EECS 450: Internet Security

Electrical Engineering & Computer Science Department

NORTHWESTERN
UNIVERSITY

RiskCog

Implicit and Continuous User Identification on Smartphones in the Wild

Final Project Report

Fall, 2016

Team:

Sandeep Raju Prabhakar
srp@u.northwestern.edu

Jiham Lee
jihamlee2017@u.northwestern.edu

Karan K Shah
karan.shah@u.northwestern.edu

Mentors:

Prof. Yan Chen
ychen@northwestern.edu

Zhengyang Qu
ZhengyangQu2017@u.northwestern.edu

Table of Contents

Abstract

Everyone is using smartphones for mobile payment, storing personal photos, using online services. Integrating the sensitive applications and files to the smartphone introduces new security risks, for example, the attacker pays with the victim's account using the device stolen, downloads sensitive folders or installs a virus on the phone. By depicting the device owner with a diverse set of features, such as the locations the user frequently visits, face snapshot and fingerprint verification, the existing risk management for mobiles is harder to get bypassed than the traditional user authentication mechanism with the simple passkey or simple pattern. However, it involves the heavy usage of user's personal information and raises the user's concern on privacy leakage. Moreover, the current risk management countermeasure are deployed at the application layer.

In this report, we continue the work done on RISKCOG, which solves the problem of identifying the authorized device owner by the data collected from the acceleration sensor, gyroscope sensor, and gravity sensor with a learning-based approach. We find the set of features that is effective to train the classifier to fingerprint the device owner. The feature set only leverages the motion sensors, which are commonly available on smartphone and smartwatch. By doing more research we have found out ways to improve the accuracy of the application. We present our findings, the challenges we faced along with their respective solutions and future work to be done in detail.

Introduction

Smartphones provide users with many different functionalities, which include the growing popularity of mobile payments. Although beneficial and convenient, they introduce various security threats, which can be categorized into two groups. Data-driven risks are usually caused by leaks of sensitive data from sources from the client side by malware, network-channels by man-in-the-middle attacks, and the server. The other category of risks concern human-driven risks that roots from when those who are not the user have access to the phone and is able to use payment functions for their own benefit. According to data from LexisNexis, mobile transactions accounted for 14% of total transactions and 21% of fraudulent transaction. An effective solution to mobile payment security is urgently needed.

The RiskCog application strives to solve the problem of implicitly identifying the authorized user based on a learning approach from data collected by the user's mobile phone. The theory behind it has three requirements: (1) least user privacy requirement; (2) deployment as a third-party service at the device level; (3) capability of enforcing verification with various motion states and dynamic device placement. It is assumed that the device contains an acceleration sensor, gyroscope sensor, and gravity sensor and that they are all working normally.

The main challenges concern the implementation of the application are the lack of feature of gait verification in other related works, data availability and dynamic device placement to recognize and classify different data, imbalanced data set when differentiating between positive user data and negative intruder data due to huge number of negative specific data, and the unlabeled data from unsupervised learning without definite number of clusters that will cause high time latency, hindering application to be implemented as a real time service.

To overcome the challenges proposed above, the following contributions have been implemented:

- Fifty six features from the motion sensors of smartphones are set to be found to effectively recognize the user of the smartphone accurately. The features do not involve in-app invocation or user tracking and is independent of the user's state. This allows system to not need developer support and has high portability as it is able to verify user's mannerisms of device handling from a steady state.
- A data collection mechanism was implemented to resolve the data availability issue so that it recognizes when phone it is in use by making it event based once the application starts. Unhelpful data is filtered out by sensing if unimportant applications are used and if the sensored are below certain thresholds that represent lack of use.
- To solve the issue of imbalanced data set with a huge number of negative samples from other users, stratified sampling was applied to construct negative sample construct, which in turn helps model the true user's data
- A semi-supervised online learning algorithm was designed to combat the high latency when handling unlabeled data with unsupervised methods. The classifier is trained to incrementally collect the data in chunks.

This project and application strives to be one of the first to achieve high accuracy for implicit user identification. Good performance implies the possible application scenarios to securing mobile payments and beyond.

Problem Statement

As mobile phones are becoming ubiquitous, there is a recurring need to make them secure. Many recent surveys have shown that people are using their mobile phones for financial transactions than ever and this trend is only set to increase. This introduces an higher sense of urgency to tackle the problem of mobile security. It is this motivation that drives us to devise a robust mechanism to prevent mobile phone theft and unauthorized usage by using concepts in Machine Learning and Network Security.

With the given premise, the problem statement can be defined as follows -

Given various data points collected from an individual's mobile phone over a period of time, devise a mechanism to detect at any point of time if the person using the mobile phone is the owner of the phone or an intruder / unauthorized user.

With this given common goal, the problem statement can be further subdivided into sub problems based on different components of the system.

1. **Server Side:** Devise a way to analyse the mobile sensor data to predict if the person using the phone is the owner of the phone or not. This involved machine learning of the sensor data sent from the phone.
2. **Client Side Mobile Application:** Build a mobile application that will be able to periodically collected mobile sensor data and sent it to the server for training. It also performs checks with the server to know who is using the phone.
3. **SmartWatch:** A smartwatch application that can detect the user based on the sensor data from smart watch. This again collects the sensor data from the smart phone and sends it to the server via the mobile application connected via bluetooth.

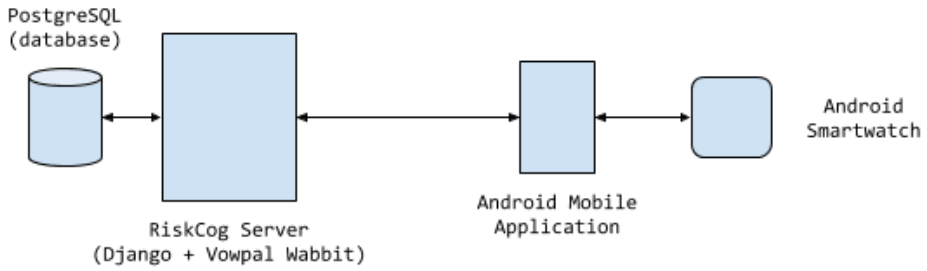
With the above problem statements clearly defined, the next section describes the overview of the entire RiskCog system in detail.

Overview of the System

The current RiskCog system can be split into three components. These three components have be divided based on the functionality they provide in the system. The broad components of the system are divided as follows:

- The server side framework.
- The client side android mobile application.
- The client side android watch application.

The above components can be illustrated by the diagram below:



System Design of RiskCog

From the diagram, we can see the interaction between the components. The interaction between components are explained in detailed in the points below:

1. Initially, the android mobile application is installed on the user's smartphone.
2. The user configures a password so that the application can prompt for it when required.
3. The application runs in the background and starts collecting sensor data when the user is using the phone.
4. This sensor data that is collected periodically is sent to the RiskCog server once enough data points are collected
5. The RiskCog server trains a machine learning model for a user.
6. Once the mobile application collects enough data files, then training is complete. The device periodically then checks if the user is the true user
7. The smartwatch collects the data using accelerometer and gyroscope sensor which is used for checking if the user is the owner or an impostor.

The server side framework

A more detailed look into the working of the server side framework is explained below:

1. The server side framework is a Python Django web application.

2. The server side framework uses PostgreSQL database to store the log of all the files uploaded by the clients.
3. The database only stores the metadata log (path of where the file is stored) but the actual file is stored in the file system (HDD).
4. The server side framework also uses the machine learning framework Vowpal Wabbit (VW) for the purpose of training the models.
5. The server side framework exposes HTTP APIs for performing various operations like - train, test, ask_trained, query, manual_fix.

The client side android mobile application

The following is a detailed framework and detailed procedures of the client side:

1. The Android application was targeted for SDK Version 21
2. After opening the application, a password must be initialized and confirmed. The user can change password even after initializing
3. Training automatically begins but the user can manually start and stop the training
4. The device collects data for 3 seconds and all the data from the sensors are stored into a .txt file in the device in a folder called SensorDemoData and sent to server “garuda.cs.northwestern.edu” port 8080 and undergo machine learning to further decipher the user
5. Once training is complete after 100 files have been written, then a lock button is visible
6. The phone then periodically checks if the user is the true user and asks for password to verify
7. False user data is sent to the server and negative user databases are created

The client side android watch application

The working of the watch application is given below:

1. The client side android watch application uses the sensors located in the watch.
2. The sensors measure the real time data of the wrist and sends it to the phone using bluetooth connection.
3. A specific gesture or movement can be ‘tagged’ which helps in obtaining the values.
4. A .csv file is created on the mobile phone that contains the real time data of accelerometer, gyroscope and gravity.
5. The file can be exported for further analysis of data.

Problems / challenges with the current system

The current system is far from perfect and requires a lot of changes and fixes. To understand what the fixes are, it is imperative to enumerate all the existing problems in the current system. The problems in the current system can be split into the following broad sections:

1. Server side challenges related to the framework
2. Client side challenges related to the android mobile application
3. Client side challenges related to the android watch

Server side challenges

1. The current framework being used for Machine Learning purposes on the server side is Vowpal Wabbit. This server side framework cannot be run on Android due to compatibility issues. If vowpal wabbit has to be used on Android phone, the mobile phone has to be rooted. Since this is not an option for majority of users in the market, there is a need to find an alternative solution for running the classification framework on the client side.
2. The reason for running the classifier on the client side is very simple. If there is interruption in the internet connectivity for the application, the system ceases to work. This is a potential problem since the whole motive of the application is to promise continuous authentication.
3. The only way to run the machine learning model currently on the android phone is via the framework called Tensorflow. Tensorflow can generate a model file of the trained classifier that can be run on the mobile phone to perform classification solely on the client side without involving any external party on the server side.
4. This means that the current server side framework had to be migrated to Tensorflow.
5. In the current framework, there was no check on accuracy, precision or recall. All the values were hardcoded (like, num. of files required to train the model, the check to tell if the model is trained or not).
6. The framework had very high rates of false positives and false negatives. The mobile application was unusable to an extent that, there was 5 to popups every second asking if the user is owner or not. This high rate or false alarms caused the users to uninstall the application. Bottom line was that, the system wasn't working as it was shown in theory.
7. The server side framework used to crash with errors on certain input data sets which leads to inconsistent models and very low accuracy. There was no way to check why the error occurred.
8. The server side code is very complex in architecture and hard to modify or debug. It is written without much abstraction and lot of reliance on the commands that are run as subprocess from Python. A lot of these commands are black box to the main risk server framework and thus cannot be debugged easily. This introduces a lot of complexities.
9. Since the framework works on data collected from sensors which is uploaded from the mobile application, there is no way to test the framework currently to check if the framework works or not. There might be cases where the accuracy drops to 0 due to a bug and still the mobile application shows that the system is trained and performs wrong detections.

10. The commit - rollback model does not work as it is proposed in the paper. The commit - rollback model has to essentially perform adaptive training using the semi-supervised algorithm as proposed in the paper. This does not work due to lack of implementation in the server side framework.

Mobile application challenges

1. The initial setup for the application started training and collecting data every two times the user switched applications. The problem with this is that it took a very long time for the application to finish training
2. The UI is not very intuitive and it was hard to understand what exactly is going on when one presses the train button and confusing what the incrementing number next to Data represents
3. SDK between phone and the server have been conflicting
4. Mobile device has been shutting down and stops running when the training finishes
5. Some devices have been collecting data that produces empty files and experiences network errors with server as the file numbers that have been sent are not valid
6. Initializing password for some phones causes crashes

Android watch challenges

1. The application stopped unexpectedly after an update to the watch.
2. The battery life of the android watch is around 6-7 hours. But during the data collection, the battery life decreased to 3-4 hours.
3. The application Sensor Dashboard installed on the phone does not work in the background during collection of data.
4. When the phone screen times out or the application is pushed to the background, the data collection stops.
5. The data from android watch is different from that of apple iwatch.

Solution to the problems

There was a need to come up with solutions to the problems defined in the previous section. These solutions tackled the problem completely in some contexts and in others they helped mitigate the problem for the time being. The solution to the problems are listed here.

Server side

On the server side, the problems were classified. Initially for the first two weeks, the goal was to use the application and make a list of all the problems that we faced while using the application. Very soon after installing the application that we realized that the application does not work at all. There were myriad of issues from the training not happening to the application generating a lot of false positives and false negatives.

Migration to Tensorflow

As a first task, an evaluation had to be conducted to port the entire framework to use Tensorflow. This framework allows for efficient usage of the classification model on the client side (android phone). Tensorflow exposes two functions specifically for the purpose of loading and dumping the model files generated as a part of the Saver class in tensorflow:

- `saver.save` - The API that is used to save the given (training in-progress) model to a file.
- `saver.restore` - The API that is used to restore (or load) the given model from a file onto the memory.

Using the above two functions, it was possible to online learning and at the same time use the saved model in the mobile application.

To experiment the feature of running a tensorflow model on Android application, first we had to build a simple model to test the model on the application. For this, a simple example of perceptron based model for XOR was chosen. This model, implemented in tensorflow was able to help us learn about how to generate a pb file to dump the tensorflow graph into a file.

The tensorflow function `tf.train.write_graph()` provides a way to dump the graph only (without weights) to a file. Later a way to generate a pb file with the trained model was learnt by referring to one of the existing implementation of MNIST digit classification example code that was designed to run on Android.

The following example shows a simple way to dump the graph in Tensorflow:

```
# Create new graph for exporting
g_2 = tf.Graph()
with g_2.as_default():
      # Reconstruct graph
```

```
x_2 = tf.placeholder("float", [None, 784], name="input")
W_2 = tf.constant(_W, name="constant_W")
b_2 = tf.constant(_b, name="constant_b")
y_2 = tf.nn.softmax(tf.matmul(x_2, W_2) + b_2, name="output")

sess_2 = tf.Session()

init_2 = tf.initialize_all_variables();
sess_2.run(init_2)

graph_def = g_2.as_graph_def()

tf.train.write_graph(graph_def, './tmp',
        'model.pb', as_text=False)
```

The most important section in the snippet above is the last two lines which shows how to get the graph definition to dump the definition onto a file. Once the model.pb file is written, this can be used on the android application.

A simple snippet that shows how to use this in the android application has been shown below:

```
public class DigitDetector {
    static {
       System.loadLibrary("tensorflow_mnist");
    }

    private native int init(AssetManager assetManager, String model);
    public native int detectDigit(int[] pixels);

    public boolean setup(Context context) {
       AssetManager assetManager = context.getAssets();

       // load the model
       int ret = init(assetManager, "file:///android_asset/model.pb");
       return ret >= 0;
    }
}
```

The highlighted lines in the above snippet shows how the library is loaded dynamically and then the model is initialized to be used in the application. The above two snippets are given as example in the Github repository given here: https://github.com/miyosuda/TensorFlowAndroidMNIST

To verify if this model was working, the simple example application was deployed on an android mobile phone. Once we know that the model is working, the next task was to use this to build the RiskCog model to be deployed on the mobile phone.

Debugging Accuracy

After the initial evaluation of Tensorflow, the next task before migrating the current framework to tensorflow is to debug the issue with accuracy drop. During the two weeks of evaluating the mobile application, it was clear that the application accuracy was very low even after training with the data.

To find out the cause of the low accuracy, the first approach was to set up the framework on the local system and start debugging the framework by understanding the code. But the problem here was that, since the framework is setup on a local system, it is very hard to send the sensor data from the mobile application. Also, to perform testing, the mobile application has to be used - this is a hard task to do while the motive was to debug the accuracy. To easen the process of testing the server side framework, a small emulation client was written in Python.

The emulation client uses the files with sensor data on the server already uploaded to emulate the requests which the mobile application makes so that it is easier to test the framework in a predictable, repeatable way.

The source code of the emulation client can be found here:
https://gist.github.com/sandeepraju/662158a2f051efb14d4581a949dddf20

A simple log of the interface is shown below:

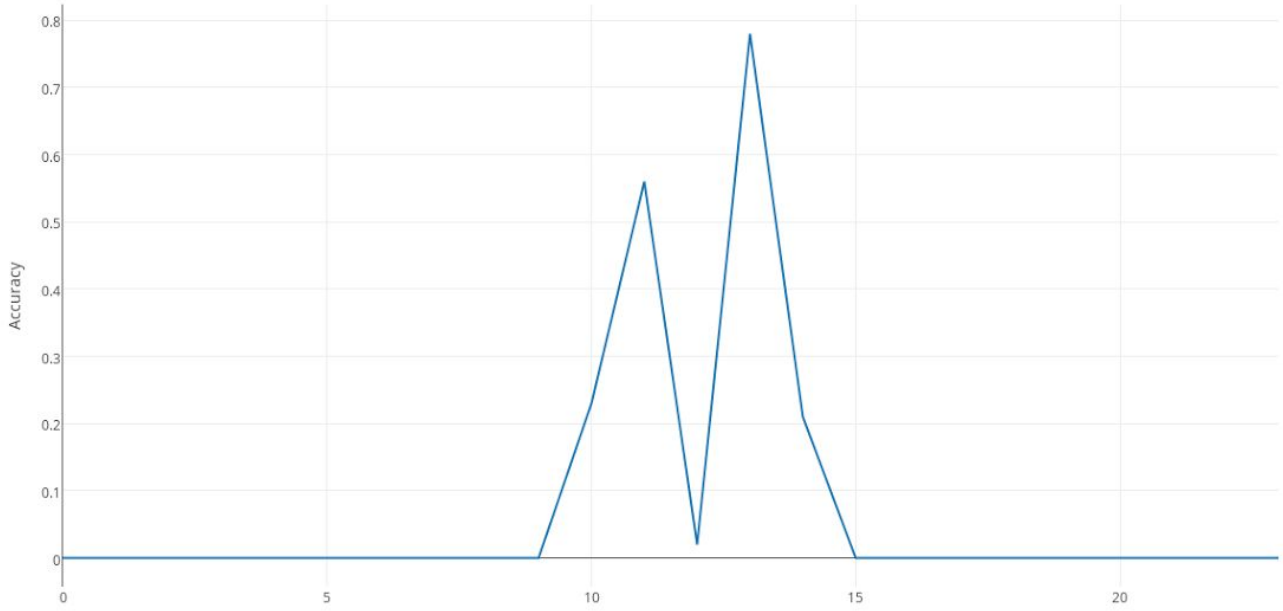
```
python test.py
Enter IMEI >>jiham01
{"message": "hello"}
loading data point number: 0
working on imei: jiham01
pick your choice:
* train
* test
* ask
* query
* manual
* exit
choice >>train
{"imei": "jiham01", "is_lie": false, "numfiles": 1, "result": 0}
{"sit_model_exist": false, "sit_trained": false, "walk_trained": false, "walk_model_exist": false}
loading data point number: 1
working on imei: jiham01
pick your choice:
* train
* test
* ask
* query
* manual
* exit
choice >>train
{"imei": "jiham01", "is_lie": false, "numfiles": 2, "result": 0}
{"sit_model_exist": false, "sit_trained": false, "walk_trained": false, "walk_model_exist": false}
```

After a few runs of data points to train, the model then reaches a point where the sit model is created. At this point the model can start to train and values of accuracy and recall can be obtained as below:

```
loading data point number: 13
working on imei: jiham01
pick your choice:
* train
* test
* ask
* query
* manual
* exit
choice >>test
{"max_version": 2}
latest_version 2
{"version": "2", "result":
"sit_accuracy=0.994974874372,walk_accuracy=0.0\nsit_precision=1.0,walk_precision=0.0\nsit_recall=0.
994974874372,walk_recall=0.0"}
loading data point number: 14
working on imei: jiham01
pick your choice:
* train
* test
* ask
* query
* manual
* exit
```

A live demo of this framework is available at the link here:
https://drive.google.com/open?id=0B6OojObu3NUWNXNNcHN2ekw1UW8

With this framework in place, testing the framework was a lot simpler compared to what it would have taken to debug the framework using the actual mobile application. Using this framework, further tests were conducted. Upon reading the code and running further experiments, we observed the following:



The above graph indicates that the accuracy remains 0 until the 1st 10 points have been trained. Then the model starts to train and it shows an initial accuracy. This accuracy is very intermittent and it does not persist. We can see that the data point 12 has brought the accuracy back to a very low value and then the next data point gets it to a very high value.

Another point to node here is that, the accuracy dropped to 0 from training run 15 and it never recovered back. The model was simply returning 0% accuracy regardless of how much training happens.

Along with this the commit rollback model was never working as shown in the paper. A demo showing this has been included in this link: https://drive.google.com/open?id=0B6OojObu3NUWTW1pWVJwU0QwNEU

Also, the way to determine if the model is trained or not was wrong. Initially the model was termed as trained as soon as there are 10 training data sets on the server. This is wrong since upon 10 files the model will be trained might be a very wrong assumption to make.

```
# enough datasets exist, start converting and training
if number >= data_source.number2start: # this number was hardcoded to 10
    parse_data(imei, dest_dir)
```

Also, the API check if the model was trained was returning wrong results - it just checks if there a model for either walk or sit model and returns True which is wrong since walk or sit might not be trained at all.

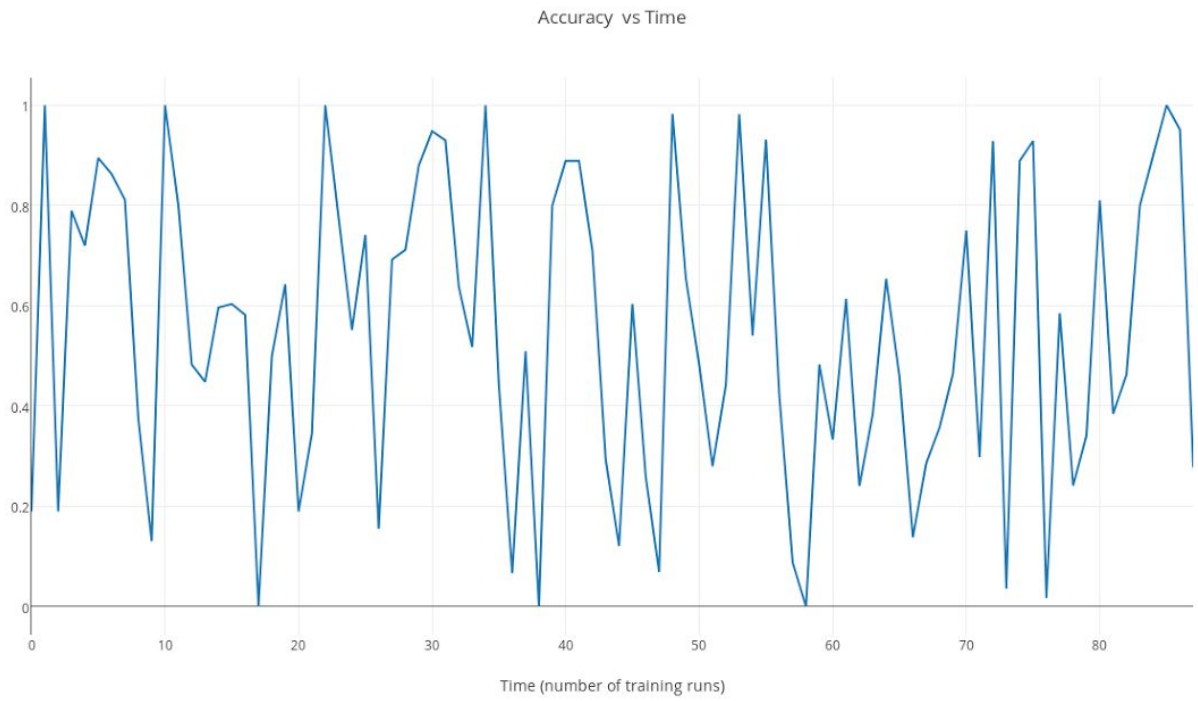
```
# the below condition is wrong
if os.path.exists(target_walk_file) or os.path.exists(target_sit_file):
      return_res ['trained'] = True
else:
      return_res['trained']  = False
```

The query API was returning just the max accuracy available which is wrong.

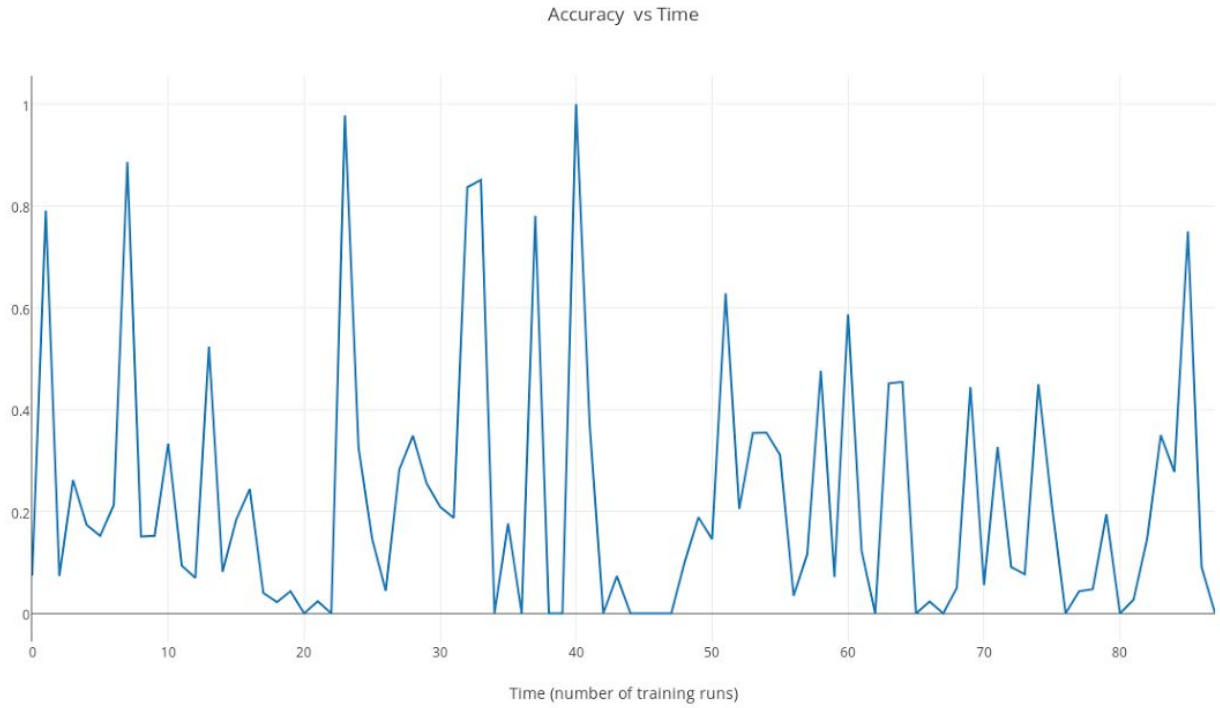
```
res1 = solve_state(result_dir, imei, 'sit', dest_name)
res2 = solve_state(result_dir, imei, 'walk', dest_name)
res = max(res1, res2) # this is wrong

fp = open(target_summary_file, "w")
fp.write(str(res))
fp.close()
```

Due to this the reported accuracy though seemed high clearly failed to work. We can see the reported accuracy graph below:



But upon further investigation, when individual accuracies were logged, it was found that the actual accuracy was found to be as shown below:



From the above graph, we can see that the actual accuracy was even bad compared to the reported accuracy.

There was some conflicts on how the API should behave. After discussion the following changes were proposed to the API:

- The ask_train should return if both the sit and walk model have completed the training. For checking if the training is completed, there is a accuracy threshold flag that is set. This flag checks the accuracy each time the model is trained. Once the model is fully trained and the accuracy crosses the set threshold, the api returns saying that the model is trained.
- The query and ask_trained should be merged to just return the accuracy once the model is termed as trained. When the testing has started and there is a summary file available, the API will return the accuracy, recall and threshold of the model for both sit and walk.

This design makes the API flexible so that the client can implement any functionality as required based on specific cases. Also, this ensures that the accuracy threshold is modified on the fly without updating the mobile application.

Also, a fix for commit rollback model was conceptualized. The initial solution was that, when a particular data file is marked as other, the framework should find all the files similar to that file and delete that file and retrain the model. When this is done, the accuracy is expected to increase. Unfortunately, the code never performed any training after fixing the data files. This code to perform training was added at the end of rollback.

Even after a lot of such changes, the accuracy increases wasn't very high. There was still a problem of accuracy dropping to zero.

To further investigate, why the framework differs so much from the tests on Tencent data. The reason for this was found out after checking the interval for which data collection was done.

Tencent collected the data as follows:

- Check if the current application is QQ or WeChat
- Collect the sensor data for 1 minute

RiskCog's way of collecting data is as follows:

- Check if any application is running
- Collection data for 6 seconds

Due to the change in the data collection duration, there might have been discrepancies in the data collection phase. To confirm this, the existing android application was modified to change the sensor collection duration to 30s. Over time, the data was collected and was used to perform tests.

The emulation client was used to run experiments to debug the drop in accuracy again. As soon as the debugging started, the following issues were seen and resolved:

- Some data points in the ARFF format had NaN. This is caused due to data corruption. This causes the framework to throw errors and in turn corrupt the entire model which was trained so far.

  A simple solution to this problem is skipping the lines which has NaN values. This seemed to solve the issue of accuracy dropping to zero for many cases.

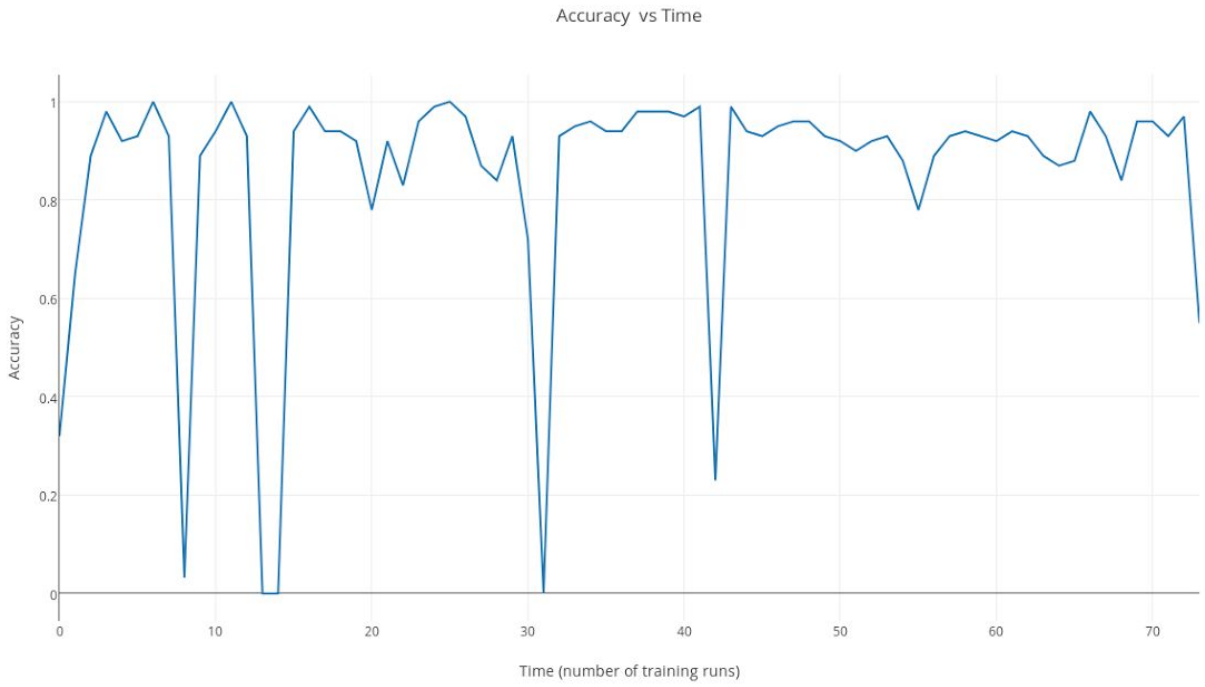
- Next, some of the data points which was used from Tencent's data to generate 1:4 dataset for training had corrupted data points whose value were in the millions range (compared to all other data points being in -1 to 1 range. This caused the VW framework to throw errors in certain cases.

  Again, the solution in this case is to simply ignore those data points and skip them while generating the vw input files.

- The third type of data related issue was, due to some dimensions missing in the data, the values are empty strings. This makes the framework read / parse the data wrongly.

  Just like the previous approaches, if there is a null string in place of a float where expected, then that data point is skipped.

Once these cases were handled, most of the problems of drop in accuracy is solved. Having said that, there is still drop in the accuracy at certain points. This is illustrated in the plot below:



If we see the plot in the previous case, we can see that compared to the one shown previously, the current plot shows that the accuracy is generally high but sometimes takes a plunge and comes back up.

Upon closer inspection of the points where the accuracy drops, the following observations were made:

- The number of data points in those cases were < 100
- Sometimes, all the data points belong to same class

This is due to the fact that some data can be corrupted and hence it is removed. Thus, the remaining data might be very less. Currently, a check has been added to ensure to print a log line when the data is very less.

While debugging and fixing all these bugs, it was found that the code is currently in a very bad state. Fixing one bug leads to breaking 10 unrelated things and is a cause of concern for anyone working on this codebase.

Android Mobile Application

One problem was that although the goal for the application was to ensure that it collects data when the user was using the phone, the process resulted in taking a very long time, which is not practical in real time settings. As a result, instead of using the approach that a file of data is collected when the user changes every two applications, it was changed so that it would be more time-based instead of event-based. This way, when the user opens the app, he/she understands that the application is undergoing training and will use the phone during the length of training. The training for each file takes around 3 seconds and buffer of 3 seconds. So in total it takes around 6 minutes to complete training to 100 files.

In addition, the UI is not very intuitive and it confuses the user on what is exactly going on. As a result, I have added another UI page called Info where if the user presses it, it opens a new page on the procedure and what

is expected to happen during training and after training. This way, the user understands how the application is supposed to work and will know if there is a problem if the process does not follow the Info page.

There are multiple bugs between the client and the server. Some phones do not work with the application because the application requires Android phones that have gravity, gyroscope, and accelerometer sensors. Even some phones that have the sensors sometimes have broken sensors and do not send compatible data files, which the server rejects. In addition, the server's SDK and the client's SDK have originally been mismatched and many functions have been deprecated and have been used. To combat this, a whole new server setup has been structured by Sandeep. But on the client side, I am still undergoing the process of changing and adapting the code so that the client and server can match in the new data collection method. The port of the new server is 8080. Instead of collecting data based off the number of files taken, the client would take data from a query from the server. It would ask the server if both "sit_trained" and "walk_trained" models have been trained. If it replies, false for any one of the two, it continues to train until both are stated yes. Once both are stated yes, it means that there is enough data for the server to accurately predict the next data set to know if the user is the true user or not. This is still an ongoing process and I hope to complete this by the end of the quarter. If this problem is fixed, then the application can finally work.

Android Smartwatch

For the data collection, the problems were easy to solve but the main problem was the data collection from the iWatch. The solution to the above mentioned problems is explained in detail.

The battery life decreases because of continuous data collection which was solved by decreasing the data points that were collected. Before the data collected was at every second, but by increasing the data collection to every 10 sec, the battery life increased by 1 hour. Thus, after defining the amount of data that is required, the time interval can be changed which leads to increase in battery life.

The 2nd problem that was faced was the stoppage of data collection after the update of android watch. This happened because the API of the application and that of the watch deferred. This was solved by changing the API using Android Studio and building the application. After the required changes, data was collected from the watch.

The other problem of running the application in background was solved temporarily but due to different updates and lower version of android, the working of the application in background was not suitable. Thus, not much changes were made for running the application in background.

The important challenge faced was the collection from iWatch. The data collected was very different from that of android watch. This was because the applications used were different. There were 2 ways to tackle the problem.

- Porting the Sensor Dashboard application from android to iOS using J2ObjC
- Programming the application from scratch in Objective C using Xcode in iOS

The 1st solution was downloading the J2ObjC software and installing it on Mac. But during the installation, many extensions were required that were either not supported by the iOS or outdated. Thus, it was not a feasible solution running j2objc software.

The 2nd solution is to program the application in objective C. It is not a feasible solution but the only way to get the same readings as android watch. The work done for programming is given in the readme file. It uses the iwatch OS 2.2 along with iOS 9.0 and above for programming purposes. Only accelerometer sensor has been configured but it shows a lot of error and there is work to be done..

Potential avenues to explore

Though the scope of this project for the quarter has been fulfilled, there are lot more things to accomplish. Some of the future avenues to explore and challenges to tackle has been listed below.

1. As explained previously, due to the data corruption mitigation strategies that include skipping the data points, there is a skew introduced into the data. This skew might be of two kinds

    a. The number of data points is very less such that the accuracy upon evaluation of these data points are either near to 0 or near to 1 which is a false representation of the system

    b. When data points are removed due to corruption, there might be cases where data of one label (ex. -1) is almost or even completely removed. Due to this, since the remaining data is of only one label, correct evaluation is not possible due to

    A mitigation strategy for this is to skip the dataset which was recorded if there is high amount of corruption. This implementation has been completed after the presentation and is ready before the report submission.

2. The current implementation has very simple approach to compute accuracy. The accuracy of the currently trained model is computed on the training dataset. Though this simple approach seems to work, it can fail or cause overfitting in some cases and can have undesirable consequences. There is a need to get a real picture of accuracy by picking a new data set and performing cross validation by picking and testing random data.

    This implementation is completed after the presentation and is ready before the report submission.

3. The commit roll-back model in the current implementation makes certain assumptions and updates the training data files based on that. Though the concept of commit-rollback works in theory, it does not seem to work in practice. This is a hard problem to test and there is a need to come up with a experimental strategy to perform a fair evaluation of the commit-rollback model

4. As explained in the previous section, there are changes made to the API. This new API interface has to be incorporated into the mobile application such that the request being made from the mobile application can leverage the full potential of the flexibility provided by the new API.

5. The current server side framework relies on Python Threads to perform some asynchronous computation that takes significant time, like testing, commit-rollback etc. During this process, it write and reads from multiple files in the file system which might cause race conditions if more than one thread is active at a given time. This race condition causes the model to be corrupt and thus lead to incorrect results.

    One of the ways to prevent this is to use a queueing system like RabbitMQ, RQ or Celery to queue these

operations and perform it async and serially rather than performing it on threads. This will ensure that the jobs are handled serially and thus will not corrupt any data or model.

6. The completion of the application for iOS so that data can be collected and verified with that of android data.
7. Configuring the RiskCog application so that the data is collected from the watch and sent to the server for training, authenticating and verification purposes.

References

1. http://scikit-learn.org/stable/modules/model_evaluation.html
2. http://scikit-learn.org/stable/modules/cross_validation.html
3. https://pymotw.com/2/threading/
4. https://www.tensorflow.org/versions/r0.12/get_started/index.html
5. https://www.tensorflow.org/versions/r0.12/how_tos/variables/index.html
6. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android
7. https://github.com/miyosuda/TensorFlowAndroidMNIST
8. https://developer.apple.com/reference/healthkit
9. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide
10. https://developer.apple.com/library/prerelease/content/navigation/
11. https://developer.apple.com/library/content/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/motion_event_basics/motion_event_basics.html
12. https://developer.apple.com/library/content/samplecode/AccelerometerGraph/Introduction/Intro.html
13. https://www.bignerdranch.com/blog/watchkit-2-hardware-bits-the-accelerometer/

Details of the source code

The updated source code for the Risk Server can be found on GitLab at the following link:

https://gitlab.com/riskcog/riskserver/tree/presentation

The updated source code for the client application can be found

https://www.dropbox.com/sh/9l9miarpdy1br44/AACaJBoniU6MwS6qa4uHfX6za?dl=0

The source code for Android Smartwatch can be found at the following link:

http://pan.baidu.com/s/1hsM9iza

The source code for Apple Smartwatch can be found at the following link:

https://drive.google.com/drive/folders/0Bxy-NcdMPFDwVV9vQXBYTDQwTFk?usp=sharing