AUGMENTING AUTHENTICATION WITH BEHAVIORAL BIOMETRICS IN A MOBILE BANKING APPLICATION

(MOBİL BANKACILIK UYGULAMASINDA DAVRANIŞSAL BİYOMETRİ İLE ARTIRILMIŞ KİMLİK DOĞRULAMA)

by

Okan Engin BAŞAR, B.S.

Thesis

Submitted in Partial Fulfillment
of the Requirements

for the Degree of

**MASTER OF SCIENCE
in**

COMPUTER ENGINEERING

in the

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

Nov 2019

This is to certify that the thesis entitled

AUGMENTING AUTHENTICATION WITH BEHAVIORAL BIOMETRICS IN A MOBILE BANKING APPLICATION

prepared by **Okan Engin BAŞAR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering** at the **Galatasaray University** is approved by the

Examining Committee:

Assoc. Prof. Dr. Özlem DURMAZ İNCEL (Supervisor)
Department of Computer Engineering
Galatasaray University

[illegible]

Assoc. Prof. Dr. Gülfem IŞIKLAR ALPTEKİN
Department of Computer Engineering
Galatasaray University

[illegible]

Assist. Prof. Dr. Ayşegül TÜYSÜZ ERMAN
Department of Computer Engineering
Işık University

[illegible]

Date: -------------------------

ACKNOWLEDGEMENTS

This thesis becomes reality with the kind support and help of many individuals. I would like to extend my sincere thanks to all of them.

I would like express my gratitude towards my family for the encouragement which helped me in completion of this thesis. My beloved and supportive wife, Fulya who is always by my side when times I needed her most and helped me a lot in making this study.

I would like to thank to my supervisor, Assoc. Prof. Dr. Özlem Durmaz İncel for imparting her knowledge and expertise in this study.

My thanks and appreciations also go to my colleague and people who have willingly helped me out with their abilities.

This work has been supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant number 5170078 and by the Galatasaray University Research Fund under grant number 17.401.004 and by Yapı Kredi Teknoloji, Istanbul.

Nov 2019
Okan Engin BAŞAR

TABLE OF CONTENTS

LIST OF SYMBOLS

10-CV	: 10-Fold Cross-Validation
ADB	: Android Debug Bridge
BKG	: Biometric Key Generation
CPU	: Central Processing Unit
CSV	: Comma Separated Values
EER	: Equal Error Rate
FAR	: False Acceptance Rate
FPR	: False Positive Rate
FRR	: False Rejection Rate
ML	: Machine Learning
PCA	: Principal Component Analysis
PIN	: Personal Identification Number
RBF	: Radial Basis Function
SDK	: Software Development Kit
SMS	: Short Messaging System
SVM	: Support Vector Machines
TPR	: True Positive Rate
WEKA	: Waikato Environment for Knowledge Analysis

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Smartphones have become very important and essential tools for our daily lives. They are not used just to communicate but each of them act as a smart personal assistant. We are playing, working and socialize wherever and whenever by using them. Even we make our banking transactions by using them instead of going to a bank branch. However, some possible security and privacy issues come in mind at this point. One of the greatest issues would be the theft or seizure of the smartphone by a third person. Banks take care of most of the cases by forcing the customer set a password which is not easily guessable. Also the banks send SMS messages or instant notifications to the customers as a second layer of security. These are good to have but may not be sufficient. Assume a customer unlocks the phone, crosses the second security layer into the banking application and then a thief steals the phone. The thief not only has the phone but also has the money that in the bank accounts at this point. This action is known as Account Takeover Attack in the literature. This thesis highlights the potential risks that occur when smartphones are stolen or seized at this kind of moments, and provides a solution to account takeovers by using continuous authentication concepts, like continuous user identification via touch and micro movements, and the mechanisms of behavioral biometrics. The solution is implemented inside a mobile banking application and the data is collected with this application. The collected data is modeled utilizing a machine learning algorithm. The details of the augmentation process and the test results in terms of authentication performance and resource consumption are also provided.

ÖZET

Akıllı telefonlar hayatımızda çok önemli hale geldi. Sadece iletişim kurmak için değiller. Her biri akıllı bir kişisel asistandır. Oynuyoruz, çalışıyoruz ve nerede olursak olalım, onları kullanırken sosyalleşiyoruz. Hatta bir banka şubesine gitmek yerine onları kullanarak bankacılık işlemlerimizi yapıyoruz. Bununla birlikte, bu aşamada bazı olası güvenlik ve gizlilik sorunları akla geliyor. En büyük sorunlardan biri de akıllı telefonun üçüncü bir kişi tarafından çalınması veya ele geçirilmesidir. Bankalar müşterilerini kolaylıkla tahmin edilemeyecek bir şifre belirlemeye zorlayarak bu tür durumların çoğunu bertaraf eder. Ayrıca bankalar müşterilere ikinci bir güvenlik katmanı olarak SMS mesajları veya anlık bildirimler de gönderir. Bunların olması güzel ama yeterli değil. Bir müşterinin telefonun kilidini açtığını, ikinci güvenlik katmanını geçerek bankacılık uygulamasına girdiğini ve ardından bir hırsızın telefonu çaldığını varsayalım. Hırsız bu noktada sadece telefonu değil aynı zamanda banka hesaplarındaki parayı da elde etmektedir. Bu eylem literatürde Hesap Devralma Saldırısı olarak bilinmektedir. Bu tez, akıllı telefonların çalınması, izinsiz ele geçirilmesi ve yetkisiz kişiler tarafından işlem yapılması ile ortaya çıkabilecek olası risklere karşı, dokunma ve cihazın mikro hareketlerini izleyerek eğitilen bir davranış modeli ile hesap devralmayı engelleyen sürekli kimlik doğrulaması yapılan yeni bir güvenlik katmanı çözümü sunmaktadır. Bu çözüm bir mobil bankacılık uygulaması içerisinde uygulanmış ve tüm veriler bu uygulama ile toplanmıştır. Toplanan veriler bir makine öğrenme algoritması kullanılarak modellenmiştir. Doğrulama performansı ve kaynak tüketimi açısından değerlendirilen test sonuçları ve tüm uygulama süreçlerinin detayları da belirtilmiştir.

1. INTRODUCTION

Smartphones have become important gadgets used in our daily lives. With a variety of apps, they are not only used for communication but also for different purposes, such as accessing social networks, browsing through the Internet, watching videos, navigation, step counting, bank transfers, etc. Often, users store personal/private information, such as photos, videos, or sensitive information such as passwords on these devices. Due to their small sizes compared to personal computers or tablets, they are also prone to get stolen or be lost and can be accessed by non-owners. If an non-owner or intruder has physical access to a device, he/she can cause monetary or non-monetary damage to the owner. Therefore, protecting the security and privacy of smartphone users against unauthorized access and providing secure authentication on these devices are important issues.

Mobile banking applications are one of the most sensitive apps for secure authentication and they are widely used by customers due to ease of access and use. These applications are required to perform remote authentication using user credentials consisting of user-name and password. Moreover, usually a confirmation password is required by the bank after the completion of a transaction as an additional security measure. Although, additional passwords bring a benefit, the use of a confirmation password extends the processing time, which may cause difficulty in the application usage for users. As an alternative method, continuous authentication using behavioral metrics can be utilized in mobile banking applications. Users exhibit different patterns while interacting with apps, or mobile devices and behavioral biometrics aim to identify users according to their unique patterns. Data related to keystrokes, touch-screen use and sensor data can be used to identify these patterns. Compared to physical biometrics, behavioral biometrics provide lower levels of security for authentication but they have the advantages of working in the background and not disrupting the user experience, and working over a session rather than a one-shot authentication. Moreover, rather than replacing a username-password based authentication, they can be used as an additional measure for authentication or they can replace the additional passwords required after transactions.

In this thesis, the augmentation of a mobile banking application with continuous authentication using behavioral biometrics is investigated. A mobile banking application from a local bank in Turkey as part of a research project is utilized. Particularly, the augmentation of the app with a data logging tool is discussed. The logger collects touch data while the user interacts with the screen, such as finger pressure, finger size and X&Y coordinates on the screen, as well as sensor data from accelerometer, gyroscope and magnetometer. 20 users collected the initial data for a pilot study. By preprocessing, the features are extracted from raw data. These extracted features were analyzed using machine learning algorithms to identify users. Especially one-class SVM classification is evaluated in detail. In addition to these, the resource consumption of the logger-augmented mobile banking application is explored in comparison to the app without the logger to investigate the overhead of data logging. As the metrics, power/battery consumption, CPU and memory usage were considered. Moreover, the resource usage with using accelerometer alone, as well as in combination with a gyroscope and magnetometer were analyzed. The resource consumption with varying sensor sampling rates was also tested. A scenario simulating a real-user was followed on the mobile banking app while collecting the data. The performance results show that touchscreen is the cheapest one in every aspect of resource consumption. Accelerometer is the least consumer among three sensors and as the sampling rate increases, the energy consumption and CPU usage can also dramatically increase after some point. Hence, sensor sampling rates should be kept at minimum in order not to disrupt user experience.

The importance of the smartphones and the potential security and privacy issues that occur when smartphones are stolen or seized are highlighted. The current techniques that are used and the methodologies that are followed to prevent these issues are explained. Continuous authentication concepts, like continuous user identification via touch and micro movements, are mentioned in detail. This thesis puts behavioral biometrics on the table and shows its importance in terms of security. The main topic of this thesis is how to increase the security level in a mobile banking application by using behavioral biometrics.

In order to provide security, the user is asked for the username and password information to provide security for the bank. Although the access to the mobile banking application is password protected, additional security controls are also necessary taken into account to prevent a fraudulent account. For this purpose, after the initial authentication, as an additional security control, a confirmation password is issued to complete the transaction. Using a confirmation password extends the processing time and complicates the bank customer's application usage.

In this thesis, it is suggested to use behavioral biometrics methods that are new for the Bank. These authentication methods will be considered as an additional control layer. In the related academic literature, the so-called 'behavioral biometry' method is based on the analysis of human-device interactions to protect users and their data and aims to distinguish the person behind a session. Among the types of behavior and interaction planned to be followed, there will be examples such as typing speed and style, pressure applied to the screen, transitions between screens. Thus, in mobile banking application, the Bank will be able to skip the confirmation password from the customer in the acceptable transactions for the bank, and the customer can be authenticated so that both fraud cases can be prevented and the use of the banking application will be facilitated.

This thesis provides five main outputs:

1. A software to be integrated into the mobile banking application that collects data on user behavior
2. User data collected during testing process of this software (raw data)
3. Feature set calculated by preprocessing the raw data
4. Results of the classification performance analysis
5. Results of the resource consumption analysis.

All the outputs mentioned above are explained in detail in this thesis. The rest of the thesis is organized as follows:

In Section II, the related work is presented.

In Section III, the components of the client application and its augmentation process with the Logger are described.

In Section IV, the data collection and analysis methods are explained. The data preprocessing and methodology being followed are also described. Feature engineering details are provided.

In Section V, the results of the performance evaluations are presented. SVM classification performance and the results of the application resource consumption analysis are provided.

Finally, in Section VI, the conclusion and future work are presented.

2. RELATED WORK

The use of continuous authentication based on behavioral biometrics is a promising area of study to enhance the security of the smartphones and there are numerous studies that have used behavioral biometrics. Although there are many types of behavioral biometrics, these studies mainly focused on hand waiving, key stroke and touchscreen behaviors. These studies extract various features such as pressure, finger size, touch location, and gesture type.

Alzubaidi A. and Kalita J. (2016) mentioned about potential risks that occur when smartphones are stolen or seized in their survey. The current approaches and mechanisms of behavioral biometrics with respect to methodology, associated datasets and evaluation approaches were analyzed.

Stylios I., et al. (2016) reviewed a summary of selected published material on this subject, with additional disclosure, critical analysis of contents and, in some cases, the main conclusions of each study. Their work can support new scientists and researchers by giving them hypothetical and viable viewpoints on Continuous Authentication utilizing behavioral biometrics.

Bo C., et al. (2014) presented a new framework called SilentSense to authenticate users using the user touch behavior biometrics and the micro-movement of the device caused by the user's touch screen actions. They have created a touch-based biometric model of the owner and then confirmed whether the current user is the owner or the guest / attacker. While using the smartphone, some unique working dynamics of the user are learned by collecting sensor data and silently touching events. Micro movement of mobile devices identifies touching from large-scale movements of the user. Touch-based

based biometrics can be neutralized. To address this, they integrated a movement based biometrics for each user with previous touch-based biometrics. They also conducted extensive evaluations of their approaches on the Android smartphone and showed that the user identification accuracy is over 99%.

Sitová Z., et al. (2016) introduced HMOG, a set of behavioral features to continuously authenticate smartphone users. HMOG was featured with micro-movements and orientation dynamics. According to this study, how a user grasps, holds, and taps on the smartphone can be used to identify the user. They also evaluated authentication and BKG performance of HMOG features. Sitting and walking data were collected from 100 subjects while they were typing on a virtual keyboard. They compared EERs of the activities of subjects.

Meng W., et al. (2015) investigated the use of both physical and behavioral biometric data. The data classes such as signature activity, walking pattern, applications used, the pattern of use of the phone keys, the pattern of the touch gestures of the screen were examined and the advantages and disadvantages of each method were discussed. In addition, different evaluation metrics (error rate, false acceptance rate, etc.) were used in studies and their performance against possible security attacks were investigated.

Patel V. M., et al. (2016) emphasized that the sensors, such as camera, microphone, etc. can be used to collect physical data, while components such as accelerometers, gyroscopes, touch screens can be used to collect behavioral biometric data, such as walking, screen touch gestures, and hand movements. In the literature, it has been stated that user validation using behavioral biometry is expressed in different ways, such as continuous authentication, active authentication, indirect authentication, and transparent authentication. The advantages of authenticating with behavioral data, as well as physical data such as fingerprints, compared to active authentication, have been defined as ease of use and continuous operation.

Rahman F., et al. (2014) suggested to use user's location, walking pattern, and the use of an image in the context authentication with different data modalities. However, an application and performance analysis has not been presented.

Fridman L., et al. (2017) constructed a dataset comprising of the text entered with the virtual keyboard, the applications used, the visited websites, and the physical location data of the device from 200 users.

Fridman L., et al. (2015) performed the analysis of keyboard usage dynamics and mouse movements as well as stylometry. In this study, laptops were considered instead of mobile devices.

Buriro A., et al. (2017) investigated a mobile banking application for continuous authentication. Touch-strokes' timing-differences and the phone-movements during the process of entering PIN/password were collected. However, the study focused on the performance of identification and neglected resource consumption. Moreover, data was collected only when entering PIN and password, while we collect data throughout a complete session, from login to logout.

To sum up, some of the related works were surveys and the others mainly focused on physical and virtual keyboard usage. Although Buriro A., et al. (2017) investigated mobile banking application for continuous authentication, this work was only related to login screen. In this thesis, a mobile banking application is investigated in much more detail. The continuous authentication mechanism is applied not only to login screen but to the all screens during a secure session. Moreover, the results of the resource consumption analysis are also provided.

3. EXTENDING THE MOBILE BANKING APPLICATION WITH A LOGGER

In this section, the components of the client application and its augmentation process with the Logger are described. Figure 3.1 shows the component diagram of the system.

This thesis is mainly focused on the client application which is the mobile banking application and the logger implementation. All the details related to the mobile banking application is provided in Section III. The Logger is explained and the implementation is provided in Section IV. The preprocessing of the raw data is also explained in Section IV. The authentication performance and the resource consumption analysis are provided in Section V.

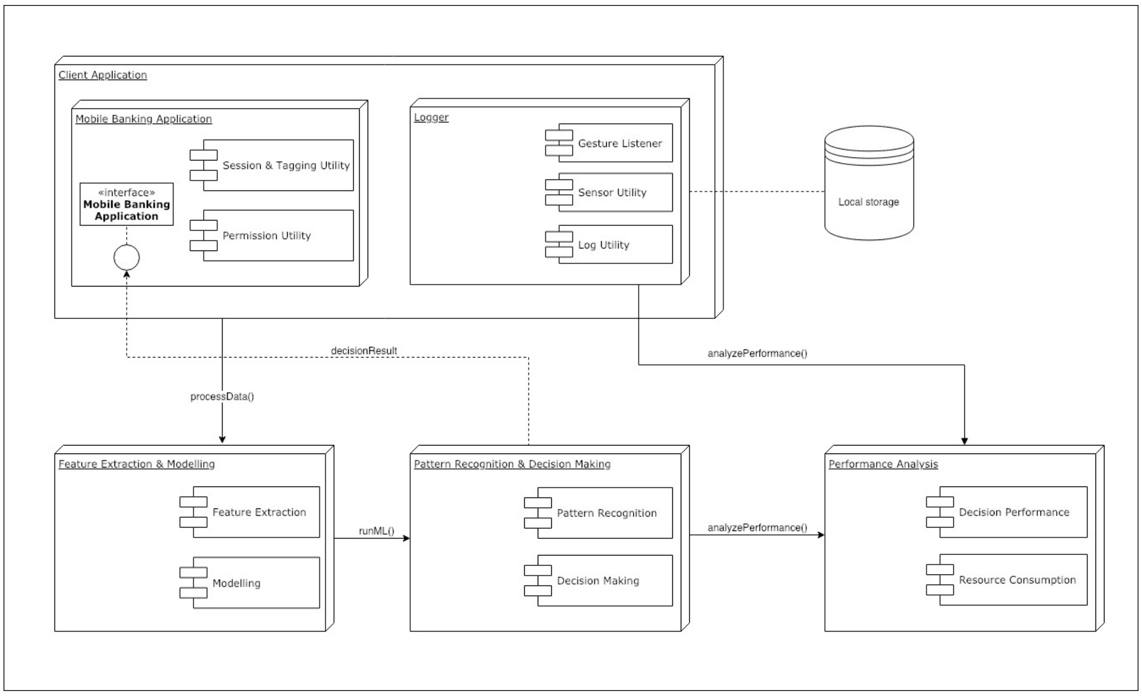


Figure 3.1: Component diagram of the system

3.1. The Mobile Banking Application

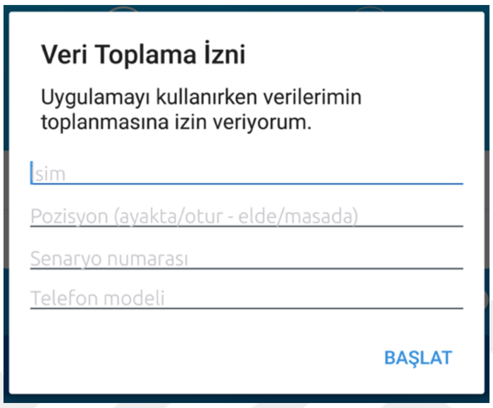
The base application, which is a mobile banking application, has been natively implemented by the bank in Java programming language to operate in mobile phones having Android operating system. By default, there is neither continuous authentication mechanism nor usage of sensors installed on device. Also, the touch events are not being logged.

In order to extend this application to have continuous authentication mechanism by using behavioral biometrics, the required data should be continuously collected. The details of the data being collected is provided in Section IV.

The base application is utilizing the fragment and the activity lifecycles. Although the application is huge, in other words it has thousands of screens, there are only couple of activities for different purposes. The rest of the screen designs are utilizing fragment architecture. For example, the NonSecureActivity is activity used for the fragments before the login screen, and MainActivity is used for the fragments after the login. All the activities inherit from the BaseActivity. Because of this, we make use of BaseActivity in order to access and keep in touch with all the fragments implemented inside the base application. The BaseActivity is used for application wide operations like activity management, fragment management, flow management, user permissions, etc.

3.1.1. Session & Tagging Utility

Before collecting any data, the user consent should be taken. Figure 3.2 shows the screenshot of a sample consent with some tagging parameters. The implementation is provided in Appendix A.



Veri Toplama İzni

Uygulamayı kullanırken verilerimin toplanmasına izin veriyorum.

İsim

Pozisyon (ayakta/otur - elde/masada)

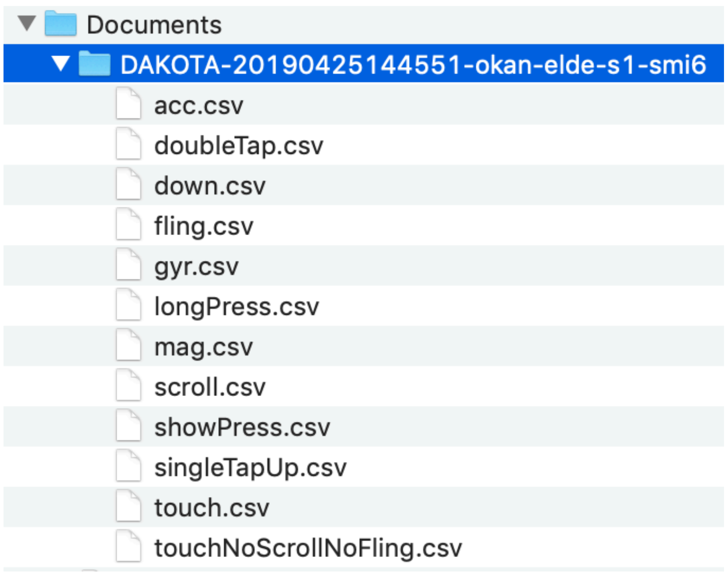
Senaryo numarası

Telefon modeli

BAŞLAT

Figure 3.2: The screenshot of the user consent pop-up

After the consent is taken, data collection starts with folder and file creation process. The data coming from sensors and touch events are recorded to the filesystem. Before the data is generated, session utility creates the corresponding file structure (Figure 3.3). This is mentioned as Session & Tagging Utility in the component diagram.



```
Documents
  DAKOTA-20190425144551-okan-elde-s1-smi6
    acc.csv
    doubleTap.csv
    down.csv
    fling.csv
    gyr.csv
    longPress.csv
    mag.csv
    scroll.csv
    showPress.csv
    singleTapUp.csv
    touch.csv
    touchNoScrollNoFling.csv
```

Figure 3.3: The file structure

3.1.2. Permission Utility

In order to create the file structure and save the obtained the sensor and the touchscreen data to the filesystem, the external storage write permission is taken in the BaseActivity. Figure 3.4 shows how to ask for a runtime permission. This is mentioned as Permission Utility in the component diagram.

```
@RuntimePermissions
public abstract class BaseActivity extends AppCompatActivity {
 ...
 @NeedsPermission({Manifest.permission.WRITE_EXTERNAL_STORAGE})
 public void setupDAKOTA(BaseActivity activity, String name, String position,
 String scenario, String phoneModel) {
     DakotaLogUtil.setupFiles(activity, name, position, scenario, phoneModel);
     DakotaLogUtil.isDakotaFileSetUpCompleted = true;
 }

 @Override
 public void onRequestPermissionsResult(int requestCode, String[] permissions,
 int[] grantResults) {
     super.onRequestPermissionsResult(requestCode, permissions, grantResults);
     BaseActivityPermissionsDispatcher.onRequestPermissionsResult(this,
 requestCode, grantResults);
 }
 ...
}
```

Figure 3.4: Usage of runtime permissions

3.2. The Logger

The Logger is the component that listens gestures and sensors on the device and logs all the gathered information. It is designed to be part of the client application in order to collect the behavioral data of the users of the application. It consists of 2 utilities and 1 listener. The utilities are Sensor Utility and Log Utility. These utilities are responsible for the sensor operations and the data logging respectively. The sensors listened on the device are the accelerometer, gyroscope and magnetometer. The Gesture Listener, on the other hand, is responsible for detecting gestures like scroll, fling, long press, show press, single tap down, single tap up and double tap.

3.2.1. Gesture Listener

Android SDK has built-in class called GestureDetector. It provides us GestureListener interfaces to implement and SimpleOnGestureListener class to extend in order to obtain the gestures listed above. The implementation is provided in Appendix E.

Sometimes the interactions are not identified as gesture. Therefore, all the motion events are also dispatched in the BaseActivity to get every touchscreen interaction data by extending dispatchTouchEvent method coming from AppCompatActivity which is a built-in activity in Android SDK (Figure 3.5). The implementation is also provided in Appendix B.

```
public boolean dispatchTouchEvent(MotionEvent ev);
```

Figure 3.5: TouchEvent dispatcher method signature in Android SDK

3.2.2. Sensor Utility

The sensor events are listened by registering the corresponding event listeners to the Sensor Manager, a built-in class in Android SDK, to access device sensors. Figure 3.6 shows how the listener registration takes place in onResume, activity callback method which is also inherited from AppCompatActivity. This callback method is called when your activity is just started to interact with the user.

There are 3 sensors that are listened; accelerometer, gyroscope and magnetometer. The sensor delay is the parameter that defines the sensor sampling rate. SENSOR_DELAY_NORMAL (200ms delay) is nearly equivalent to 5Hz sampling rate. In Section V, the importance of the sampling rate is emphasized and the comparison of different sampling rates, specifically 5 Hz, 20 Hz and 100 Hz, is made in terms of resource consumption.

```
@Override
public void onResume() {
    super.onResume();
    ...
    if (DakotaSensorUtil.sensorManager != null &&
!DakotaSensorUtil.sensorListening) {

        if (DakotaSensorUtil.gyroscopeSensor != null &&
DakotaSensorUtil.gyroscopeEventListener != null) {

DakotaSensorUtil.sensorManager.registerListener(DakotaSensorUtil.gyroscopeEven
tListener, DakotaSensorUtil.gyroscopeSensor,
SensorManager.SENSOR_DELAY_NORMAL);
        }

        if (DakotaSensorUtil.accEventListener != null) {

DakotaSensorUtil.sensorManager.registerListener(DakotaSensorUtil.accEventListe
ner, DakotaSensorUtil.accSensor, SensorManager.SENSOR_DELAY_NORMAL);
        }

        if (DakotaSensorUtil.magEventListener != null) {

DakotaSensorUtil.sensorManager.registerListener(DakotaSensorUtil.magEventListe
ner, DakotaSensorUtil.magSensor, SensorManager.SENSOR_DELAY_NORMAL);
        }

        DakotaSensorUtil.sensorListening = true;
    }
    ...
}
```

Figure 3.6: Registration of sensor event listeners in onResume

Since the sensors are serious energy consumers, as described in Section V, they should be active as short as possible. For this reason, the event listeners are unregistered from Sensor Manager in onPause activity callback method as shown in Figure 3.7 in order to suspend data collection while the application is not active. This callback method is called when your activity is not visible the user.

```
@Override
public void onPause() {
    ...
    if (DakotaSensorUtil.sensorManager != null) {

        if (DakotaSensorUtil.gyroscopeSensor != null &&
DakotaSensorUtil.gyroscopeEventListener != null) {

DakotaSensorUtil.sensorManager.unregisterListener(DakotaSensorUtil.gyroscopeEve
ntListener);
        }

        if (DakotaSensorUtil.accSensor != null &&
DakotaSensorUtil.accEventListener != null) {

DakotaSensorUtil.sensorManager.unregisterListener(DakotaSensorUtil.accEventList
ener);
        }

        if (DakotaSensorUtil.magSensor != null &&
DakotaSensorUtil.magEventListener != null) {

DakotaSensorUtil.sensorManager.unregisterListener(DakotaSensorUtil.magEventList
ener);
        }

        DakotaSensorUtil.sensorListening = false;
    }

    super.onPause();
}
```

Figure 3.7: Unregistration of sensor event listeners in onResume

3.2.3. Log Utility

After the data collection consent and the write permission is taken, the session is started by the Session & Tagging Utility. At this point, data is started to be generated by the Gesture Listener and the Sensor Utility as long as the base application is running and visible on device screen. This data should be kept for a while on the device, until all the data is exported to the server.

As mentioned, there exists two main data sources; the Gesture Listener and the Sensor Utility. The corresponding implementations are provided in Appendix E and Appendix D.

The current data structure is the generated 12 CSV files on local file system as listed in Figure 3.3. The column names are also listed in Table 3.1. Each sensor data has X, Y and Z axis values. Each touchscreen data, on the other hand, has finger size, finger pressure, and X and Y coordinates of the screen. Fling and scroll gestures have also velocity information for each axis. Therefore these information have their own columns in the corresponding files. All the files have also the Time and Fragment Name columns to keep track of when and where the action takes place. The implementation of Log Utility is attached as Appendix C.

Each session has a folder containing these files. When a new session is started, also a new folder is created for it. Each folder has some tagging information on it. The timestamp, the username, the position, the scenario number and the phone model. The timestamp is generated automatically. The other parameters are optional and currently obtained manually in the consent popup to get an idea about the data. When the application is deployed to production environment, the user information which already exist in the bank’s database will be used. The other parameters will not be necessary.

Accelerometer: - Time - X Axis - Y Axis - Z Axis - Fragment Name	Gyroscope: - Time - X Axis - Y Axis - Z Axis - Fragment Name	Magnetometer: - Time - X Axis - Y Axis - Z Axis - Fragment Name
Touch: - Finger Pressure - X Axis - Y Axis - Finger Size - Time - Fragment Name	TouchNoScrollNoFling: - Finger Pressure - X Axis - Y Axis - Finger Size - Time - Fragment Name	Down: - Finger Pressure - X Axis - Y Axis - Finger Size - Time - Fragment Name
Fling: - Finger Pressure - X Axis - Y Axis - Finger Size - X Axis Velocity - Y Axis Velocity - Time - Fragment Name	Double Tap: - Finger Pressure - X Axis - Y Axis - Finger Size - Time - Fragment Name	Long Press: - Finger Pressure - X Axis - Y Axis - Finger Size - Time - Fragment Name
Scroll: - Finger Pressure - X Axis - Y Axis - Finger Size - X Axis Velocity - Y Axis Velocity - Time - Direction - Fragment Name	Show Press: - Finger Pressure - X Axis - Y Axis - Finger Size - Time - Fragment Name	Single Tap Up: - Finger Pressure - X Axis - Y Axis - Finger Size - Time - Fragment Name

Table 3.1: The raw data saved by logger to CSV files

4. DATA COLLECTION AND ANALYSIS

In this section, the data collection and analysis methods are explained. The data preprocessing and methodology being followed are described. Feature engineering details are provided.

According to the analysis performed on the current customers of bank, the mostly used functions in the application are determined and combined together to generate scenarios. The measured average session duration is around 3 minutes. To achieve this duration, the customer behavior is simulated by following 5 scenarios listed below.

1) Account and credit card balance control on dashboard,
2) Searching accounts on account list and balance control,
3) Money transfer from one account to another,
4) Foreign exchange buy operation,
5) Credit card debt payment.

The data is collected by following these scenarios after login in each session. Although there are unlimited number of different stances, three of them are selected to be the most common ones to simulate the scenarios. Every user has completed scenarios in each of the following stances.

1) Holding the phone in hand and standing
2) Holding the phone in hand and sitting
3) Holding the phone on the table and sitting

3 device models are used in this thesis. The devices used to collect data are Samsung Galaxy S9 and Xiaomi Mi8. The resource consumption analysis, on the other hand, is performed on Samsung Galaxy S3 Neo since the other models do not support the power estimation utilities.

The data is collected from 20 users. Each user has followed the above-mentioned 5 scenarios in corresponding sessions. Each user then alters the stance, and repeats the same scenarios, and then alters the stance again to repeat ones more.

That makes 15 sessions per user. 1 user has 13 sessions. 18 users have 15 sessions. 2 users have 30 sessions to see the importance of training dataset size in classification performance. Table 4.1 shows the list of users. In total, 47.31 MB of data is collected from 20 users.

User ID	Gender	Phone Model	Session Count	Data Size (bytes)
2	F	Samsung S9	15	2,696,179
3	F	Samsung S9	15	3,258,477
4	M	Samsung S9	15	2,655,264
5	M	Samsung S9	15	2,885,752
6	M	Samsung S9	13	3,351,233
8	M	Samsung S9	15	2,901,635
30	M	Xiaomi Mi8	30	3,916,120
31	F	Xiaomi Mi8	15	2,257,572
50	M	Xiaomi Mi8	15	2,117,162
51	M	Xiaomi Mi8	15	2,529,388
52	M	Xiaomi Mi8	15	2,494,544
53	M	Xiaomi Mi8	15	1,976,520
54	M	Xiaomi Mi8	15	3,101,618
55	F	Xiaomi Mi8	15	1,783,096
56	F	Xiaomi Mi8	15	2,343,062
57	M	Xiaomi Mi8	15	2,496,702
70	F	Xiaomi Mi8	30	3,951,858
71	F	Xiaomi Mi8	15	2,433,167
72	M	Xiaomi Mi8	15	2,515,956
73	M	Xiaomi Mi8	15	1,932,334

Table 4.1: The list of users

4.1. Feature Extraction

The raw data is preprocessed and 66 features are extracted. These features are introduced by Sitová Z., et al. (2016) and the details of each feature can be found in their work. Since the dataset in this thesis is very similar to the one in HMOG paper, the same features are extracted and utilized. The extracted features are listed on Table 4.2 and they are obtained from only scroll events and the corresponding sensor data. Although other gestures might contribute to another machine learning model by their own feature set, this is left as future work. For this reason, only the transactions that consist scrolling on screen can be authenticated by this machine learning model. The scrolling event comes from every kind of list. For example; the account list, account transactions list, credit card list, credit card transactions list, funds list, stocks list, etc. It does not matter whether the user has lots of accounts or credit cards. If the operation includes any kind of list scrolling, that session can be authenticated with this approach. The Python source codes used for feature extraction and feature merging are attached as Appendix F, Appendix G, Appendix H, and Appendix I.

X_acc_mean	X_acc_std	X_acc_median
Y_acc_mean	Y_acc_std	Y_acc_median
Z_acc_mean	Z_acc_std	Z_acc_median
X_gyr_mean	X_gyr_std	X_gyr_median
Y_gyr_mean	Y_gyr_std	Y_gyr_median
Z_gyr_mean	Z_gyr_std	Z_gyr_median
X_mag_mean	X_mag_std	X_mag_median
Y_mag_mean	Y_mag_std	Y_mag_median
Z_mag_mean	Z_mag_std	Z_mag_median
Acc_Mag_mean	Acc_Mag_std	Acc_Mag_median
Gyr_Mag_mean	Gyr_Mag_std	Gyr_Mag_median
Mag_Mag_mean	Mag_Mag_std	Mag_Mag_median
START_X_first	CURRENT_X_last	CURRENT_X_maxdev
CURRENT_X_dev20	CURRENT_X_dev50	CURRENT_X_dev80
START_Y_first	CURRENT_Y_last	CURRENT_Y_maxdev
CURRENT_Y_dev20	CURRENT_Y_dev50	CURRENT_Y_dev80
V_pairwise20	V_pairwise50	V_pairwise80
A_pairwise20	A_pairwise50	A_pairwise80
V_medianVelocity LastThree	A_averageAccFirstFive	pairwiseDisplacement_ lengthOfTrajectory
CURRENT_PRESSURE_ median	CURRENT_SIZE_median	distance
directionOfEndtoEnfLine	ratio	duration
averageVelocity	MeanResultantLength	AverageDirectionEnsemble

Table 4.2: The list of extracted features

4.2. Feature Transformation

In order to improve authentication performance, feature transformation with Principal Component Analysis is performed. It is a statistical procedure that transforms a number of possibly correlated variables into a smaller number of linearly uncorrelated variables called principal components without losing any information. By applying PCA, 66 features are transformed into 23 and better SVM classifier performance achieved on reduced number of features.

4.3. Classification

SVM is a supervised ML algorithm which can be used for classification, regression purposes and mostly for the pattern recognition. When it is used for classification, this algorithm tries to find the optimal hyperplane which strictly classifies the data points with a maximum margin in N-dimensional space where N is the number of features. Figure 4.1 shows the possible hyperplanes in 2-dimensional space.

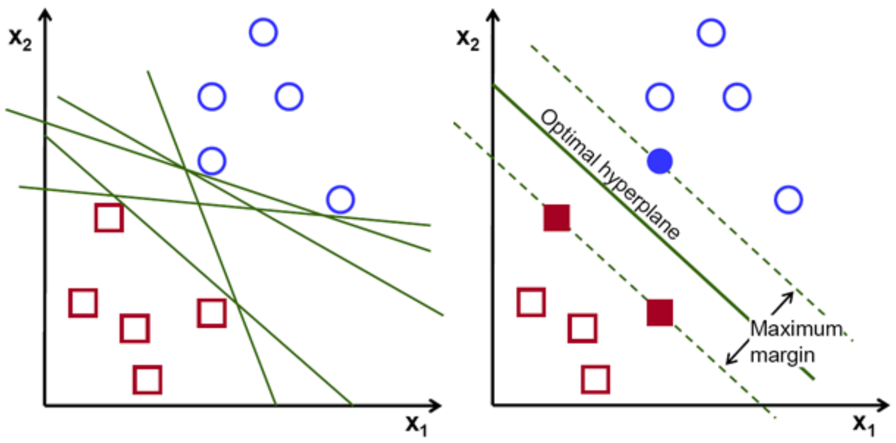


Figure 4.1: Possible hyperplanes in SVM in 2-dimensional space

One-class SVM classification, on the other hand, is an unsupervised learning algorithm which is used to distinguish the target class from all other classes by using only target class training data. If the outliers are not represented well in the training set, this algorithm is much suitable. The aim is to separate the data from the origin in the N-dimensional predictor space and detect the outliers. Rana, D. (2015) compared two SVM classification algorithms and this comparison is given in Table 4.3.

One-Class SVM	**SVM**
Contains data from only one class, target class.	Contains data of two or more classes.
Goal is to create a description of one class of objects and distinguish from outliers.	Goal is to create hyperplane with maximum margin between two classes.
Decision boundary is estimated in all directions in the feature space around the target class.	Hyperplane is created in between datasets to indicate which class it belongs to.

Table 4.3: One-class SVM vs SVM

Since the model ought to be trained with the single user's data and the purpose is to classify new data whether they belong to this class or not, one-class SVM classification algorithm was used. WEKA libSVM library contains one-class-SVM implementation. This implementation is used in experiments.

When training an SVM classifier with the kernel type RBF, nu and gamma parameters must be considered. The influence amount of a single training instance on the classification is defined as gamma.

The nu parameter, is upper bounded by the fraction of outliers and lower bounded by the fraction of support vectors. This parameter should be optimized to tune the trade-off between overfitting and generalization.

All experiments showed that the lowest possible gamma value, 1.0E-7, has the best authentication performance. In order to be consistent in SVM, all the target attributes are normalized by setting the normalize parameter to true. All other parameters are left as default in WEKA 3.8. Future experiments are made based on these parameters.

Classifier configuration:

weka.classifiers.functions.LibSVM -S 2 -K 2 -D 3 -G 1.0E-7 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -Z -model "C:\\Program Files\\Weka-3-8" -seed 1 -output-debug-info

4.4. Testing & Validation

Cross-validation is widely used statistical method to estimate the machine learning model performance. It is a resampling procedure used when the data sample is limited. k-Fold cross-validation is a term where the k is the number of groups/folds that a given data sample is to be split into randomly. In each iteration, 1 fold is selected for testing and the k-1 folds are used for training. This is repeated k times for each fold.

10-fold cross-validation (10-CV) is used on training data to set the parameters for PCA and SVM. 10-CV is also used in testing to optimize the TPR values for each user.

In order to optimize the authentication performance in terms of TPR, all different nu values in the range of 0.01-0.99 with the step size 0.01 are examined for each PCA transformed training data. According to the result matrix of TPR and nu values, a best nu value is assigned to each user. These nu values are used in one-class SVM classification.

4.5. Metrics

There are 3 metrics that are evaluated in this thesis. True Positive Rate (TPR), False Positive Rate (FPR) and Equal Error Rate (EER).

TPR is an outcome where the model correctly predicts the positive class. TPR value is high if the data being tested is correctly identified as belonging a target class. Higher TPR means, better classification and hence better authentication performance.

True Positive Rate (TPR) is defined as follows:

$$TPR = \frac{TP}{TP+FN} \tag{1}$$

FPR is an outcome where the model incorrectly predicts the positive class. FPR value is low if the data being tested is correctly identified to be an outlier. Lower FPR means, better classification and hence better authentication performance.

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP+TN} \tag{2}$$

EER, on the other hand, is a biometric security system algorithm to determine the thresholds for its False Acceptance Rate (FAR) and its False Rejection Rate (FRR). When FAR and FRR are equal, the common value is referred as EER. The relation between EER, FAR and FRR is shown in Figure 4.2. Lower EER means, better classification and hence better authentication performance.

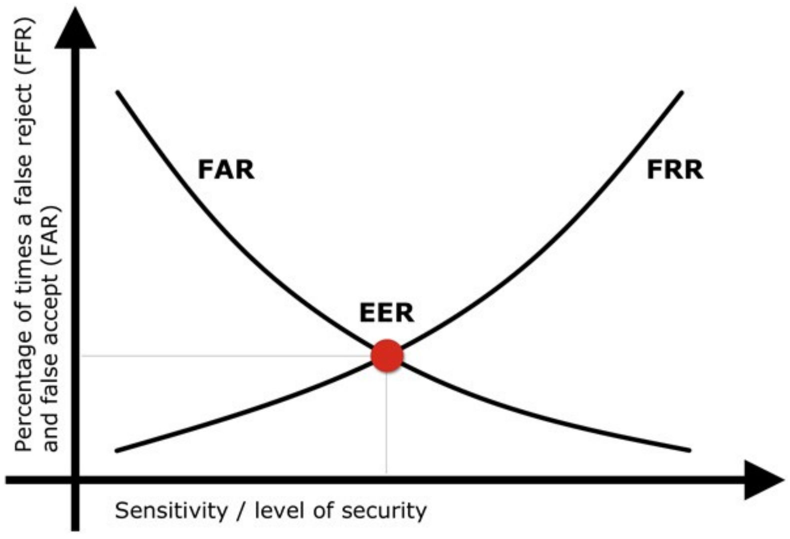

Percentage of times a false reject (FFR) and false accept (FAR)
FAR
FRR
EER
Sensitivity / level of security

Figure 4.2: Definition of Equal Error Rate (EER)

5. PERFORMANCE EVALUATION

In this section, the classification performance and the resource consumption analysis are provided. The effects of PCA and SVM parameters on the classification performance are investigated. TPR and/or FPR results are provided with each configuration. Results of the resource consumption analysis are also presented.

5.1. Classification Performance Analysis

The experiments are held on the collected data of 20 users. For each user, one class SVM is performed with the default parameters of WEKA libSVM. SVM models are trained with user's own data and tested with the data of other users.

48.85% TPR is observed on average as seen in Table 5.1 without PCA and without SVM parameter optimization.

SVM parameter optimization is very effective on classification performance. Gamma value is determined to be best at 1.0E-7. A user-specific nu value that has the best authentication performance on user data is assigned for each user. These nu values are determined by automated tests in Weka Experimenter.

Table 5.2 shows that the TPR values obtained vary between 70% to 96% when the features are transformed with PCA and the SVM parameters are optimized for each user. 83.13% TPR is observed on average.

User ID	TPR
2	50.85%
3	51.60%
4	48.80%
5	48.49%
6	49.19%
8	49.35%
30	47.49%
31	48.69%
50	49.07%
51	48.91%
52	47.47%
53	49.38%
54	48.47%
55	47.40%
56	49.17%
57	48.61%
70	47.56%
71	49.34%
72	48.23%
73	48.84%
Avg.-->	48.85%

Table 5.1: TPR performance without PCA & with default SVM parameters

SVM classifier parameters that are used in experiments:

kernelType	: Radial Basis Function (RBF)
gamma	: 1.0E-7
nu	: "an optimized value in range 0.01 - 0.99"
normalize	: true
others	: WEKA libSVM defaults

PCA is performed on all features. The following default WEKA configuration parameters are used.

Attribute selection configuration for the PCA evaluator:

weka.attributeSelection.PrincipalComponents -R 0.95 -A 5

Attribute selection configuration for the PCA search:

weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1

User ID	# of training instances	# of testing instances	# of total instances	Best nu	Best TPR %
2	264	29	293	0.25	82.22
3	225	25	250	0.40	94.80
4	225	25	250	0.04	96.00
5	329	37	365	0.50	83.66
6	278	31	309	0.50	94.83
8	275	31	306	0.04	87.82
30	197	22	219	0.60	75.24
31	240	27	267	0.20	91.70
50	194	22	216	0.50	80.67
51	166	18	184	0.20	72.43
52	195	22	217	0.40	87.84
53	219	24	243	0.72	81.23
54	147	16	163	0.50	73.90
55	156	17	173	0.25	81.27
56	216	24	240	0.25	90.42
57	226	25	251	0.50	89.09
70	203	23	225	0.86	70.79
71	204	23	227	0.50	86.17
72	203	23	226	0.84	72.13
73	194	22	215	0.50	70.39
Avg.TPR -->					**83.13**

Table 5.2: TPR performance with PCA & with optimized SVM parameters

The importance of the training instance count is also examined. User #30 and User #70 has collected data from 15 more sessions. Table 5.3 shows that User #70 has shown great improvement on TPR performance, although the TPR performance of User #30 does not change much.

User ID	# of training instances	# of testing instances	# of total instances	Best nu	Best TPR %
30	197	22	219	0.60	75.24
30	418	46	464	0.19	74.77
70	203	23	225	0.86	70.79
70	440	49	489	0.30	98.78
Avg.TPR -->					**84.51**

Table 5.3: TPR performance comparison for more training data

PCA and SVM parameter optimization have increased SVM performance and so does the authentication performance. The training data size has also affected the performance. The comparison is given in Table 5.4.

FPR is also crucial for authentication. However, it can only be obtained by testing one-class SVM model with other users' instances. Since the model is generated with PCA, to maximize the TPR, the testing instances should also be generated with the same PCA attribute optimization formula. This is achieved in WEKA by using attribute selected classifier. It takes training feature dataset and testing feature dataset separately and apply training PCA to the testing instances, too.

Classifier configuration:

```
weka.classifiers.meta.AttributeSelectedClassifier -E
"weka.attributeSelection.PrincipalComponents -R 0.95 -A 5" -S
"weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1" -W
weka.classifiers.functions.LibSVM -output-debug-info -- -S 2 -K 2 -D 3 -G 1.0E-7 -R
0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -Z -model "C:\\Program Files\\Weka-3-8" -
seed 1 -output-debug-info
```

Table 5.5 shows the corresponding FPR performance of TPR optimized one-class SVM model. Although the ratio of number of training instances to the number of testing instances is very low, the FPR values are less than 3% for the half of the users.

	Without PCA, Without SVM parameter optimization	With PCA, With SVM parameter optimization	The effect of more training data
User ID	**TPR**	**TPR**	**TPR**
2	50.85%	82.22%	-
3	51.60%	94.80%	-
4	48.80%	96.00%	-
5	48.49%	83.66%	-
6	49.19%	94.83%	-
8	49.35%	87.82%	-
30	47.49%	75.24%	74.77%
31	48.69%	91.70%	-
50	49.07%	80.67%	-
51	48.91%	72.43%	-
52	47.47%	87.84%	-
53	49.38%	81.23%	-
54	48.47%	73.90%	-
55	47.40%	81.27%	-
56	49.17%	90.42%	-
57	48.61%	89.09%	-
70	47.56%	70.79%	98.78%
71	49.34%	86.17%	-
72	48.23%	72.13%	-
73	48.84%	70.39%	-

Table 5.4: Effects of PCA, SVM parameter optimization and training data size

User #50, User #52, User #30 and User #56 have very high FPR values and it is necessary to train better models for these users before making any model eligible to authenticate the corresponding user. This might happen if there is no user characteristic throughout the scenarios and the limited data may not reflect the user properly.

User ID	# of training instances	# of testing instances	# of total instances	# of classified instances	nu (TPR optimized)	Best TPR	FPR (TPR optimized)
2	293	5055	5348	109	0.25	82.22%	2.16%
3	250	5098	5348	187	0.40	94.80%	3.67%
4	250	5098	5348	781	0.04	96.00%	15.32%
5	365	4983	5348	554	0.50	83.66%	11.12%
6	309	5039	5348	198	0.50	94.83%	3.93%
8	306	5042	5348	0	0.04	87.82%	0.00%
30	464	4884	5348	1062	0.19	74.77%	21.74%
31	267	5081	5348	8	0.20	91.70%	0.16%
50	216	5132	5348	4381	0.50	80.67%	85.37%
51	184	5164	5348	166	0.20	72.43%	3.21%
52	217	5131	5348	2620	0.40	87.84%	51.06%
53	243	5105	5348	330	0.72	81.23%	6.46%
54	163	5185	5348	519	0.50	73.90%	10.01%
55	173	5175	5348	165	0.25	81.27%	3.19%
56	240	5108	5348	1117	0.25	90.42%	21.87%
57	251	5097	5348	0	0.50	89.09%	0.00%
70	489	4859	5348	40	0.30	98.78%	0.82%
71	227	5121	5348	138	0.50	86.17%	2.69%
72	226	5122	5348	4	0.84	72.13%	0.08%
73	215	5133	5348	16	0.50	70.39%	0.31%
		Avg.FPR -->					12.16%

Table 5.5: FPR performance of TPR optimized one-class SVM model

In this work, FAR equals FPR and FRR equals 1 minus TPR. The values can be seen in Table 5.6. FAR values are listed in descending order and FRR values are listed in ascending order.

Since FAR and FRR distributions overlap, the intersection point gives us the EER value, which is nearly 11% as seen in Figure 5.1.

FAR	FRR
85.37%	4.00%
51.06%	5.17%
21.87%	5.20%
21.74%	8.30%
15.32%	9.58%
11.12%	10.91%
10.01%	12.16%
6.46%	12.18%
3.93%	13.83%
3.67%	16.34%
3.21%	17.78%
3.19%	18.73%
2.69%	18.77%
2.16%	19.33%
0.82%	24.76%
0.31%	26.10%
0.16%	27.57%
0.08%	27.87%
0.00%	29.21%
0.00%	29.61%

Table 5.6: FAR and FRR values

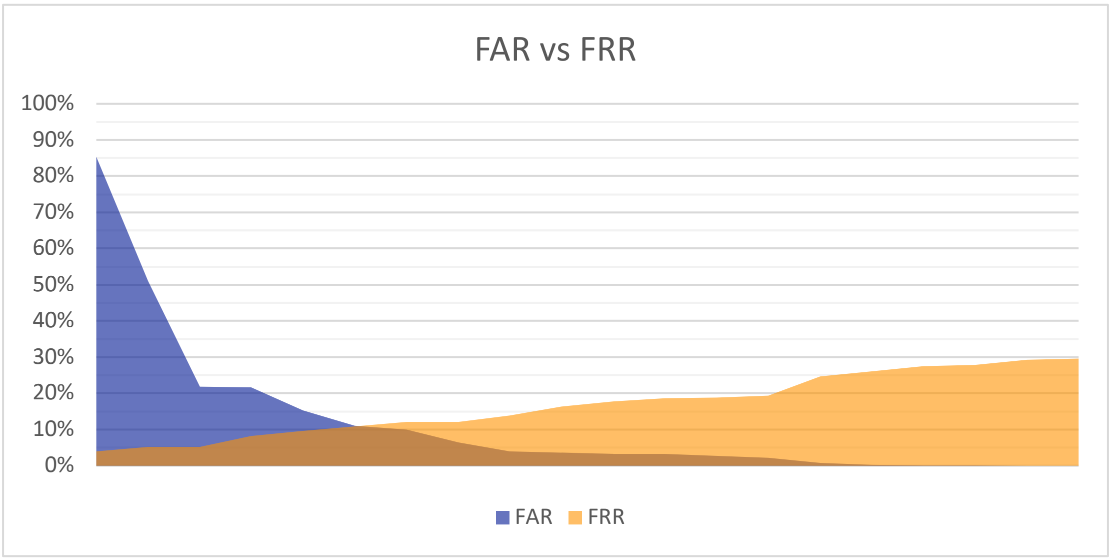


Figure 5.1: Equal Error Rate (EER)

5.2. Resource Consumption Analysis

Since the platform is a mobile phone, the monitored resources are power consumption, CPU usage, I/O usage, memory usage and network usage. All the test scenarios mentioned in Section 3 are followed and the measurements are taken on Android OS version 4.4.2 installed on Samsung S3 Neo (GT-I9301Q) phone. In each test, the predefined and exactly the same scenario is followed. Each test session has a different duration, varying from 3 minutes to 4 minutes, depending on the user speed and the web service latency.
The power consumption measurements are taken by the PowerTutor[1] app. It is a power monitoring application for Android-based mobile platforms developed by University of Michigan Ph.D. students. Zhang L., et al. (2010) estimated power usage of the applications running on device with this app they developed. The estimation is based on a built-in model which estimates within 5% of actual values. Total energy usage is obtained from PowerTutor for each session. Since each test session has different duration, varying from 3 minutes to 4 minutes, the total energy consumption (in joules) is divided by session duration (in seconds) to get the power consumption (in watts).
The CPU usage is obtained by using the dumpsys[2]. It is a tool that runs on Android devices and provides information about system services. "adb shell dumpsys cpuinfo" command is called from the command line to get diagnostic output using the Android Debug Bridge (ADB). The output shows CPU usage of every application running on the device, "grep" command is used to filter out the others.

The command is called in every 10s by using the "watch" command which can be used to automate commands on a regular basis. All the outputs show the average CPU usage and then their average is used to compare different test sessions.

[1] PowerTutor - A Power Monitor for Android-Based Mobile Platforms
http://ziyang.eecs.umich.edu/projects/powertutor/

[2] dumpsys – A tool that runs on Android devices and provides information about system services
https://developer.android.com/studio/command-line/dumpsys

5.2.1. Power Consumption and CPU Usage

In terms of power consumption, 39% increase is observed in the complete logger activated application compared to the base application (Figure 5.1). Measurements in Figure 5.2 show that complete logger activated application has used 5% more CPU on average compared to the base application. Since the logging is not a CPU intensive task, most of the power consumption is due to the I/O workload. At the rate of 5 Hz, the average I/O rate is 250 bytes/s. It also increases directly proportional to the sampling rate for each sensor. The I/O rate for the touchscreen usage is very small compared to the even 5 Hz sensor data generated. Hence, it is neglected. Table 1 also shows the schema of the data written by each sensor and the touchscreen gesture in every data change.

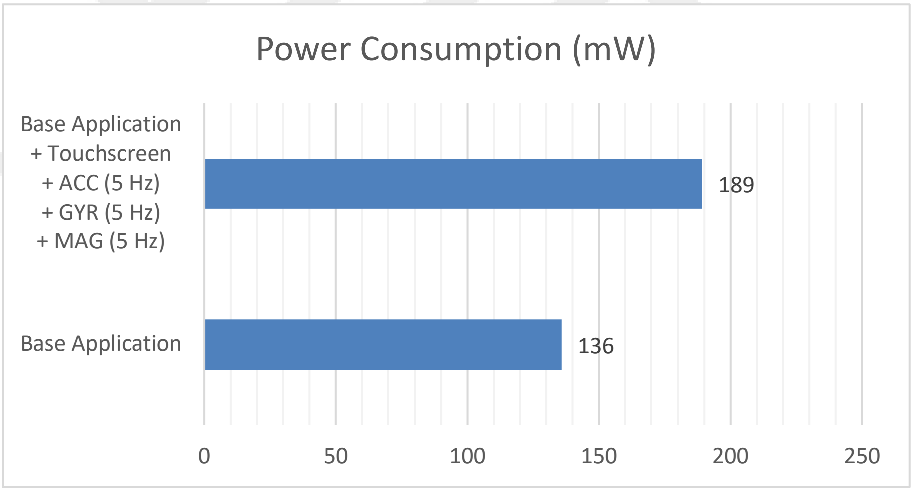


Figure 5.2: Overall power consumption

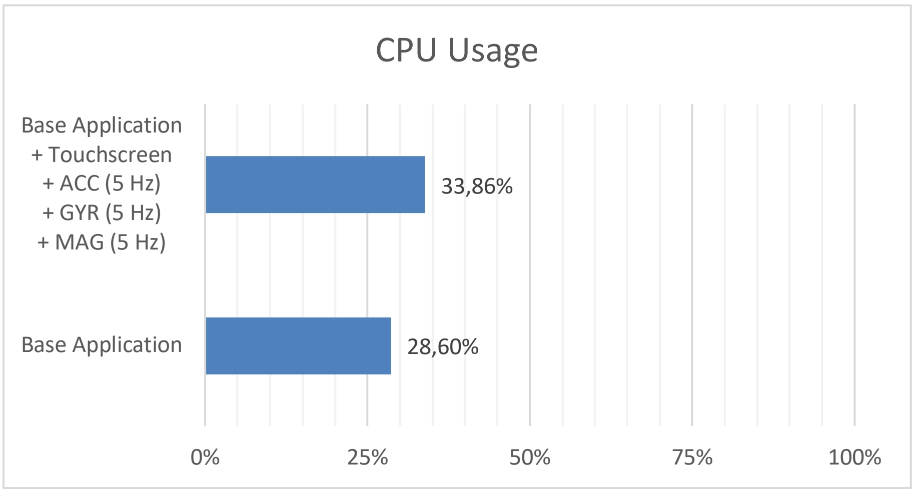


Figure 5.3: Overall CPU usage

5.2.2. Impact of Touchscreen (Normal usage vs Under stress)

Figure 5.3 and Figure 5.4 show that touchscreen increases power consumption only 3% with normal usage according to the scenarios. To maximize the generated data size, the application is also tested under stress. This is achieved by touching and scrolling aggressively and randomly on touchscreen without following the predefined scenario. The power consumption in stress test is 70% more than the normal touchscreen usage. The results show that touchscreen has very little effect on both power consumption and CPU usage. Corresponding differences are 4 mW and 0.5% respectively.

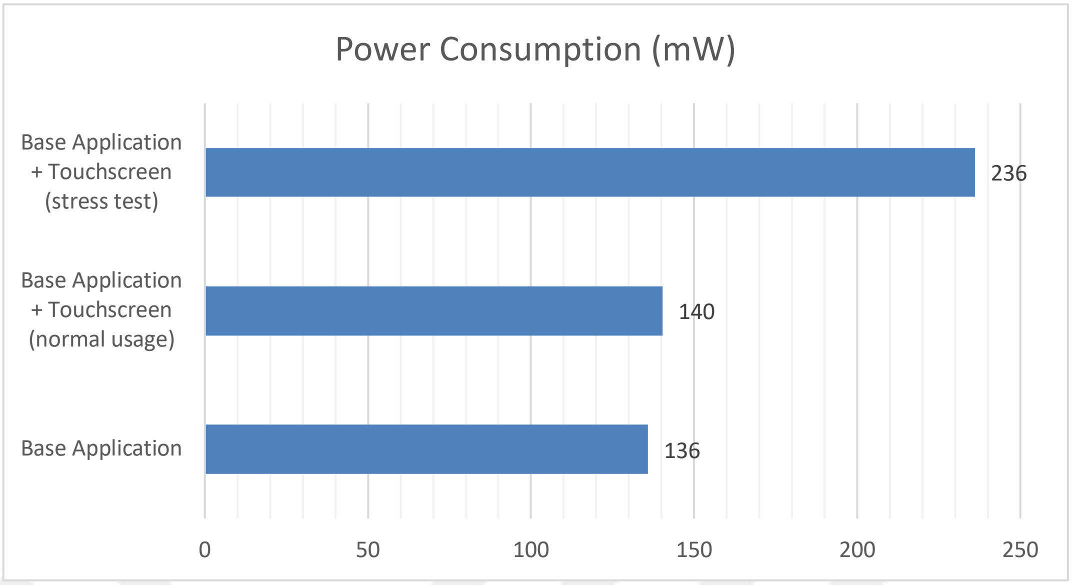


Figure 5.4: Impact of touchscreen on power consumption

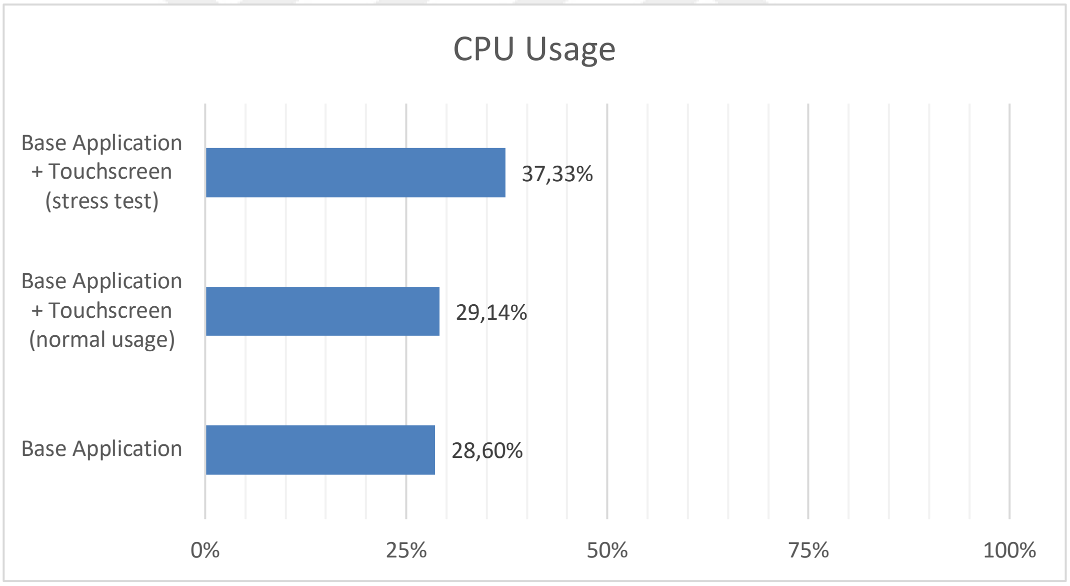


Figure 5.5: Impact of touchscreen on CPU usage

5.2.3. Impact of Accelerometer (Sampling rate: 5 Hz vs 20 Hz vs 100 Hz)

Although accelerometer can consume large amount of energy in 100Hz, which is the maximum sampling rate in our device, it is the least energy consumer among the other sensors that are observed. At the rate of 5Hz, the power consumption of the accelerometer is only 9mW. As the sampling rate increases, the power consumption and the CPU usage are also increased accordingly, as seen in Figure 5.5 and Figure 5.6. The dramatic increase in CPU usage at the rate 100Hz is due to the number of the created async tasks and the I/O wait time.

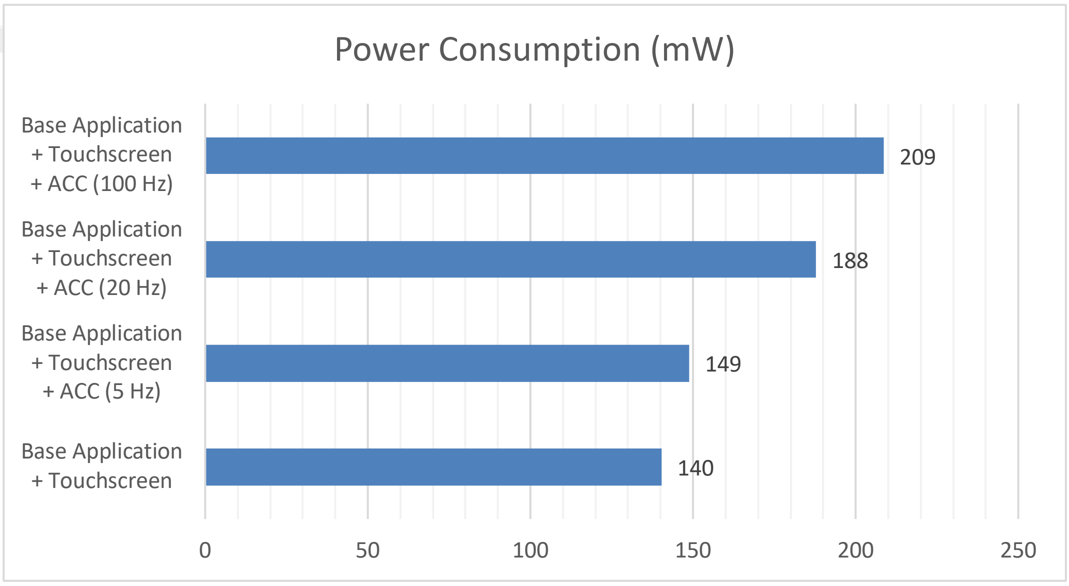


Figure 5.6: Impact of accelerometer on power consumption

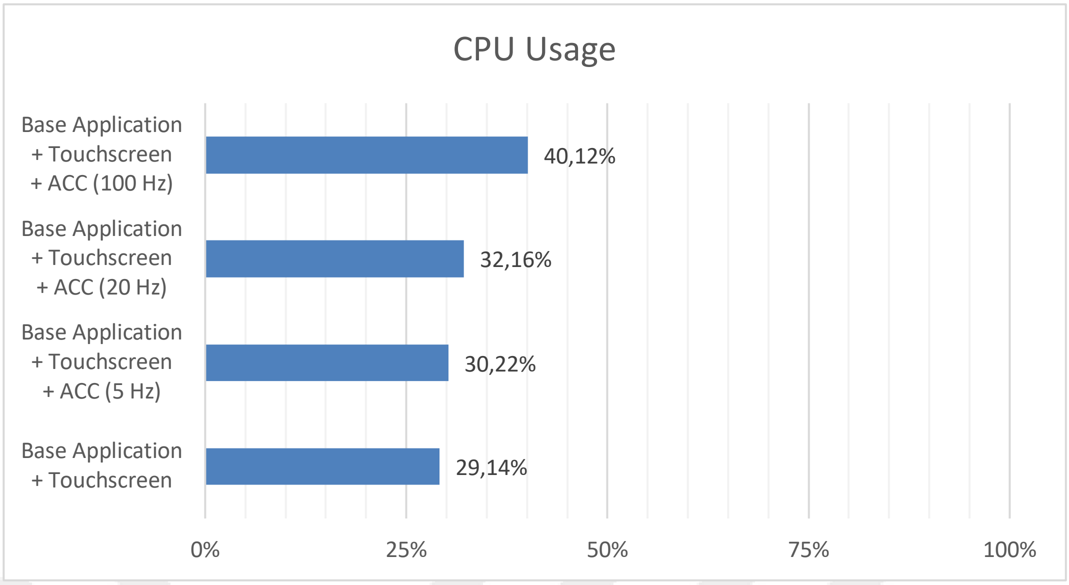


Figure 5.7: Impact of accelerometer on CPU usage

5.2.4. Impact of Gyroscope (Sampling rate: 5 Hz vs 20 Hz vs 100 Hz)

Even with a 5 Hz sampling rate, very large amount of energy is consumed by the 9-axis gyroscope. Figure 5.7 and Figure 5.8 show that 30mW increase in power consumption and 21% more battery demand occur compared to the base application. Although these numbers are device specific, it is crucial to reduce the active time of the gyroscope sensor. All the sensors are active all the time the application is being used when these measurements are taken.

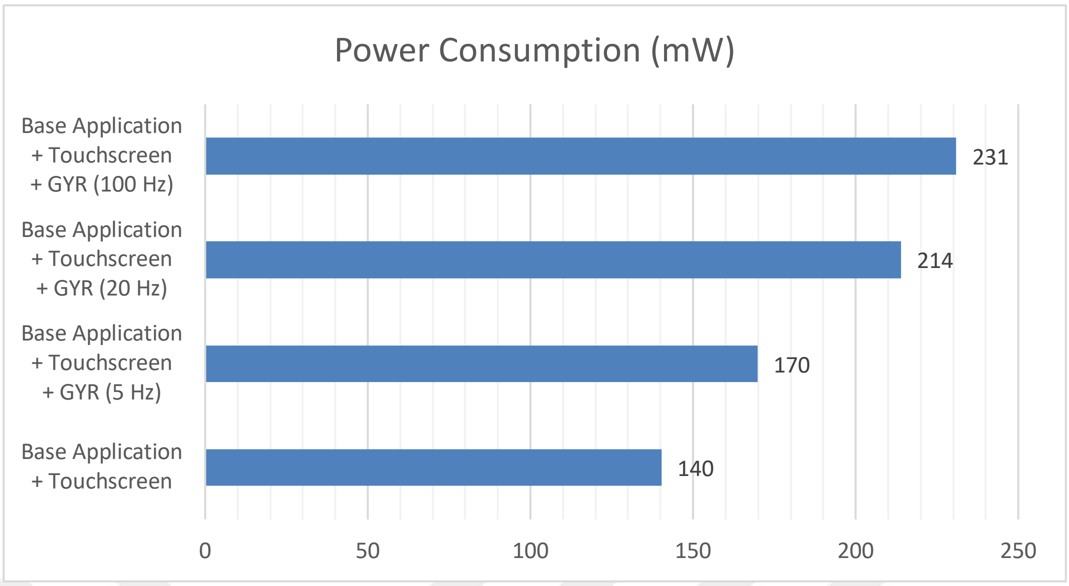


Figure 5.8: Impact of gyroscope on power consumption

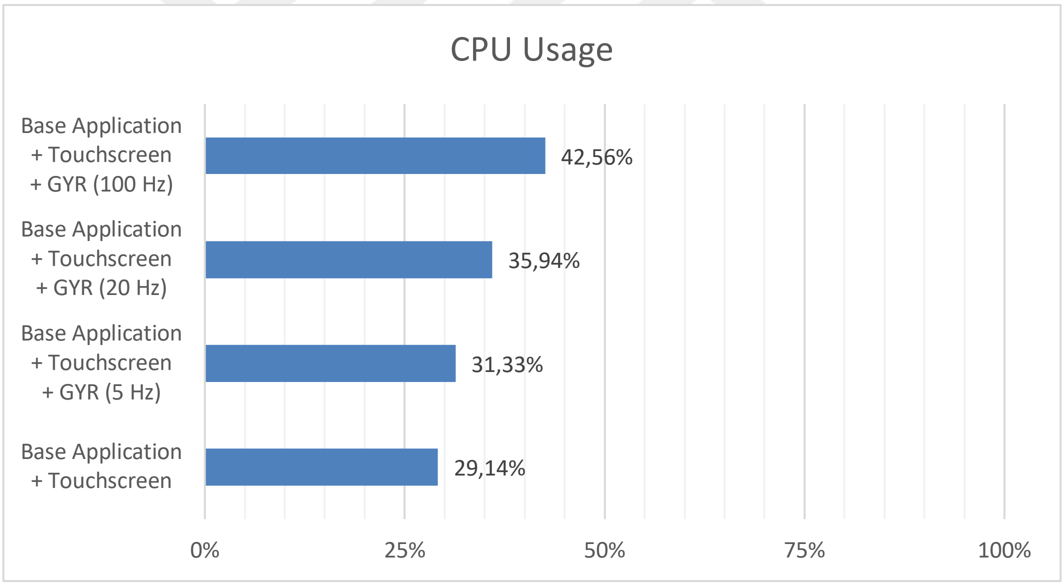


Figure 5.9: Impact of gyroscope on CPU usage

5.2.5. Impact of Magnetometer (Sampling rate: 5 Hz vs 20 Hz vs 100 Hz)

Similar to the gyroscope, the magnetometer is also an energy-hungry sensor. Even with a 5 Hz sampling rate, 27mW power and 2.6% CPU demand arise compared to the base application, as seen in Figure 5.9 and Figure 5.10. With a 100 Hz sampling rate, on the other hand, dramatic increase in the power consumption and CPU usage can be unacceptable for a user.

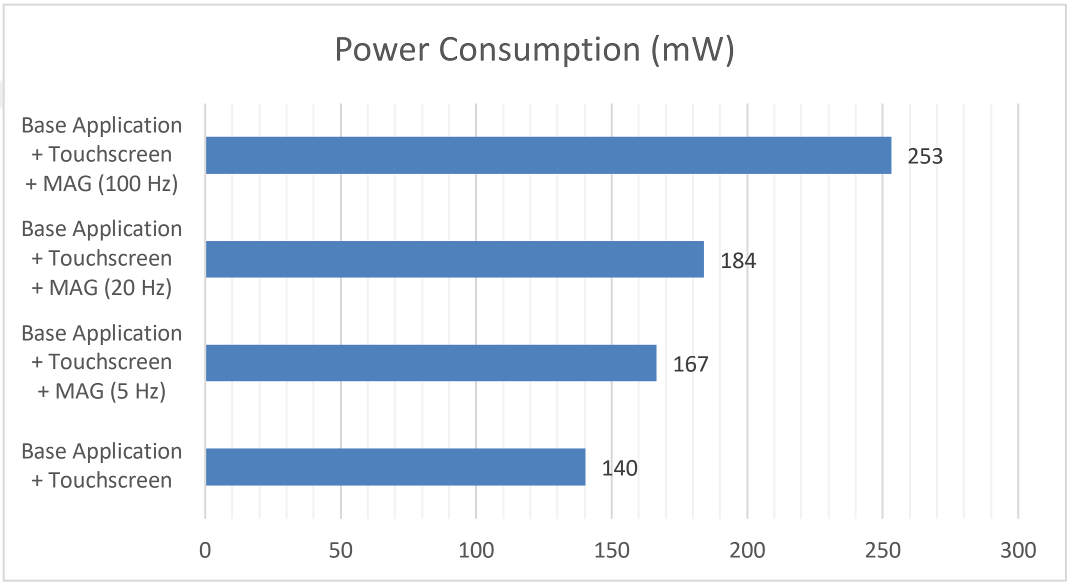


Figure 5.10: Impact of magnetometer on power consumption

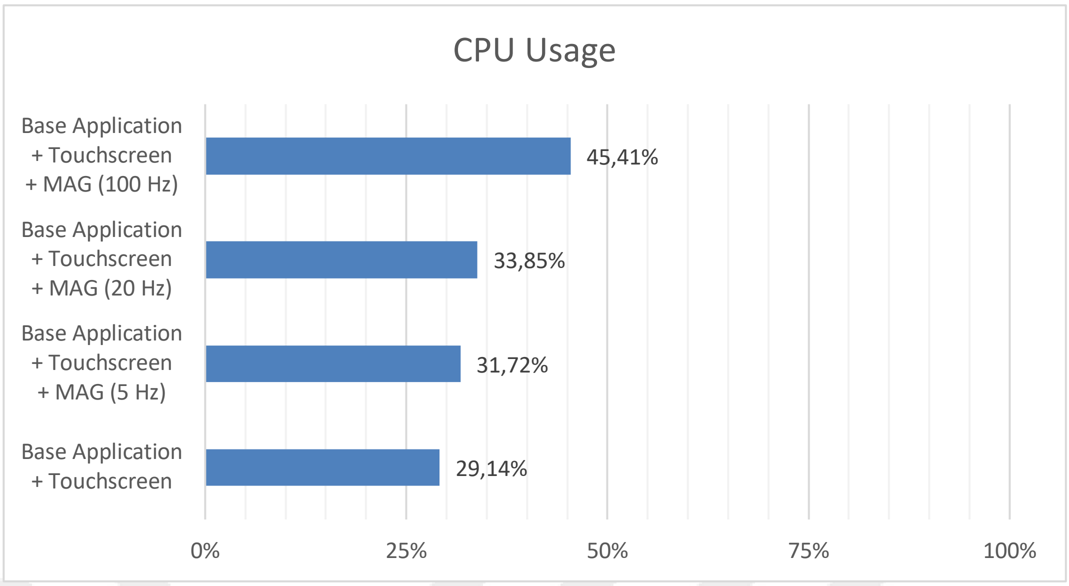


Figure 5.11: Impact of magnetometer on CPU usage

6. CONCLUSION & FUTURE WORK

In conclusion, the augmentation of a mobile banking application with continuous authentication using behavioral biometrics is investigated in this thesis. Data is collected from three phone sensors and touchscreen. 20 users participated in data collection by following 5 scenarios in 3 stances. It is very crucial for a bank to have all the necessary data without affecting the user experience. The performance evaluations are made under these circumstances. In order to be sure about the maximum effect, we measured each sensor at maximum sampling rate. For the touchscreen, on the other hand, stress test was conducted to push the limits. According to measurement results, it is acceptable to collect all the data at 5 Hz and logger augmented app does not bring significant overhead in terms of resource consumption. Sampling the sensors at 100Hz on the other hand, brings a serious overhead on CPU and power consumption. Although these metrics are affected seriously, the modern phones are capable enough to run on device without affecting the user experience. The only effect visible to the user is the battery consumption. The authentication performance, on the other hand, is %83 on average with SVM parameter optimization and PCA. It is also observed that the training data size can has positive effects on SVM performance. However, there are still some points that are required to be optimized, like magnetometer and gyroscope deactivation. I/O rate can be optimized by utilizing a database instead of CSV files in the filesystem. Collected data can be transferred as small pieces to the backend server. Feature extraction can be automatized on server and the features can be stored in a more secure database. When the count of the training instances reaches a predefined threshold, then the SVM model is auto trained with PCA. The tests can be done automatically by using random testing instances from all users feature instances. The model is then marked to be eligible to authenticate the user if the TPR and FPR values falls in an acceptable range. These automations on the server side are left as future work.

REFERENCES

A. Buriro, S. Gupta and B. Crispo, "Evaluation of Motion-Based Touch-Typing Biometrics for Online Banking," 2017 International Conference of the Biometrics Special Interest Group (BIOSIG), Darmstadt, 2017, pp. 1-5.

Alzubaidi A., Kalita J. (2016). Authentication of smartphone users using behavioral biometrics. IEEE Communications Surveys Tutorials, vol. 18, no. 3, pp. 1998-2026, third quarter 2016.

Bailey, N. T. J. (1952). Study of queues and appointment systems in outpatient departments, with special reference to waiting-times., *Journal of the Royal Statistical Society : Series B* 14(2) : 185–199.

Bo C., Zhang L., Jung T., Han J., Li X.-Y., Wang Y., in Proc. of IEEE IPCCC. Continuous user identification via touch and movement behavioral biometrics (IEEEPiscataway, New Jersey, USA, 2014).

Dumpsys Android Developers, Android Developers. (2019). https://developer.android.com/studio/command-line/dumpsys (accessed 30 April 2019).

F. Rahman, M. O. Gani, G. M. T. Ahsan and S. I. Ahamed, "Seeing Beyond Visibility: A Four Way Fusion of User Authentication for Efficient Usable Security on Mobile Devices," 2014 IEEE Eighth International Conference on Software Security and Reliability-Companion, San Francisco, CA, 2014, pp. 121-129. doi: 10.1109/SERE-C.2014.30

Fridman, Lex, et al. "Multi-modal decision fusion for continuous authentication." Computers & Electrical Engineering 41 (2015): 142-156.

L. Fridman, S. Weber, R. Greenstadt and M. Kam, "Active Authentication on Mobile Devices via Stylometry, Application Usage, Web Browsing, and GPS Location," in IEEE Systems Journal, vol. 11, no. 2, pp. 513-521, June 2017. doi: 10.1109/JSYST.2015.2472579

Rana, D. “One Class SVM Vs SVM Classification”, in International Journal of Science and Research (IJSR), vol. 4, issue 6, pp. 1350-1352, June 2015.

Sitová Z., Šeděnka J., Yang Q., Peng G., Zhou G., Gasti P., Balagani K.S., "HMOG: New behavioral biometric features for continuous authentication of smart-phone users", IEEE Trans. Inform. Forensics Security, vol. 11, no. 5, pp. 877-892, May 2016.

Stylios I., Thanou O., Androulidakis I., Zaitseva E., A Review of Continuous Authentication Using Behavioral Biometrics, ACM SEEDA-CECNSM 2016, September 2016, ACM. URL:https://www.researchgate.net/publication/303974524_A_Review_of_Continuous_Authentication_Using_Behavioral_Biometrics

V. M. Patel, R. Chellappa, D. Chandra and B. Barbello, "Continuous User Authentication on Mobile Devices: Recent progress and remaining challenges," in IEEE Signal Processing Magazine, vol. 33, no. 4, pp. 49-61, July 2016. doi: 10.1109/MSP.2016.2555335

W. Meng, D. S. Wong, S. Furnell and J. Zhou, "Surveying the Development of Biometric User Authentication on Mobile Phones," in IEEE Communications Surveys & Tutorials, vol. 17, no. 3, pp. 1268-1293, third quarter 2015.

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R.P., Mao, Z.M., et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis; CODES/ISSS ’10. ACM. ISBN 978-1-60558-905-3; 2010, p. 105–114.

APPENDICES

Appendix A. Showing a consent pop-up to start data collection

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ...

    if (!DakotaLogUtil.isDakotaFileSetUpCompleted) {

        AlertDialog.Builder alert = new AlertDialog.Builder(this);

        alert.setTitle("Veri Toplama İzni");
        alert.setMessage("Uygulamayı kullanırken verilerimin toplanmasına izin
veriyorum.");


        final DakotaDataCollectionInputView input = new
DakotaDataCollectionInputView(this);

        alert.setView(input);

        alert.setPositiveButton("Başlat", new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                String name = input.getName();
                String position = input.getPosition();
                String scenario = input.getScenario();
                String phoneModel = input.getPhoneModel();


BaseActivityPermissionsDispatcher.setupDAKOTAWithPermissionCheck(BaseActivity.
this, BaseActivity.this, name, position, scenario, phoneModel);
            }
        });

        alert.show();
    }

    DakotaSensorUtil.setupSensors(BaseActivity.this);

    DakotaSensorUtil.mGestureDetector = new GestureDetector(BaseActivity.this,
new DakotaGestureListener());
}
```

Appendix B. Dispatching the touch events

```
@Override
public boolean dispatchTouchEvent(MotionEvent ev) {

    boolean eventConsumed =
DakotaSensorUtil.mGestureDetector.onTouchEvent(ev);
    if (eventConsumed)
    {
        String text = DakotaGestureListener.currentGestureDetected;
        text = text.concat(";" + (getCurrentPageFragment() != null ?
getCurrentPageFragment().getSimpleClassName() : "-"));

        for (String fileName : DakotaGestureListener.fileNames) {
            DakotaLogUtil.writeToFile(text, fileName);
        }
    }


    switch (ev.getAction()) {
        case MotionEvent.ACTION_DOWN:
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_MOVE:
            String text = ev.getAction() + ";" + ev.getPressure() + ";" +
ev.getX() + ";" + ev.getY() + ";" + ev.getSize() + ";" + ev.getDownTime() +
";" + System.currentTimeMillis();
            text = text.concat(";" + (getCurrentPageFragment() != null ?
getCurrentPageFragment().getSimpleClassName() : "-"));
            DakotaLogUtil.writeToFile(text, "touch.csv");
            Log.e("Touch", text);
            break;
        default:
            break;

    }

    getCurrentPageFragment().dispatchTouchEvent(ev);
    return disableUserInteraction || super.dispatchTouchEvent(ev);
}
```

Appendix C. Log Utility

```
public class DakotaLogUtil {
    private static final String CSV_FILE_HEADER =
"Title1;Title2;Title3;Title4";
    public static boolean isDakotaFileSetUpCompleted = false;
    private static File parentDirectory;

    /* Checks if external storage is available for read and write */
    public static boolean isExternalStorageWritable() {
        String state = Environment.getExternalStorageState();
        return Environment.MEDIA_MOUNTED.equals(state);
    }

    public static void setupFiles(BaseActivity activity, String name, String
position, String scenario, String phoneModel) {
        boolean writable = isExternalStorageWritable();

        if (writable) Log.e("DAKOTA EXTERNAL STORAGE", "Writable");
        else Log.e("DAKOTA EXTERNAL STORAGE", "NOT Writable");

        SimpleDateFormat s = new SimpleDateFormat("yyyyMMddHHmmss");
        String formattedDate = s.format(new Date());

        ActivityCompat.requestPermissions(activity, new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, 1);
        parentDirectory = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUM
ENTS), "DAKOTA-" + formattedDate + "-" + name + "-" + position + "-s" +
scenario + "-s" + phoneModel);

        if (parentDirectory.exists()) {
            Log.e("DAKOTA EXTERNAL STORAGE", "Directory exists");
        } else {
            if (!parentDirectory.mkdirs()) {
                Log.e("DAKOTA EXTERNAL STORAGE", "Directory not created");
            } else {
                Log.e("DAKOTA EXTERNAL STORAGE", "Directory created");
            }
        }

        initializeHeaders();

        Log.e("DAKOTA EXTERNAL STORAGE", parentDirectory.getAbsolutePath());
    }

    private static void initializeHeaders() {
        ArrayList<String> files = new ArrayList<>();
        files.add(0, "down.csv");
        files.add(1, "fling.csv");
        files.add(2, "scroll.csv");
        files.add(3, "singleTapUp.csv");
        files.add(4, "showPress.csv");
        files.add(5, "longPress.csv");
        files.add(6, "doubleTap.csv");
        files.add(7, "acc.csv");
        files.add(8, "gyr.csv");
        files.add(9, "mag.csv");
        files.add(10, "touch.csv");
        files.add(11, "touchNoScrollNoFling.csv");
        for (int i = 0; i < files.size(); i++) {
            String fileName = files.get(i);
            File file = new File(parentDirectory, fileName);
            if (!file.exists()) {
                Log.e("DAKOTA HEADER CREATION", "FILE DOES NOT EXIST " +
file.getAbsolutePath());
                String header = "";
                switch (fileName) {
                    case "down.csv"://
                        header =
"ACTION_NAME;PRESSURE;X;Y;SIZE;EVENT_TIME;SYSTEM_TIME;FRAGMENT_NAME";
                        break;
                    case "fling.csv":
                        header =
"ACTION_NAME;PRESSURE;X;Y;SIZE;VELOCITY_X;VELOCITY_Y;EVENT_TIME;SYSTEM_TIME;FR
AGMENT_NAME";
                        break;
                    case "scroll.csv":
                        header =
"EVENT_ID;BEGIN_TIME;START_X;START_Y;START_PRESSURE;START_SIZE;EVENT_TIME;CURR
ENT_X;CURRENT_Y;CURRENT_PRESSURE;CURRENT_SIZE;SYSTEM_TIME;DISTANCE_X;DISTANCE_
Y;DIRECTION;FRAGMENT_NAME";
                        break;
                    case "singleTapUp.csv"://
                        header =
"ACTION_NAME;PRESSURE;X;Y;SIZE;EVENT_TIME;SYSTEM_TIME;FRAGMENT_NAME";
                        break;
                    case "showPress.csv"://
                        header =
"ACTION_NAME;PRESSURE;X;Y;SIZE;EVENT_TIME;SYSTEM_TIME;FRAGMENT_NAME";
                        break;
                    case "longPress.csv"://
                        header =
"ACTION_NAME;PRESSURE;X;Y;SIZE;EVENT_TIME;SYSTEM_TIME;FRAGMENT_NAME";
                        break;
                    case "doubleTap.csv"://
                        header =
"ACTION_NAME;PRESSURE;X;Y;SIZE;EVENT_TIME;SYSTEM_TIME;FRAGMENT_NAME";
                        break;
                    case "acc.csv":
                        header = "SYSTEM_TIME;X;Y;Z;FRAGMENT_NAME";
                        break;
                    case "gyr.csv":
                        header = "SYSTEM_TIME;X;Y;Z;FRAGMENT_NAME";
                        break;
                    case "mag.csv":
                        header = "SYSTEM_TIME;X;Y;Z;FRAGMENT_NAME";
                        break;
                    case "touch.csv":
                        header =
"ACTION_ID;PRESSURE;X;Y;SIZE;EVENT_TIME;SYSTEM_TIME;FRAGMENT_NAME";
                        break;
                    case "touchNoScrollNoFling.csv":
                        header =
"ACTION_ID;PRESSURE;X;Y;SIZE;EVENT_TIME;SYSTEM_TIME;FRAGMENT_NAME";
                        break;
                    default:
                        break;
                }
                try {
                    FileOutputStream outputStreamWriter = new
FileOutputStream(file, true);
                    PrintWriter bufferedWriter = new
PrintWriter(outputStreamWriter);

                    bufferedWriter.println(header);
                    bufferedWriter.flush();
                    bufferedWriter.close();
                    outputStreamWriter.close();
                } catch (Exception e) {
                    Log.e("DAKOTA HEADER CREATION", e.getMessage());
                    e.printStackTrace();
                }
            }
        }
    }

    public static void writeToFile(String data, String fileName) {
        AsyncTask asyncTask = new DakotaLogWriter();
        Log.w("DAKOTA WRITE TO FILE", "" + data + " ; " + fileName);
        asyncTask.execute(data, fileName);
    }

    /* Checks if external storage is available to at least read */
    public boolean isExternalStorageReadable() {
        String state = Environment.getExternalStorageState();
        return Environment.MEDIA_MOUNTED.equals(state) ||
                Environment.MEDIA_MOUNTED_READ_ONLY.equals(state);
    }

    public static class DakotaLogWriter extends AsyncTask<Object, Void,
String> {

        @Override
        protected String doInBackground(Object... objects) {
            try {
                String data = (String) objects[0];
                String fileName = (String) objects[1];

                File file = new File(parentDirectory, fileName);
                FileOutputStream outputStreamWriter = new
FileOutputStream(file, true);
                PrintWriter bufferedWriter = new
PrintWriter(outputStreamWriter);

                bufferedWriter.println(data);
                bufferedWriter.flush();
                bufferedWriter.close();
                outputStreamWriter.close();
                return null;
            } catch (FileNotFoundException e) {
                Log.e("DAKOTA Exception", "File not found: " + e.toString());
                return null;
            } catch (IOException e) {
                Log.e("DAKOTA Exception", "File write failed: " +
e.toString());
                return null;
            }
        }
    }

}
```

Appendix D. Sensor Utility

```java
public class DakotaSensorUtil {
    public static SensorManager sensorManager;
    public static Sensor accSensor = null;
    public static Sensor gyroscopeSensor = null;
    public static Sensor magSensor = null;
    public static SensorEventListener gyroscopeEventListener;
    public static SensorEventListener accEventListener;
    public static SensorEventListener magEventListener;
    public static GestureDetector mGestureDetector;
    public static boolean sensorListening = true;

    public static void setupSensors(final Context context) {
        sensorManager = (SensorManager)
context.getSystemService(Context.SENSOR_SERVICE);
        if (sensorManager != null) {
            accSensor =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
            gyroscopeSensor =
sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
            magSensor =
sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
        } else {
            Toast.makeText(context, "Sensor manager is not available.",
Toast.LENGTH_LONG).show();
        }

        if (gyroscopeSensor == null) {
            Toast.makeText(context, "This Device has no Gyroscope !",
Toast.LENGTH_LONG).show();
        } else {
            gyroscopeEventListener = new SensorEventListener() {
                @Override
                public void onSensorChanged(SensorEvent event) {
                    String log = System.currentTimeMillis() + ";" +
event.values[0] + ";" + event.values[1] + ";" + event.values[2] + ";" +
(((BaseActivity) context).getCurrentPageFragment() != null ? ((BaseActivity)
context).getCurrentPageFragment().getSimpleClassName() : "-");

                    DakotaLogUtil.writeToFile(log, "gyr.csv");
                }

                @Override
                public void onAccuracyChanged(Sensor sensor, int accuracy) {

                }
            };
        }

        if (accSensor == null) {
            Toast.makeText(context, "This Device has no Accelerometer !",
Toast.LENGTH_LONG).show();
        } else {
            accEventListener = new SensorEventListener() {
                @Override
                public void onSensorChanged(SensorEvent event) {
                    String log = System.currentTimeMillis() + ";" +
event.values[0] + ";" + event.values[1] + ";" + event.values[2] + ";" +
(((BaseActivity) context).getCurrentPageFragment() != null ? ((BaseActivity)
context).getCurrentPageFragment().getSimpleClassName() : "-");
                    DakotaLogUtil.writeToFile(log, "acc.csv");
                }

                @Override
                public void onAccuracyChanged(Sensor sensor, int accuracy) {
                }
            };
        }

        if (magSensor == null) {
            Toast.makeText(context, "This Device has no Magnetometer !",
Toast.LENGTH_LONG).show();
        } else {
            magEventListener = new SensorEventListener() {
                @Override
                public void onSensorChanged(SensorEvent event) {
                    String log = System.currentTimeMillis() + ";" +
event.values[0] + ";" + event.values[1] + ";" + event.values[2] + ";" +
(((BaseActivity) context).getCurrentPageFragment() != null ? ((BaseActivity)
context).getCurrentPageFragment().getSimpleClassName() : "-");
                    DakotaLogUtil.writeToFile(log, "mag.csv");
                }

                @Override
                public void onAccuracyChanged(Sensor sensor, int accuracy) {
                }
            };
        }
    }
}
```

Appendix E. Gesture Listener

```
public class DakotaGestureListener extends
GestureDetector.SimpleOnGestureListener {
    private static final int SLIDE_THRESHOLD = 100;


    public static String currentGestureDetected;
    public static List<String> fileNames = new ArrayList<>();
    public float x, y;

    // Override s all the callback methods of
GestureDetector.SimpleOnGestureListener
    @Override
    public boolean onSingleTapUp(MotionEvent ev) {
        String s = "SINGLE_TAP_UP;" + ev.getPressure() + ";" + ev.getX() + ";"
+ ev.getY() + ";" + ev.getSize() + ";" + ev.getDownTime() + ";" +
System.currentTimeMillis();
        currentGestureDetected = s;
        fileNames.clear();
        fileNames.add("singleTapUp.csv");
        fileNames.add("touchNoScrollNoFling.csv");
        Log.e("Logger - onSingleTapUp", s);
        return true;
    }

    @Override
    public void onShowPress(MotionEvent ev) {
        String s = "SHOW_PRESS;" + ev.getPressure() + ";" + ev.getX() + ";" +
ev.getY() + ";" + ev.getSize() + " ;" + ev.getDownTime() + ";" +
System.currentTimeMillis();
        currentGestureDetected = s;
        fileNames.clear();
        fileNames.add("showPress.csv");
        fileNames.add("touchNoScrollNoFling.csv");
        Log.e("Logger - onShowPress", s);


    }

    @Override
    public void onLongPress(MotionEvent ev) {
        String s = "LONG_PRESS;" + ev.getPressure() + ";" + ev.getX() + ";" +
ev.getY() + ";" + ev.getSize() + ";" + ev.getDownTime() + ";" +
System.currentTimeMillis();
        currentGestureDetected = s;
        fileNames.clear();
        fileNames.add("longPress.csv");
        fileNames.add("touchNoScrollNoFling.csv");
        Log.e("Logger - onLongPress", s);

    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
float distanceY) {

        double distance = Math.sqrt(Math.abs(distanceX * distanceX - distanceY
* distanceY));
        Log.e("Logger - DISTANCE -----", "" + distance);
        String eventID = String.valueOf(e1.getDownTime()); // Event ID

        String beginTime = String.valueOf(e1.getDownTime()); // first down
time
        String startX = String.valueOf(e1.getX());
        String startY = String.valueOf(e1.getY());
        String startPressure = String.valueOf(e1.getPressure());
        String startSize = String.valueOf(e1.getSize());

        String eventTime = String.valueOf(e2.getEventTime()); // time btw
events
        String currentX = String.valueOf(e2.getX());
        String currentY = String.valueOf(e2.getY());
        String currentPressure = String.valueOf(e2.getPressure());
        String currentSize = String.valueOf(e2.getSize());
        String systemTime = String.valueOf(System.currentTimeMillis()); //
system time
        String distX = String.valueOf(distanceX);
        String distY = String.valueOf(distanceY);

        String s = eventID + ";" +
                beginTime + ";" +
                startX + ";" +
                startY + ";" +
                startPressure + ";" +
                startSize  + ";" +
                eventTime  + ";" +
                currentX  + ";" +
                currentY  + ";" +
                currentPressure  + ";" +
                currentSize + ";" +
                systemTime + ";" +
                distX + ";" +
                distY;

        try {
            float deltaY = e2.getY() - e1.getY();
            float deltaX = e2.getX() - e1.getX();
            String temp = "";

            if (Math.abs(deltaX) > Math.abs(deltaY)) {
                if (Math.abs(deltaX) > SLIDE_THRESHOLD) {
                    if (deltaX > 0) {
                        // the user made a sliding right gesture
                        temp += ";RIGHT";
                    } else {
                        // the user made a sliding left gesture
                        temp += ";LEFT";
                    }
                }
            } else {
                if (Math.abs(deltaY) > SLIDE_THRESHOLD) {
                    if (deltaY > 0) {
                        // the user made a sliding down gesture
                        temp += ";DOWN";
                    } else {
                        // the user made a sliding up gesture
                        temp += ";UP";
                    }
                }
            }
            if ("".equals(temp.trim())) {
                temp = ";-";
            }
            s += temp;
        } catch (Exception exception) {
            Log.e("ERR", exception.getMessage());
        }

        currentGestureDetected = s;
        fileNames.clear();
        fileNames.add("scroll.csv");
        Log.e("Logger - onScroll", s);
        return true;
    }

    @Override
    public boolean onDown(MotionEvent ev) {
        String s = "DOWN;" + ev.getPressure() + ";" + ev.getX() + ";" +
ev.getY() + ";" + ev.getSize() + ";" + ev.getDownTime() + ";" +
System.currentTimeMillis();
        currentGestureDetected = s;
        fileNames.clear();
        fileNames.add("down.csv");
        fileNames.add("touchNoScrollNoFling.csv");
        Log.e("Logger - onDown", s);
        return true;
    }

    @Override
    public boolean onDoubleTap(MotionEvent ev) {
        String s = "DOUBLE_TAP;" + ev.getPressure() + ";" + ev.getX() + ";" +
ev.getY() + ";" + ev.getSize() + ";" + ev.getDownTime() + ";" +
System.currentTimeMillis();
        currentGestureDetected = s;
        fileNames.clear();
        fileNames.add("doubleTap.csv");
        fileNames.add("touchNoScrollNoFling.csv");
        Log.e("Logger - onDoubleTap", "DOUBLE TAP");

        return true;
    }

    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
float velocityY) {

        String s = "FLING;" + e2.getPressure() + ";" + e2.getX() + ";" +
e2.getY() + ";" + e2.getSize() + ";" + velocityX + ";" + velocityY + ";" +
e1.getAction() + ";" + e1.getDownTime() + ";" + System.currentTimeMillis() +
";" + e2.getAction() + ";" + e2.getDownTime() + ";" +
System.currentTimeMillis();
        currentGestureDetected = s;
        fileNames.clear();
        fileNames.add("fling.csv");
        Log.e("Logger - onFling", s);

        return true;
    }
}
```

Appendix F. Feature Extraction in Python

```
#!/usr/bin/env python
# coding: utf-8

#from pathlib import Path
from scipy import signal
import pandas as pd
import glob
import re
import os
import numpy as np


def circ_r(vals, axis=0):
    alpha = np.array(vals, dtype='f8')
    # sum of cos & sin angles
    t = np.exp(1j * alpha)
    r = np.sum(t, axis=axis)
    # obtain length
    r = np.abs(r) / alpha.shape[axis]
    return r


def circ_mean(vals):
    x = np.sum(np.cos(vals))
    y = np.sum(np.sin(vals))
    return np.arctan2(y, x)


current_dir = os.getcwd()
data_dir = current_dir + '\\Data\\'
print("Current Data Dir: ", data_dir)

sessions = glob.glob(data_dir + "\\*\\")
print("Sessions", sessions)

for tl in sessions:
    print("Session --> ", tl)
    userid = int(re.search(r"\\(\d*)\\$", tl).group(1))
    print("User: ", userid)
    for sf in glob.glob(data_dir + "{}\\*\\".format(userid)):
        print("Calculating: {}".format(sf))
        try:
            exists = os.path.isfile("{}features ".format(sf))

            if exists:
                print('Feature file is already calculated.\n')
                continue

            scroll_exists = os.path.isfile("{}scroll.csv".format(sf))

            if not scroll_exists:
                print('No scroll file.\n')
                continue

            accelerometer_df = pd.read_csv(
                '{}acc.csv'.format(sf),
                delimiter=';',
                header=0,
                names=['SYSTEM_TIME', 'X_acc', 'Y_acc', 'Z_acc',
'FRAGMENT_NAME_acc']
            )
            accelerometer_df = accelerometer_df.sort_values('SYSTEM_TIME')

            gyroscope_df = pd.read_csv(
                '{}gyr.csv'.format(sf),
                delimiter=';',
                header=0,
                names=['SYSTEM_TIME', 'X_gyr', 'Y_gyr', 'Z_gyr',
'FRAGMENT_NAME_gyr']
            )
            gyroscope_df = gyroscope_df.sort_values('SYSTEM_TIME')

            magnetometer_df = pd.read_csv(
                '{}mag.csv'.format(sf),
                delimiter=';',
                header=0,
                names=['SYSTEM_TIME', 'X_mag', 'Y_mag', 'Z_mag',
'FRAGMENT_NAME_mag']
            )
            magnetometer_df = magnetometer_df.sort_values('SYSTEM_TIME')

            scroll_df = pd.read_csv(
                '{}scroll.csv'.format(sf),
                delimiter=';',
                header=0,
                names=[
                        'EVENT_ID',
                        'BEGIN_TIME',
                        'START_X',
                        'START_Y',
                        'START_PRESSURE',
                        'START_SIZE',
                        'EVENT_TIME',
                        'CURRENT_X',
                        'CURRENT_Y',
                        'CURRENT_PRESSURE',
                        'CURRENT_SIZE',
                        'SYSTEM_TIME',
                        'DISTANCE_X',
                        'DISTANCE_Y',
                        'DIRECTION',
                        'FRAGMENT_NAME'
                ]
            )

            scroll_df = scroll_df.sort_values('SYSTEM_TIME')

            if scroll_df.shape[0] == 0:
                print('Scroll file has no data.\n')
                continue
            if scroll_df['EVENT_ID'][0] == -1:
                print('Scroll file has -1 data.\n')
                continue

            sensor_merged = pd.merge_asof(accelerometer_df, magnetometer_df,
on='SYSTEM_TIME', direction='forward')
            sensor_merged = pd.merge_asof(sensor_merged, gyroscope_df,
on='SYSTEM_TIME', direction='forward')

            del sensor_merged['FRAGMENT_NAME_acc']
            del sensor_merged['FRAGMENT_NAME_gyr']
            del sensor_merged['FRAGMENT_NAME_mag']
            sensor_merged = sensor_merged.sort_values('SYSTEM_TIME')

            merged = pd.merge_asof(scroll_df, sensor_merged, on='SYSTEM_TIME',
direction='forward')

            merged['SYSTEM_TIME'] = merged['SYSTEM_TIME'].astype(float)

            # Scroll features

            merged['xDisplacement'] = signal.lfilter([1, -1], 1,
merged['CURRENT_X'])
            merged['xDisplacement'] = np.where(merged['xDisplacement'] >= 100,
0, merged['xDisplacement'])
            merged['yDisplacement'] = signal.lfilter([1, -1], 1,
merged['CURRENT_Y'])
            merged['yDisplacement'] = np.where(merged['yDisplacement'] >= 100,
0, merged['yDisplacement'])
            merged['pairwiseTimeDiff'] = signal.lfilter([1, -1], 1,
merged['SYSTEM_TIME'])
            merged['pairwiseTimeDiff'] = np.where(merged['pairwiseTimeDiff']
>= 100, 1, merged['pairwiseTimeDiff'])
            merged['pairwiseAngle'] = np.arctan2(merged['yDisplacement'],
merged['xDisplacement'])
            merged['pairwiseDisplacement'] =
np.sqrt(pow(merged['xDisplacement'], 2) + pow(merged['yDisplacement'], 2))
            merged['V'] = merged['pairwiseDisplacement'] /
merged['pairwiseTimeDiff']
            merged['A'] = signal.lfilter([1, -1], 1, merged['V']) /
merged['pairwiseTimeDiff']

            # Scroll Features end

            # Sensor Features

            merged['Acc_Mag'] = np.sqrt(merged['X_acc'] ** 2 + merged['Y_acc']
** 2 + merged['Z_acc'] ** 2)
            merged['Gyr_Mag'] = np.sqrt(merged['X_gyr'] ** 2 + merged['Y_gyr']
** 2 + merged['Z_gyr'] ** 2)
            merged['Mag_Mag'] = np.sqrt(merged['X_mag'] ** 2 + merged['Y_mag']
** 2 + merged['Z_mag'] ** 2)

            aggs = merged.groupby('EVENT_ID').agg({
                'X_acc': ['mean', 'std', 'median'],
                'Y_acc': ['mean', 'std', 'median'],
                'Z_acc': ['mean', 'std', 'median'],
                'X_mag': ['mean', 'std', 'median'],
                'Y_mag': ['mean', 'std', 'median'],
                'Z_mag': ['mean', 'std', 'median'],
                'X_gyr': ['mean', 'std', 'median'],
                'Y_gyr': ['mean', 'std', 'median'],
                'Z_gyr': ['mean', 'std', 'median'],
                'Acc_Mag': ['mean', 'std', 'median'],
                'Gyr_Mag': ['mean', 'std', 'median'],
                'Mag_Mag': ['mean', 'std', 'median'],
                'START_X': 'first',
                'CURRENT_X': ['last',
                            ('maxdev', lambda x: (x - x.mean()).max()),
                            ('dev20', lambda x: (x -
x.mean()).quantile(0.20)),
                            ('dev50', lambda x: (x -
x.mean()).quantile(0.50)),
                            ('dev80', lambda x: (x -
x.mean()).quantile(0.80))],
                'START_Y': 'first',
                'CURRENT_Y': ['last',
                            ('maxdev', lambda x: (x - x.mean()).max()),
                            ('dev20', lambda x: (x -
x.mean()).quantile(0.20)),
                            ('dev50', lambda x: (x -
x.mean()).quantile(0.50)),
                            ('dev80', lambda x: (x -
x.mean()).quantile(0.80))],
                'V': [('pairwise20', lambda x: x.quantile(0.20)),
                    ('pairwise50', lambda x: x.quantile(0.50)),
                    ('pairwise80', lambda x: x.quantile(0.80)),
                    ('medianVelocityLastThree', lambda x:
np.median(x.tail(3)))],
                'A': [('pairwise20', lambda x: x.quantile(0.20)),
                    ('pairwise50', lambda x: x.quantile(0.50)),
                    ('pairwise80', lambda x: x.quantile(0.80)),
                    ('averageAccFirstFive',
                    lambda x: np.median(x.head(5)) if x.size >= 5 else
np.median(x))],
                'pairwiseDisplacement': [('lengthOfTrajectory', 'sum')],
                'SYSTEM_TIME': ['first', 'last'],
                'CURRENT_PRESSURE': 'median',
                'CURRENT_SIZE': 'median',
            })

            aggs = aggs.fillna(0)

            aggs['distance'] = np.sqrt(
                pow(aggs['CURRENT_X']['last'] - aggs['START_X']['first'], 2) +
                pow(aggs['CURRENT_Y']['last'] - aggs['START_Y']['first'], 2)
            )
            aggs['directionOfEndtoEnfLine'] =
np.arctan2((aggs['CURRENT_X']['last'] - aggs['START_X']['first']),

(aggs['CURRENT_Y']['last'] - aggs['START_Y']['first']))

            #division by 0 (= inf) case eliminated
            #aggs['ratio'] = aggs['distance'] /
aggs['pairwiseDisplacement']['lengthOfTrajectory']
            aggs =
aggs.assign(ratio=np.where(aggs['pairwiseDisplacement']['lengthOfTrajectory']
!= 0, aggs['distance'] / aggs['pairwiseDisplacement']['lengthOfTrajectory'],
0))

            aggs['duration'] = aggs['SYSTEM_TIME']['last'] -
aggs['SYSTEM_TIME']['first']

            #division by 0 (= inf) case eliminated
            #aggs['averageVelocity'] =
aggs['pairwiseDisplacement']['lengthOfTrajectory'] / aggs['duration']
            aggs = aggs.assign(averageVelocity=np.where(aggs['duration'] != 0,
aggs['pairwiseDisplacement']['lengthOfTrajectory'] / aggs['duration'], 0))

            aggs['MeanResultantLength'] = circ_r(merged['pairwiseAngle'])
            aggs['AverageDirectionEnsemble'] =
circ_mean(merged['pairwiseAngle'])

            del aggs['SYSTEM_TIME']

            flatten_df = pd.DataFrame()

            for i, index0 in enumerate(aggs.keys().get_level_values(0)):
                if aggs.keys().get_level_values(1)[i] != '':
                    flatten_df[index0 + '_' +
aggs.keys().get_level_values(1)[i]] = aggs[index0][
                        aggs.keys().get_level_values(1)[i]]
                else:
                    flatten_df[index0] = aggs[index0]

            flatten_df['LABEL'] = "user"# + str(userid)

            flatten_df.to_csv("{}features.csv".format(sf), index=False)
            print('Done: ' + "{}features.csv".format(sf))
        except IndexError as e:
            print(e)
        except Exception as e:
            print(e)
#"""
```

Appendix G. Feature Merger in Python

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import glob
import re
import os

current_dir = os.getcwd()
data_dir = current_dir + '\\Data\\'
print("Current Data Dir: ", data_dir)

sessions = glob.glob(data_dir + "\\*\\")
print("Sessions", sessions)

for tl in sessions:
  try:
    print("Session --> ", tl)
    userid = int(re.search(r"\\(\d*)\\$", tl).group(1))
    print("User: ", userid)

    sf in glob.glob(data_dir + "{}\\*\\".format(userid))

    files = glob.glob("{}\\*\\features.csv".format(tl))
    df = pd.concat((pd.read_csv(f) for f in files))
    output = "{}\\merged_features_{}.csv".format(data_dir, userid)
    print("writing to: {}merged_features_{}.csv".format(data_dir, userid))
    df.to_csv(output, index=False)
    print('done')
    print()
  except Exception as e:
    print(e)
```

Appendix H. Feature of Other Users Merger in Python

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import glob
import re
import os

current_dir = os.getcwd()
data_dir = current_dir + '\\Data\\'
print("Current Data Dir: ", data_dir)

sessions = glob.glob(data_dir + "\\*\\")
print("Sessions", sessions)

for tl in sessions:
  try:
    print("Session --> ", tl)
    userid = int(re.search(r"\\(\d*)\\$", tl).group(1))
    print("User: ", userid)

    sf in glob.glob(data_dir + "{}\\*\\".format(userid))

    files = glob.glob("{}\\*\\features.csv".format(tl))
    df = pd.concat((pd.read_csv(f) for f in files))

    for user_folder in sessions:
        user_folder_id = int(re.search(r"\\(\d*)\\$", user_folder).group(1))
        if userid != user_folder_id:
            output = "{}\\merged_features_others_{}.csv".format(data_dir,
user_folder_id)
            print("writing to:
{}merged_features_others_{}.csv".format(data_dir, user_folder_id))

            #df['LABEL'] = "others"

            exists = os.path.isfile(output)

            if exists:
                df.to_csv(output, mode='a', index=False, header=False)
            else:
                df.to_csv(output, index=False)

            print('done')
            print()


  except Exception as e:
    print(e)
```

Appendix I. Feature of All Users Merger in Python

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import glob
import re
import os

current_dir = os.getcwd()
data_dir = current_dir + '\\Data\\'
print("Current Data Dir: ", data_dir)

sessions = glob.glob(data_dir + "\\*\\")
print("Sessions", sessions)

for tl in sessions:
  try:
    print("Session --> ", tl)
    userid = int(re.search(r"\\(\d*)\\$", tl).group(1))
    print("User: ", userid)

    sf in glob.glob(data_dir + "{}\\*\\".format(userid))

    files = glob.glob("{}\\*\\features.csv".format(tl))
    df = pd.concat((pd.read_csv(f) for f in files))

    output = "{}\\merged_features_all.csv".format(data_dir)
    print("writing to: {}merged_features_all.csv".format(data_dir))

    exists = os.path.isfile(output)

    if exists:
        df.to_csv(output, mode='a', index=False, header=False)
    else:
        df.to_csv(output, index=False)

    print('done')
    print()

  except Exception as e:
    print(e)
```

BIOGRAPHICAL SKETCH

Okan Engin Başar was born on September 26, 1987 in Ankara, Turkey. After graduating from Yunus Emre High School in 2005, he began to study in Department of Computer Engineering in Middle East Technical University in Ankara, Turkey. He graduated from Computer Engineering Department on Jan, 2012. After 2 years of work experience in Garanti Technology as a Computer Engineer, he enrolled in M.Sc. program in Computer Engineering Department in Galatasaray University. At the same time, he continued to work as a Software Developer in Garanti Technology and later as a Senior Software Project Engineer in Yapi Kredi Technology. His professional expertise as a Software Engineer is the development of mobile applications.