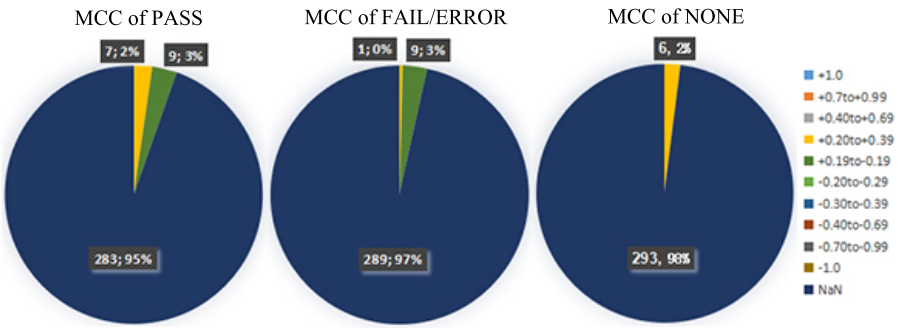

Figure 4.5: The MCC results of 10-fold cross validation for small data set 16_90pct.

From the above pie charts, it is very clear that most of calculations returned the value "NaN", which means those MCC values cannot be computed. We argue that the reason is that source data set is relatively imbalance. As shown in previous heat maps, most of the test cases never failed, as the result, the formula for calculating MCC would have value "0" as its denominator (formula is described in Section 3.5.1), and thus the calculation cannot be performed. Those kind of MCC results are not helpful for evaluating the quality of the built models (Table 3.3).

During this phase, we found most of MCC values were not calculable due to imbalanced data. We argue that such situation could due to two main reasons: the data in analyzed set is extremely imbalanced and the automatic cross validation of Weka is not suitable for our case.

For the first reason, since the source data includes historical information from real industry, the data sets could be unbalanced due to company's use case. We did not modify the source data, because filtering the data that we do not expect manually is not a proper way to do data mining; And for the second reason, it is because the processed data set includes information of the same test cases that were used in different time. When Weka randomly selects instances for cross validation, there are situations that is using future data, the test cases that applied in later jobs, as the training sets to predict the test cases that applied in earlier jobs, the past data. This kind of obviously unreasonable cases will seriously affected the forecast results and later prioritization. To solve this problem, we modified our scripts to manually select the training sets and the test sets based on calendar order.

In this small data set, job instances are listed in the order of their execution time, so the earlier data is in the forefront. Thus we selected 66% of data as training set, and the rest as test set. The new MCC values are presented in Figure 4.6.

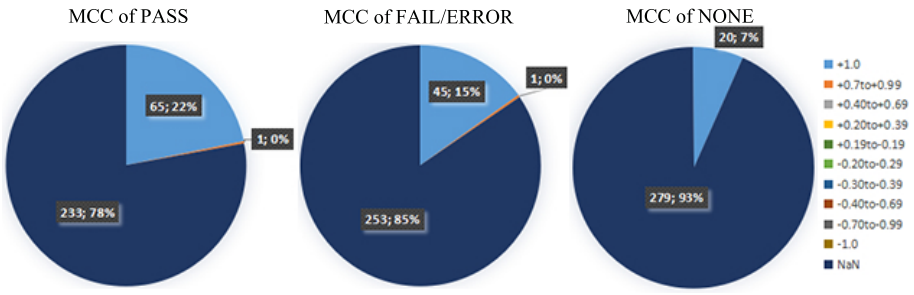

Figure 4.6: The improved MCC results for small data set 16_90pct.

By comparing with the outcomes in Figure 4.5, the improved MCC results from Figure 4.6 are more satisfying. As can be seen, the proportions of calculable MCC values are increased, and most of their values close or equal to one. The above chart shows 85% of MCC values in category of FAIL/ERROR cannot be calculated (indicates "NaN"), which consistent with the characteristics described in Figure 4.4, where roughly 85% of tests did indeed never fail. It proves that the new models are doing very good in predicting (Table 3.3). Due to above relatively satisfactory results, we are going to apply the same method when processing larger data set, which is the overall data.

4.3 Processing the Data Sets with Large Size

After receiving good results from processing small data set, we started to test our method with larger data sets (introduced in Table 3.1). It is worth mentioning that

the new data sets have better structure: the previous data sets only contained the changed files and related executed test cases, but the new also include information of changed test cases. We also started with visualizing overviews of the entire data sets by using heat maps.

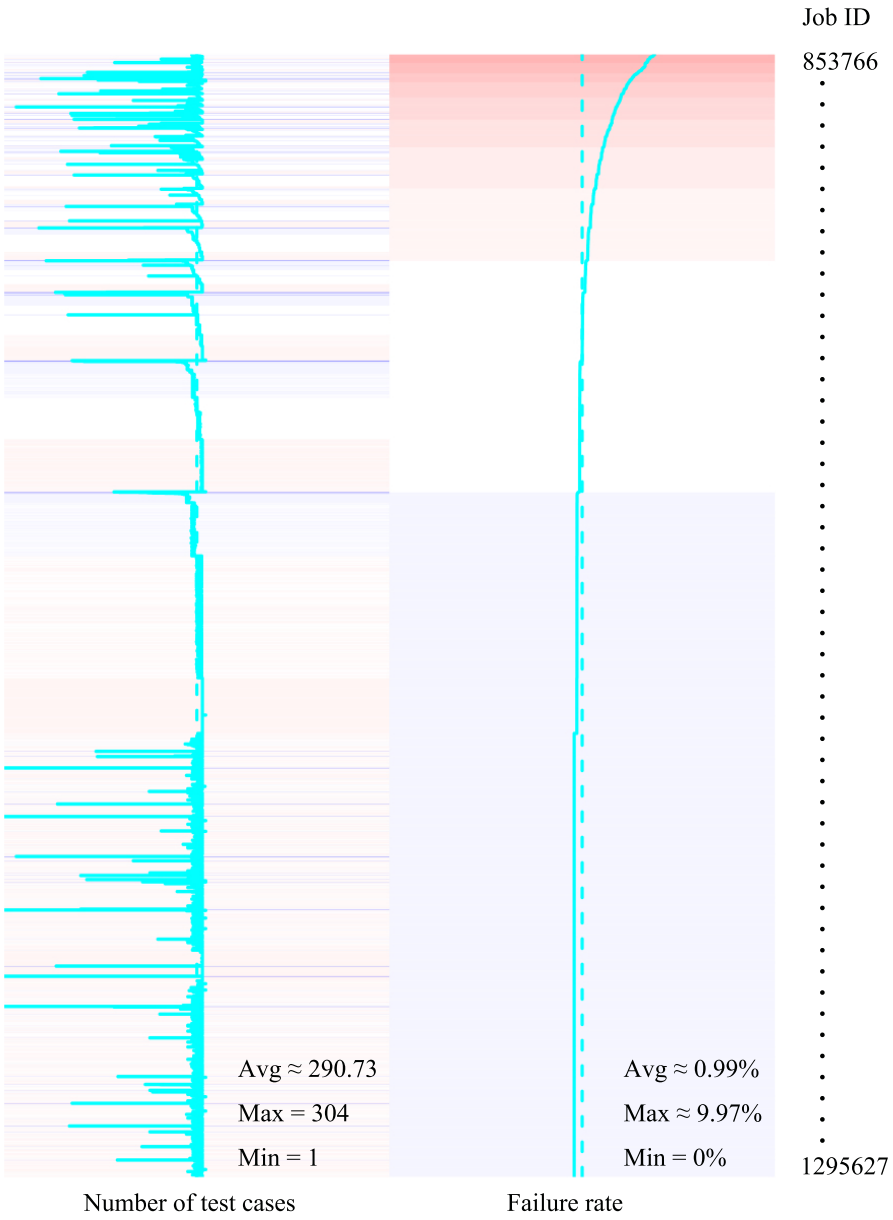

Figure 4.7: The number of test cases and failure rate based on each job in large data set 14_90pct.

The Figure 4.7 shows that every job includes at least one test case, while the average failure rate is below 1%. Similar to the small data set, only a small part of right column are in color red and most part of solid line stay to the left of the dotted line.

The Figure 4.8 shows the correspondence between jobs and test cases in this large data set.

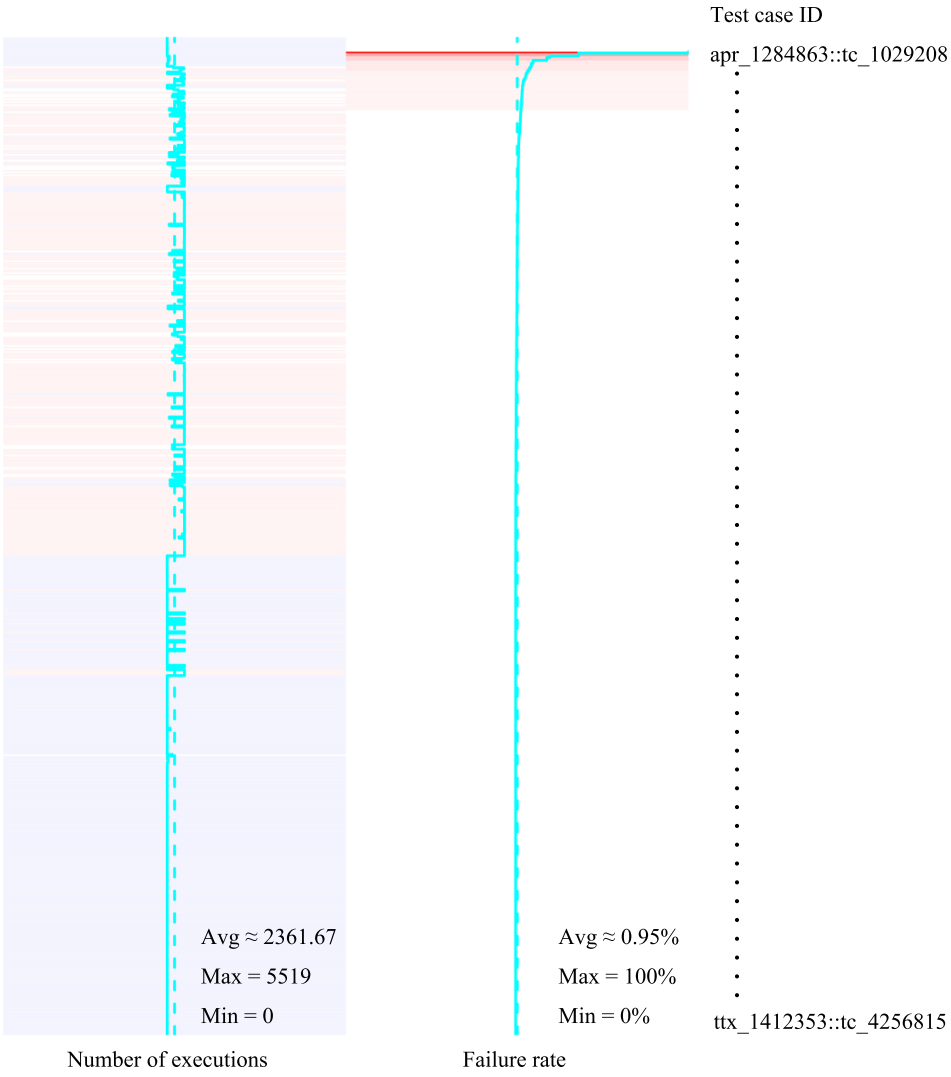

Figure 4.8: The number of executions and failure rate based on each test case based in large data set 14_90pct.

From above heat map, we learned that some test cases were never executed, and the average failure rate for each test case is around 0.95%. By observing the right column, we find there are test cases that always failed, as well as some with zero failure records. There is a blank area on the top of the right column, it is because some test cases were never executed. Those unexecuted test cases have zero execution and "NaN" failure rate. Again, according to the distribution of colors and blue lines, we can see that only a small part of these test cases failed.

Both Figure 4.7 and Figure 4.8 are heat maps of the data set that only contains test cases with over 90% pass rate. After observing them, we found their situations were similar to the small data set, in which most of the tests never or seldom failed.

For having more complete understanding of the characteristics of the new data sets and considering the industrial environment, we also generated heat maps for the data set which contains all test cases regardless of their pass rate, i.e. Large data set 14_all (introduced in Table 3.1).

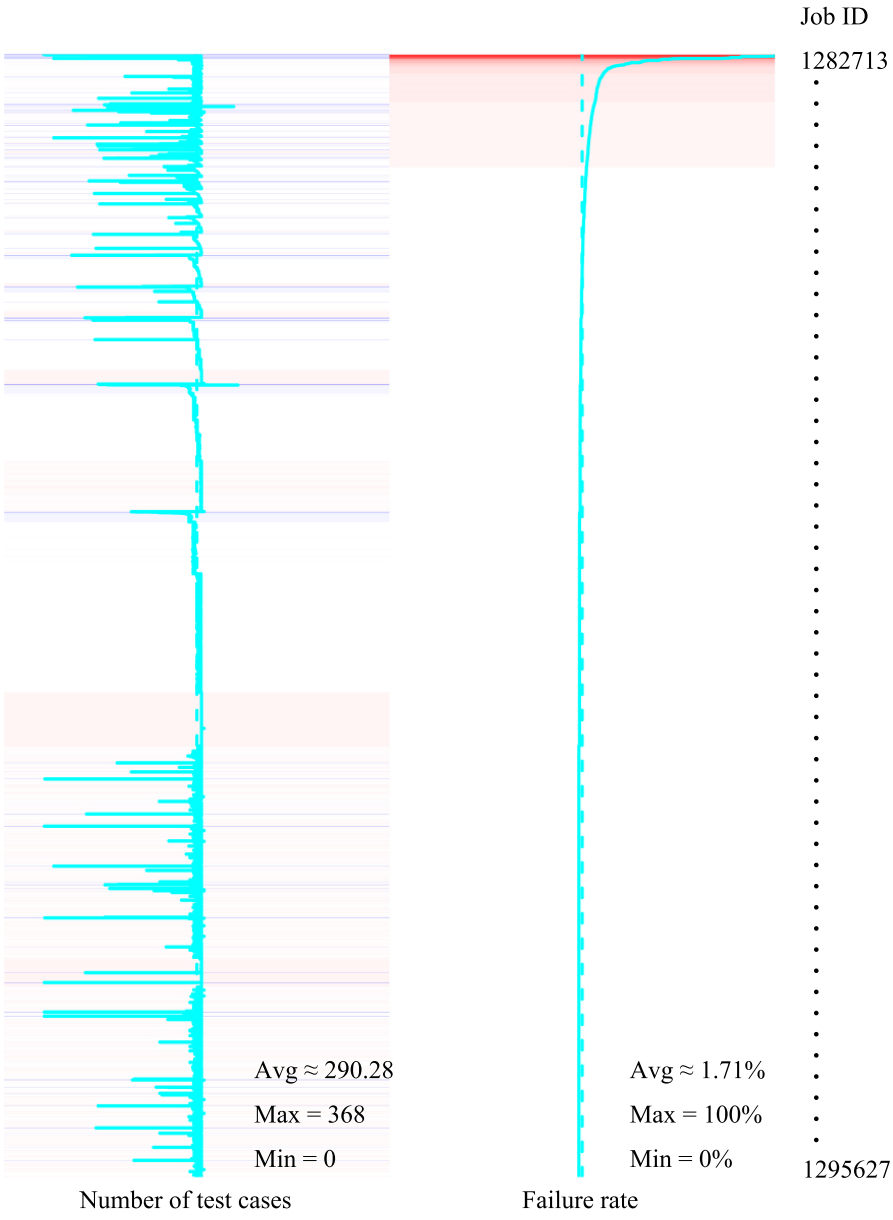

Figure 4.9: The number of test cases and failure rate based on each job in large data set 14_all.

As shown in Figure 4.9, the analysis results of new data set is a bit different. First of all, we noticed the average number of test cases in each job is very close to previous data set, but in the new one, some jobs contain zero test cases. For some jobs, all test cases they contain are failed. Since jobs with high failure rate were not removed, the average value of failure rate is doubled. However, there were still only

a few instances that have failed records.

In order to do a comprehensive comparison, we also created the test case based heat map for this complete data set.

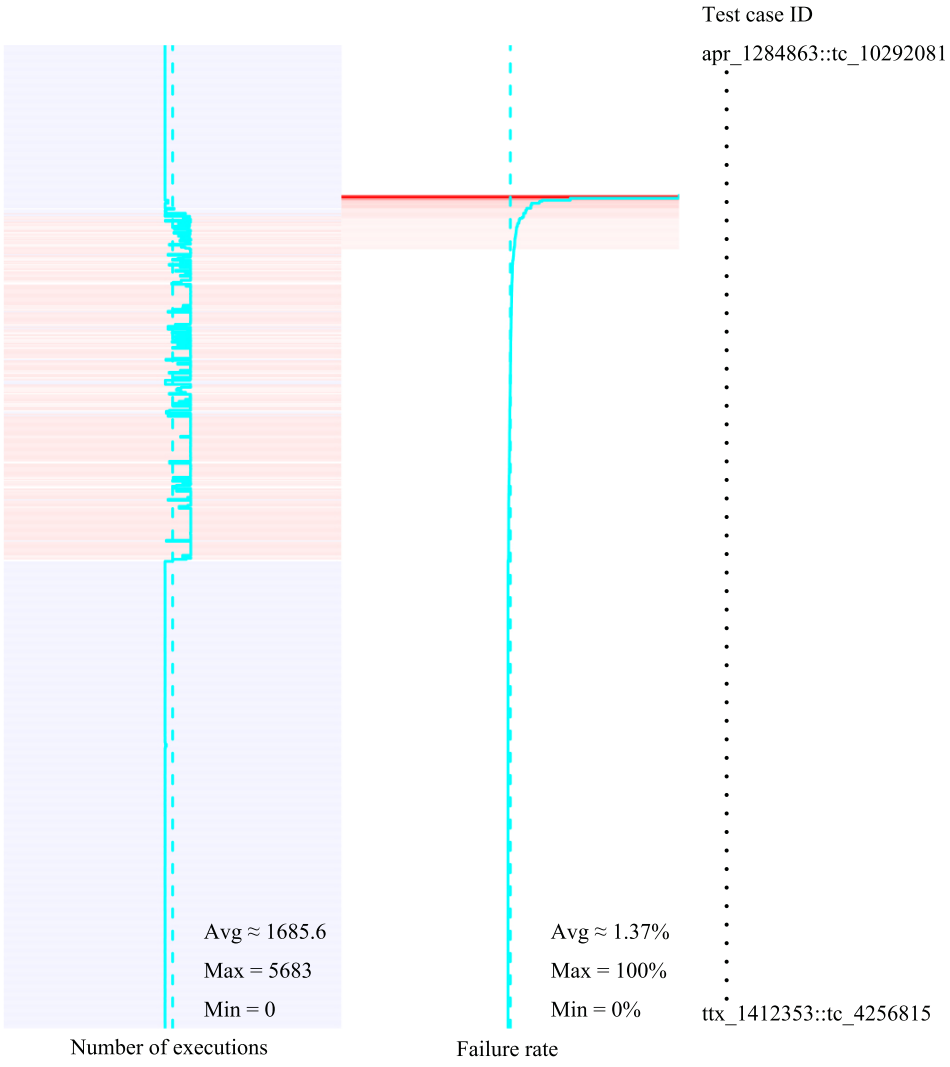

Figure 4.10: The number of executions and failure rate based on each test case in large data set 14_all.

Comparing the results of Figure 4.8 with the picture above, by including all test cases, the average number of test case executions is significantly decreased, and the average failure rate increased. The blank area in the failure column becomes larger due to more unexecuted test cases were included. Both the Figure 4.9 and Figure 4.10 help to prove that even in the data set with complete information, there is still only a very small part of the test cases that failed.

With the help of company's powerful hardware, we successfully processed the large data set that contains all test cases, built up models and calculated MCC values.

Building up models for 300 jobs from the large data set took around one week, and using the following 100 jobs as test set to evaluate the prediction only took around two hours. The results were also visualized into pie charts and shown as following.

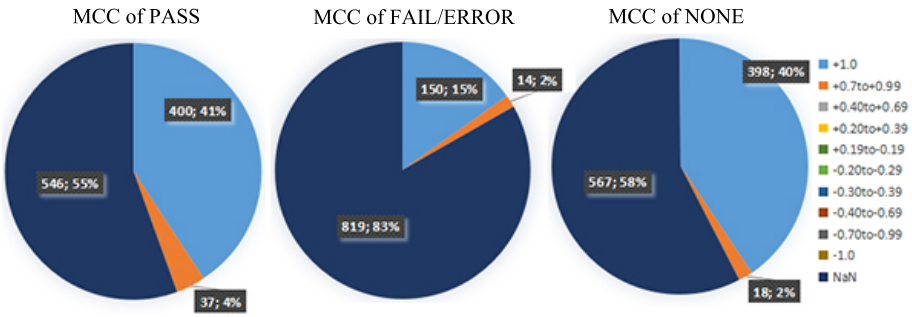

Figure 4.11: The MCC of large data set with large data set 14_all.

Comparing the results in Figure 4.11 with previous outcomes (shown in Figure 4.5 and Figure 4.6), the new one looks much better. For the large data set, data is relatively more balanced since it contains all test cases, thus more MCC of test cases were computable while most of them were good values (Table 3.3). The light blue part in the chart indicates test cases with "one" as their MCC means the prediction for them are 100% accurate, which is unlikely to occur in real industrial. The feedback shows our predictions of those test cases are indeed very accurate.

Apart from the perfectly predicted tests cases mentioned above, quite a few cases also have rather high MCC values, between +0.7 and +0.99. Also, we foundd that there are more "NaN" values in the FAIL/ERROR category than the two other. It is because the prediction tend to be only PASS or NONE, and few test cases are predicted to be FAIL/ERROR, which is consistent with the heat map we showed before (Figure 4.10).

All in all, comparing with the MCC results from cross-validation, the new results are more satisfactory. With the new method, we successfully built models, implemented the prediction method and its evaluation.

5
Results

As described in Section 4, we processed different data sets and collected corresponding results. Starting with the small data set, after successfully building up models, we first used them to make predictions for each test case and inferred their failure rate in different jobs, we then prioritized test cases for each job. As shown in Figure 5.1, it is an example of prioritization result where test cases are sorted based on their failure rate from Job 7, the test case with the highest probability to fail is listed in the first row. In following example, the probability to fail of the first test case is 77%.

Test Case ID	Job: 1	Job: 2	Job: 3	Job: 4	Job: 5	Job: 6	Job: 7	Job: 8	Job: 9
lte_8094186::	0.21	0	0.01	0.03	0.04	0.01	0.77	0.07	0.03
c1i_7297540::	0.17	1	0.05	0.01	0.02	0.03	0.72	0.06	0.02
c1i_7297540::	0.17	1	0.05	0.01	0.02	0.03	0.72	0.06	0.02
cdr_239864::t	0.06	0	0.1	0	0.01	0	0.69	0.03	0
rf_3291280::t	0.03	0	0.11	0.1	0	0	0.04	0.01	0
gtp_13653653:	0.03	0	0.1	0	0	0	0.03	0.01	0
gtp_13653653:	0.03	0	0.1	0	0	0	0.03	0.01	0
gtp_5628961::	0.03	0	0.1	0	0	0	0.03	0.01	0
gtp_5628961::	0.03	0	0.1	0	0	0	0.03	0.01	0
gy_6767498::t	0.03	0	0.1	0	0	0	0.03	0.01	0
gy_6767498::t	0.03	0	0.1	0	0	0	0.03	0.01	0
lte_12930956:	0.03	0	0.1	0	0	0	0.03	0.01	0
lte_12930956:	0.03	0	0.1	0	0	0	0.03	0.01	0
lte_12930956:	0.03	0	0.1	0	0	0	0.03	0.01	0
lte_12930956:	0.03	0	0.1	0	0	0	0.03	0.01	0
lte_12930956:	0.03	0	0.1	0	0	0	0.03	0.01	0
lte_16481587:	0.03	0	0.1	0	0	0	0.03	0.01	0
apr_12597690:	0	0	0	0	0	0	0	0	0
c1ci_14216099	0	0	0	0	0	0	0	0	0
c1ci_14216099	0	0	0	0	0	0	0	0	0
c1i_7297540::	0	0	0	0	0	0	0	0	0
c1i_7297540::	0	0	0	0	0	0	0	0	0

Figure 5.1: The Prioritization Results Example.

In conjunction with Figure 4.6, these predicted results are more credible. By referencing the APFD (introduced in Section 2.2.3), we adopt a scoring method which rates points for each instances and then visualized results into scoring curves. In order to prove the efficiency of our method (APCIT), we compare its score with scores of three other priority ways:

1.Optimal order, the ideal list where test cases with failures would be put in the top.

2.Natural order, the original list of test cases that is sorted alphabetically.

3.Random order, the list contains test case that sorted by the system randomly.

4.APCIT, the list sorted by our method.

The score for each job is calculated by computing the sum of the ordinals of the failing test cases in the sorted list. In the optimal list, test cases with failures would be put in the top of it. E.g. When a job contains ten test cases and three of them fail. These three shall be listed as the first three and have "1", "2" and "3" as their ordinal respectively. For calculating its optimal score, we have "1 + 2 + 3", and the result point is "6"; In APCIT list, if failures were predicted and ranked as "1st", "2nd" and "4th", then the score would be "1 + 2 + 4 = 7". The scores of natural order and random order are calculated in the same way. The closer the score of a sorting method and the optimal result of jobs, the more accurate and valuable the predicted prioritization.

The reason we introduce the natural order and random order is that their results can reflect current situation of functional testing in the case company. At the same time, comparing with the optimal results in scoring curve can also highlight the advantages of APCIT. The following picture shows the evaluation results of prediction for small data set.

Figure 5.2 and 5.3 show the curves for small data set 16_90pct.

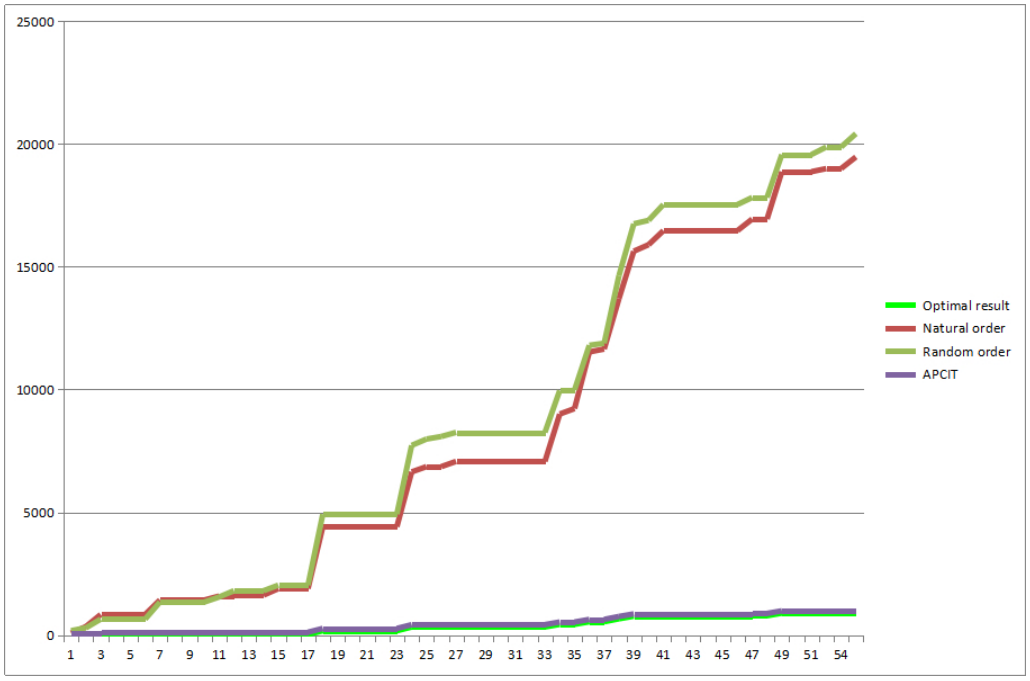

Figure 5.2: The scoring curves for small data set 16_90pct.

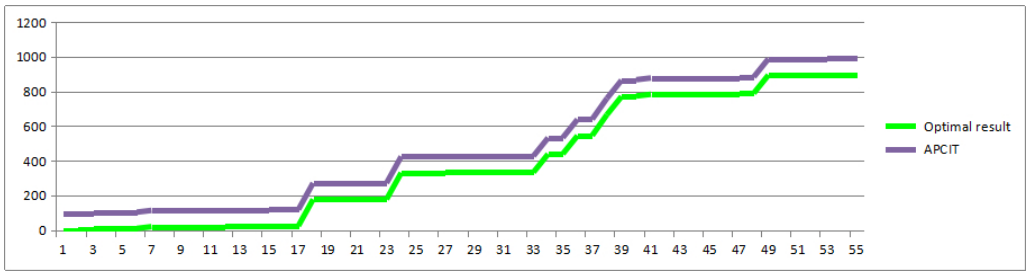

Figure 5.3: The magnified scoring curves for small data set 16_90pct.

The Figure 5.2 shows scoring curves for small data set 16_90pct. In the graph, the x-axis represents jobs and the y-axis is total score. From its upper part, it is clear that natural order curve and random order curves are very close and do not have much differences. It is because in the source data, test cases are sorted alphabetically, thus they are already listed like random order. Meanwhile, the APCIT scores are much lower than the scores of both natural and random order, and also near to optimal results on expense of very high computational effort.

In order to distinguish the curve better, we also zoomed in APCIT and optimal curves as the Figure 5.3. As shown in the second figure, although these two curves do not completely overlap, they are very close to each other and their trends of changing are also the same. Their heap scores are much less than natural or random order. Moreover, not overlapping also means the results are not overfitting. Therefore, we believe that the prediction is very accurate, and we can apply this method

to large data set 14_all.

The following figure shows scoring curves for the large data set, it has the same expressions and similar characteristics as the previous figures.

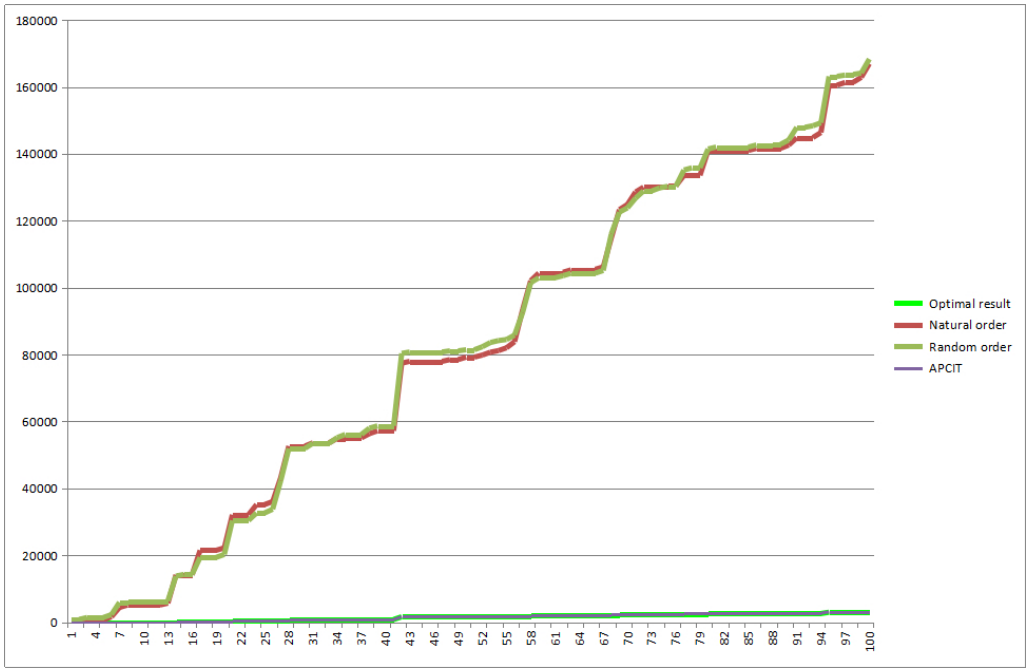

Figure 5.4: The scoring curves for large data set 14_all.

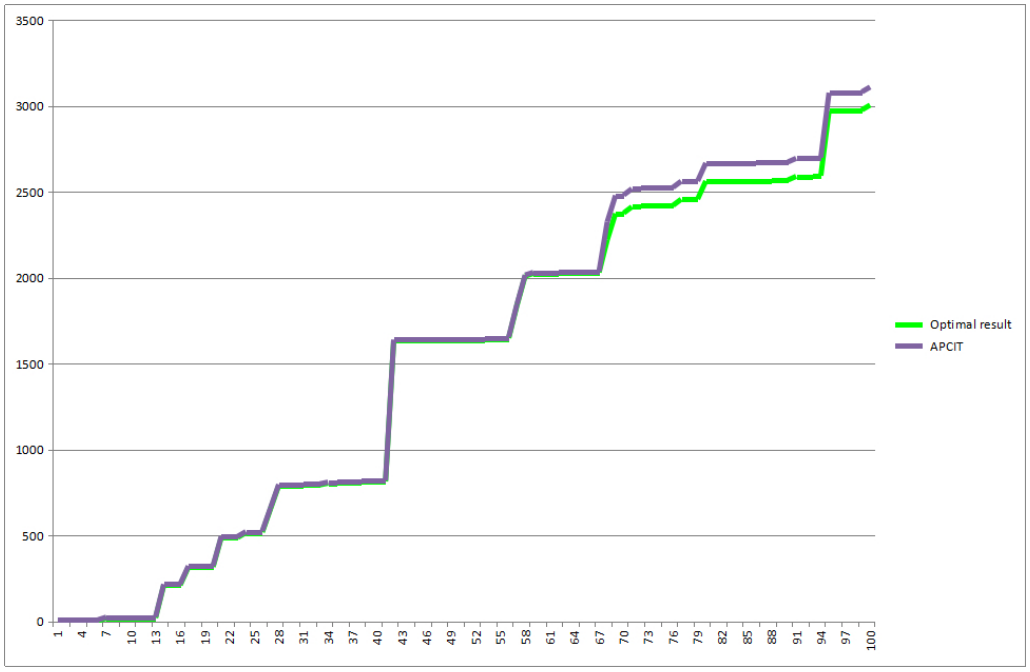

Figure 5.5: The magnified scoring curves for large data set 14_all.

The figures illustrate that the curve of APCIT is far superior to the curves of natural and random order, while also very close to the optimal one. However, since the amount of data is very large, the scores are quite high which lead to many overlaps, even in the enlarged part. But it also shows that the results of APCIT and optimal are very close, which proves the accuracy and reliability of our predictions.

6

Discussion

6.1 Bottlenecks and Challenges

We think the biggest bottleneck in this study was during the first half phase. At that time, we were not aware of the need to use powerful hardware from the case company, and thus, we were unable to carry out analysis of large-scale data nor do proper testing. After we realized the this need and having had a discussion with the case company, we were able to mitigate this problem and remove the bottleneck with their help.

The biggest challenge in this study was to find an efficient way to handle and analyze the data sets. Each execution of scripts would cost a lot of time. Although, building up models using the small data set 16_90pct only took around two hours and twenty minutes for evaluation and prediction, finishing the tasks for the first 300 jobs in large data set 14_all took around one week and three hours respectively. We assume generating models for the entire large data set 14_all would need around two months. Multiple debugging and testing iterations were costly and difficult to apply during the development process, so we had to be careful with making every change to the scripts during the design process and make reasonable arrangements for the schedule.

Besides, the built models are non-updateable. Even though the frequency of how frequent the model needs to be rebuilt and how large the training set shall be are good questions, and need to be answered before the method is applied in industry, we were not able to verify them within this thesis.

Further, since the analyzed data sets contain relatively large amount of information, the analysis actions require strong hardware support. During the process of parsing 300 instances, the physical memory used for building models were around 7GB, where the final models occupied around 13GB of disk space.

6.2 Trade-offs

6.2.1 Selection of Algorithms

As explained in Section 2.2.2, we decided to use RandomForest as our main focus after studying the relevant literature. But we did make another attempt since we found that Weka supports an algorithm named NaiveBayesUpdateable, which can not only support quick updates to the model but also computes the results rapidly. The figure below shows a part of the MCC results from using the NaiveBayes classifier.

MCC(PASS)	MCC(FAIL/ERROR)	MCC(NONE)
NaN	NaN	NaN
-0.031788262	-0.031788262	NaN
-0.005988024	-0.005988024	NaN
-0.01043423	-0.01043423	NaN
NaN	NaN	NaN
NaN	NaN	NaN
-0.005988024	-0.005988024	NaN
-0.008493813	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.030443128	-0.029054103	NaN
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024
-0.005988024	NaN	-0.005988024

Figure 6.1: The MCC results of NaiveBayesUpdateable.

In Figure 6.1, it shows that many MCC values for test cases can be calculated, but they are between -0.19 and +0.19, which means these results are like random guessing and not credible. By comparing with the outcomes of the results from RandomeForest, we decided to abandon using NaiveBayes.

6.2.2 Filtering Data Sets

In the early stages of the study, we analyzed the small data set. By counting, there were 299 test cases and only 55 of them had failed. As discussed in Section 6.1, the analysis program requires a lot of time. For the purpose of time-saving, we removed all none-failure test cases before using Weka for data mining. However, in later experimenting process, we found that those removed test cases did not consume too

much time to process. For the large data sets, the analysis time of such test cases is only tens of seconds. Thus in order to maintain integrity of data sets, we skipped filtering of test cases in the rest of the study.

6.2.3 Scoring System

As mentioned before, our scoring system referred to the calculation of Average Percentage of Faults Detected (APFD), and we believe it is scientific. We used our own methodology instead of APFD since we found the latter is improper for the use case in this study. According to Praveen [29], the formula of APFD is:

$$APFD = 1 - \frac{TF_1 + TF_2 + ... + TF_m}{nm} + \frac{1}{2n}$$

Where the value "m" is the number of faults contained in the program under test P, the value "n" is the total number of test cases, and the value "TF_i" is the position of the first test in T that exposes fault "i".

This APFD formulation can only be implement when at least one failed test case was included in each job. In our use case, most of the test cases never fail (reference heat maps in Section 4), thus the m value in APFD would be zero and the entire calculation cannot be carried out. The same problem does not exist in our scoring system because we are only grading and counting failed test cases. Due to this, we did not completely apply APFD in this study.

6.3 Feasibility, Applicability and Practicality

Since doing data mining on the source data sets resulted acceptable MCC values, we believe that the method of predicting the failure probability of test cases and to prioritize them by mining the historical data of changed files and associated test case results is feasible.

Meanwhile, by doing data selection, preprocessing, transformation, data mining and evaluation, we produced reasonable and valuable results that also recognized by the case company. Thus we argue that the result of this study is applicable and valuable.

In addition, the scoring curves show in Section 5 also confirm that the method described in this report can provide very accurate predictions, which has a high practicality.

To answer the RQ1. Statements above prove that the case company is capable to provide us historical data for automatic classification and prediction.

To answer the RQ3, we presented and discussed our results with the case company. The company recognized our outcomes and agrees on using this method could be very helpful. After fully implemented, APCIT can increase productivity of functional testing by decreasing the feedback time from failing test cases.

6.4 Restrictions

As mentioned in Section 6.1, the time for this study is very limited. Therefore, it is difficult for us to make a deep analysis or verification of different data sets. In this situation, we cannot build up a complete program for data mining and analysis. But instead, we do provide a working prototype and satisfactory conclusions based on the given use cases.

Since this method has relatively high demands on hardware, especially the physical memory, a personal computer would have difficulties with carrying out the processing of large data sets. At the same time, due to this method is very time consuming, updates of the models could also be very expensive, thus built models are not likely to be updated frequently, which could limit its scope of application. We have selected what we believe is the most efficient way from existing algorithms, if in the future, a more efficient algorithm is developed, these restrictions might be solved. All above are the main limitations that we found during our study.

The answer for RQ1 in Section 6.3 already prove that APCIT is suitable for the use case. The discussion in this section supplementary answered the RQ2, which proposes some notes when this method is applied to other use cases.

6.5 Threats to Validity

We performed semi-structured interviews and non-systematic literature review in this study, because they can help us to correctly understand the use case and gain basic domain knowledge. They are not our main focuses, instead, we highlighted our processes of building up models and evaluation.

This study is based on the use case of the case company. Data distribution in analyzed source data sets are very uneven, most of the test cases tend to pass. Thus we only selected RandomForest for our data analysis. In other cases, if a more even-distribution historical data was used for processing, other algorithms might also be able to build excellent models. Therefore, we cannot guarantee that our method will applicable for other use cases.

Our method is designed for and tested with the use case of the case company. The risks of applying APCIT to other use cases are mentioned in Section 6.4.

This study was conducted by two master students of software engineering domain. During the research processes, they monitored and promoted each other to guarantee the accuracy and reliability of the results. Meanwhile, the final outcomes were verified by the case company which further enhance their value.

7
Conclusion

This study was designed to help the case company to prioritize their test cases. The company will not deliver products to their customers until all tests are executed and passed. However, during development, it might be good to not always run all test cases. Also, if all tests shall be run, it could be very helpful for giving feedback as fast as possible. And by prioritizing test cases that are likely to fail, the feedback loops can be reduced. We tried to make prioritization by focusing on finding the relevance between results of test case executions and changed files by using the technology of data mining. As the result, after successfully predicted probabilities to fail for all test cases in different jobs, we are able to provide prioritization lists for all test cases in different jobs.

Our APCIT method can provide accurate predictions. After inspection, the accuracy of prioritization is far superior to natural and random order. In order to use this method, a user needs to extract historical data in a certain format, and then executes the corresponding scripts to complete all the remaining steps, including transforming the format data, building up the models, obtaining and verifying the results. The built models can then be utilized to generate prioritization list for new data that include changed files and related test case results until the model out of date. But it is worth mentioning that the process of building models takes a lot of time and the models are not easily updatable, users need to weigh whether and how to apply it according to their specific situations.

Overall, after processing as much different data sets as possible in limited time and verifying their results scientifically and rationally, we believe that APCIT can provide excellent test case prioritization for the use case of the case company, and it is a highly desirable method. Also, this method has the potential to be further implemented and applied to other use cases.

7.1 Future Work

In this thesis, we focus on probing the feasibility, applicability and practicality of APCIT by analyzing the characteristics of source data sets from the case company and providing prioritization lists based on prediction of failure probabilities for test cases in different jobs. As discussed in Section 6.1, parsing data sets is very time-consuming and we can only study on a few cases. If we were to continue with this

study in the future, we would be able to process more data sets and evaluate the models with various of test sets, which further help us to examine how often the models need to be updated (regenerated).

It would also be interesting to have further discussions with the case company on how good (accurate) prediction is required. Our method can provide precise forecasting, but if the company was less critical with the accuracy of predictions, then, finding other solutions that can parse data faster could be another valuable study. Meanwhile, once models have been generated, prioritizing new inputs would only costs relatively short time, which makes daily execution feasible. Thus investigating how often the evaluation shall be applied with the company would also be another valuable future study.

Bibliography

[1] Martin Fowler (2006). Continuous Integration. Available from: <http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/2013-14/lecturas/10_Fowler_Continuous_ Integration.pdf>. [8 January 2016].

[2] Sean Stolberg (2009). Enabling Agile Testing through Continuous Integration. IEEE Conference Publications.

[3] Agneta Nilsson, Jan Bosch, and Christian Berger (2014). Visualizing Testing Activities to Support Continuous Integration: A Multiple Case Study. Springer International Publishing Switzerland.

[4] Adam Debbiche, Mikael Diener, and Richard Berntsson Svensson (2014). Challenges When Adopting Continuous Integration: A Case Study. Springer International Publishing Switzerland.

[5] Rudolf Ramler, Michael Felderer (2016). A Process for Risk-Based Test Strategy Development and Its Industrial Evaluation. Product-Focused Software Process Improvement.

[6] Robert Feldt, Greger Wikstrand, Jeevan Kumar Gorantla, Wang Zhe (2009). Dynamic Regression Test Selection Based on a File Cache - An Industrial Evaluation. International Conference on Software Testing Verification and Validation.

[7] Linlin Wang (2015). Implementation and Evaluation of an Automatic Recommender for Integration. Chalmers University of Technology.

[8] Osmar R. Zaiane (1999). Introduction to Data Mining. CMPUT690 Principles of Knowledge Discovery in Databases.

[9] Jiawei Han, Micheline Kamber (2001). Data Mining: Concepts and Techniques. London:Academic Press, 5.

[10] Ian H. Witten, Eibe Frank (2005). Data Mining - Practical Machine Learning Tools and Techniques - 2nd Ed. Morgan Kaufmann Publishers is an imprint of Elsevier.

[11] Felderer M, Schieferdecker I (2014). A taxonomy of risk-based testing. International Journal on Software Tools for Technology Transfer.

[12] Tin Kam Ho(1995). Random Decision Forest. Proceedings of the Third International Conference on Document Analysis and Recognition, vol.1, p.278-282.

[13] Tin Kam Ho (1998). The Random Subspace Method for Constructing Decision Forests. IEEE Transaction Pattern Analysis and Machine Intelligence 20 (8), p.832-844.

[14] Huotao Deng, George Runger, Eugene Tuv (2011). Bias of importance measures for multi-valued attributes and solutions. Proceedings of the 21st International Conference on Artificial Neural Networks.

[15] David Martin (2011). 'Evaluation: From Precision, Recall and f-measure to ROC, Informedness, Markedness & Correlation. Journal of Machine Learning Technologies. 2 (Issue 1), p.37-63.
[16] Deborah J. Rumsey (2011). Statistics For Dummies, 2nd Edition. Wiley Publishing, ch 10.
[17] Ahmed Hassan (2008). The road ahead for mining software repositories. FoSM: Frontiers of Software Maintenance, p.48-57.
[18] Ahmed Hassan (2006). Mining Software Repositories to Assist Developers and Support Managers. ICSM 2006: 22nd IEEE International Conference on Software Maintenance.
[19] Mathias Meyer (2014). Continuous Integration and Its Tools. IEEE SOFTWARE, vol. 31, issue 3.
[20] Ripon K. Saha, Lingming Zhang, Sarfraz Khurshid, Dewayne E. Perry (2015). An Information Retrieval Approach for Regression Test Prioritization Based on Program Changes. ICSE 2015.
[21] Cem Kaner, Jack Falk, Hung Quoc Nguyen (1999). Testing Computer Software 2nd Ed. Wiley Computer Publishing, p.480.
[22] Shalal-Esa, Andrea, Maureen Bavdek (2013). Investigators to travel to Pratt facility in Phoenix: NTSB. Reuters.
[23] Kaner, Falk, Nguyen (1999) Testing Computer Software. Wiley Computer Publishing, p.42.
[24] Paul Ammann, Jeff Offutt (2008). Introduction to Software Testing. Cambridge, p.215.
[25] Tin Kam Ho (2002). A Data Complexity Analysis of Comparative Advantages of Decision Forest Constructors. Pattern Analysis and Applications, p.102–112.
[26] Tao Shi, Steve Horvath (2006). Unsupervised Learning with Random Forest Predictors. Journal of Computational and Graphical Statistics, vol.15(1), p.118–138.
[27] Andy Liaw, Matthew Wiener (2002). Classification and Regression by randomForest. R News, Vol.3(2), p.18-22.
[28] Gregg Rothermel, Roland H. Untch, Chengyun Chu, Mary Jean Harrold (2001). Prioritizing Test Cases For Regression Testing. IEEE Transactions on Software Engineering, Vol.27(10).
[29] Praveen Ranjan Srivastava (2005). Test Case Prioritization. Journal of Theoretical and Applied Information Technology.
[30] Yafeng Lu, Yiling Lou, Shiyang Cheng, Lingming Zhang, Dan Hao, Yangfan Zhou, Lu Zhang (2016). How Does Regression Test Prioritization Perform in Real-World Software Evolution?. ICSE 2016.
[31] Eibe Frank, Mark Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H. Witten, Len Trigg (2005). WEKA - A Machine Learning Workbench for Data Mining. Data Mining and Knowledge Discovery Handbook, p.1305-1314.