Automated Testing of Android Applications Using Machine Learning

Dhruv Mehta, Afamefuna P Umejiaku, Anubhav Tiwari, Ayodeji Ejiade

# Introduction

Android applications occupy a dominant share in the mobile device market reaching over 70 billion downloads in 2018 and a yearly growth of about 7% [2], this has caused the android software industry to be faced with the ever-increasing need for effective and efficient testing methodologies for these applications, this is very critical given the various configuration of devices available in the industry.

Manual software testing methods have been traditionally used and have proven to be very expensive and less efficient, as significant resources are needed to hardcode test cases for every single application. Sustenance of manual testing methods could become impracticable especially with the intense proliferation of the market with different devices and ever-changing mobile platforms. Asides that, most of these test testing methods are mostly capable of finding technical bugs with low capabilities for catching logic bugs in the android applications.

The challenge on how to efficiently and effectively test android applications is not new to researchers, several researchers have been working on the automation of the software testing processes and companies now use multiple automated tools to ensure the effectiveness of the testing processes, very little work has been done in utilizing the vast capabilities of machine learning tools for software testing processes simplification.

The work we will model leveraged on the fact that android applications share a similar pattern in design of the user interfaces and the authors explored this advantage to develop a test tool that can be used across different android applications even with very little knowledge of the application structure. We will adopt Rosenfeld et al approach on classification of activities on android applications into seven pre-defined types by extracting features of the UI, we will develop a tool similar to that developed by them to run automated tests for each activity type that have very high likelihood of finding functional bugs. We will utilize TestProject, which is a cloud hosted, free test automation platform built on Selenium and Appium, used by testers and developers to perform user interface tests.

# Implementation

Having studied related literature, we have broken down the work into phases and the first release includes the following:

1. **Identification of the 7 types of activities on android applications:** For this project, we referred to the preliminary study of the published paper [3] and confirmed by manually searching for common patterns, structures and behaviors in the different activities. Based on this, we adopted the seven activity types. These activity types are: Splash, Advertisement, Browser, Mailing, Login, Portal and To-do List activity.
2. **Construction of a features vector:** By examining the basic activity templates from the Android studio activity, we can say that each activity screen can be divided into 3 parts: the top, the middle and the bottom. The division can be defined as 20%-60%-20% from top to bottom.

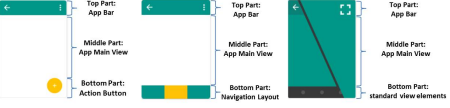


Figure 1: Screen Divisions across common activities templates [3]

1. **Identifying the different feature groups in the applications:** We would divide the elements into following element groups

* Clickable elements: Elements that are responsive to the user touch click.
* Horizontal swipe able elements: Elements that can be swiped by the user left and right.
* Vertical swipe able elements: Elements that can be swiped by the user up and down.
* Text field elements: Elements that the user can type text into them.

Our feature dataset will look something like the below table once we have identified feature groups and create our training dataset.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Feature Location** | **Feature Group** | **Count** | **Application Name** |
| 1. | Middle Section | Clickable | 5 | Gmail |
| 2. | Middle Section | Vertical Swipe able | 2 | Gmail |
| 3. | Top Section | Text field | 2 | Browser |
| 4. | Bottom Section | Horizontal Swipe able | 1 | Contact |

## Android Activities

Unlike currently well-established programming paradigms where applications are launched with a ***main()*** method, Android systems use activity instances to initiates codes. Android developer's website guide states that *“Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.”*

The use of activity instances is necessitated because unlike user interaction in desktop applications, a mobile application does not always begin in the same place. This allows an application to invoke another applications activity rather than the application as an atomic whole. In this way, the activity serves as the entry point for an application’s interaction with the user. Hence developers are expected to implement an activity as a subclass of the Activity class where an activity provides the window in which the application draws its UI [1].

Android Developers using Android Studio 3 are expected to use one of the below listed activities when developing applications for Phones and tablets as templates.

1. Basic Activity - creates a new activity with an Application Bar.
2. Empty Activity – creates a new activity that’s empty.
3. Bottom Navigation Activity – creates a new activity with bottom navigation used to create UI that allow Horizontal scrolling
4. Fragment + ViewModel – creates a new activity with a fragment and a view model. This is used to handle UI related changes in a lifecycle conscious manner. Each activity could have several fragments that share one ViewModel allowing communication between fragments
5. Full Screen Activity – creates a new activity that toggles the visibility of the systems UI and action bar upon user interaction
6. Master/detail Flow- creates a new master/detail flow, allowing users to view collection of objects as well as details of each object. The template creates two activities, a master fragment and a detail fragment. On larger screens like tablets, the flow is presented using two column and smaller screens one column.
7. Navigation Drawer Activity – creates a new activity with a navigation drawer.
8. Google map Activity – used to create activities with map.
9. Login Activity – used to create Registration and login activities. Its design in a manner that to handle passwords, emails and other registration information.
10. Scrolling Activity – creates a new Vertical scrolling activity
11. Tabbed Activity – creates a blank activity with tabs.

Rosenfeld et al in their paper stated in 2017 that having compared most application developed for android devices and the type of activities they perform, they stated that by classifying these activities into seven Activity types stated below the were accounting for about 80% of android applications. The seven Activity types could be likened to those in Android Studio with added features such as text boxes, Images and buttons. They classified the applications in this manner so that they could develop test cases that will be applicable for various applications using same activity types.

## Introducing the seven activity types

In this section, we will be looking at the seven activity types earlier mentioned, highlight the linkage with the activities on android studio, specific features and recommend test cases for each of them.

1. **Splash Activity:** Splash activity is the screen that is displayed initially when opening the activity, it displays an image or text while the application is loading in the background. If it exists, its normally the first screen in the application.

## 

## Basic Android Activities:

* An Empty Activity

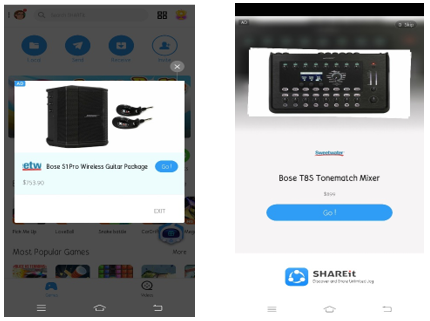
**Feature Groups:**

* Image/Animation containing the logo of the current application.
* TextView field containing the name of the application

**Test Case:**

* The test for this activity is to make sure it directs to the next desired page in the application.

1. **Advertisement Activity:** Android developers commonly incorporate advertisements in their applications in order to compensate for free downloading of the applications. Most common source of advertisements in android application is through Google AdSense. These advertisements can pop-up at any point in the application and need to be classified and identified properly. Testing the ad should take into consideration that in most of the cases clicking on ads leads to exit from the application to a browser and thus testing should ensure that it does not exit from the scope of the application.



**Basic Android Activities:**

* + An Empty Activity
  + A Fragment + View Activity

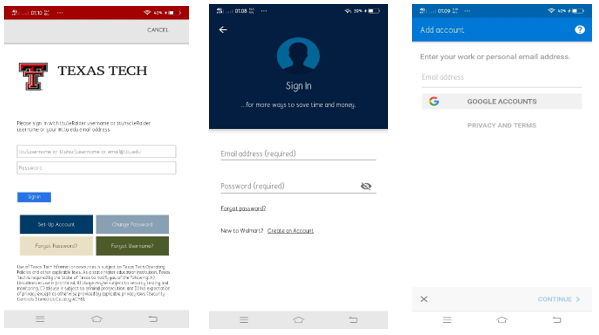
**Feature Groups:**

* **Clickable elements:** Close/Skip button to exit the advertisement.
* Image/Videos/Animations containing the advertisement.
* Timer to display the exit time of the advertisement.

**Test Case:**

* To ensure that the close/skip button ensures the closure of the advertisement and takes the user back to the desired page.

1. **Login Activity:** Most android applications contain the login activity that includes the username and password fields to ensure correct user login into the application.



**Basic Android Activities:**

* A login activity

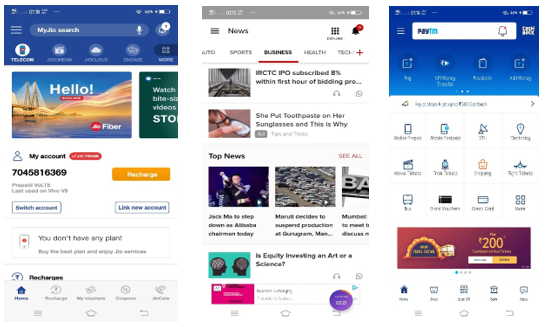
**Feature Groups:**

* **Clickable elements:** Button to view the entered password, Hyperlink to change the password (Forgot Password), Button to Sign In/Login after the credentials have been entered, Button to Create an account for the new user
* **Text field elements:** Text field to enter the Username/Mobile Number/Email Id, Password field to enter the Password
* Image/Animation that displays the logo of the application

**Test Case:**

* Click on the Sign In button without entering any credentials to ensure that an error message is displayed to tell the user to enter the credentials.
* To enter either of the Username/Password and then trying to sign in to the system to ensure that an error message is displayed to tell the user to enter the value in the required field.
* To enter invalid credentials to ensure that when the user tries to sign in to the system an error message is shown to enter the valid credentials.
* To enter valid credentials to ensure that the user is directed to the correct page after login.
* To click on the hyperlink of Forgot Password to ensure that user is directed onto the page to change the existing password.
* To click on the Create an account button to ensure that user is directed to a page to enter the data for a new account.

1. **Portal Activity:** Android applications of today’s communication medias such as websites, newspapers, tv channels etc. contain a portal activity that is used to display all the information to the user which can be accessed by them to accomplish various tasks. This screen consists of navigation bars, screen that can be swiped left or right to access different sections of contents, scrolled up or down to access various articles, they have buttons that are used to accomplish their designated tasks etc.



**Basic Android Activities:**

* An Empty Activity
* A Fragment + View Activity
* A Bottom Navigation Activity
* A Fullscreen Activity
* A Master-Detail Flow Activity
* A Navigation Drawer Activity
* A Scrolling Activity
* A Tabbed Activity

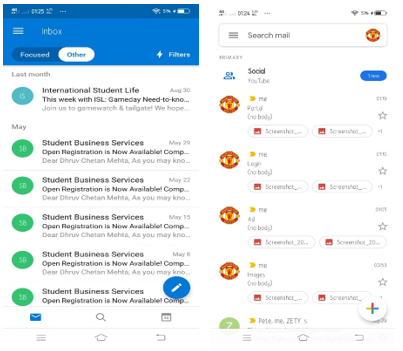
**Feature Groups:**

* **Clickable Elements:** Navigation bars that can be at the top/bottom of the screen, buttons that can be at either of the top/middle/bottom of the screen to accomplish a certain task.
* **Horizontal swipeable elements:** Screen can be swiped left/right to access different sections of contents
* **Vertical swipeable elements:** Screen can be scrolled up/down to access different articles on the screen

**Test Cases:**

* To ensure that the navigation bar with different menus opens as required when the button for the navigation bar is clicked.
* To ensure that buttons when clicked lead to the desirable page or complete their desirable task.
* To ensure that an article can be opened from the screen
* To check whether the screen can be swiped left/right to different sections of content
* To check whether the screen can be scrolled up/down to access different articles.

1. **Mail Activity:** It is the hub screen of the android mail applications which have fixed and limited functionalities such as message management, composition, and reception etc. which allows us to create standard set of test cases.



**Basic Android Activities:**

* A Basic Activity
* A Fragment + View Activity
* A Navigation Drawer Activity
* A Scrolling Activity
* A Tabbed Activity
* A Login Activity

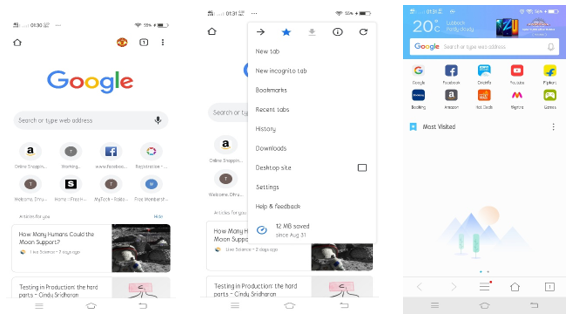
**Feature Groups:**

* **Clickable elements:** Navigation bars that can be at the top/bottom of the screen, buttons that can be at either of the top/middle/bottom of the screen to accomplish a certain task.
* **Horizontal swipe able elements:** The mails on the screen can be swiped left/right to perform various functionalities such as to archive them, delete them etc.
* **Vertical swipe able elements:** Screen can be scrolled up/down to access different articles on the screen
* **Long-clickable elements:** The mails on the screen can be considered as long-clickable elements which when pressed for long display various options either to delete the mail, archive the mail etc.

**Test Case:**

* To browse through a list of inbox mails
* To open a mail at random from the list of inbox mails
* To compose a new mail
* To send a mail without entering the recipient mail id to ensure an error is shown to enter the recipient mail id
* To send a mail by entering invalid recipient mail id to ensure an error is shown to enter valid recipient mail id
* To send a mail by entering valid recipient mail id to ensure a mail is sent
* To click on the navigation bar and to navigate to the sent email list to ensure the mail is sent and the navigation menus work properly
* To test the long clickable elements by long clicking the email and performing one of the available functionalities such as deleting the mail, archiving the mail

1. **Browser activity:** A browser activity can be used in applications such as web browser to access the internet on mobile devices for tasks such as visiting websites and to do activities within them such as login, view multimedia, link from one page to another, send and receive mails etc.



**Basic Android Activities:**

* An Empty Activity
* A Fragment + View Activity
* A Bottom Navigation Activity
* A Fullscreen Activity
* A Master-Detail Flow Activity
* A Navigation Drawer Activity
* A Scrolling Activity
* A Tabbed Activity
* A Google Maps Activity
* A Login Activity

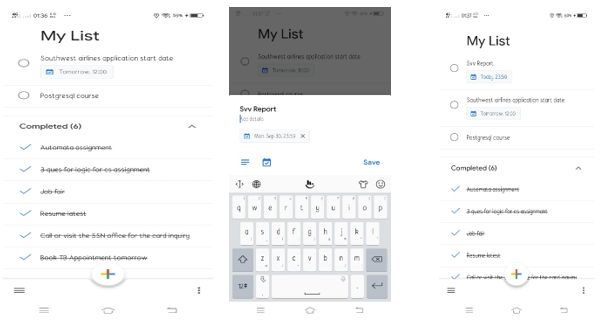
**Feature Groups:**

* **Clickable elements:** Buttons for performing various functionalities such as moving backward and forward, back to home, opening a new tab, refresh a page, bookmark a page etc. Navigation Bar for displaying various functionalities menus such as view history, downloads, settings etc. Images/animations on the browsers can be clicked to view them in a separate frame or to get more information about them.
* **Horizontal swipe able elements:** To swipe left/right across various contents on the websites
* **Vertical swipe able elements:** To scroll up and down on the webpage to view the article
* **Text field elements:** To enter the URL of the website to be visited
* **Long-clickable elements:** The images/videos/animations can be long pressed which will mark them and show options to share them, save them, open them in a new tab etc.

**Test Case:**

* To test all the buttons for their desired functionalities such as moving backward, forward, back to home, download etc.
* To test the navigation bar to ensure all the menu functionalities in it works as desired
* To click on the images/animations to ensure they are directed to the correct page
* To swipe left/right across web pages
* To scroll up/down the page to view the whole content on the screen
* To enter an invalid URL to ensure an error message is shown to enter the correct URL
* To enter a valid URL to ensure the correct web page opens up
* To ensure when the image/videos/animations are pressed for a few seconds they display options to share, save them etc.

1. **To Do List Activity:** To do list android applications allow user to store their list of tasks that have to be completed within a specific time frame.



**Basic Android Activities:**

* An Empty Activity
* A Basic Activity
* A Navigation Drawer Activity
* A Scrolling Activity

**Feature Groups:**

* **Clickable elements:** Navigation bar for various functionalities, button to add new tasks, checkbox to tick off completed tasks, delete a task etc.
* **Horizontal swipe able elements:** Some of the to-do list android applications allow to tick off a task as completed by swiping the task either in left/right direction
* **Vertical swipe able elements:** To scroll up/down to view the list of tasks
* **Text field elements:** To enter the task and list of subtasks to be completed, date and time field to enter the date and time by which the task must be accomplished.

**Test Case:**

* To test the navigation bar to check all its functionalities are executed properly.
* To test the buttons to add a new task, delete an existing task.
* To test the checkbox to ensure when it is ticked the task is removed from the list of tasks pending and added to the list of completed tasks.
* To test the horizontal swipe functionality such that a tasks checkbox is ticked off when swiped and its added to the completed tasks list.
* To test the scrolling functionality to view list of tasks
* To enter a task and list of sub tasks and to save them and to ensure they are added to the list of pending tasks.
* To ensure that the application does not allow saving of a blank task
* To enter the date in an invalid format along with the tasks to ensure an error message is shown letting the user know that date is entered in the wrong format
* To enter the task along with the date in the valid format

# Dataset Creation

## Introduction [Ayodeji Ejiade]

To enable our tool extract features and classify android activities effectively, we need to train our machine learning model with real-world data, this part of the work is very crucial as we need to get the right activity classification before we can execute the test cases on the individual activities, recall that we have adopted the seven predefined activity types to classify all activities in an android application and will be using the features vector introduced in the second chapter of this report.

We have picked 5 android applications to build an initial dataset for all different activity types. Given that an application can contain several activities, while this should be enough to train the model, we shall however improve this by building more data when we review the accuracy of the final tool. We have picked our samples from very popular android applications like Gmail, yahoomail, outlook, quick memo and Sololearn to build the dataset.

To get accurate data from our dataset, we employed both the ***manual*** and ***automated*** methodsby running through all the activities of the application and at the same time using the ***element explorer*** in the Test Project framework to extract the properties of the features in the application and storing these data in a CSV file, the methods to extract these features will be described in later sections.

Using feature groups described in chapter 2 together with the location of the feature and general features independent of location adopted in the experiment we are implemented; we came up with 15 features described below to classify our activities.

| ID | Features | Unit |
| --- | --- | --- |
| F01 | Number of clickable elements in the top section of the screen | Integer |
| F02 | Number of clickable elements in the middle section of the screen | Integer |
| F03 | Number of clickable elements in the bottom section of the screen | Integer |
| F04 | Number of vertical swipeable elements in the top section of the screen | Integer |
| F05 | Number of vertical swipeable elements in the middle section of the screen | Integer |
| F06 | Number of vertical swipeable elements in the bottom section of the screen | Integer |
| F07 | Number of horizontal swipeable elements in the top section of the screen | Integer |
| F08 | Number of horizontal swipeable elements in the middle section of the screen | Integer |
| F09 | Number of horizontal swipeable elements in the bottom section of the screen | Integer |
| F10 | Number of text fields elements in the top section of the screen | Integer |
| F11 | Number of text fields elements in the middle section of the screen | Integer |
| F12 | Number of text fields elements in the bottom section of the screen | Integer |
| F13 | Number of general elements | Integer |
| F14 | Number of long-clickable elements | Integer |
| F15 | Does the activity have a navigation Drawer | Boolean |

Table 3‑1: Mapping of all features to be used for classification of activities

Using the raw data from the test applications, we calculated the average data from each of the features and checked for patterns across specific features in the dataset, this will help us in determining weights to each of the features.

|  |  |
| --- | --- |
| **#** | **Activity Type** |
| **1** | Portal |
| **2** | Splash |
| **3** | Login |
| **4** | Mail |
| **5** | Browser |
| **6** | To Do |
| **7** | Advertisement |

Table 3‑2: Activity numbers and names

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Activity** | **F01** | **F02** | **F03** | **F04** | **F05** | **F06** | **F07** | **F08** | **F09** | **F10** | **F11** | **F12** | **F13** | **F14** | **F15** |
| **1** | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | F |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | F |
| **3** | 1 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 7 | 0 | F |
| **4** | 3 | 0 | 4 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 9 | 0 | T |
| **5** | 5 | 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | T |
| **6** | 8 | 2 | 2 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 12 | 3 | T |
| **7** | 1 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | F |

Table 3‑3: Average data for pre-defined activity types

The above summary was generated using raw data below gotten from the applications we surveyed.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Activity** | **F01** | **F02** | **F03** | **F04** | **F05** | **F06** | **F07** | **F08** | **F09** | **F10** | **F11** | **F12** | **F13** | **F14** | **F15** |
| **1** | 1 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | F |
| **2** | 5 | 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | T |
| **3** | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 8 | 0 | F |
| **3** | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 7 | 0 | F |
| **3** | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 7 | 0 | F |
| **3** | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 6 | 0 | F |
| **4** | 3 | 0 | 5 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 10 | 0 | T |
| **4** | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 8 | 0 | F |
| **4** | 3 | 0 | 5 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | T |
| **5** | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | F |
| **5** | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | F |
| **5** | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | F |
| **5** | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | F |
| **5** | 0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | F |
| **5** | 0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | F |
| **5** | 0 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | F |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | F |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | F |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | F |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | F |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | F |
| **7** | 5 | 5 | 1 | 0 | 1 | 0 | 2 | 4 | 2 | 0 | 0 | 0 | 10 | 6 | T |
| **7** | 9 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 13 | 1 | T |
| **7** | 9 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 13 | 1 | T |

Table 3‑4: Initial raw Data for dataset

We will be loading the generated data into our machine learning classifier algorithm to train the model which will be used to identify the type of any android activity we test.

## Methods of Feature Extraction [Dhruv Mehta]

**Feature Extraction using PageSource property and Java Xpath Parser for XML:**

1. Once we have connected the application under test (AUT) we need to extract the features of the elements present on the specific page of the application.
2. There are two ways the following thing can be implemented; in this section we will explain how we can fetch the features on the page using the PageSource property and Java Xpath Parser for XML.
3. In the first step we will set all the mobile capabilities required such as the Platform name, unique device ID, application package and activity. After this we create a driver object and provide the URL of the appium server to connect with.

//Creation of the android device capability object  
DesiredCapabilities dc = DesiredCapabilities.android();

//Setting the device capability properties  
dc.setCapability(MobileCapabilityType.PLATFORM\_NAME, Platform.ANDROID);  
dc.setCapability(MobileCapabilityType.UDID, "52b5900b");  
dc.setCapability(MobileCapabilityType.NO\_RESET,false);  
dc.setCapability(AndroidMobileCapabilityType.APP\_PACKAGE, "com.linkedin.android");  
dc.setCapability(AndroidMobileCapabilityType.APP\_ACTIVITY, ".authenticator.LaunchActivity");

//Creating the driver object and setting the url property to the appium server url  
AndroidDriver<AndroidElement> driver = null;  
driver = **new** AndroidDriver<>(**new** URL("http://localhost:4723/wd/hub"), dc);

1. Initially, when we open the app, we reset the app to get it into its initial state. After that we use the **getPageSource()** property of the driver to extract the xml page structure. The return type of the property is in string format which was stored in a string variable. If the string extraction is successful, we create a document with element and its features else an error message is displayed. Once the extraction is complete, we close the driver to terminate the operation.

//**Reset** the application data **to** **set** it **to** the intital state  
driver.resetApp();  
   
//**To** **fetch** the xml page structure **and** **create** the element-feature structure **or** **to** **show** an error **in** **case** **of** failure   
String PgSource = driver.getPageSource();  
**if** (PgSource != " ")  
 createDocument(PgSource);  
**else**  
 **System**.**out**.println("Error in opening the Application!!!");  
  
//**Close** the driver once the operation **is** complete  
driver.quit();

1. In order to create the element-feature structure we use the Java XPath Parser for XML. The steps for are used while parsing the XML document in XPath Parser:

* XML related packages should be imported
* To create a document Builder
* To create a document from a file or a stream of data
* To create an XPath object and XPath expression to fetch the all the matching nodes
* To compile the expression and store the list of fetched nodes
* To iterate over the list of fetched nodes
* To fetch the element and the values of their properties (Features)

1. ZIn the first step we import the packages to facilitate the use of Java XPath Parser for XML.

import org.w3c.dom.\*;   
import org.xml.sax.\*;   
import javax.xml.parsers.\*;   
import javax.xml.xpath.\*; import java.io.\*;

1. We create a Document builder object and using the writeUsingFiles(String Data) function we store the data in a input.xml file.

//Store xml data in file

public static void writeUsingFiles(String Data)  
 {  
 try {  
 Files.write(Paths.get("C:/Users/dhruv/eclipse-workspace/AppiumDemo/input.xml"), Data.getBytes());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }

//Creating the Document Builder object

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.**new**Instance();  
DocumentBuilder dBuilder;  
dBuilder = dbFactory.**new**DocumentBuilder();

//Create a file to store the string in the XML file in the UTF-8 encryption

writeUsingFiles(XMLSource);

1. Build XPath Factory for the data.

XPath xPath = XPathFactory.newInstance().newXPath();

1. In the next step we build the expression so as to navigate each tag in the xml and to store each element as a node in the nodelist.

String expression = "/\*";   
NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(  
doc, XPathConstants.NODESET);

1. In the next step we iterate over the NodeList to fetch each element with its feature

// Iterate over the NodeList

for (int i = 0; i < nodeList.getLength(); i++)

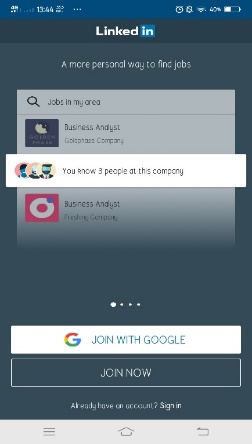
{  
 Node nNode = nodeList.item(i);  
 System.out.println("\nCurrent Element :" + nNode.getNodeName());  
   
 **if** (nNode.getNodeType() == Node.ELEMENT\_NODE)

{  
 Element eElement = (Element) nNode;  
 System.out.println("Element Name :" + eElement.getAttribute("knownSuperClass"));  
 System.out.println("Clickable : " + eElement.getAttribute("clickable"));  
 System.out.println("Long Clickable : " + eElement.getAttribute("long-clickable"));  
 System.out.println("Password : " + eElement.getAttribute("password"));  
 System.out.println("Scrollable : " + eElement.getAttribute("scrollable"));

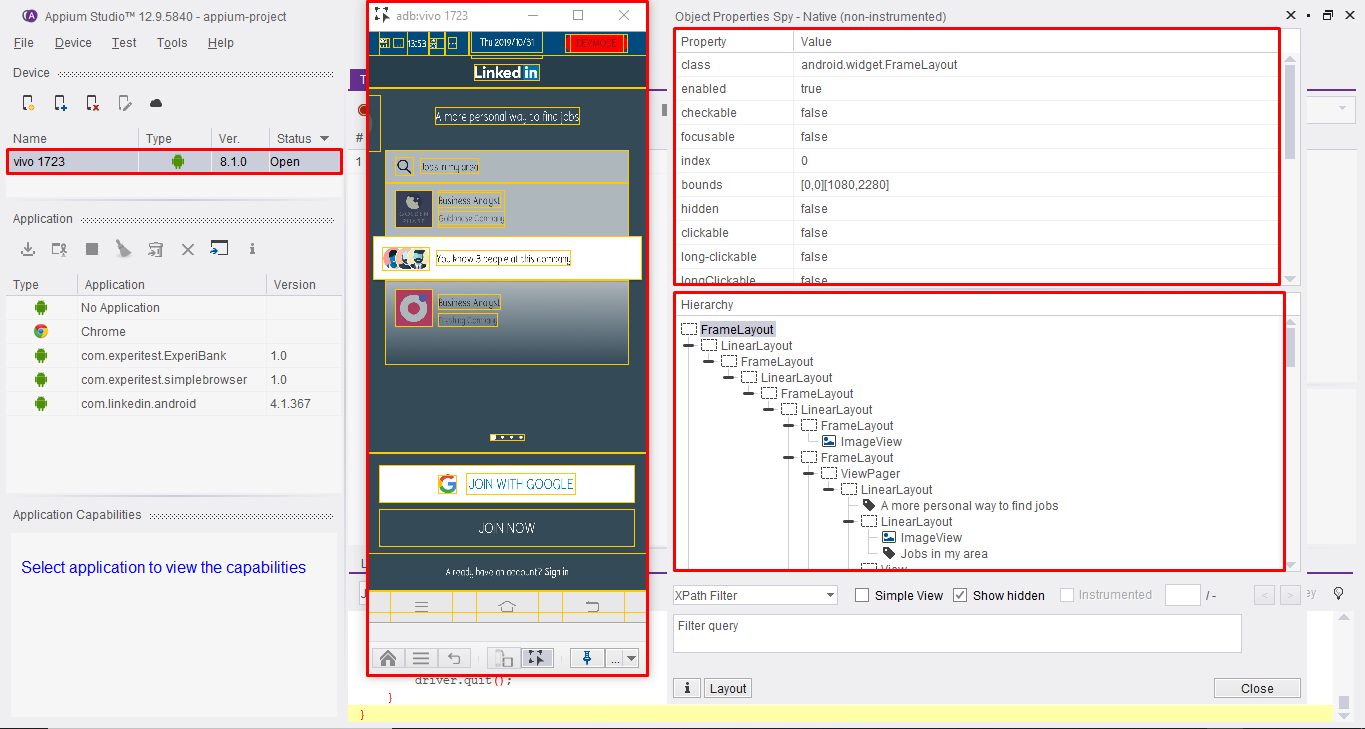
}

}

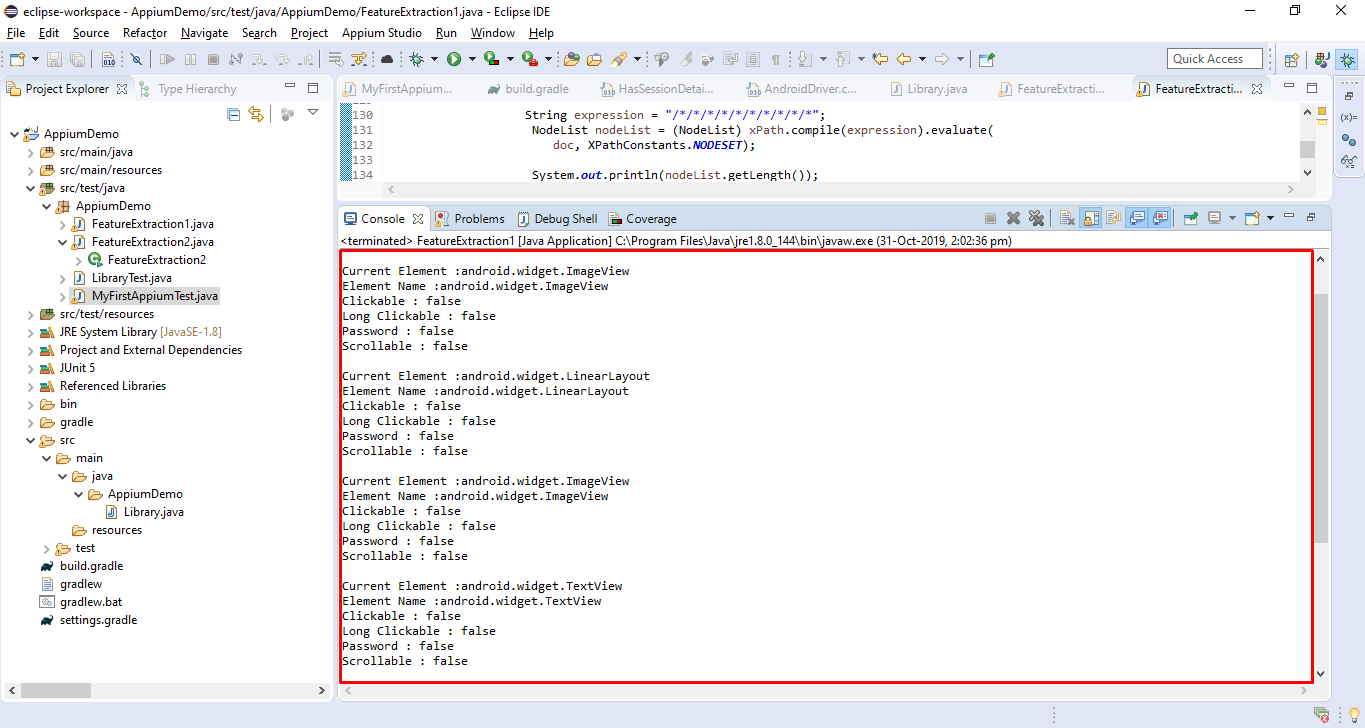
1. The list of screenshots below depicts the appium object spy strutcure, Linkedln page whose xml structure is fetched and the element-feature list fetched using the script written in eclipse.



**Fig 1: Linkedln Page**



**Fig 2: Appium Object Spy with the Linkedln Page Structure**

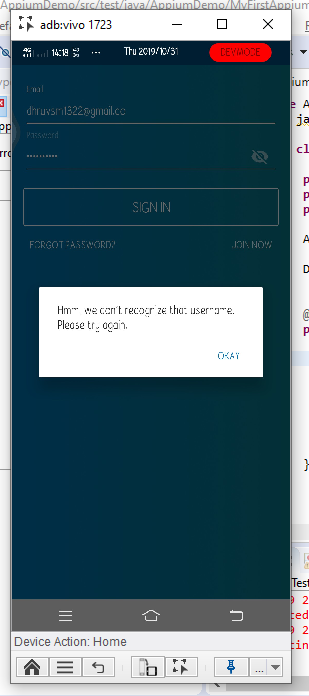
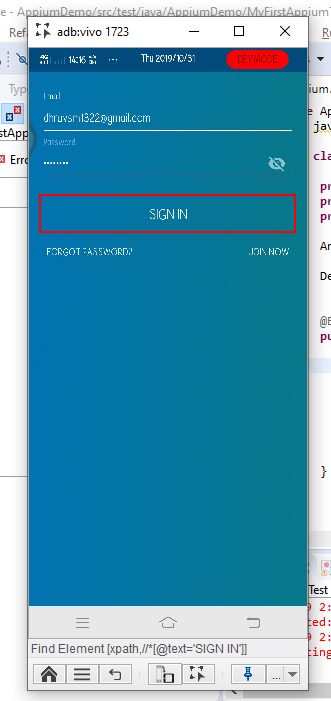


**Fig 3: Element-Feature Extraction**

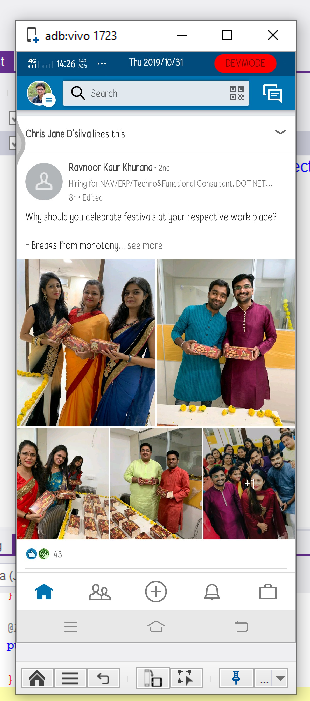
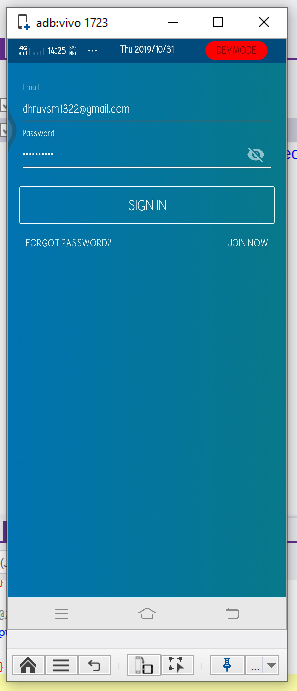
**Testing Linkedln Login :**

* 1. In this part, we have written the test cases to test the Linkedln Login portal. In the first part we enter incorrect credentials and in the second part correct credentials are entered.

Part 1 :



Part 2:



Code:  
package AppiumDemo;  
**import** io.appium.java\_client.remote.AndroidMobileCapabilityType;  
**import** io.appium.java\_client.android.AndroidDriver;  
**import** io.appium.java\_client.android.AndroidElement;  
**import** io.appium.java\_client.remote.MobileCapabilityType;  
**import** org.openqa.selenium.support.ui.ExpectedConditions;  
**import** org.openqa.selenium.support.ui.WebDriverWait;  
**import** org.openqa.selenium.remote.DesiredCapabilities;  
**import** org.openqa.selenium.By;  
**import** org.openqa.selenium.Platform;  
**import** org.openqa.selenium.interactions.InputSource;  
**import** org.junit.jupiter.api.\*;  
**import** java.net.URL;  
**import** java.nio.file.Files;  
**import** java.nio.file.Paths;  
**import** java.net.MalformedURLException;  
**import** java.util.List;  
**import** java.util.logging.Level;  
  
**public** **class** **MyFirstAppiumTest** {  
   
 **private** String reportDirectory = "reports";  
 **private** String reportFormat = "xml";  
 **private** String testName = "Untitled";  
   
 AndroidDriver<AndroidElement> driver = null;  
  
 DesiredCapabilities dc = DesiredCapabilities.android();  
  
   
 @BeforeEach  
 **public** void setUp() **throws** MalformedURLException {  
 dc.setCapability("reportDirectory", reportDirectory);  
 dc.setCapability("reportFormat", reportFormat);  
 dc.setCapability("testName", testName);  
 dc.setCapability(MobileCapabilityType.UDID, "52b5900b");  
 dc.setCapability(AndroidMobileCapabilityType.APP\_PACKAGE, "com.linkedin.android");  
 dc.setCapability(AndroidMobileCapabilityType.APP\_ACTIVITY, ".authenticator.LaunchActivity");  
 driver = new AndroidDriver<>(new URL("http://localhost:4723/wd/hub"), dc);  
 driver.setLogLevel(Level.INFO);  
 }   
   
 @Test  
 **public** void testUntitled() {  
 driver.findElement(By.xpath("//\*[@id='growth\_login\_join\_fragment\_email\_address']")).sendKeys("dh ruvsm1322@gmail.com");  
 driver.findElement(By.xpath("//\*[@id='growth\_login\_join\_fragment\_password']")).sendKeys("########");  
 new WebDriverWait(driver, 30).until(ExpectedConditions.presenceOfElementLocated(By.xpath("//\*[@text='SIGN IN']")));  
 driver.findElement(By.xpath("//\*[@text='SIGN IN']")).click();  
 driver.findElement(By.xpath("//\*[@text='OKAY']")).click();  
 driver.executeScript("seetest:client.deviceAction(\"BKSP\")");  
 driver.findElement(By.xpath("//\*[@id='growth\_login\_join\_fragment\_password']")).sendKeys("##########");  
 driver.findElement(By.xpath("//\*[@text='SIGN IN']")).click();  
 }  
  
 @AfterEach  
 **public** void tearDown() {  
 driver.quit();  
 }  
  
 }

**Feature Extraction Approach 2 (Find Element by xpath) [Anubhav Tiwari]**

The environment setup for the program required creation of a new Android driver object with the desired capabilities parameters. These parameters which are automatically generated by appium include:

1. PLATFORM\_NAME – Name of platform to be tested which is Android.
2. UDID – This is the unique device ID of the target device on which application is tested.
3. NO\_RESET – TRUE or FALSE depending on whether application should be reset every time program executes or not.
4. APP\_PACKAGE – Main application package which is different for every android application.
5. APP\_ACTIVITY – The activity name of that application

For extracting the features, another approach that we took is of going to android application’s source and finding all the elements by xpath. Using Appium's method of finding elements by xpath and giving the expression to match as (“.//\*”), we can get all the elements of a specific activity of the android application. The output of this method will be a list of all the android elements in the current activity.

Next challenge was to get the required features of those elements from the whole hierarchy of application activity which was the output of findElementByXpath method. To do this we traversed through each element in the list and extracted the following properties of those elements.

1. Known Super Class – Tells the name of super class or parent class of the current element in the hierarchy.
2. Tag Name – Name of the current element. (for e.g., Frame Layout, Linear Layout, Text View, Image View, View Pager, View Group or Button)
3. Clickable – TRUE or FALSE depending on the element behavior if its clickable or not.
4. Long-Clickable – TRUE or FALSE depending on the element behavior if its long-clickable or not.
5. Scrollable – TRUE or FALSE depending on the element behavior if its scrollable or not.
6. Password – TRUE or FALSE depending on the element behavior if it is a password field or not.
7. Text-Field – TRUE or FALSE depending on the element behavior if it accepts a text type input or not.

These extracted features were converted into a CSV file for further processing. For the CSV conversion we have used the FileWriter Class of java. Below screenshot shows the launch activity of LinkedIn app and the output file will show the elements of the same activity.

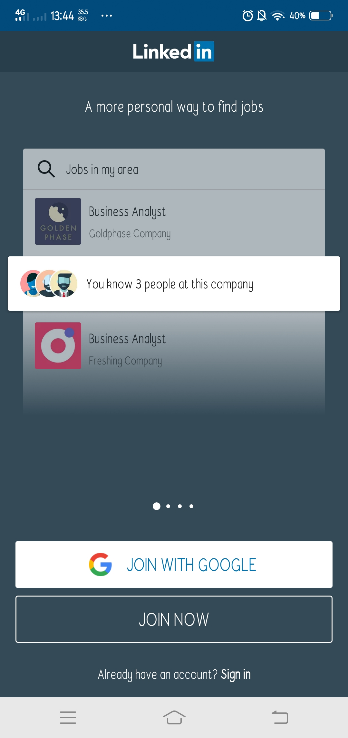
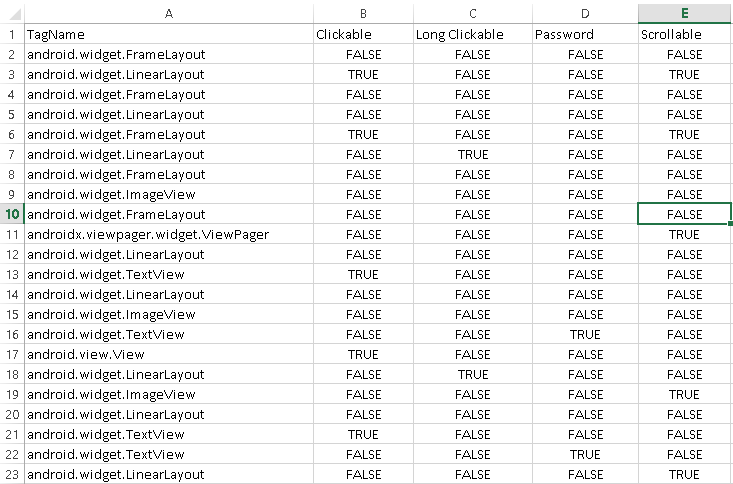


Fig. Launch Activity of LinkedIn Application

Following is source code of the java class used for Feature Extraction:

**public** class FeatureExtraction2 {  
      
        **public** **static** **void** main(**String** []args) **throws** Exception{  
        AndroidDriver<AndroidElement> driver = **null**;  
        // Setting Up Android Driver Object  
        DesiredCapabilities dc = DesiredCapabilities.android();  
        dc.setCapability(MobileCapabilityType.PLATFORM\_NAME, Platform.ANDROID);  
        dc.setCapability(MobileCapabilityType.UDID, "52b5900b");  
        dc.setCapability(MobileCapabilityType.NO\_RESET,**false**);  
        dc.setCapability(AndroidMobileCapabilityType.APP\_PACKAGE, "com.linkedin.android");  
        dc.setCapability(AndroidMobileCapabilityType.APP\_ACTIVITY, ".authenticator.LaunchActivity");          
        driver = **new** AndroidDriver<>(**new** URL("http://localhost:4723/wd/hub"), dc);  
        // Find Element By Xpath  
        List<AndroidElement> PgSrc = driver.findElementsByXPath(".//\*");  
        **try** (PrintWriter writer = **new** PrintWriter(**new** File("test.csv"))) {  
            StringBuilder sb = **new** StringBuilder();  
            sb.append("TagName");  
            sb.append(',');  
            sb.append("Clickable");  
            sb.append(',');  
            sb.append("Long Clickable");  
            sb.append(',');  
            sb.append("Password");  
            sb.append(',');  
            sb.append("Scrollable");  
            sb.append('\n');  
            **for** (int i = 0; i < PgSrc.size();i++) {  
                System.out.print(i);  
                **String** tagName = PgSrc.get(i).getTagName();  
              sb.append(tagName);  
              sb.append(',');  
            sb.append(PgSrc.get(i).getAttribute("clickable"));  
            sb.append(',');  
            sb.append(PgSrc.get(i).getAttribute("long-clickable"));  
            sb.append(',');  
            sb.append(PgSrc.get(i).getAttribute("password"));  
            sb.append(',');  
            sb.append(PgSrc.get(i).getAttribute("scrollable"));  
            sb.append('\n');  
            // Converting to CSV  
            writer.write(sb.toString());  
            System.out.println("done!");          }   
            **catch** (FileNotFoundException e) {  
            System.out.println(e.getMessage());  
          }  
        driver.resetApp();  
        driver.quit();  
       }  
}

The following screenshot shows the output CSV file generated from the above class :



# Activities coverage in our test case design [Afamefuna Umejiaku]

An Android application consists of various activities linked together, thus far we have been considering the individual activities separately, developing a system with the aid of machine learning capable of detecting what activity type based on our activity type classification and also developing relevant test cases to be executed on the various individual activities as described above.

The need to develop and incorporate into our software testing tool a means to transition through the various activities in an android application, testing each activity and producing an output that depicts all the activities in the application, the possible means of transitions in the application and the results of the various test performed in the application is the focus in this section.

To do this, we considered possible means of transitions between the various activity types in an application and created a finite machine (moore machine). Each state in the Moore machine represents an activity type, upon transitioning into a state, the required test cases are performed. The possible transition between the relevant activities types is depicted in Figure 2.1 below.

Other than the splash state where the transition is expected to perform automatically, transitions between the states involves either clicking a Clickable element, swiping a horizontal swipeable element or a Vertical swipeable element, filling a text field and/or a combination of two or more actions. The transition from a state could be back to the same state or a different state, a transition to the same state means we moved from one activity to another activity of the same type while transition to different activity type results in movement to a state that depicts that activity type. Considering that certain activities such as mailing could not be effectively tested with performing an action and revisiting the activity at a later time, some activities will include special instructions in the algorithms to perform such tests.

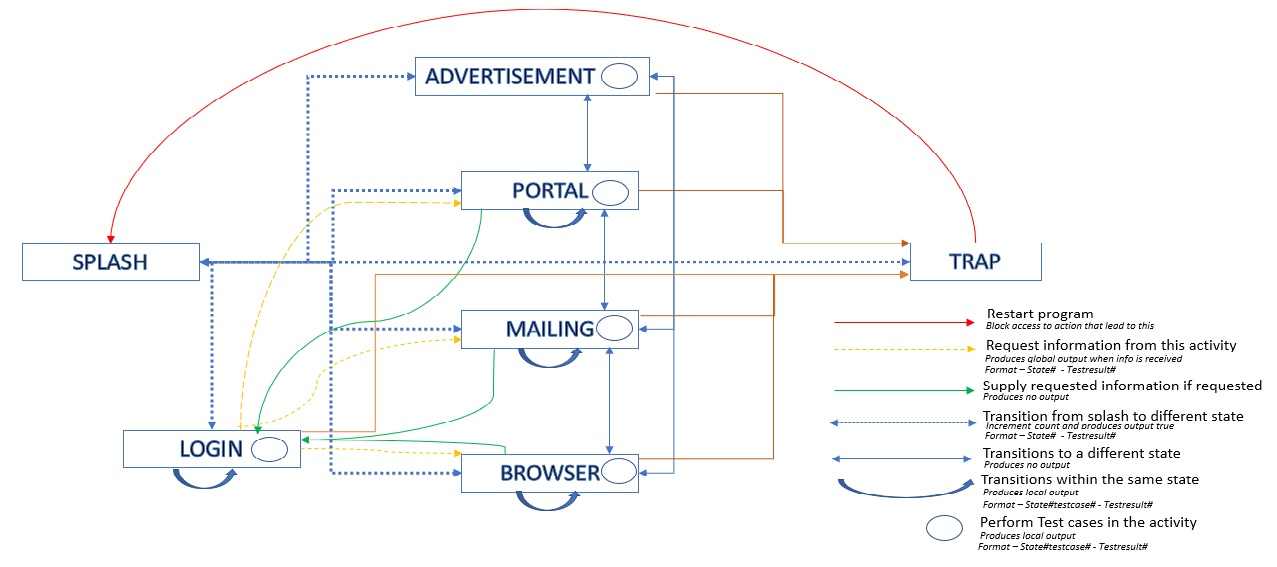
The output in the moore machine depicts the results of the result obtained each time we transition into that activity type and perform the necessary test cases. A counter is introduced to count the number of times a state is transitioned into and to depict the output of the state with the count. The final report after testing an application would show the number of times a state was transitioned into and the test result each time it was transitioned into.

if for any reason we encounter a situation we are forced to move to a different application from the one being tested or an activity type not considered in our design, we consider that to be a trap state and would expect the tool to backtrack and skip the feature in the application that lead to entering the trap state.

**How the states are tested.**

While the various states in the automata have different test cases based on its activity type, some states could have all the required test cases tested independently on an activity, in some states, certain tests cases in those states require information from other states(Activity type) or from an earlier activity in that state. Here we discuss they various states are tested and the expected results.

1. Splash State- the test for this state is detecting a splash activity and moving to a different state, hence information from other states (activities) is not required in this state to detect if this state passes its test cases. Hence the output in this state can be set to always true (1) unless there is no transition to another state. Hence the output of testing at this state shows the count each time the splash state is entered and if it passes of fails. For example - Splash1-True, Splash-False, Splash3-true
2. Advertisement State- the test for this state is clicking a close/skip button and remaining in the Application but at a different state, hence information from other states (activities) is not required in this state to detect if this state passes its test cases. Hence the output in this state can be set to always true (1) also and changes to zero(failed) if we cannot transition to another state by clicking a button. Hence the output of testing at this state shows the count each time the state is entered and if it passes of fails. For example - Advert1-True, Advert2-False
3. Trap State – this state does not have any test case as this state represents states we did not consider in our design as well as a situation where an action leads to a different application. Once a we enter this state, the action the lead to entering this state is marked and the application testing is restated. This produces no output result just other than recording a trap state was entered.
4. Login State – having proposed 6 different test cases for the login activity, with some of the test cases performed in the same activity, some a different activity but of the same type and some a different activity type hence moving to a different state, the state would have 6 different output results with the 5 local tests that don’t move to a different activities type producing outputs similar to that of splash having a local counter and its result and for tests that move to different states having a global counter producing its results. The output result here will have to forms. An example of the global cases will be Login1-False, Login2-true and for local will be Login1test1-true
5. Portal State – the portal state has 8 different test cases but unlike the login state, in the portal state there is no need to have information from another state or activity to verify that the test cases where successful other than the fact that there was a transition to a different state. The output here would all be local for example portal1test1-true , portal1test2-true , portal2test1-true
6. ToDoList state -– the ToDoList state has 9 different test cases, with seven of the test cases on the same activities but to verify that the 2 of the test cases where successful we would need information from other activities possible in the same state or in a different state and not just that there is a transition to a different state. The output here would all be local for example list1test1-true and global for 2 cases.
7. Mailing State – testing in the mailing state is similar to that of the Portal state as there is no need to have information from another state but unlike the portal state, to verify that the some of the test cases where successful we would need information from other activities in the same state than the fact there was a transition to a different state. The output here would all be local for example portal1test1-true, portal1test2-true , portal2test1-true.
8. Browser state- testing in the browser state involves 9 tests cases with all the cases involving moving from one activity to another activity. The output for all these test cases could be stored locally.



**Machine Learning Classifier (using SVM algorithm): [Anubhav Tiwari]**

For Classifying the activities into the seven activity groups and predicting the type of activity, we have built a classifier model based on SVM (Support Vector Machines) algorithm. Next section explains about the SVM in brief.

**Support Vector Machine:**

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification and regression challenges. It can solve linear and non-linear problems and work well for many practical problems.

Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a coordinate. Then, we perform classification by finding the hyper-plane that differentiate the classes very well.

**Implementation**:

This section describes how we have implemented the SVM algorithm for classifying our dataset rows into one of the activity types. For this implementation, we have used “sklearn” library of python which comes with the SVM and metric modules. These modules consist of SVC (Support Vector Classifier), classification\_report and confusion\_matrix functions which we have used in this classifier code.

Also, we are using pandas and numpy library of python for all the operations related to CSV (Comma Separated values) files. These are the steps for setting up and running the following classifier:

1.) Install the libraries in the system using pip install. The libraries to install are pandas, numpy and sklearn.

C:\ pip install pandas

C:\ pip install numpy

C:\ pip install sklearn

2.) Convert the training and testing dataset files into dataframes by using read\_csv function of pandas.

**data** = pd.read\_csv("../datasets/dataset\_v2.csv")

   test\_data = pd.read\_csv("../datasets/test1.csv")

3.) Replace the Null values in both the dataframes with zero.

**data** = **data**.fillna('0')

4.) Now we have to do **Data Pre-Processing.**

Data preprocessing involves:

(i) Dividing the data into attributes and labels and

(ii) Dividing the data into training and testing sets.

To divide the data into attributes and labels, execute the following code:

X = **data**.drop(['Activity\_Type'], axis=1)

y = **data**['Activity\_Type]

Here X is row of the attributes and and y has all the labels or the activity types.

5.) **Training the algorithm:**

Scikit-Learn contains the svm library, which contains built-in classes for different SVM algorithms. Since we are going to perform a classification task, we will use the support vector classifier class, which is written as SVC in the Scikit-Learn's svm library. This class takes one parameter, which is the kernel type. In our classifier, we are using gaussian kernel due to its improved accuracy over polynomial.

This parameter can have following values:

(i) ‘linear’ for linearly separable data.

(ii) In case of non-linearly separable data, we use Kernel SVM which may have polynomial, Gaussian, and sigmoid as parameters.

The fit method of SVC class is called to train the algorithm on the training data, which is passed as a parameter to the fit method.

svclassifier = SVC(kernel='rbf')

svclassifier.fit(X\_train, y\_train)

6.) **Making Predictions:**

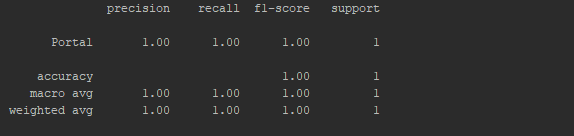
To make predictions, the predict method of the SVC class is used which is shown by the following code:

y\_pred = svclassifier.predict(X\_test)

7.) **Evaluating the algorithm:**

Confusion matrix, precision, recall, and F1 measures are the most used metrics for classification tasks. In our Classifier algorithm have used classification report to display the precision, recall and F1 scores for the result.

So, the result will look something like this:



Following is source code of the python file used as classifier for our program:

**import** pandas **as** pd  
**import** numpy **as** np  
from sklearn.svm **import** SVC  
from sklearn.metrics **import** classification\_report, confusion\_matrix  
  
  
def clean\_dataset(df):  
 assert isinstance(df, pd.DataFrame)  
 df.dropna(inplace=True)  
 indices\_to\_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(1)  
 return df[indices\_to\_keep].astype(np.float64)  
  
  
**data** = pd.read\_csv("../datasets/dataset\_v2.csv")  
test\_data = pd.read\_csv("../datasets/test1.csv")  
**data** = **data**.fillna('0')  
test\_data = test\_data.fillna('0')  
  
X = **data**.drop(['Activity\_Type'], axis=1)  
X\_test = test\_data.drop(['Activity\_Type'], axis=1)  
y\_test = test\_data['Activity\_Type']  
y = **data**['Activity\_Type']

X\_train = X  
y\_train = y  
  
svclassifier = SVC(kernel='rbf')  
svclassifier.fit(X\_train, y\_train)  
  
y\_pred = svclassifier.predict(X\_test)  
print(y\_pred[0])  
print(confusion\_matrix(y\_test,y\_pred))  
print(classification\_report(y\_test,y\_pred))

**Machine Learning Classifier (using KNN Algorithm): [Ayodeji Ejiade]**

KNN Algorithm stands for k-Nearest Neighbor, this algorithm is very simple and applicable to problems like this. It is referred to as lazy learning method, because we need to hold the entire dataset and is only used when a prediction is needed.

# Methodology

We store all our training dataset in a CSV file and the k-most similar records are selected from the training dataset during testing, a prediction is made from these neighbors. In our case, we use k = 5.

To measure the similarities between these data, several methods can be used, however, for this work, we used the Euclidean distance, which is highly applicable for tabular data like this.

Once, we get the neighbors, prediction is made by getting the most represented data from the neighbors, this makes KNN very applicable for solving both classification and regression problems.

# KNN Steps

Overall, the algorithm is divided in 4 major parts

## Load Training Data

During our actual software testing, features are extracted and stored in a CSV file, the machine learning program picks the CSV file and loads this and stores it a list.

## Calculate Euclidean Distance.

The next step is to calculate the Euclidean distance between the test row and every, the Euclidean distance between 2 vectors is calculated as the modulus between the two vectors which is the square root of the sum of the squared differences between the two vectors.

Where x1 is the first row of data, x2 is the second row of data and i is the index to a specific column as we sum across all columns. Euclidean distance will allow us to calculate how similar 2 data is.

## Get Nearest Neighbors.

In the KNN algorithm, we are interested in getting the k-nearest neighbors, to do this, we calculate the Euclidean distance (defined above) between the test data and each row of the training dataset.

After calculating all these, we sort all the distances and select the top k to return as the most similar neighbors

Neighbors for a new piece of data in the dataset are the k closest instances, as defined by our distance measure.

Once distances are calculated, we must sort all of the records in the training dataset by their distance to the new data. We can then select the top k to return as the most similar neighbors.

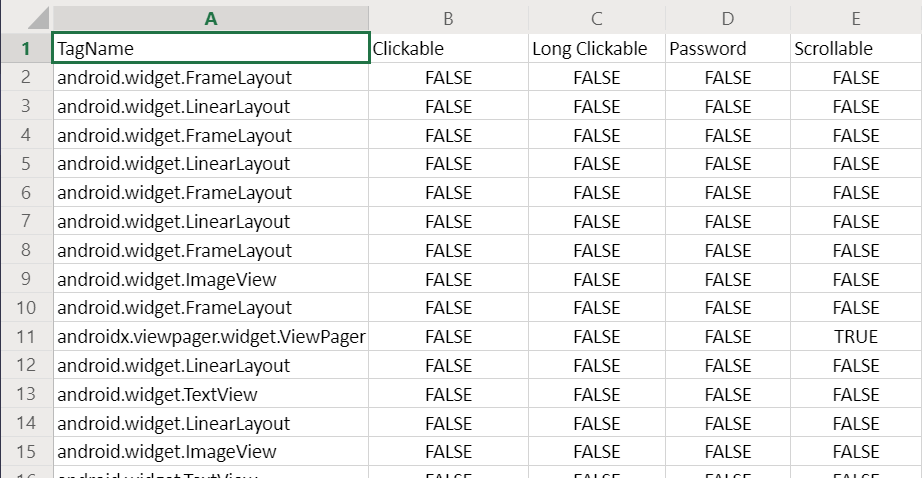
## Make Predictions.

To make predictions, we find the most represented among the neighbors, we achieved this by performing the max() function on the list of output values from the neighbors. Given a list of class values observed in the neighbors, the max() function takes a set of unique class values and calls the count on the list of class values for each class value in the set.

s# Make Predictions  
from csv import reader  
from math import sqrt  
import pandas as pd  
  
# Load a CSV file  
def load\_training\_data(filename):  
 training\_dataset = list()  
 with open(filename, 'r') as file:  
 csv\_reader = reader(file)  
 for data in csv\_reader:  
 if not data:  
 continue  
 training\_dataset.append(row)  
 return training\_dataset  
  
  
# Convert string column to float  
def str\_column\_to\_float(dataset, column):  
 for row in dataset:  
 row[column] = float(row[column].strip())  
  
  
# Convert string column to integer  
def str\_column\_to\_int(dataset, column):  
 class\_values = [row[column] for row in dataset]  
 unique = set(class\_values)  
 lookup = dict()  
 for i, value in enumerate(unique):  
 lookup[value] = i  
 print('[%s] => %d' % (value, i))  
 for row in dataset:  
 row[column] = lookup[row[column]]  
 return lookup  
  
  
# Find the min and max values for each column  
def dataset\_minmax(dataset):  
 minmax = list()  
 for i in range(len(dataset[0])):  
 col\_values = [row[i] for row in dataset]  
 value\_min = min(col\_values)  
 value\_max = max(col\_values)  
 minmax.append([value\_min, value\_max])  
 return minmax  
  
  
# Rescale dataset columns to the range 0-1  
def normalize\_dataset(dataset, minmax):  
 for row in dataset:  
 for i in range(len(row)):  
 row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])  
  
  
# Calculate the Euclidean distance between two vectors  
def euclidean\_distance(row1, row2):  
 distance = 0.0  
 for i in range(len(row1) - 1):  
 distance += (row1[i] - row2[i]) \*\* 2  
 return sqrt(distance)  
  
  
# Locate the most similar neighbors  
def get\_neighbors(train, test\_row, num\_neighbors):  
 distances = list()  
 for train\_row in train:  
 dist = euclidean\_distance(test\_row, train\_row)  
 distances.append((train\_row, dist))  
 distances.sort(key=lambda tup: tup[1])  
 neighbors = list()  
 for i in range(num\_neighbors):  
 neighbors.append(distances[i][0])  
 return neighbors  
  
  
# Make a prediction with neighbors  
def predict\_classification(train, test\_row, num\_neighbors):  
 neighbors = get\_neighbors(train, test\_row, num\_neighbors)  
 output\_values = [row[-1] for row in neighbors]  
 prediction = max(set(output\_values), key=output\_values.count)  
 return prediction  
  
  
# Make a prediction with KNN on Iris Dataset  
filename = 'dataset\_v2.csv'  
dataset = load\_training\_data(filename)  
for i in range(len(dataset[0]) - 1):  
 str\_column\_to\_float(dataset, i)  
# convert class column to integers  
str\_column\_to\_int(dataset, len(dataset[0]) - 1)  
# define model parameter  
num\_neighbors = 5  
  
  
# define a new record  
test ='test1.csv'  
test\_data = pd.read\_csv(test)  
row = test\_data.iloc[0].values  
#row = str\_column\_to\_int(row,0)  
#print(row)  
  
# predict the label  
label = predict\_classification(dataset, row, num\_neighbors)  
print('Data=%s, Predicted: %s' % (row, label))

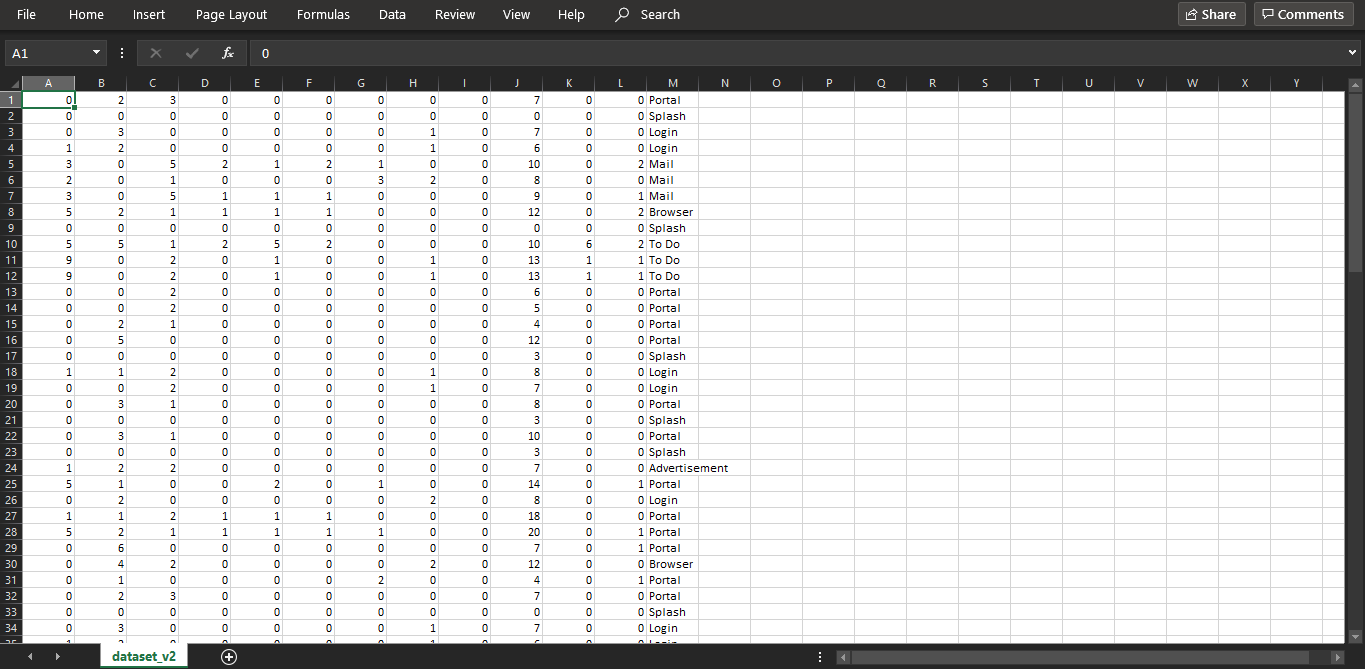
**Creation of Feature Extraction Testing Dataset [Dhruv Mehta]**

**1) Initially we create a Test.csv file to check the values of different features.**

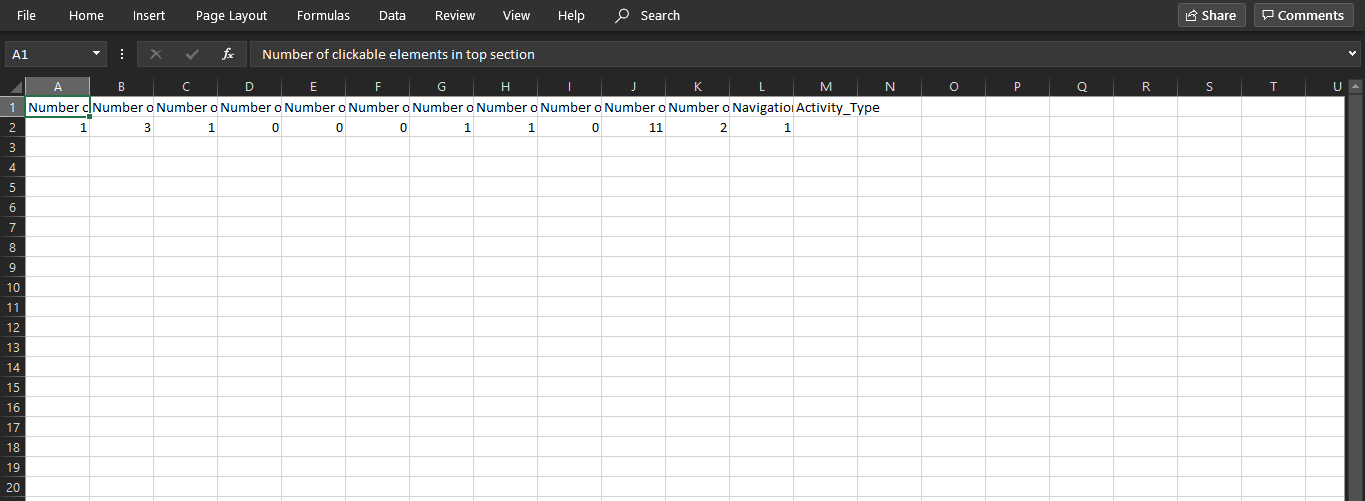


**Fig 3.1: CSV file with the features and their values**

2) Then we have created a code where using these features, we can calculate the count of various features such as clickable elements in the top, middle and bottom part; scrollable elements in the top, middle and bottom part, text elements in the top, middle and bottom part; long-clickable elements, general elements and navigation bar. For that we require two files one for training the algorithm and one for testing the algorithm.



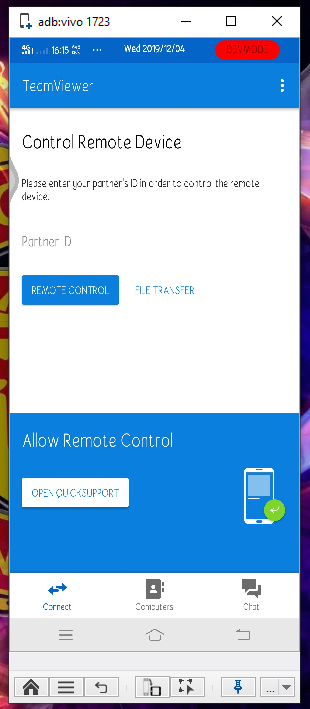
**Fig3.2 : Training Dataset**



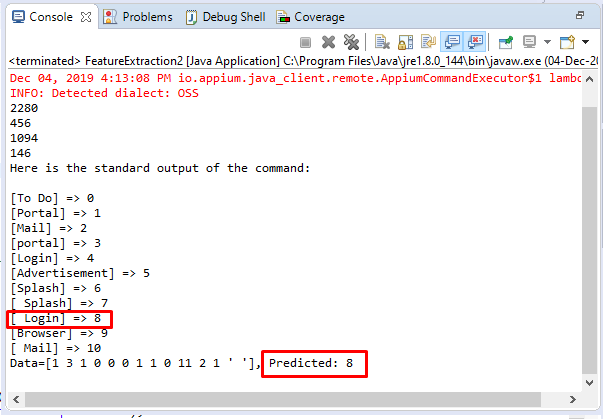
**Fig 3.3: Testing Dataset that is extracted from the TeamViewer Login Activity Page**

3) After this the machine learning algorithm starts as soon as the file above with the testing dataset is created. We make use of the KNN classification machine learning algorithm. It creates a dictionary of activity type and on the basis of what it has learnt from the training dataset it predicts the activity type of the testing dataset.

**For Login Activity Type:**

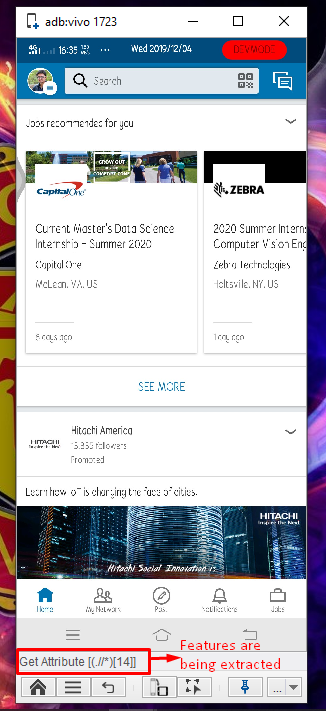


**Fig3.4: TeamViewer Login Page**

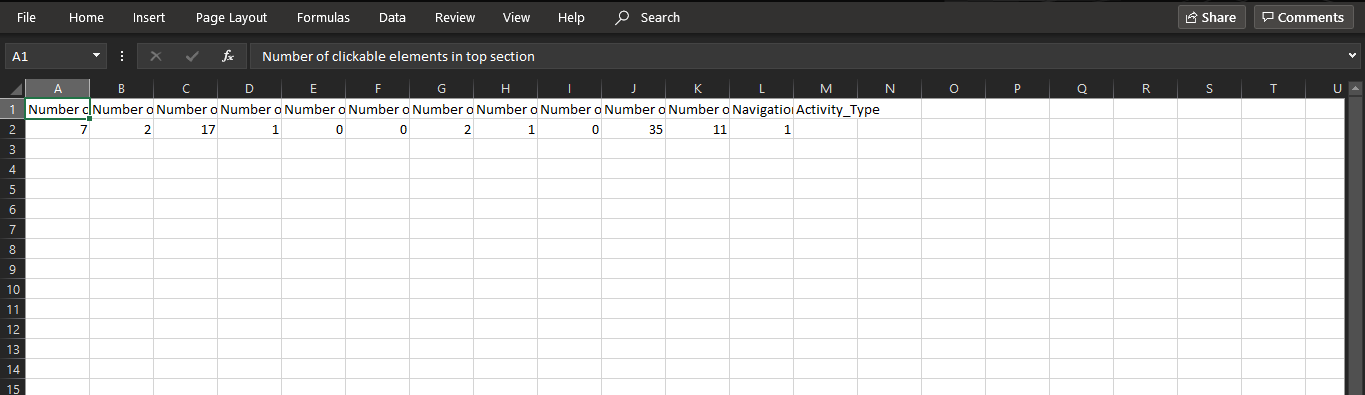


**Fig 3.5: Output of the machine learning algorithm that predicts the Activity type is Login**

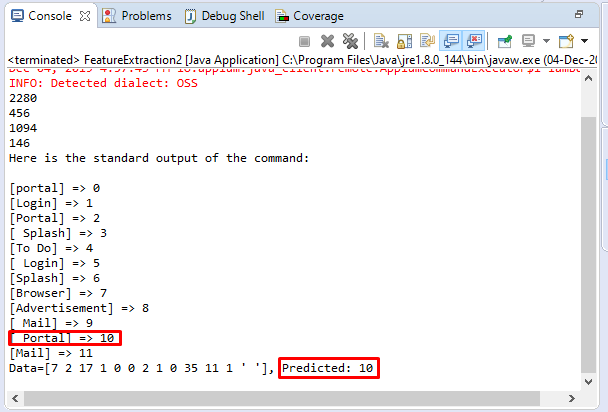
**For Portal Activity Type:**



**Fig 3.6: Feature extraction from the Linkedin Portal page**



**Fig 3.7: Testing Dataset that is extracted from the Linkedin Portal Activity Page**



**Fig 3.8: Output of the machine learning algorithm that predicts the Activity type is Portal**

**OUR WORK [Umejiaku Promise Afamefuna]**

We successful developed a white board testing too capable of detecting an activity type on an android application and performing relevant test on that activity and uploaded to github. To check out the repository you can go to  <https://github.com/Dhruvmehta2213/Automated-Android-Test-Tool-using-Machine-Learning> .

**Running our Tool**

Following the steps below you can run our tool

On an Android Phone**:**

* Enable Developer Option (Enabling this option is phone dependent but in most phones, in the settings option you can search for build, and click build number seven times.
* Enable USB Debugging on your phone.

On your laptop

* Install Appium Studio in your Computer if not installed.
* Open Appium Studio.
* Add your Device you want to test.
* Select device name and click open icon. This will open in ADB (Android Debug Bridge) used for android testing
* Open the application to test using ADB
* You can clone our git
* Run our Tool on your IDE.

**Future Work**

Unable to get a dataset related to our research considering the uniqueness of our classification criteria, we had to develop one within the Limited time we had. The process we deployed in creating our data sets was time consuming hence we were only able to create a dataset with 90 rows as a result our machine learning algorithm was not able to achieve our targeted accuracy of 95% for all activity types.

While we noticed an accuracy of 100% in Splash Activity, Advertisement Activity and Portal Activity, we experienced 60% accuracy in Login Activity and Mailing Activity and only 20% in Browser Activity as a result we felt our dataset was highly inadequate.

We also noticed that we need to add certain activities to our activity types due to the uniqueness of our test cases for some activities and the number we of times we encounter those activities. We propose to add payment/registration activity, map Activity and calculator activity.

Also, in the design of our automata, we were not able to implement transfer of information from other states(Activity types) and using such information to perform tests, this affected our ability to perform global tests cases in login and mailing activity.

Finally, we propose merging both the browser and portal activities due to their similarities as we noticed the have similar features and our test cases were similar.

**Conclusion**

We have developed a tool capable of testing android software on phones. We employed three different methods when initially designing our database and decided to use find Element by Xpath for the final product. We also employed two different machine learning algorithms KNN (K nearest neighbor) and SVM (support vector machine) when the initially developing our machine algorithm and observed KNN had a better accuracy rate hence was adopted. We would also like to use this opportunity to thank you for guiding us through this process.

# References

1. Android Developers Guide Retrieved from <https://developer.android.com/guide/components/activities/intro-activities> accessed on Sep 27, 2019
2. Developing an appium project and using Appium Page Object Model from <https://www.youtube.com/watch?time_continue=2158&v=fN1ItWV7mNI>
3. Fetching elements from the Android applications automatically : <https://discuss.appium.io/t/how-can-i-fetch-all-elements-from-a-screen-automatically/17269/3>
4. Implementing Page Object Model using Appium and Java for tests using <https://blog.testproject.io/2018/07/31/page-object-model-appium-java-android/#DemoApp>
5. Mansoor Iqbal (2019) “App Download and Usage Statistics (2019)” Retrieved from <https://www.businessofapps.com/data/app-statistics/>
6. Rosenfeld A., Kardashov O., Zang O. (2017) ACAT: A Novel Machine-Learning-Based Tool for Automating Android Application Testing. In: Strichman O., Tzoref-Brill R. (eds) Hardware and Software: Verification and Testing. HVC 2017. Lecture Notes in Computer Science, vol 10629. Retrieved from <https://link.springer.com/content/pdf/10.1007%2F978-3-319-70389-3_14.pdf>
7. Writing a CSV file in java using OpenCSV : <https://www.geeksforgeeks.org/writing-a-csv-file-in-java-using-opencsv/>
8. Yuanchun L, Ziyue Y & Yao G. (2019). *“A Deep Learning based Approach to Automated Android App Testing”*. arXiv preprint arXiv:1901.02633. Retrieved from<https://arxiv.org/pdf/1901.02633.pdf>