IoTSSC Project – Cloud-powered Wearable Fitness Tracking

Xu Zhang
s1604556@sms.ed.ac.uk
University of Edinburgh
United Kingdom

KEYWORDS

Internet of things, Human activity recognition, Classifier, Google Cloud, STM32, Android

1 INTRODUCTION

The first part of this report talks about background research made on Human Activity Recognition. The main body of the report walks through every single component implemented in the system. For each component, I will talk about the design decision(why I choose the method and tool), the actual implementation and potential future improvement for the component. In the end, the evaluation method and the result are discussed.

2 BACKGROUND RESEARCH

2.1 Human Activity Recognition[1]

Human Activity Recognition is also known as HAR is one of the most popular research areas in the past couple of years due to the increased accessibility of sensors used in wearable and IoT devices. Human activities that can be easily recognized using the sensor data usually include walking, running, resting, standing, sleeping and many more. The state of art HAR method includes the typical machine learning classification e.g. Decision tree or Neural network-based method e.g. Convolutional Neural Network. Most HAR method can achieve over 99% accuracy when the dataset is well collected

However, despite the high accuracy of these methods, there are still many challenges in the area of HAR. Many challenges are crucial if HAR was to become commercially available.

The first big challenge is the dataset used for machine learning, the type of sensor used and placement of the sensor on the body greatly affect the collected data and the accuracy of trained models, and even with a good dataset, real-life sensor readings are sometimes still quite different, therefore it is very hard to generalize a prediction model for everyone.

Secondly, many machine learning method suffers from either overfitting or underfitting when the training dataset is small or has low variance, so the actual implementation of the machine learning model can vary significantly depending on the dataset and scope of the activities.

Finally, even if we have a perfect dataset and a model is trained without significant problems, it is still very difficult to recognize multiple activities that happened simultaneously e.g. user change activities rapidly within seconds

Table 1: Reading unit for each sensor[6]

Sensor	Reading unit
temperature	Degree Celsius from 15 to +40 °C ± 0.5
Humidity	Relative humidity from 20 to +80% rH ± 3.5
Accelerometer	milli-gs with format (x, y, z)
Gyroscope	millidegrees per second with format (x, y, z)
Magnetometer	mill-gauss with format (x, y, z)

There are many more challenges that include multi-person activities recognition, vision-based activity recognition and time-complexity of the classification process increases significantly if high accuracy is required

2.2 Sensors for HAR

In one of the previous research[2] on various sensors for human activity recognition (HAR), it was shown that many of the researches on HAR were done using Accelerometer, Gyroscope and Magnetometer. On top of these three, temperature and humidity sensor reading may contribute toward HAR if we are measuring the human body temperature and humidity caused by sweat.

However, it is very clear that a microphone is not needed as we are not recording any sound. The barometer only measures air pressure, which does not change much if we are at sea level. The time-of-flight sensor is more commonly used for obstacle avoidance on the robot.

For the remaining 5 sensors, to process the sensor values correctly for our classifier, we need to first understand the unit/format of which the sensor values (see Table 1) are presented on the Pelion platform. This information was found on the user manual and sensor datasheets.[3][4][5]

2.3 HAR classifier

Common machine learning model used for classifying includes, support vector machine (SVM), decision tree, k-nearest neighbours (KNN), Naïve Bayes classifier and Multilayer perceptron (MLP), all of which has pros and cons depending on the usage scenario. In one of the studies[2], a comparison of HAR classification accuracy was made between various models. It was found that the performance of most classification models is similar when the model is trained and tested using the sensor data collected from a single subject. However, when the model is trained using 9 different subjects and tested on one other subject, NB, Random Forest (RF) and J48 decision tree gave the best classification accuracy. The latter scenario would

be more realistic in normal usage as we will not be able to retrain a model for every single new subject.

This previous research was done with only accelerometer, gyroscope, and magnetometer sensors. Considering we have also had data for timestamp, temperature and humidity, the best classifier can be different.

3 SYSTEM COMMUNICATION DESIGN

The technologies/tools used to implement every component of this project are shown and the communication between each component is shown by the arrows (See Figure 1).

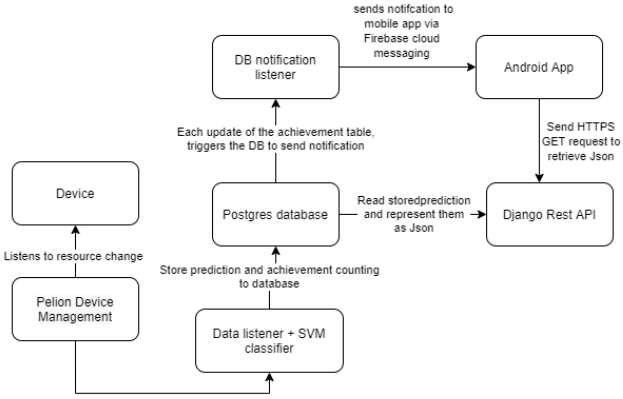

Figure 1: Communication design

4 CLASSIFIER COMPARISON

I have developed 5 commonly used classifiers using Tensor flow[7] and Scikit-learn[8] including a linear support vector machine, an artificial neural network with two hidden layers ([30,10], step=10000), decision tree, k-nearest neighbours (with k=3) and Naïve Bayes classifier. For these models, I used the dataset mentioned in the coursework[9], and randomly split the dataset into 70% training set and 30% testing set with a random seed of 42.

In the dataset, we have 'date' and 'time' available. However, it was not possible to train models with Datetime objects, the solution for this is to convert date into ordinal and storing hours, minutes, and seconds individually.

For example, '30/06/2017' would be stored as 736510 and '13:51:15:847724020' would be stored in 4 different columns (hours, minutes, seconds and millisecond) like 13, 51, 15 and 8477240. The nanosecond component is omitted due to the limitation of the DateTime library in python and it is very insignificant anyway for our usage.

The decision tree performs the best for both including and excluding 'date' and 'time'. Whereas because feature column increased by 5. SVM was not possible to train on a local machine for a reasonable time (see Figure 2).

Model	Accuracy	Training time(seconds)
SVM	0.858	46
ANN	0.935	14.87
KNN	0.985	0.1
Decision tree	0.985	0.45
NB	0.958	0.01

Table 1. Dataset excluding date and time.

Model	Accuracy	Training time
SVM	N/A	N/A
ANN	0.515	21.4
KNN	0.875	0.22
Decision tree	0.996	0.26
NB	0.674	0.013

Table 2. Dataset including date and time.

Figure 2: real time prediction endpoints

Due to the dataset used, the classifier is limited to recognize only "walking" and "running". Ideally, I should collect new data using the system and tag them into different activities group to retrain the model. However, this was not possible due to COVID Lockdown.

5 EMBEDDED FIRMWARE

The embedded application developed is derived primarily based on the code made available from the lab material[10] and the official tutorial on Pelion[11]. The firmware is capable of reading from all sensors available on the board for a chosen interval, these sensor reading are then sent over WIFI to the Pelion device management platform via M2M communication.

The interval chosen is 5 seconds, as it does not seem logical for a user to change their activity frequently within 5 second time frame, so this sufficient for our usage, and doing this also will help reduce the database storage usage. The firmware was tested with intervals of 0.5 and 1 second, so if required this can always change

The M2M communication is established between the device and Pelion device management platform by doing the following steps:

- Creating API key with administrator rights on Pelion
- Applying the API key in Mbed Studio
- Creating a developer certificate on Pelion
- Including client registering code in the firmware

This Client firmware will then support pushing updates to all devices(with WIFI enabled) registered on the Pelion platform, which is a convenient feature for project scaling.

After successful client registering, the sensor readings from each device can be view on Pelion device management platform on preset resource path e.g., resource path /3333/0/5704 for temperature readings(see Table 2). For 3-axis sensors, three reading can be sent to 5702, 5703, 5704 which represents x, y, z respectively. By doing it this way, request to Pelion API path e.g. /3334/0 can retrieve all 3 values simultaneously.

All of these resources will be subscribed by the web app hosted on google cloud app engine to access the reading and classify the activity.

Table 2: Path for sensors on Pelion[6]

Resource	Path
temperature	/3333/0/5704
Humidity	/3000/0/5604
Accelerometer	/3334/0/5702-5704
Gyroscope	/3313/0/5702-5704
Magnetometer	/3314/0/5702-5704

6 CLOUD SERVER

The Cloud Server consisted of 4 parts,

- Pelion API requesting script + SVM classifier
- Django Web application
- PostgreSQL Database
- Database listener + Firebase Cloud messaging script

6.1 Pelion API and SVM classifier

6.1.1 Design choice.

Originally I planned to use Pelion REST APIs directly by using HTTP PUT to subscribe to resource changes as shown by their tutorial. For some reason, there are very few third-party learning resources available and their documentation are somewhat misleading and badly managed(with links to pages returning 404). After constantly getting unauthorized status code, I moved on to use their old SDK solution[12], which is no longer officially supported but after some test, it works very well and there is enough learning resource available in their Github repository.

As mentioned in the previous section, the decision tree classifier had the highest accuracy when predicting the testing set. However, when passing real time sensor readings to a decision tree classifier, it did not manage to predict the activities accurately, my assumption is that the high accuracy is a result of over-fitting to the Kaggle dataset and therefore it does not work well with new readings. Furthermore, including time information did not help classification very much, my assumption on this issue is that the Kaggle dataset is collected over around 10 days and I was testing the system at the time that is very different from the dataset. In the final version of the prototype, the SVM model(regardless of kernel mode, as they perform similarly) is used as it is shown to make better predictions.

6.1.2 Implementation.

This API requesting script is based on the example given on the Pelion-SDK-Python repository and is hosted on Google compute engine[13]. The script will first request the registered device on the Pelion platform, then it starts a while true loop(forever running), which request resource value with the given device id and resource path. The script checks if the new response received is different from the last request, if so, new resource values are passed to the SVM classifier for human activity recognition, the classification result is stored to Postgre Database alongside the current time. At the same time, depending on the classification result walking or running, increasing the respective field in the achievement table by 1(see Algorithm 1), the entire process is done asynchronously by waiting for the response to complete and catch potential errors that can halt the process.

Data received from the Pelion platform is quite different from the Kaggle dataset. After doing some research online, I found that the sensor reading needed to be divided by 1000 to match the Kaggle dataset[14]. Additionally, instead of using unit 'Degree' for gyroscope readings, we need to convert degree to radians as the Kaggle dataset is collected using IOS and IOS uses radians for their built-in gyroscope[15].

Algorithm 1: API requesting main logic

```
Initialize API objects;
Establish connection with notification channel;
Retrieve all registered devices;
while True do
    Get Accelerator resource values;
    Get Gyroscope resource values;
    while Accelerator or gyroscope retrieving is not done do
        Wait for 0.1 second;
    end
    if error occurred during retrieving then
        Log the error;
        Continue the loop;
    end
    if new reading not equal to previous reading then
        Get the current time;
        Extract x,y,z values from reading;
        Converting values to correct unit;
        Classifying on these values;
        Store the classification result to Database;
        if result is walking then
            increase walking field by 1;
        else
            increase running field by 1;
        end
    end
end
```

6.1.3 Future improvement.

The future improvement includes, migrating the current API requesting to use REST APIs directly, the temporal information used to train classifier should be personalized for each user rather than generalized, and decrease the weight of the temporal information in the training process.

6.2 Web application

6.2.1 Design choice.

Initially, I planned to have a classifier included within the web app, and since the classifier is made with Python, using a python web framework would make integration much easier during development. Therefore I have decided to use Django[16] to implement the REST API endpoint[17] as I am much more familiar with Django than Flask.

I have decided to host the web app using the gcloud app engine, because it supports HTTPS encryption out of the box with SSL certificate renewed automatically[18], this ensures that the connection is always secure if this web app was to be deployed for long term use. Furthermore, as the app engine is Platform as a Service(PaaS), it supports autoscaling, this means the instances required for the web app will increase/decrease depending on the traffic, this will help to reduce cost and maintain the stability of the system.

6.2.2 Implementation.
The implemented Django API has 2 endpoints, "/prediction/realtime" (see Figure 3) and "/prediction/historical" (see Figure 4). As the name suggested, the realtime endpoint returns real-time prediction and the historical endpoint returns prediction history.

When the "/realtime" endpoint is called, Django will access the database and retrieve the latest classification result, and return the data as JSON. When the "/historical" endpoint is called, Django will access the database and retrieve all classification results stored, and return data as JSON, to ensure JSON is formatted correctly, data are passed to a custom serializer for validation and serialization

To increase the security of these endpoints, I have implemented token authentication within Django[19]. A secrete token can be generated for a user created by Django, to access these endpoints, the secret token must be included in the Header of each GET request, anyone trying to access these endpoints without valid tokens will receive HTTP 401 Unauthorized code.

6.2.3 Future improvement.
The secret token used for authorization is currently valid infinitely long, which will cause a problem if such token is acquired maliciously, therefore the next step for the web app is to create another endpoint e.g. user/login, which on request from a registered user give out temporary tokens.

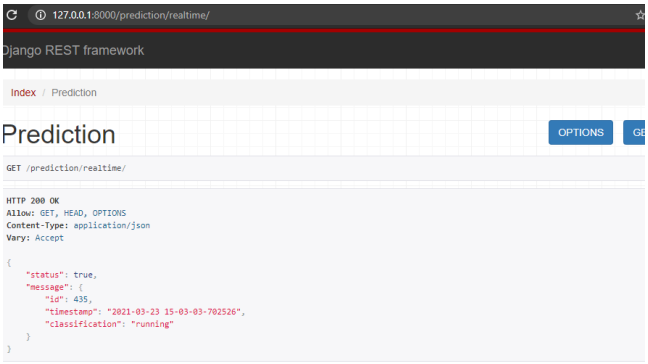

Figure 3: real time prediction endpoints

6.3 PostgreSQL

6.3.1 Design choices.
PostgreSQL Database is used for this project simply because Django uses this by default, for our usage, all SQL database can work and

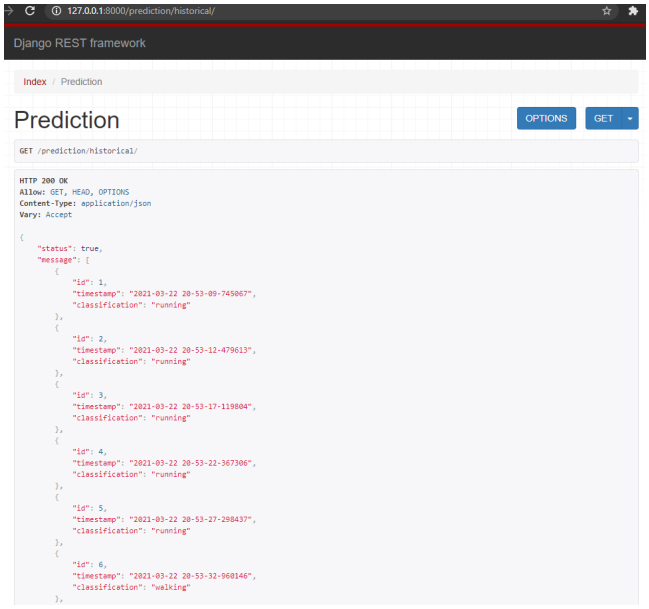

Figure 4: Historical prediction endpoints

considering this is a light-weight system, there are no significant performance differences.

This Database is hosted using Cloud SQL[20] as it is fully-managed so like app engine it supports autoscaling where it increases the storage availability as it fills up, it also provides auto backups, data encryption and integrates with app engine natively.

6.3.2 Implementation.
The database consists of two tables, "classification" and "achievement". The classification table contains two columns "timestamp" and "classification", both are stored as strings, "timestamp" represents the time that the prediction was made, and "classification" represents the result of the prediction, it is either "walking" or "running".

The achievement table contains two columns, "walking" and "running" both are stored as integers, this table only has one row, each time a prediction is made, we update the value in the corresponding column by 1. The achievement table also contains a trigger, a notification function is triggered after each update is made to the achievement table, this notification function make a notification on channel 'achievement' with a payload containing the latest values of both "walking" and "running" columns. (see Figure 5)

6.4 Push Notification

6.4.1 Design choices.
There are many solutions available to push notifications to the android app, However, many of these services have a monthly cost. I have decided to use Firebase Cloud Messaging(FCM)[21], which is a free product and since Firebase is a Google product, it integrates nicely with the Google Cloud platform where the rest of the project is hosted on. This functionality is also implemented as a python

```sql
drop table if exists classification;
create table classification(
    id serial PRIMARY KEY NOT NULL ,
    timestamp varchar(255),
    classification varchar(255)
);

drop table if exists achievement;
create table achievement(
    id serial PRIMARY KEY NOT NULL ,
    walking INTEGER,
    running INTEGER
);

CREATE OR REPLACE FUNCTION mytrigger()
  RETURNS trigger AS
$BODY$
begin
    PERFORM pg_notify('achievement', row_to_json(new)::text);
    return new;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE;

CREATE TRIGGER mytrigger
    AFTER INSERT OR UPDATE ON achievement
    FOR EACH ROW
    EXECUTE PROCEDURE mytrigger();
```

Figure 5: SQL initialization script

script and hosted on the same compute engine as the Pelion API script.

6.4.2 *Implementation.*

This script makes use of two libraries, psycopg2[22], for making connection to the database and pyfcm, to communicate with FCM service using API key. This script will first establish a connection with FCM and Postgres database, then start to listen to the channel "achievement". A while True loop is started to make sure this script is running forever. Whenever a notification is sent to channel "achievement", retrieve the payload and compare the values with the preset achievement, if an achievement value is reached, ask FCM to push to notification using the provided messages. (see Algorithm 2)

6.4.3 *Future improvement.*

One major issue with the current push notification implementation is that past achievement is not recorded, it is impossible for a user to obtain a list of their obtained achievements. A possible solution is to have another table storing achievement and the date of obtainment.

7 MOBILE APP

The android app consists of three pages, the home-screen page which set up the app to receive and display notification. There are two buttons in the home-screen taking the user to real-time or historical page to view corresponding activity predictions.

All 3 activity within the Android app has error catching implemented (using try, except clauses), making sure that regardless of what went wrong in the server-side, the app does not crash.

Algorithm 2: Push notification main logic

```
Establish conncetion with FCM using api key;
Setting up target app id;
Setting up achievement goals;
Establish connection with database;
Start listening to channel "achievement";
while True do
    Connection Pool created;
    while DB make a notification do
        Get payload from the notification;
        Convert payload from dict string to JSON;
        Read JSON to get latest values in achievement table;
        if walking value is reached 10 then
            Call FCM to push notification "walking 10
              achievement reached"
        end
        if running value is reached 10 then
            Call FCM to push notification "running 10
              achievement reached"
        end
        Setting more achievement using the above format as
          required;
    end
end
```

7.1 Receive Notification

On startup of the android app, FCM connection is established and an FCM token is generated, this token is used by the server to identify the app and device and so the notification is pushed to the correct app on a correct mobile device. (see Figure 6 & 7)

7.2 Real-time prediction

The real-time prediction page is very simple, there is a digital clock showing the current time of the user's timezone. On the startup of this page, the app makes an HTTPS GET request to the endpoints served on the App engine to retrieve the real-time prediction result and show it on the screen. There is a refresh button in the bottom right if the user wishes to make another real-time prediction. (see Figure 8)

7.3 Historical prediction

The historical prediction page is similar to the real-time page. On startup, the app also makes an HTTPS GET request to Django endpoint, however, since this time we are receiving all past predictions, we cannot simply display it using a TextView, instead, using a combination of CardView and RecyclerView, each prediction is JSON is parsed and displayed as a card. User can scroll up and down to view every single past prediction. There is also a refresh button if the user wishes to make update the page (see Figure 9)

7.4 Future improvement

The most important change needed for the app is the UI, currently, there is no design at all, white background with no animations. Additionally, the historical prediction page can be changed to accept

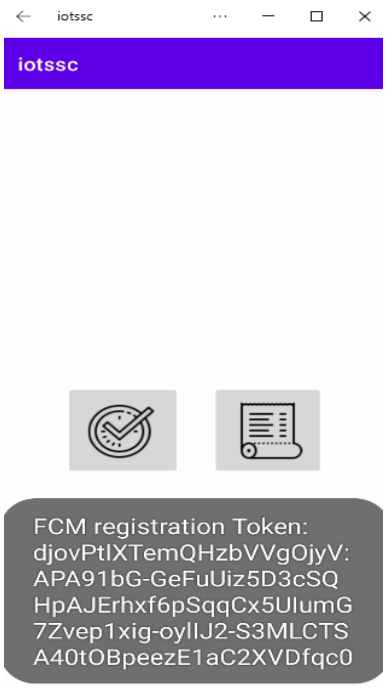

Figure 6: real time prediction endpoints

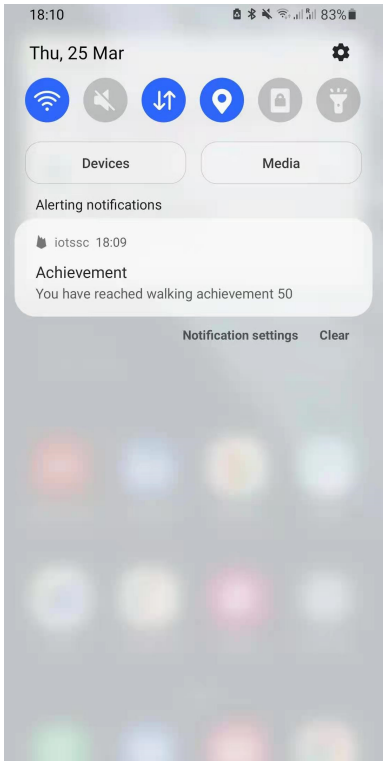

Figure 7: Showing Notification

a date range input and show only predictions within the range instead of showing every single past prediction and maybe implement a visualization graph instead of RecyclerView.

8 EVALUATION

The system is evaluated in four aspects, CPU usage, memory usage, security and the accuracy of classification.

8.1 Usage

All components for the cloud server together utilized 1.8% of the CPU(see Table 3), although this is a single-user scenario, it is clear that all tasks are not CPU intensive, the memory used by all component for the cloud server is 156MB, considering in 2021, most mobile phone has at least 8GB memory, these tasks are also not memory intensive.

I could not find a way to test the CPU usage of an Android app. The 55.7MB memory usage on the other hand is actually a little bit too high compared to other apps with much more functionalities. For example the BBC app uses around 60MB when running in the background. My assumption is that I have used three activities where the app can be easily made with one activity with reusable fragments. However, so far I did not find a source confirming fragment uses less memory compared to activities.

Table 3: Tested on AMD Ryzen 9 3900X and Samsung S21

Functionality	CPU Usage	Memory Usage
Django API	1.2%	69.8MB
Pelion API + Classifier	0.5%	70.8MB
Push notification	0.1%	15.4MB
Android app	N/A	55.7MB

8.2 Security

The main security issue in the project is the publicly accessible Django web app, the security of the remaining component are all managed by platforms and there are not much to be done from the developer's perspective.

The Django app is set to allowing HTTPS request only, this will make sure all connection is secure and does not suffer from attacks that HTTP is vulnerable for e.g. man-in-the-middle. To ensure access to API endpoints are limited to registered users only, token authorization is implemented, anyone accessing the publicly accessible Django API without valid token receives 401 error code, this still does not present attacks like DDoS, but at least the data stored is relatively safe.

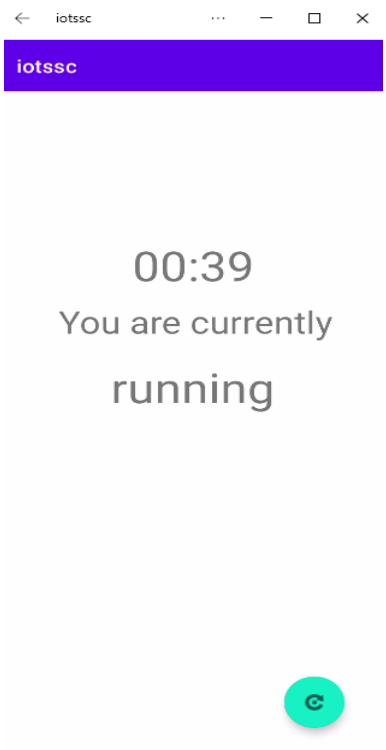

Figure 8: real time prediction endpoints

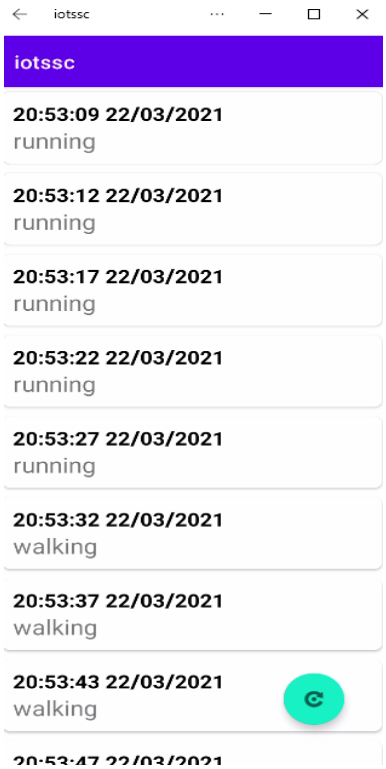

Figure 9: real time prediction endpoints

Table 4: Classification accuracy

Model	Accuracy
SVM(linear)	89%
SVM(Poly)	85%
Decision tree	78%
KNN	76%

Some other potential security issue with the current system includes, infinitely long valid token, no proper user login authorization and all API keys used are clear text in the code on Github. However, these are acceptable for a prototype system.

8.3 Classification accuracy

To evaluate the classification accuracy for real-time data, I run the system for 50 classifications while periodically change the between walking and running and record the current activities on my phone, then I manually match the classification result to work out the accuracy. The result shows that linear SVM is performing the best(See table 4)

REFERENCES

[1] Charmi Jobanputra, Jatna Bavishi, and Nishant Doshi. Human activity recognition: A survey. *Procedia Computer Science*, 155:698–703, 2019. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2019.08.100. URL https://www.sciencedirect.com/science/article/pii/S1877050919310166. The 16th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2019),The 14th International Conference on Future Networks and Communications (FNC-2019),The 9th International Conference on Sustainable Energy Information Technology.

[2] Warren souza and R. Kavitha. Human activity recognition using accelerometer and gyroscope sensors. *International Journal of Engineering and Technology*, 9: 1171–1179, 04 2017. doi: 10.21817/ijet/2017/v9i2/170902134.

[3] B-l475e-iot01a - stm32l4 discovery kit iot node, low-power wireless, ble, nfc, subghz, wi-fi - stmicroelectronics. URL https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html#documentation.

[4] Um2153 user manual discovery kit for iot node, multi-channel communication with stm32l4, 2019. URL https://www.st.com/resource/en/user_manual/dm00347848-discovery-kit-for-iot-node-multichannel-communication-with-stm32l4-stmicroelectronics.pdf.

[5] This is information on a product in full production. digital output magnetic sensor: ultra-low-power, high-performance 3-axis magnetometer datasheet -production data features, 2017. URL https://www.st.com/resource/en/datasheet/lis3mdl.pdf.

[6] Open mobile alliance - lwm2m registry. URL https://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html.

[7] tf.estimator.dnnclassifier | tensorflow core v2.4.1. URL https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier.

[8] scikit-learn 0.22.1 documentation, 2019. URL https://scikit-learn.org/stable/user_guide.html.

[9] Viktor Malyi. Run or walk. URL https://www.kaggle.com/vmalyi/run-or-walk.

[10] iotssc-labs-20-21. URL https://git.ecdf.ed.ac.uk/s1443541/iotssc-labs-20-21.

[11] Command-line tutorial for the advanced device management client example with an mbed os device - connecting devices | pelion device management documentation. URL https://developer.pelion.com/docs/device-management/current/connecting/mbed-os.html.

[12] Pelioniot/mbed-cloud-sdk-python, 11 2020. URL https://github.com/PelionIoT/mbed-cloud-sdk-python.

[13] Compute engine: Virtual machines (vms) | google cloud, 2012. URL https://cloud.google.com/compute.

[14] Mark Williams. Converting values from an accelerometer to g, 06 2016. URL https://ozzmaker.com/accelerometer-to-g/.

[15] Apple developer documentation, . URL https://developer.apple.com/documentation/coremotion/getting_raw_gyroscope_events.

[16] Django documentation | django documentation | django. URL https://docs.djangoproject.com/en/3.1/.
[17] Tom Christie. Django rest framework, 2011. URL https://www.django-rest-framework.org/.
[18] App engine, . URL https://cloud.google.com/appengine.
[19] Vitor Freitas. How to implement token authentication using django rest framework, 11 2018. URL https://simpleisbetterthancomplex.com/tutorial/2018/11/22/how-to-implement-token-authentication-using-django-rest-framework.html#implementing-the-token-authentication.
[20] Cloud sql: Relational database service. URL https://cloud.google.com/sql.
[21] Firebase cloud messaging | firebase, 2019. URL https://firebase.google.com/docs/cloud-messaging.
[22] Federico Di Gregorio. psycopg2. URL https://pypi.org/project/psycopg2/.