Homework 3 -- MUC Winter 2015/2016

Deadline: November 22nd, 23:59.
Note: The assignments are to be done strictly individually.

In this homework we continue with the development our app for monitoring a user's wireless connectivity, and collecting geotagged information about the wireless connection quality. We will present this information on an interactive map, and via a list of entries. In addition, the app will ensure data persistency by using remote cloud services.

In homeworks 1 and 2 we have completed:

- Collecting a user's (manual) input
- Storing and retrieving the data from the application-allocated memory
- Interacting with the user through Tabs and Fragments
- Displaying a Google Map to a user
- Supporting conditional navigation, so that the app behaves differently depending on whether it was launched for the first time or not.
- Sensing a user's location
- Showing information on the map
- Scheduling periodic repetitive tasks
- Saving data in a local SQLite database
- Using a third-party machine learning library in your project

In homework 3 we will work on:

- Sensing the type and the quality of a user's wireless connection
- Labelling sensor data with learnt labels
- Merging location and WiFi connectivity data
- Displaying the merged data in a list and on the map
- Saving valuable raw data to a remote cloud

You should work on the same github repository that you used for the first homework. You just need to extend the app to support the requirements of homework 3 and push your code to the repository. Don't forget to add new resources and source files (including a new backend module) to the repository!

Background sensing connectivity quality

In the previous homework we were sensing location in the background. Since we are interested in wireless connectivity, we are going to extend periodic sensing with the gathering of information regarding user's connectivity.

You already have an AlarmManager that periodically fires a task for sensing a user's location. Assuming you have an IntentService for that task (other solutions are also adequate, such as managing your own Threads and Runnables), create another class which also inherits from IntentService, and implement connectivity sensing in this class. We are interested in sensing *whether a user is connected to WiFi*, and if so, *what is the value of the received signal strength indicator (RSSI)*. **RSSI** is a negative often on the lower side of the -100 to 0 interval. In the labs we went over checking the type of connectivity and getting RSSI. However, we are also interested in *the name of the access point (AP)* a user is connected to. As you know (or will soon learn) from the lectures, WiFi APs have a descriptive name called **SSID** (such as "eduroam") and the hardware interface name/address called **BSSID** (e.g. "6c:40:08:b0:2b:1a").

You can get the information about the SSID and BSSID, and of course the RSSI, using WiFiInfo object referring to the AP a phone is connected to:

```
WifiManager wifiManager = (WifiManager)
                          context.getSystemService(Context.WIFI_SERVICE);
WifiInfo wifiInfo = wifiManager.getConnectionInfo();
String SSID = wifiInfo.getSSID();
String BSSID = wifiInfo.getBSSID();
int RSSI = wifiInfo.getRssi();
```

The data should be saved in the local SQLite database, so create another table, e.g. “wifi” and store the data there. We want to relate the data about WiFi connectivity and the data about a user’s location. Since we have data in two different tables (assuming your original table for GPS coordinates is “location”), *we need to introduce a common column for the two tables*. The easiest way is to have a **trigger_ID** field that will label the location sensing and the WiFi sensing task triggered at the same time with the same value (a positive integer, for example), and put that value in both tables. trigger_ID should be made persistent (using SharedPreferences) and incremented every time a sensing session is started.

Labelling with the learnt information

Originally we sampled the location, labelled late night samples as “home”, and daylight, workday samples as “work”, and then applied machine learning to figure out where a user’s home and work locations are. Now, we have already trained the learner, so we can use it to label location sampled at any time just by comparing the distance between that sampled location’s coordinates and the coordinates of work/home. So, modify the app to do that:

- If the app has not collected enough samples and has not trained the classifiers (you still don’t know where home/work area), label the data using the time of day, as before.
- If the app has collected enough samples, and has already trained the classifier, label the data so that samples that are within 200m from home/work are labelled as home/work. If the coordinates are not close to home or work coordinates, the sample should be labelled as “other”.
  NOTE: classifiers from the machine learning toolkit have “classify(Instance)” method implemented, however, you shouldn’t use it in this case, as classification in DensityClustering case means returning the closest centroid, irrespective of how far the point is from the centroid (e.g. if you’re 5km from home and 6km from work, “classify” would return “home).

Make sure that you “remember” that you have trained the classifier, so you can start using it. SharedPreferences are a good class for saving such information.

Tune sensing parameters

This is probably a good time to revise the sensing strategy. “Sampling interval” setting in “Settings” tab should be connected with the actual interval between consecutive sensing points. The user should be able to change this time, so that if she notices that the app is draining too much battery can reduce sensing to say, every 45 minutes. Your app should detect when this value is changed (for example, by using SharedPreferences.OnSharedPreferenceChangeListener) and adjust the AlarmManager accordingly.

Display sensed information in a list

We've reserved one of the fragments for showing statistics about WiFi connectivity. When a user clicks on the corresponding tab, the app should:

- Calculate and display the average RSSI at "work"
- Calculate and display the average RSSI at "home"
- List (up to) top ten sensed access points, display their SSID, RSSI and label (home/work/other). When a user clicks on one of the listed entries, the map fragment should appear with the AP location marked.

Google Cloud Service

Over time your app will collect a significant amount of valuable data, we don't want to risk losing it. Google provides a variety of services on their cloud platform. We will use datastore in this homework -- it will allow us to keep data securely in the cloud.
NOTE: You need to sign up to google cloud services in order to use it on the public Web, and although it is free it will ask you for a credit card number.

We will cover Google cloud services in the labs, but if you're curious, you can go over a nice tutorial here:
https://cloud.google.com/solutions/mobile/how-to-build-mobile-app-with-app-engine-backend-tutorial/

Add a new module to your project where you will develop the backend part of the application. The backend part should take the data from the mobile, and store it in datastore. Transferring large amounts of data while the user is connected to a cellular network is not a good idea as it may increase the user's telephone bill. Transferring large amounts of data while connected to a poorly performing WiFi is not a good idea either, as it leads to broken transmissions and low energy efficiency. Thus, your app should transfer the data to the backend only when a user is **connected to one of the top WiFi access points**. In addition, the app should not transfer data more than once every six hours.

Your app might be installed by multiple users. Use the email provided at the registration time to link the data with the user who generated it.

Extra credit: Create a backend webpage on Google Cloud Service that will allow the user to get the information about her top ten access points directly from the cloud.

Things to keep in mind

- Google maps require project registration and an API key. Google cloud services require another kind of registration, and another API key in case you are using Google cloud messaging.
- You can test your backed app even before releasing it on the public Web. Once you add the backend module to your app a new run configuration appears in Android Studio (called "backend", if that's how you named your module). Running it starts your backend on the local machine.
- Ensure that your app state is preserved. If you trained classifier once and learned the locations of home/work, do not perform the same task the next time the app is restarted.

The deliverables of the first homework

- The source code of your application will be checked out from the git account you provided at 23:59 on November 22nd.
- An APK of your app, to be uploaded via ucilnica.