國立臺灣大學電機資訊學院電機工程學研究所
碩士論文
Graduate Institute of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

網頁和手機程式自動化測試智能技術

Intelligent Techniques for Automated Testing of Web and Mobile Applications

吳啓允
Chi-Yun Wu

指導教授：王 凡 博士
Advisor: Farn Wang, Ph.D.

中華民國 105 年 7 月
June 2015

誌謝

首先最先感謝我的父母，有了爸爸媽媽的支持與照顧，在我低潮的時候鼓勵我勉勵我，我才能順利完成我的學業。我也要特別感謝指導教授王凡老師，總是爲我指引研究的方向。老師教導我軟體測試的專業知識，也教導我進行研究的方法和態度，還提供許多機會讓我開廣見聞。我還要感謝俊偉學長、庭芬學姊和汶宏學長，因爲他們的幫助，我才能順利完成程式的開發，對於程式的觀念也進步許多。我也要感謝修博、宗儒和上誼，這兩年一起修課，在困難的時候一起扶持，一起面對挑戰，因爲有你們的幫助，我才能堅持完成這一切。最後感謝所有實驗室的同學，帶給我歡樂的研究室生活。謝謝大家。

摘要

在網路普及的時代，各式各樣用途的網頁相繼開發出來，人們的生活也越來依靠各種網站。因此對開發者而言，爲了確保網站程式的品質，如何快速有效的進行網頁測試就顯得重要。我們在此篇論文中開發了一套工具來自動化測試網頁，使用者不需撰寫測試腳本，就可以點擊和填值的方式自動瀏覽動態網頁，並且記錄下瀏覽的紀錄。然後我們提出了一個方法來建構測試準則，我們應用了機器學習中支撐向量機的技術，從網頁和手機的使用軌跡中抓取特徵值，順練出預測模型來自動判斷測試紀錄是否通過。藉由這個工具和驗證方法，我們可以有效地降低人力成本達到自動化測試的目的。

關鍵字：軟體測試、網頁測試、自動化測試、測試準則

Abstract

In the recent days, the Intertnet is becoming more popular. A wide range of web applications have been developed. People spent lots of time on Internet. Thus, it becomes an important problem to verify the web applications to developers. In this paper, we propose a tool for automated web testing. The developers do not need to write the test scripts. The tool can explore the dynamic webpages by clicking buttons and insert values and record the test traces. We propose a system to construct test oracle of web applications and mobile applications using the support vector machines. The system extracts features from the traces and builds aa predictive model to classify the passed traces and failed traces. With the automated testing tool and system, we can effectively reduce human cost to achieve the purpose of automated testing.

Keywords: Software testing, Web testing, Automated testing, Test oracle

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Web Applications become more and more indispensable in our life. People can easily get informations with the development of the Internet. It becomes more convenient for people to search new knowledge, buy goods and chat with friends. People spent lots of time on the Internet for working, studying, shopping, socialozong and playing. As the result, more engineers tend to develop the applications on the Internet. On the other hand, Mobile Applications is also a choice for software developer. People can carry smart phone and enjoy the applications everywhere. Although people can visit web applications by browser on the smart phone, a special mobile application has a better performance. Companies may develop the applications on both the Internet and the mobiles. In order to guarantee the performance and user experience of the applications, the testing of the applications is required. With generating lot of test cases, it can help engineers find the bugs in their applications from the failed test cases.

1.2 Motivation

There are serveral tools developed fro testing applications nowadays, for example, developers can use selenium to test that their website is working successfully. However, the current techniques of testing take lots of works and times. Because the applications are complicated and dynamic, they have different reaction with different inputs. Engineers need to write the test scripts case by case according to the each functions. They also need basic knowledge of software testing. To reduce the work time of testing, the automated testing become essential for engineers. But the challenge is that it is too difficult to generate test case on black box testing. Without the detailed information of functions, We may not know how to make a test script. Thus, to find a method for solving the problem of automated tesing dynamic applications is important. The automtated testing should help developers generate a variety of traces and try to predict which traces is failed and

1.3 Purpose

In this paper, we propose a technique to automatically generating traces on dynamic web applications. The program named WebTraceCollector can collect the informations on web pages and try to guess the suitable inputs. We construct a input databank with some examples of input, so the program will find a similar example as the fitting input value of the web page. With the suggested input values, the program can explore the website automatically and build the finite state machine to represent the website, and record the every step and web pages during the testing and generate the traces for testing.

In order to automatically evaluate traces collected from web and mobile applications, we propose a teachnique to extract the feature from the traces and transform the traces into feature vector. We constrcut a keyword dictionary, which collects keywords that represent the common sense behaviors of action and screen from applications, so the traces can be expressed in a vector format. Then, we use the method of machine learning to evaluate traces. We use SVM to classify the traces into passed traces and failed traces.

1.4 Organization

The rest of this paper is organized as follows. Chapter 2 shows the related work of testing, and we introduce the tool for generating mobile traces and the method used for evaluating in Chapter 3. In Chapter 4, we propose a technique to generate traces on dynamic web applications. Chapter 5 shows the framework to evaluate traces from the web and mobiles. Then, the experiment results of collecting traces from web applications and predicting traces are shown on the Chapte 6, and the conclusion is shown in Chapter 7.

Chapter 2

Related Works

Our purpose is to construct an automated web testing tool and evaluation method. Web testing and test evaluation have been studied many years since 2001[1]. We survey the similar tools[2] and researchs about testing web applications and test oracles. They are shown as the following.

2.1 Web testing

Testing tools likes Jmeter[3] record the script of user behaviors and replay it on the web pages. Jmeter is not focus on functional testing but on load testing. It creates many threads to test the specific websites simultaneously or replay the test lots of times. It measure the performance of the web applications and the servers.

Selenium[4] is a tool which can record test scripts with friendly interface on firefox. Except for recording, selenium also can set assertions in the test cases. Selenium is a powerful library for web testing. User can write their test scripts on most of language likes C#, Python, Ruby, Java and Perl.

TestStudio[5] and TestComplete[6] is platform which has frienfdly GUI interface for user. It can record scripts and implement functional testing and performane testing. During testing, it captures not only snapshots but also informations of objects. User can also write their own plugin to control the test scripts.

VeriWeb is tool presented by Benedikt et al.[7] for automatically exploring dynamic

websites and execution paths including form submissions and client-side script. To testing websites used Ajax, which is based on asynchronous communication with server, Marchetto et al.[8] proposed state-based testing and construct a finite state machines to explore websites.

Crawljax[9] presented by Mesbath et al. is a java open source tool which can automatically explore dynamic websites and collect the invariants of the Document Object Model of the web pages to construct finite state machine. It records traces by Depth-first search algorithm and generates state flow graph. User can write own API to control test scripts or add assertions.

2.2 Test evaluation

In software testing, testers use an oracle as a mechanism for determining whether a test has passed or failed in order to reduce human cost. The test oracle should capture the properties of the system to find test cases and counter examples. Assertions may be used to check the specified behaviors of systems at runtime. Barr et al.[10] gathererd the oracle problem in software testing into four categories.

Kanewala et al.[11] reasearch for finding the metamorphic relations, which focus on the relations between different executions of the applications and inferred from white box inspection of the SUT, using machine learning techniques. Derived test oracle distinguishes between the correct and incorrect behaviors of the SUT based on the properties pd th system. Daikon is a tool, which is presented bt Ernst et al.[12], to find likely invariants from traces automatically.

Chapter 3

Preliminaries

In this chapter, We not only construct a tool for automated testing, but also use other tools to help us to generate traces and use the algorithm of machine learning to help us to evaluate traces. The tool and the algorithm are introduced as the followings.

3.1 SpecElicitor

SpecElicitor is a tool made by Yuan-Hong Lo, which can help user test Mobile Application with friendly GUI interface.

When user start using SpecElicitor, each time user do an action, such as clicking a button, on the Mobile Application, the tool will ask for labeling the action and the screen shown on the interface. By using SpecElicitor, we can choose the specified action at every step, label the meaning of the action, and label the exception situation if it happen and stop the trace at any time we need. The interface of the SpecElicitor is shown in Fig[3.1].

The traces made by SpecElicitor include automata, screenshots, XML files and labels. The automata is a json file record every states and edges on the traces. A state has a screenshot and a XML file dumped from the Mobile, and an edge records the action user did such as clicking element, typing text or exiting the Application. The traces also record labels on every states and edges, so we can recognize how many label we focus on happened on the trace. An example trace of Recipe Application made by SpecElicitor is shown in Fig[3.2].

Figure 3.1: The GUI interface of SpecElicitor.

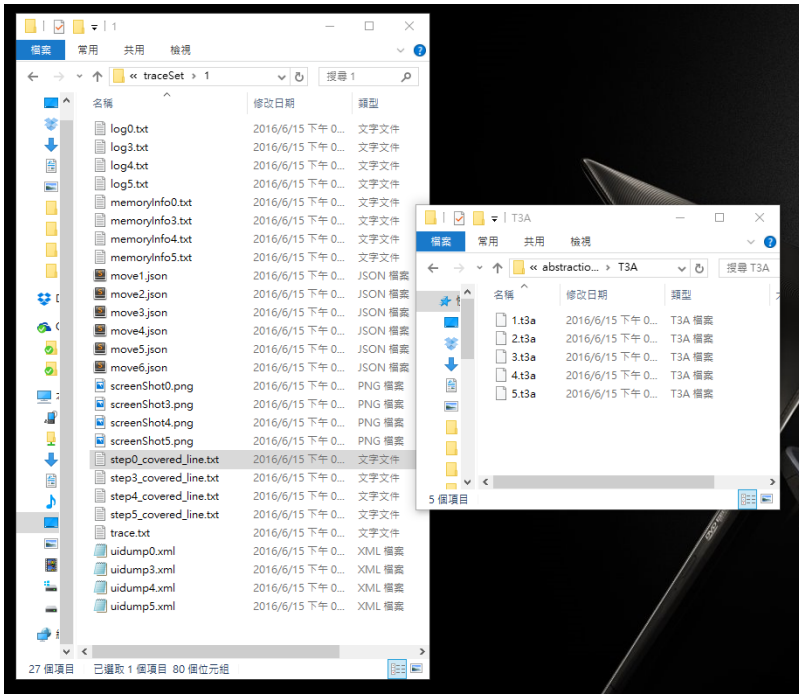

Figure 3.2: An example of the traces.

3.2 Normalized keywords

We construct a keyword dictionary of Normalized keywords, which represents the behaviors of applications. There are different kinds of applications, however, people will expect that same kind of applications perform the similar behaviors and generate similar results. By generalizing those similar applications with behaviors, we transform the behaviors of applications into the common sense model and use normalized terms to represent the common sense.

The keyword dictionary collects normalized keywords from several kinds of applications. A common sense, defined as a normalized keyword, represent a concept of a screen shown on the platform or an action for a clicking event. For instance, the keyword dictionary of FileManager application is listed in Table 3.1. Take FileManager as example, the button "REMOVE" and "DELETE" represent the same concept of behavior, removing the selected file. People except this two buttons have same performance when people use in the two different FileManager applications. We define this concept of action and represented by a normalized keyword "delete-file". With the help of keyword dictionary, we can use the normalized keywords as features and extract the features of the traces, which consist of screens and actions, from different appliactions and convert the traces into feature vectors.

Screen labels	Action labels
Folder	Change-folder
Create-query	Open-file
Search-Query	Create-file
Compress-query	Creare-folder
Delete-query	Search-file-or-forder
Rename-query	Select-all
Copy-query	Select-file-or-folder
Cut-query	Deselect
Paste-query	Compress-file-or-folder
Item-information	Copy-file-or-folder

Table 3.1: Common sense keywords of FileManager Application.

3.3 Support Vector Machine

Machine learning is widely used in recognition and classification problems. For example, handwriting recognition can predict the letter without using keyboard. Generally, the dataset used by machine learning would be divided into training set and testing set. the training set would be analyzed and used to construct a model, which can make prediction on the testing set.

In our work, we use the Support Vector Machine[13] as our machine learning algorithm to solve the classification problem of traces. Support Vector Machine is a well-known algorithm in machine learning and widely used for classification and regression analysis. SVM is a supervised learning algorithm and it needs the training data labeled. SVM find a hyper-plane in a high dimensional space in order to seperate the instances. The optimal hyper plane has the largest margin, which means the distance between the hyper plane and the closest trainging data. We use the python library in LIBSVM which is provided by Chang et al [14] to implement SVM algorithm in the paper.

We collect the testing traces from web and mobile applications. Because each instances in the data set should be formalized to a set of feature vector, We use the keyword dictionary mentioned above as the featuer vector and transform the traces to vectors consist of keyword features. In order to evaluate traces, we train the SVM with the labeled traces. Then we can predict unlabeled traces and classify them into passed traces and failed traces.

Chapter 4

WebTraceCollector

In order to testing web applications, we need collects a variety of test cases for finding the bugs, analyzing the reasons why the failure happen and ensuring the application can work successfully in any circumstances. However, recording the behaviors on browser manually costs lots of time and work. It is important to improve the efficiency of web testing by generating test traces automatically. Thus, we need a tool that can automatically explore web applications and record the traces.

We make a tool named WebTraceCollector that will be introduced in this chapter for automatically generating traces of Websites. WebTraceCollector is wrote on Python 3 and using Selenium as Library. It can open a browser with selenium and explore the Website with the browser automatically. Depending on the Document Object Model[15] of the current web page,it will find the suitable clickable elements to click. WebTraceCollector can also work fine on the dynamic websites with Ajax[16] application, and insert texts into every input text fields with the fitting data.

4.1 Framework

There are several kinds of work, such as taking snapshot, inserting values and deciding the next element to click, during the web testing. To manage all works, WebTraceCollector collects all events by a list. It also constructs a finite state machine to record every web page in the website and represent the link between web pages by the edges and states. The framework of WebTraceCollector is shown in Fig 4.1.

At the first step of testing, WebTraceCollector go to the target URL and start to recognize the first web page. The web page will be converted into a state of a finite state machine. In order to explore the website deeper, the clickable elements and input fields should be identified and the next action event depending on the clickable elements will be added into the event list. WebTraceCollector constantly get an event from the list and implement a event at once until the list is empty.

After implementing an action, WebTraceCollector will check the status of the webpage. If the web page changes, the new web page will be converted into a new state and saved in the automata. Similarly, the clickable elements of new web page should be identified and the events of new clickables will be added into event list.

The algorithm of WebTraceCollector is shwon in algorithm 4.4.

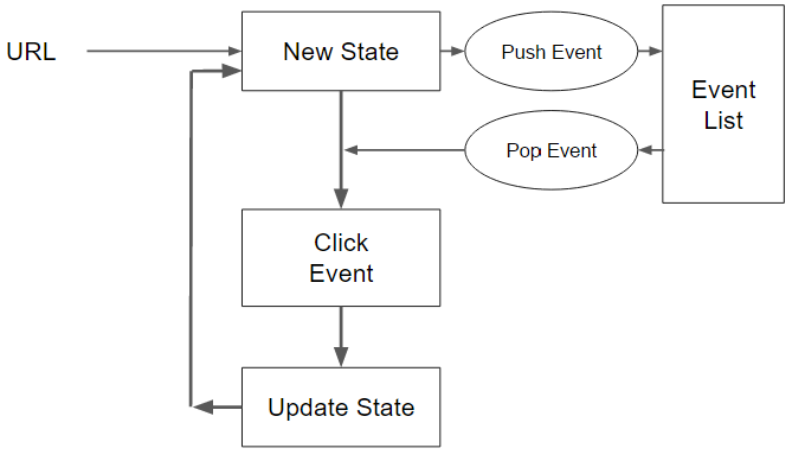

Figure 4.1: The framework of WebTraceCollector.

Input: URL, algo

```
Initial(URL);
Current_State ← Web_Page;
Automata.add_state( State );
while Event_List ≠ Empty do
    Event ← Event_List.Get_Event();
    Do_Event( Event );
    Next_State ← Web_Page;
    if Next_State ≠ Current_State then
        Automata.add_state( Next_State );
        Event_List.Add_Event( Next_State );
    Current_State ← Next_State;
```

Algorithm 1: Overview

4.2 Webpage identification

WebTraceCollector record the current page of the browser into a state in the finite state machine. In order to identify a web page, we distinguish the web pages according to the DOM tree. The DOM tree is the tree structure of the web elements. However, we can not directly access the DOM tree of the web page by the browser, the method to solve this problem is downloading the page source of web page by selenium and rebuild the DOM tree by the library BeautifulSoup[17].

After building the DOM tree, the elements in the web page are known. The clickable elements are defined as <a> tag, <button> tag and <input> tag with button attribute. WebTraceCollector find all the clickable elements in the DOM tree and convert them into event objects. The events will be added into event list, so the testing will continue to click these element later.

In some complex case, the website is not only work by the simple buttons but also javascript functions that have bind mouse listener on other elements. It is almost impossible to guess which element have the javascript function, becuase we only do black box testing of the web applications and we do not know how the web applications work for understanding the javascript code in the web page is too difficult. Even so, most of the web applications still can tested by this method, and automated testing still saves lots of human effort.

4.3 Event

During the testing, WebTraceCollector repeats getting an event from the event list and implements the action of the event until the event list is empty. The procedure of event is shown in Fig 4.2. An event consists of action, state and depth. The action implies which clickable element should be clicked. The state implies where web page the element locates in. The depth implies how deep this event is in the automata. Every event has only one action, which means it only click one button at one time. According to the DOM tree of the web page, the clicking action will insert the value into the input fields.

Before the event start, WebTraceCollector checks the current state on the browser. Because if the current state is not equal to the state recorded on the event, the wrong element will be clicked and the change of the state will be recorded and lead to wrong automata. If the current state is not equal to the state on the event, it should change the web page to where the clickable element locates. Most of the time, it can reached by trace back history from the browser. But in the case of complex website ,it will be hard to retrun the correct web page. In that case, the automata find the simple past from the initial state of wepsite to the target state, and WebTraceCollector will go through the simple path to reach the target web page.

Once the arrive to the correct web page, WebTraceCollector will start to clicke the target element. However, web pages not only have clickable elements, the dynamic web pages also have input fields need to insert and lead to different reaction according to the value that user inserted. To work fine on those dynamic websites that not only have buttons to click but also have other input fields, WebTraceCollector needs to insert values into those elements when it implements the clicking action. The input field elements are defined as <input> tag, <select> tag, <radio> tag and <checkbox> tag. Because the Website may only accept certain words as input values likes, we can not just make random string to test the Website. we construct a database to analysis the element and find the most suitable string. The database has some example values of the common inputs we generalize from websites. Considering the element's tag, id, name and other siblings tags, WebTraceCollector will use the most similar string to the examples as the input value of

the element. The algorithm of implementing the action is shwon in algorithm[2].

Input: action, state, depth

if *Current_State* ≠ *state* **then**

    Back_Track(state);

Clickable ← action.Get_Clickable();

Inputs, Selects, Radios, Checkboxes ← state.getFormElements();

foreach *Element in Inputs, Selects, Radios, Checkboxes* **do**

    Value ← Database.Find_Value(Element);

    Set_Value(Element, Value);

Click_Clickable();

Algorithm 2: To implement the action

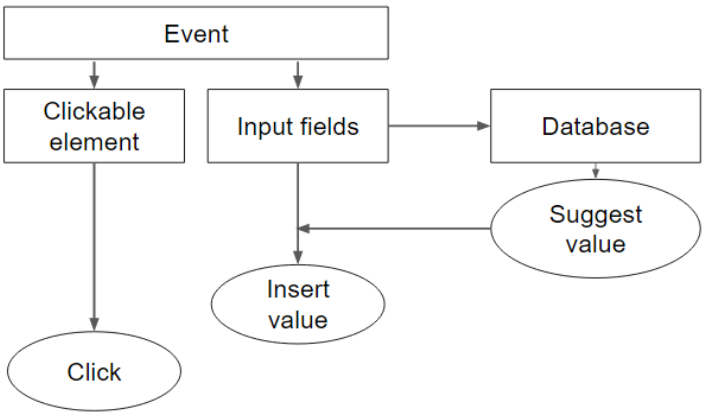

Figure 4.2: The procedure of click event.

4.4 Suggested value

The problem of guessing the correct value of specific input field is very difficult. Because we only implement the black box testing, it is hard to realize the meaning of the current web page while unknowing the whole codes of the web applications. The method we can use is analyzing the text of the element and trying to guess the meaning of the input element.

We constrcut a database which storing lots of example values of common input elements. A partial of the database is shown in Table 4.1. Each data is an example value of a common feature and each feature has several normalized keywords. For example, a feature email has keywords "email" and "信箱" and some exmple valuse like "Ann@gmail.com". For a specific input element, we analyze the element's tag name, id attribute, name attribute and text. If the attributes of the elements similar to the keywords of a feature, we will suggest that the feature is suitable to be the value of the input element. Then, one of the examples randomly selected and insert into the input field.

While an event is implementing, WebTraceCollector not only choose the target clickable element but also need to insert the suitable value into every input fields on the web page. Those input fields' detailed infomation will analyzed with the database and the sugested value will generated by string analysis.

Feature	value 1	value 2	value 3
email / 信箱	Ann@hotmail.com.twn	Bob@gmail.comn	Cindy@yahooo.com
name / 名字	Ann	Bob	Cindy
job / 職業	studentn	engineern	officer
gender / 性別	man	woman	
phone / 電話	0900123456n	0987654321n	0911001011
address / 地址	#12 abc street Tainan Taiwan	5 floor #12 sun street Pintung Taiwa	# USA orange,2323
birth / 生日	2001/02/03	1999/04/03	1988/11/30

Table 4.1: Part of the input database.

4.5 Normalization

After the action, WebTraceCollector will check what happens on the browser by comparing the current state of the web page and other states in the automata. The state objects have the information of DOM tree loading from the browser by Selenium. If the DOM tree is same after click event, the clickable element will be regarded as ineffective and no edge and reaction will happen. On the other hand, if the DOM tree changes, the clickable element will be regarded as effective link. The clickable and the input values will made as a edge and added into automata.

However, there are some problems if we just regard the two DOM trees as strings and compare. For example, there may be advertising applications, calenders, popularity numbers or catalogs shown on the website. Each time user visits the website, the DOM tree of the website may be different. To prevent recognizing a wrong state, we must preprocess the DOM tree.

The work of preprocessing is removing the elements that may confuse us and remain the elements we want to focus on. We remove the tags that are unvisible on the web page, the javascripts code, the HEAD of the html and the css style. We construct a class named normalizer, which can scan the DOM tree, find the target element and remove it. For specific website, User can set the specific normalizer to normalize the DOM tree by his own style. The examples of DOM tree and normalized DOM tree are shown in Fig[4.3].

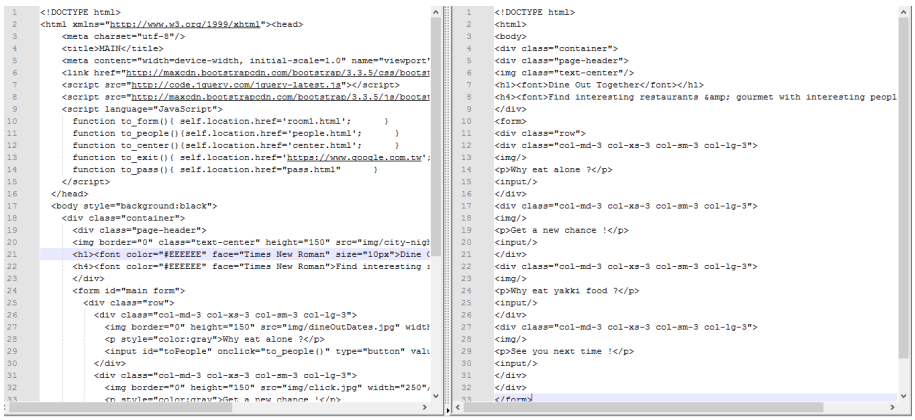

Figure 4.3: An Example of DOM tree compare with Normalize DOM tree.

4.6 Automata and traces

We construct an finite state automata, which consists of states and edges, to represent the Website with each state represent a web page and each edge represent the link from one web page to another. The state is defined based on the DOM tree of the web page. If the URL, DOM tree or any element on the web page changes, we will recognize that it goes to other state. The edge records one state goes to another state by an action, which clicks the specific clickable tag with values written in all form elements.

In order to view the automata clearly and help user to understand the automata easily, WebTraceCollector generate a HTML file to show the graph of the automata. In the html, every state represented by the snapshot of the web page, and every edge between two states are represented by the arrows connected between the two states. The example web page of automata is shown in Fig[4.4].

As the test result, WebTraceCollector generates traces information in different types of files. There are JSON files of traces and automata, screenshots of every state, DOM tree and detail information of every state and a web page of overview automata. Automata.json records all states, which have URL, ID, screenshot and DOM tree of the web page, and edges. Traces.json records the states and edges with the order The example of the test result is shown in Fig[4.5].

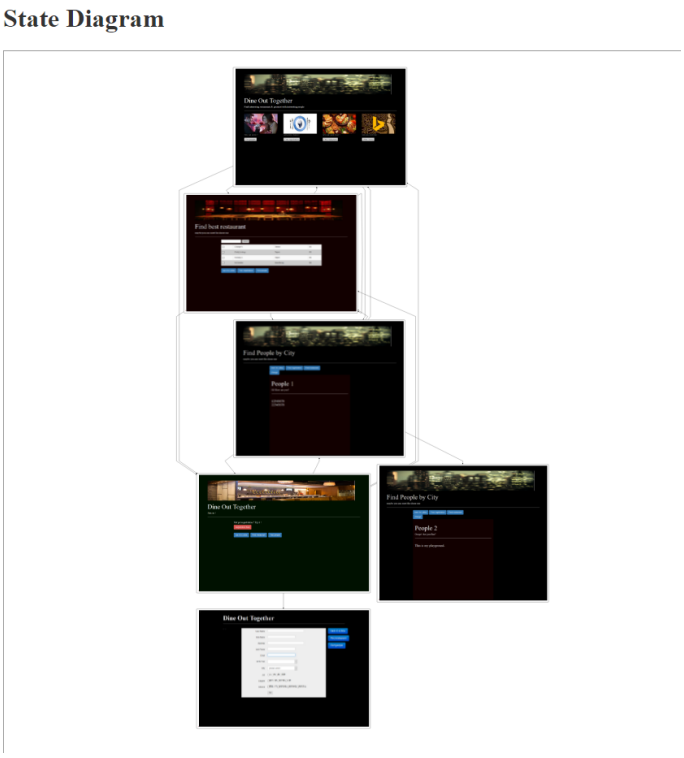

Figure 4.4: Overview of the Automata.

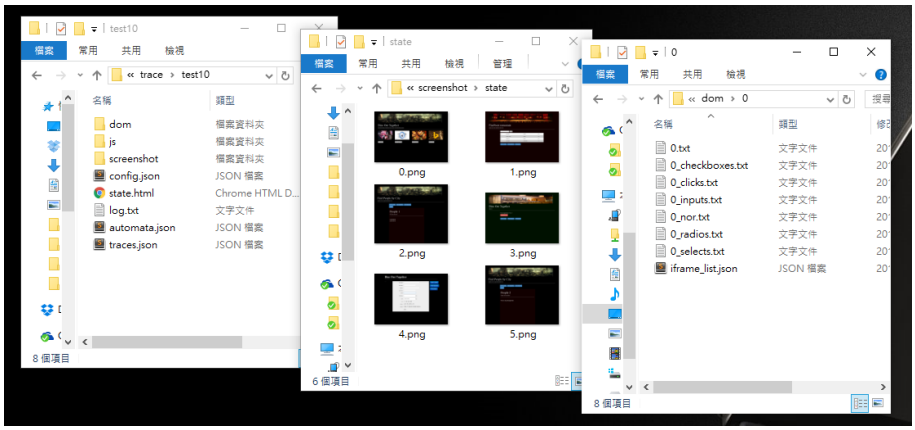

Figure 4.5: An Example traces of the website.

4.7 Trace collection

WebTraceCollector can explore the website with algorithm set by user. We make two algorithms for WebTraceCollector: Monkey and DFS. User can choose the algorithm and set to the config as an input before starting the test. WebTraceCollector will switch it's reaction by the algorithm at every stage after action during the test.

4.7.1 Monkey

The monkey algorithm simply adds a random event to the event list. Every time when WebTraceCollector detects the DOM tree changed and encounters a different state, it collects all clickable elements from the current state and randomly choose one as next event. User can set the trace length and trace amount in the config, which means the maximum edges in one trace and the total amount of all traces. The trace generated by monkey algorithm may contains a loop, because monkey algorithm may choose the same clickable.

4.7.2 Depth First Search

The disadvantage of monkey algorithm is that it may only visit certain web pages and not explore the whole website. To guarantee all clickable elements will be clicked durring the testing, We develop the DFS algorithm. However, Some Websites is too huge to explore all of it's web pages, like Wiki or News, or too complicated that makes unlimited different web pages, like chat room or other dynamic websites. User need to set the maximum depth of the DFS, so the DFS algorithm only explore the website with the depth from the initial web page less than the max depth.

Unlike the Monkey algorithm, DFS algorithm adds all evenst on the current state to the event list. When WebTraceCollector encounter a different web page, it will download the page source of the web page as a state and compare the current state with other states in the automata. For the four different situations mentioned before, the DFS algorithm has different reaction. If the current state is a new state, which means this web page does not

visited before, this state will added into the automata and all clickable elements will made as an event and added into the event list. If the currenrt state is an old state or is as same as the last state, there will be no new events added and continue next event. If the URL is out of the domain, WebTraceCollector will ignore this state and back to the last state.

Chapter 5

Trace Evaluation

The framework of testing is shown in Fig 5.1. The work of testing devided into trace collection and trace evaluation. At last chapter, we construct a tool to generate the traces of the websites automatically. We use the tool SpecElicitor to help us making traces of Mobile Applications with the labels. Even though we can easily generate lots of traces , it is still a heavy works to evaluate all traces. Because the traces may lead to different result by some slight difference, it needs people to check the traces case by case. We develop a method to teach computer how to learn predicting a trace.

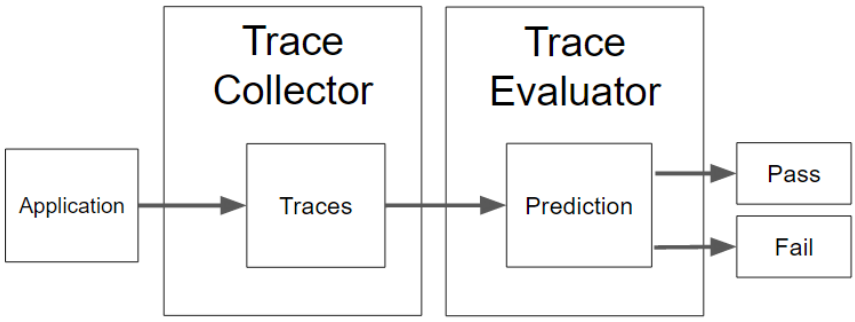

Figure 5.1: The framework of automated testing.

5.1 Procedure

The testing framework is shown in Fig[5.2]. An automated testing system can be devided into two parts, one is automatically generating traces anf the another one is automated predicting the traces. In the last chapter, we introduced a tool for automatically testing dynamic web applications. With the tool WebTraceCollector and the tool SpecElicitor introduced in chapter 3, we can collect web traces and android traces automaitcally.

In order to automated evaluating traces, we need to construct a test oracle. The test oracle may distinguishe the correct and incorrect behaviors of the system under test. The simple ways are checking the behaviors manually, inserting assertions and detecing crash, error or failure in traces. In our work, we use the machine learning method to achive the goal of evaluating traces automatically. The method of machine learning we used in this paper is Support Vector Machine, which needs collecting training set and testing set to train the predictive model. Because the SVM is a supervised machine learining algorithm, we have to labeled all traces for training.

The system will repeat the process of collecting traces of certain applications and labeled the traces for training SVM model until the accuracy of prediction is well enough. The problem we occured will be how many traces we should collect to train a predictive model and how to select the traces as training set and testing set.

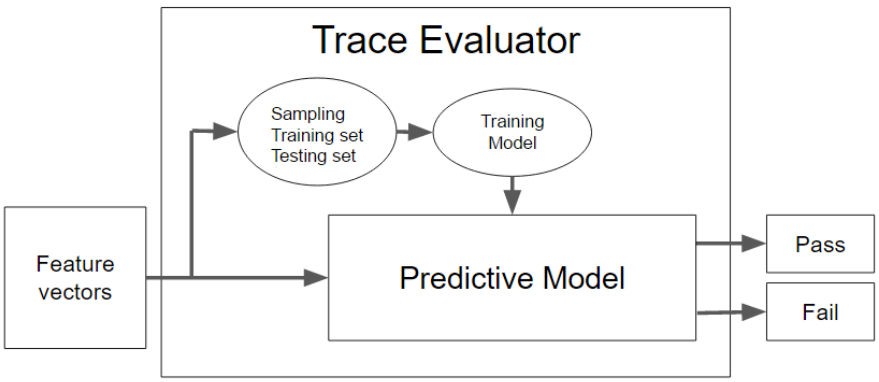

Figure 5.2: The framework of testing.

5.2 Feature vector

We hope that the method can not only predict the traces of one applications but also works on the traces between different application, even on the different platform, so we need a specified defination of features that can work on different type of traces.

The tool SpecElicitor built a common sense model for labeling screens and actions during the test. According to the common sense model, SpecElicitor can merge automata generated by different applications into one automata consist of label. We select the normalized terms user labeled during the testing. By generalizing the traces of similar applications, we construct a keyword dictionary. The keyword dictionary is independently built on the database, so we can easily updating the keywords. The more traces we collect from several similar applications, the more normalized keywords and it's synonyms we generalize.

We use the keyword dictionary as the specified feature used in trace evaluation. In the case of the Android traces generated by SpecElicitor, we directly get the feature of each states and edges. By collecting those features, we convert the Android traces into a feature vector. In the case of the traces generated by WebTraceCollector, we need to extract the feature by string comparing between keywords and it's synonyms with the DOM tree of every state.

With the keyword dictionary, we can extract the features from the traces of web applications and mobile applications and generate the feature vectors. The vector is consist of integer numbers, which means the times the specific keyword appearing in the trace, and the position of keywords in every vectors are stable. A partial of feature vectors is shown in Fig 5.3.

We generalize the similar behaviors and represent the behaviors in normalized keywords between similar applications. Therefore, the feature vectors extracted from traces of similar applications have same feature dimensions. We can repeatedly use the keyword dictionary to automated extract features for reducing human effort. After generating feature vectors, the SVM will training the model accroding the types of applications. It is obvious that the keywords can seriously affect the training result. If we can not precisely

represent the incorrect behavior or failure in the traces by normalized keywords, It will be hard to successfully extract the feature from traces. However, the work of collecting keywords is a heavy work and can not be quickly complete. We need constantly updating the keyword dictionary to increase the precision of keywords.

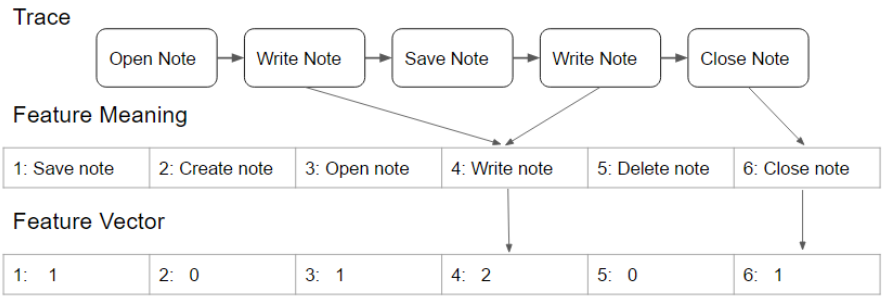

Figure 5.3: Extracting the feature vectors.

5.3 Sampling traces

Before using the traces, we need to label it. The precess of labeling traces as passed and failed is a hard work for human, but it is necessary and can not be avoided in software testing.

How to select the traces to label into training set is another important problem. If we can devide the traces precisely, it can reduce the training work and the perfomance of predicting will be great. However, it is too hard to find out which trace has the most informative feature.

In order to reduce human work, The active learning algorithm can help us find out which traces should be labeled. Active learning is a subfield of machine learning. The studies about this field are gatherd by Settles[18] The active learning algorithm depends on the sampling strategy. so we need a method to sample the trace more efficiently. There are 3 sampling methods we could use, which are randomly sampling, greedy sampling and uncertainly sampling. Randomly sampling just randomy pick a trace into training set. Greedy sampling select the trace which can maximize the feature coverage. Uncertainly sampling select the trace whose prediction is the least confident. We will implement them to find out which sampling method is the suitable to this system in the future.

Chapter 6

Experiments

We construct a tool WebTraceCollector for automatic web testing, which can explore the website automaically and analyze the DOM tree of the current web page to find the next clickable element to click, and propose a machine learning method to evaluate traces by SVM. In order to use the tool and implement the evaluation, wee need a laptop and installing some tools. The specification of the laptop we used is shown in Table[6.1], and we need to install Windows, Python 3.4.3 and MySQL database.

Laptop	AUSU X450JN
Operating System	Windows 10 (64bit)
CPU	Intel(R) Core(TM) i5-4200 @2.8GHz
RAM	4GB
Storage	800G

Table 6.1: The specification of the laptop.

WebTraceCollector is constructed based on Python, it control the browser by the selenium library and recognize the elements of the current page by the beautifulsoup library. The automatic exploring mechanism highly depends on the page source downloaded from the current web page. If the page source ot the constructed DOM tree can not match the current page correctly, we can not find the correct clickable element and the testing may fail. Thus, it is important to check how many web applications can download page soucre and construct DOM tree accurately. In the experiment 1, we find 20 websites which is famous in taiwan. We test those websites and generate traces automatically by

WebTraceCollector. Moreover, we want to check the ability of handling dynamic web pages. We choose some common web pages with lots of input fields need to be inserted to test in the experiment 2. In the experiment 3, we want to check the accuracy of the automatic prediction. We generate some traces of certain applications and use those traces to train SVM model, and we use this model to predict the traces of the applications.

6.1 Trace collection

In 50 popular applications[19] that shown in Table 6.2, there are forums, news, community platforms, games and blogs. We generate monkey traces by randomly click buttons and the length of traces is about 5. The reason that we do not completely explore the whole website is most of the scale of websites are too big. For example, a news website may have thousands of subpages and it may cost too much time to generate completely traces. We just want to check the DOM tree is constructed correctly, so we make short traces to focus on checking the ability of clicking correct clickable elements.

The result of the experiments is shown in Table[]. There are few web applications that have serious problem on testing, and most of web applications can be successfully explored with little defects. There are two unkown fail happened, however it can work after restart the test. The problems of the fail applications are too many iframes of advertising and API plugin in the webpages and buttons functions implemented by javascript. The iframes become noise in the page source and make the DOM tree constructed incorrectly, so WebTraceCollector may not find the correct clickable elements or only focus on the clickables in the advertising iframe. On the other hand, the button functions in the web pages may implemented by javascript. That means clickable elements may not be the style we thought, such as link tags or input-button tags, it could be images, div tags or any tags if javascript bind the mouse listener on. With the uncertain clickable elements, it is hard to find the correct clickable elements to explore the web and record the traces. WebTraceCollector will click the wrong elements with nothing happens and stay at the same page until the trace is end.

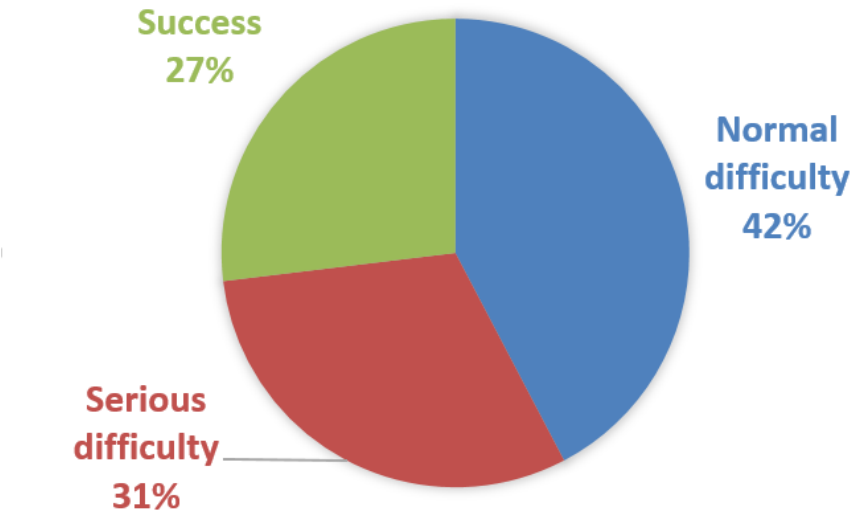

Figure 6.1: Test issue of popular applications.

編號	網站名稱	網站類型	URL
1	Facebook	Community platform	https://www.facebook.com/
2	YouTube	Entertainment	https://www.youtube.com/
3	Yahoo	Search engine	https://tw.yahoo.com/
4	Google	Search engine	https://www.google.com.tw/
5	中時電子報	News	http://www.chinatimes.com/
6	露天拍賣	Commerce	http://www.ruten.com.tw/
7	聯合新聞網	News	http://udn.com/news/index
8	巴哈姆特	Forum	http://www.gamer.com.tw/
9	Mobile01	Forum	http://www.mobile01.com/
10	蘋果日報	News	http://www.appledaily.com.tw/
11	百度	Search engine	https://www.baidu.com/
12	東森新聞雲	News	http://www.ettoday.net/
13	卡提諾論壇	Forum	http://ck101.com/
14	伊莉討論區	Forum	http://www40.eyny.com/index.php
15	Hinet	Search engine	http://www.hinet.net/
16	微軟Live.com	Search engine	https://login.live.com/
17	痞客邦	Blog	https://www.pixnet.net/
18	PChome Online	Search engine	http://pchome.com.tw/
19	淘寶	Commerce	https://world.taobao.com/
20	Life生活網	Blog	http://www.life.com.tw/
21	104人力銀行	Survice	http://104.com.tw/
22	騰訊網	Search Engine	http://www.qq.com/
23	新浪微博	Community platform	http://www.weibo.com/login.php
24	自由時報電子報	News	http://www.ltn.com.tw/
25	維基百科	Dictionary	https://www.wikipedia.org/

Table 6.2: The 20 popular websites.

26	博客來	Commerce	http://www.books.com.tw/
27	今日新聞網	News	http://www.nownews.com/
28	商業周刊	News	http://www.businessweekly.com.tw/
29	隨意窩Xuite	Blog	http://xuite.net/
30	1111人力銀行	Survice	http://www.1111.com.tw/
31	Flickr	Community platform	https://www.flickr.com/
32	Blogger	Blog	http://www.blogger.com/
33	批踢踢實業坊	Community platform	https://www.ptt.cc/index.html
34	FC2	Survice	http://fc2.com/
35	卡卡洛普	Survice	http://www.gamme.com.tw/
36	PChome商店街	Commerce	http://www.pcstore.com.tw/
37	Giga Circle	Survice	http://tw.gigacircle.com/
38	鉅亨網	News	http://www.cnyes.com/
39	momo購物網	Commerce	http://www.momoshop.com.tw/main/Main.jsp
40	蕃薯藤	Search engine	http://www.yam.com/
41	OB嚴選	Commerce	http://www.obdesign.com.tw/
42	GOMAJI	Commerce	http://www.gomaji.com/Taipei
43	愛評網	Survice	http://www.ipeen.com.tw/
44	123KUBO酷播	Entertainment	http://www.123kubo.com/
45	591租屋網	Commerce	https://www.591.com.tw/
46	TEEPR趣味新聞	News	http://www.teepr.com/
47	KKBOX	Entertainment	https://www.kkbox.com/tw/tc/index.html
48	msn台灣	Search Engine	http://www.msn.com/zh-tw/
49	微軟	Survice	https://www.microsoft.com/zh-tw/
50	T客邦	Blog	http://www.techbang.com/

6.2 Dynamic webpages

In order to test the ability of explore dynamic webpages, we select some webpages that have lots of input fields and it need to insert correct values to pass through the web pages. The selected webpages are shown in Table 6.3. Most of the input fields in those web pages are names, years, emails and so on. We can see that there are several web pages can be successfully passed through, but few web pages can not. There are 3 reasons that can not pass through the dynamic web pages are the follows and the examples arw shown in Fig 6.2,6.3 and 6.4.

1. The web pages have robot checking mechanism to prevent automatic exploring.
2. The form of input fields are different to the examples in the database.
3. The input fields are out of the range of examples we collecteed.

In reason 1, the robot checking like image recognition is developed to prevent people exploring the website automatically be computer. It is too hard to find out the correct value by program, and it is unreasonable if we can easily pass through those web pages. If we want to test those websites, the reasonable way is let the developers of the websites turn off the robot ckecking.

In reason 2, even though we have collect the examples of the input fields in the database, it is still a hard work to find out the correct form of the values. For instance, we analyze the input field and find out the value should be a string of address. But the type of input fields can be text, checkbox or selects, it is too hard to analyze user should insert the whole string of address or select each partens of area.

In reason 3, the web pages can have unlimited kinds of input fields. Although we can add as more examples and rules in the database as we possible, it is still impossible to handle all kinds of input fields by only few kinds of examples. In future work, we can develop to find out the values by other method such as maching learning.

Figure 6.2: The example of difficulty 1: Robot checking.

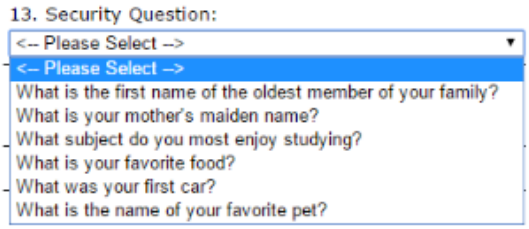

Figure 6.3: The example of difficulty 2: Input fields cannot find examples in database.

Figure 6.4: The example of difficulty 3: Unsuitable example format .

URL	Result
http://www.plurk.com/signup	Pass
https://ups.moe.edu.tw/Personal_Page/index.php	Pass
https://applyweb.collegenet.com/account/new/create	Pass
https://apply.grad.ucsd.edu/signup	Pass
https://user.gamer.com.tw/regS1.phpt	Fail
http://panel.pixnet.cc/signup/step2	Fail
https://apps.grad.uw.edu/applForAdmiss/newUserProfile.aspx?bhcp=1	Fail
https://oauth.life.tw/auth?appId=1&redirect=http	Pass
https://cart.books.com.tw/member/join_step1	Fail
https://login.yahoo.com/account/create?.src=flickrsignup	Pass
http://www40.eyny.com/member.php?mod=register	Fail

Table 6.3: The dynamic webpages.

6.3 Prediction

In this experiment, we focus on the ability of automated predicting passed traces and failed traces. We select two web applications to use in our study and analysis their results. The first web application is a small website about restraurant with simple buttons, and the second web application is a more complex website about APP store. Before comparing the results of predicting traces between different applications, we should collect the passed traces and failed traces from those web applications first. Thus, we generate the random traces automatically by the tool WebTraceCollector introduced above and select certain buttons and screens as the failed traits.

The traces with the specific buttons or screens will be labeled as failed traces and the other traces will be labeled as passed traces. The detailed information of traces collected from the two web applications is shown in Table 6.4 . We can see that the percentage of failed traces is low. The reason is that if the bugs are not occur at the main functionf of the applications, there will be only a few traces triggering the bugs.

After collecting traces, the traces should be divided into training set for training SVM model and testing set. We randomly devide the traces into two sets, and repeat the experiments with different proportion of training set. The result of prediction traces of application 1 is shown in Table 6.5. Notice that the accuracy of prediction may not be better than just predicting every traces to pass, because the number of failed trace is smaller than the number of passed trace.

Even though the prediction can not enhance the accuracy, it still can help people finding failed traces. The precision is the number of traces predicted failed that actually failed devided by the number of traces predicted as failed, which means the accuracy of predicting failed trace. With the higher precision, it will be more convinced the traces predicted to be failed is actually failed. If the precision is high enough, it can reduce human effort by predicting the failed traces with the convinced model.

The recall is the number of traces predicted failed that actually failed devided by the number of failed traces, which means the ability of finding the failed traces in all traces. With the higher recall, it will be more convinced that the system can find all the failed

traces. If the recall is high enough, it can reduce human effort by finding the failed traces with the convinced model.

We can obeserve that the accuracy of prediction in application 1 is about 0.65 0.75. The average of precision is about 0.5 and the average of recall is about 0.45 but larger range. From the result, we can say that the predictive model can find almost half of the failed traces in testing set, and the half of the predicted traces are actually failed.

The result of prediction traces of application 2 is shown in Table 6.6. We can obeserve that the accuracy of prediction in application 1 is about 0.75 0.85. The average of precision is about 0.5 and the average of recall is about 0.45 but larger range. However, there are models that have very low precision and recall in some iterations. It seems that sometimes SVM can not build a reliable model from training. Because the web application 2 is more complex than application 1 and the feature keywords is not precise enough to present the difference between passed traces and failed traces. It will be hard to learn the predictive model of application 2. The proportion of negative labels is another problem. Because the amount of failed traces of application 2 is less than the failed traces of application 1, The method of deviding traces into training set and testing set has a greater influence on learning model. If we can not find the most important trace for training, it will be difficult to learn the model and result in poor outcome.

Although the results of automated prediction shown in the above experiments is not good enough to replace all human effort, it is still a feasible method for reducing the cost of web application testing. In order to increase the accuracy of automated predicting, the future work we need to do is to collect more keywords precisely and find other method of deviding traces into training set. We can select the traces to trainging set depending on which trace can maximize the feature coverage or which trace has the least confident. We can also use other machine learning algorithm, such as deep learning, to construct the predictice model.

Web Application	Total traces	Pass traces	Fail traces	Negative Proportion
Restraurant	122	86	36	29.5%
APP Store	94	84	10	10.63%

Table 6.4: The number of passed traces and failed traces.

Application 1	Training	Accuracy	Precision	Recall
Repeat 1	0.4	0.7436	0.5	0.4
Repeat 2	0.4	0.75	0.64	0.5
Repeat 3	0.4	0.7333	0.53	0.69
Repeat 4	0.4	0.7209	0.63	0.46
Repeat 5	0.4	0.6667	0.45	0.45
Repeat 6	0.6	0.7272	0.5	0.5
Repeat 7	0.6	0.7307	0.5	0.43
Repeat 8	0.6	0.6667	0.6	0.43
Repeat 9	0.6	0.7407	0.67	0.45
Repeat 10	0.6	0.750	0.57	0.67

Table 6.5: The prediction of Application 1.

Application 1	Training	Accuracy	Precision	Recall
Repeat 1	0.4	0.833	0.4	0.66
Repeat 2	0.4	0.891	1.0	0.33
Repeat 3	0.4	0.857	0.5	0.33
Repeat 4	0.4	0.920	1.0	0.33
Repeat 5	0.4	0.8787	0.5	0.25
Repeat 6	0.6	0.85	0.3	0.5
Repeat 7	0.6	0.88	0.4	1.0
Repeat 8	0.6	0.666	0.125	0.5
Repeat 9	0.6	0.8	0.2	1.0
Repeat 10	0.6	0.6363	0.1	1.0

Table 6.6: The prediction of Application 2.

Chapter 7

Conclusion

7.1 Summary

In this paper, We present a tool named WebTraceCollector to automated test dynamic web pages in order to reduce human cost on testing. WebTraceCollector download the page source from the current web page and find the target clickable elements and input fields. We collects examples of input fields and construct a database. WebTraceCollector analyze the input fields and get the suggested value to pass through the dynamic web page.

Moreover, We propose a system to automated evaluating traces and use Support Vector Machine to predict traces. We collects the common sense behavior of certian applications and make a keyword library. The system use the keyword as features and extracts the features from the traces. with selecting the traces as training set and building a predictive model, the system can automated predict traces to passed traces and failed traces.

7.2 Limination

The tool can not successfully test all websites and dynamic webpages for some reasons. The advertising iframes and plugins make noise on the page source and lead to wrong DOM tree and functions implemented by javascript may change the tags format of clickables, so it increase the difficulty to find the correct clickable elements and input fields.

The ability of getting suggested values of input fields completely depend on the

database, so we need more and more amounts of input examples. However, the dynamic webpages can be turn into a variety of styles. It seems too hard to handle dynamic webpages only with string comparing of input examples. It is important to find out other method to guess the correct input values.

On the other hand, the prediction method is based on the support vector machine. The accuracy of prediction highly depends on the feature extracted from thr traces. If we can not find the precise keywords to represent the failed traces, it will be hard to train the SVM model and increase the accuracy.

7.3 Future work

In order to automated test web applications successfully, we need to find out other methods to filter the noise of page source and correctly recognize the actual elements with functions. To increase the ability of passing through dynamic pages, we not only need to add more input examples into the database, but also find out other methods, such as natural language processing, to guess input values more correctly.

In the future works, we have to generalize more behaviors of similar applications to make the feature more precise. We will use other machine learning methods to predict traces, such as deep learning, and compare the accuracy between different methods. We want to find out which methods is most suitable to this automated evaluation system.

Bibliography

[1] F. Ricca and P. Tonella, "Analysis and testing of web applications," in *Proceedings of the 23rd international conference on software engineering icse 2001*, 2001, pp. 25–34.

[2] *List of web testing tools*. [Online]. Available: `https://en.wikipedia.org/wiki/List_of_web_testing_tools`.

[3] *Apache jmeter*. [Online]. Available: `http://jmeter.apache.org/`.

[4] *Selenium*. [Online]. Available: `http://www.seleniumhq.org/`.

[5] *Test studio*. [Online]. Available: `http://www.telerik.com/teststudio`.

[6] *Smart bear*. [Online]. Available: `https://smartbear.com/product/testcomplete/overview/`.

[7] M. Benedikt, J. Freire, and P. Godefroid, "Automatically testing dynamic web sites," in *11th int conf. world wide web (www02)*, 2002.

[8] A. Marchetto, P. Tonella, and F. B. Kessler-IRST, "Search-based testing of ajax web applications," in *Ieee - search based software engineering*, 2009, pp. 3 –12.

[9] *Crawljax*. [Online]. Available: `http://crawljax.com/apidocs/`.

[10] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing:a survey," in *Ieee transactions on software engineering*, 2015, pp. 507–525.

[11] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *Ieee 24th international symposium on software reliability enginerring issre2013*, 2013, pp. 1–10.

[12] M. D. Ernst, J. Cockrell, W. G. Griswoldand, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," in *Iee etransactions on software engineering*, 2001, pp. 99–123.

[13] C. Cortes and V. Vapnik, "Support-vector networks," in *Machine learning 20(3)*, 1995, pp. 273–297.

[14] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," in *Acm transactions on intelligent systems and technology (tist)*, 2011, pp. 1–39.

[15] Peter-Paul and Koch, "The document object model: An introduction," in *Digital web magazine*, 2001.

[16] *Ajax: A new approach to web applications*. [Online]. Available: `http://adaptivepath.org/ideas/ajax-new-approach-web-applications/`.

[17] *Beautiful soup document*. [Online]. Available: `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`.

[18] B. Settles, “Active learning literature survey,” in *Machine learning*, 2010, pp. 201–221.

[19] *Popular websites in taiwan*. [Online]. Available: http://www.bnext.com.tw/article/view/id/35475.