UNIVERSITÀ DI PISA

DEVELOPMENT OF AN ANDROID APPLICATION ABLE TO RECOGNIZE HANDWASHING USING OPEN-SOURCE LIBRARIES

Mobile and Pervasive Systems Project

Francesco FORNAINI
Elena SCARSELLI
Gerardo ALVARO
Riccardo POLINI

A.Y. 2019-2020

Contents

1. Introduction

The idea of this project is to identify and recognize the user's hand washing. When the user goes back home, a 15-minute windows is simulated to check if an hand washing activity is detected during this time. The service is constantly waiting to receive notification regarding the user's homecoming. At this point the application is activated and checks if the user washes his hands in this period (simulated with a .csv file containing 15 minutes of data). If the washing takes place, the application notifies the module that manages the Covid-19 contagion risk index associated with the user, thereby decreasing it.

1.1 Project choices

1.1.1 Recognition of movements

The movements will be recognized through a smartphone or a smartwatch worn on the wrist. The data collected by the sensor will be verified through a classifier. Data from a day of activity was collected with habitual movements and data from ten hand washes was separately collected. The handwashing data was used to train the classifier and cross-validation of the long-lasting data was then made. False positives have been observed. For practical reasons (we didn't have a smartwatch), the application is not directly connected to a wearable device. All data was obtained through a Samsung Galaxy S5 tied to the user's wrist and saved in *csv* format.

1.1.2 Data collection

To collect data we used a smartphone that is worn as a smartwatch. In this way we collect accelerometer and gyroscope movement data. First, data relating to hand washing alone was collected, for a total of 4000 captures from the sensors and subsequently a session of normal daily activities was recorded, including hand washing, for a total of 30 thousand captures from the sensors. We have obtained datasets of different activities. The dataset contains the measurements of gyroscope and accelerometer in the form of <x, y, z> coordinates. The data were obtained by sampling at a frequency of 50Hz, so as to have fifty measurements per second.

1.1.3 Data classification

The classification is done in real time in 1 second windows. Every second the classifier checks whether the user is washing their hands by checking the gyroscope classifier and the accelerometer classifier. The two classifiers are binary, both return:

- "0" if hand washing has not been detected;
- "1" if hand washing has been detected.

More on this in chapter 1.1.7.

1.1.4 Features

We have implemented the MotionSensor class that allows us to save 50 elements in an array, sampling at 50Hz, which correspond to one second of activity, coming either from the gyroscope or from the accelerometer. The data are divided into three planes: x, y and z. The collected data are saved in arrays and for each of the three dimension, the seven feautures are calculated, so we have a total of 21 features. We extracted the following features from the accelerometer and gyroscope trace file: MEAN, MAX, MIN, STANDARD DEVIATION, VARIANCE, SKEWNESS, ZERO CROSSING RATE for each axis (x, y, z). The features are calculated as follows.

Mean

To compute the mean we use the `computeMean` function.

```
1 private double computeMean(List<Double> list)
```

The `computeMean` function calculates the mean according to this formula:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Max

To compute the max we use `max()` according to the formula MAX=max $\{x_1, x_2, .., x_{50}\}$

```
1 Collections.max(list);
```

Min

To compute the min we use `min()` according to the formula min = min $\{x_1, x_2, .., x_{50}\}$

```
1 Collections.min(list);
```

Standard Deviation

To compute the standard deviation we use the `computeStdDev` function.

```
1 private double computeStdDev(List<Double> list, double mean)
```

The `computeStdDev` function calculates the standard deviation according to this formula:

$$stdDev = \sqrt{\frac{\sum_{i=1}^{n} |x_i - \bar{x}|^2}{n}}$$

Variance

To compute the variance we use the `computeVariance` function.

```
1 private double computeVariance(List<Double> list, double mean)
```

The `computeVariance` function calculates the variance according to this formula:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

Skewness

To compute the skewness we use the `computeSkewness` function.

```
1 private double computeSkewness(List<Double> list, double mean, double
    stdDev)
```

The `computeSkewness` function calculates the skewness according to this formula:

$$Skew = \frac{n}{(n-1)(n-2)} \cdot \sum_{i=1}^{n} \frac{(x_i - \bar{x})^3}{\sigma^3}$$

Zero Crossing Rate

To compute the Zero Crossing Rate we use the `computeZeroCrossingRate` function.

```
1 private double computeZeroCrossingRate(List<Double> list, double mean,
    double std)
```

The `computeZeroCrossingRate` function calculates the zero crossing rate according to this formula:

$$ZCR = \frac{1}{n} \cdot \sum_{i=1}^{n} |sign(i) - sign(i-1)|$$

1.1.5 Risk index management

The risk index is not directly calculated by our module, but is calculated thanks to a parameter we have detected. Our application intercepts intent *homeproximity* sent in broadcast by the geolocation module. Once this intent is intercepted, a simulated timer will start indicating the time within which the user must wash his hands to lower his risk index. If an hand washing activity is detected, our module will send an *handWashed* intent message in broadcast that will be intercepted by the module that deals with modifying the risk index. By common accord with this group the message will be sent only if the activity is detected. If the user is used to wash his hands after returning home, his risk index will decrease, otherwise not.

1.1.6 Accuracy of the classifier

After the extraction of the features through some python scripts, an offline training with Weka has been performed. Weka is a java application that allow to perform the training and the preprocessing of data through a simple GUI. We found out that with the initial dataset (which has been properly balanced through weka's funcionalities) we had similar accuracy results over various classificator. The classificator that has been tester are *Multi-Layer Perceptron, Naive Bayes, decision tree, decision forest and J48*. The accuracy obtained by the classifiers for the gyroscope is 80% and for the

accelerometer is 96%. Due to the simplicity of implementantion and his accuracy results similar to the other, the J48 algortithm has been chosen for the implementation in the final result.

```
=== Summary ===

Correctly Classified Instances         654.7627          96.0063 %
Incorrectly Classified Instances        27.2373           3.9937 %
Kappa statistic                          0.9201
Mean absolute error                      0.0437
Root mean squared error                  0.1994
Relative absolute error                  8.7406 %
Root relative squared error             39.8849 %
Total Number of Instances              682

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,937    0,017    0,983      0,937    0,959      0,921    0,963     0,958     True
                 0,983    0,063    0,940      0,983    0,961      0,921    0,963     0,937     False
Weighted Avg.    0,960    0,040    0,961      0,960    0,960      0,921    0,963     0,947

=== Confusion Matrix ===

   a      b      <-- classified as
 319.42  21.58 |      a = True
   5.66 335.34 |      b = False
```

Figure 1.1: Accellerometer J48 results

```
=== Summary ===

Correctly Classified Instances         551.8263          80.9129 %
Incorrectly Classified Instances       130.1737          19.0871 %
Kappa statistic                          0.6183
Mean absolute error                      0.1942
Root mean squared error                  0.4316
Relative absolute error                 38.8448 %
Root relative squared error             86.312  %
Total Number of Instances              682

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,696    0,078    0,899      0,696    0,785      0,635    0,825     0,820     True
                 0,922    0,304    0,752      0,922    0,828      0,635    0,825     0,747     False
Weighted Avg.    0,809    0,191    0,826      0,809    0,807      0,635    0,825     0,783

=== Confusion Matrix ===

   a      b      <-- classified as
 237.41 103.59 |      a = True
  26.58 314.42 |      b = False
```

Figure 1.2: Gyroscope J48 results

1.1.7 Handwashing Recognition

Handwashing is a complex activity which lasts for a certain amount of time, usually varying between different people. Classifying every second does not yield a realiable output since many false positives can be induced by performing specific wrist motions in order to "fool" the classifier. Moreover, the accelerometer and gyroscope classifiers may produce conflicting classifications. How can we factor out these problems?
Handwashing can't be recognized over a time frame of 1 second so we decided to classify the activity on a time window of 10 seconds which, to us, seems a reasonable time for handwashing. Every second the classifier produces an output and saves it in a queue of 10 elements. There is a queue for the accelerometer and one for the gyroscope. To decide if a user has or has not washed his hands in this time frame a *weighted average* of the sum of elements (binary classifications) in both queues is computed. If this average is greater or equal to a certain *THRESHOLD* then the classifier can state that the user has washed hands.

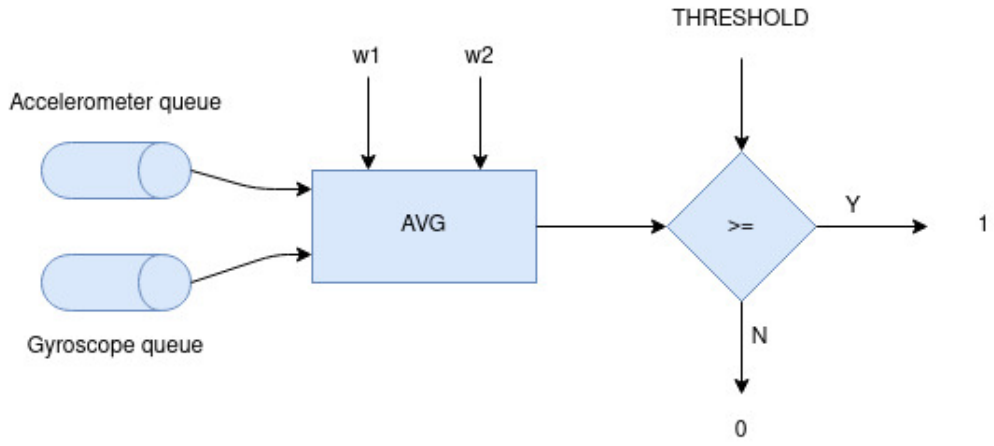

Figure 1.3: Classification scheme

The weighted average is computed with weights $\omega_1 = 0.96$ for the Accelerometer and $\omega_2 = 0.8$ for the Gyroscope, both corresponding to the accuracy of the classifiers according to *Weka*. The *THRESHOLD* is set to 7 and was obtained by trial & error.
At this point a problem arised: if a user started washing hands halfway through the queue, the list of classifications would be split in half between the current queue and the next resulting in a missed classification (false negative). To overcome this we introduced a 50% overlap between consecutive 10 second frames.
With this model our application was able to recognize handwashing in most tests with no false positives and minimal false negatives.

2. Android application

2.1 Prerequisites

Used resources

To development the application we used:

- Smartphone Samsung Galxy S5: used to collect the dataset;
- Weka: tool that allowed us to do preprocessing of the data that classification with many classifiers, we have chosen J48 decision tree classifier. The training is done automatically and an output file with the classifier structure is returned. This allows you to select the parameters to eliminate the error due to the data. It already has usable java functions. We can do both training and testing. Once a model has been created, it is possible to load and enter new data. Once our classifier is ready we insert it in the java code of our application;
- Android Studio: used to implement the application, inserting the handwash recognition and classification forms, the forms for receiving the return home message and for sending the handwashing message.

The classifiers

To choose the classifier we used the Weka tool. This produced us a binary J48 decision tree classifier aimed at recognizing hand washing. There are two classifiers used, one to recognize the data produced by the accelerometer and one to recognize the data produced by the gyroscope. Both were trained with a dataset of data collected by an accelerometer and gyroscope during a real hand washing session.

2.2 Communication between modules

Our part of the application is embedded within the global application. The CovidB application includes many features. Our module, in particular, is responsible for checking whether the user has washed his hands once he returned home. Our application interacts with the remaining part in two ways.
In the first phase, through the broadcast receiver module, the application waits to receive the *homeproximity* intent which indicates that the user is near his home. The fired intent will have an extra boolean added with key *KEYPROXIMITYENTERING*. If the value is true, the user is entering the proximity region; if false, it is exiting. In the first case our application begins its control phase to detect hand washing, this is done from a .csv file.
If in the first fifteen minutes, starting from the time of user's homecoming, the hand washing activity is detected, then an *handWashed* intent is generated and sent in brodcast. This intent is captured by the module that takes care of calculating the user's risk index. If the user does not wash his hands, no intent is sent.

Listing 2.1: onReceive

```
1  public void onReceive(Context context, Intent intent) {
2
3      Log.d("MSG","Intent received: "+intent.getAction());
4
5      if(intent.getAction().equals("home_proximity")) {
6          //User near home
7          Toast.makeText(context, "home_proximity intent received", Toast.
               LENGTH_LONG).show();
8
9          //KEY_PROXIMITY_ENTERING used to discriminate whether the user
               is returning home or leaving home
10         key = LocationManager.KEY_PROXIMITY_ENTERING;
11         entering = intent.getBooleanExtra(key, true);
12
13         //Case user is returning home
14         if(entering) {
15         Log.d("TEST","User is back home");
16             Toast.makeText(context, "User is back home", Toast.
                   LENGTH_LONG).show();
17             try {
18                 //read csv file (first 15mins) and classify
19                 BufferedReader br = new BufferedReader(new FileReader(
                       accelFile));
20                 BufferedReader br2 = new BufferedReader(new FileReader(
                       gyrFile));
21                 String separator = ",", line = br.readLine(), line2 =
                       br2.readLine();
22
23                 int linesRead = 0;
24                 //90000 lines = 15mins worth of data for both sensors
                       (45000 lines each)
25                 while((linesRead < 90000) && line != null && line2 !=
                       null) {
26                     String[] coord = line.split(separator); accelData.
```

```
                    add(Double.parseDouble(coord[0]), Double.
                    parseDouble(coord[1]), Double.parseDouble(coord
                    [2]));

                coord = line2.split(separator);
                gyrData.add(Double.parseDouble(coord[0]), Double.
                    parseDouble(coord[1]), Double.parseDouble(coord
                    [2]));

                if ( accelData.getNumData() == WINDOW_SIZE)
                {
                    accelData.computeFeatures();
                    try {
                        Double result = WekaAccelClassifier.classify
                            (accelData.getFeatures());

                        //inserting at the end, popping the first
                            element
                        accelVector.add(result);
                        accelVector.remove(0);
                        accelInserts++;

                        if(accelInserts == OVERLAP*MIN_ACTIVITY_TIME
                            ) {
                            double s1 = 0, s2 = 0;
                            for(double num: accelVector)
                                s1 = s1+num;
                            for(double num: gyroVector)
                                s2 = s2+num;

                            if( ( (s1*ACC_WEIGHT)+(s2*GYR_WEIGHT) )
                                /2 >= THRESHOLD ) {
                                //send intent
                                    sendIntent(context);
                                break;
                            }
                            accelInserts = 0;
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    accelData.flush();
                }
                if(gyrData.getNumData() == WINDOW_SIZE) {
                    gyrData.computeFeatures();
                    try {
                        Double result = WekaGyroClassifier.classify(
                            gyrData.getFeatures());

                        //inserting at the end, popping the first
                            element
                        gyroVector.add(result);
```

```
68                              gyroVector.remove(0);
69                              gyroInserts++;;
70
71                              if(gyroInserts == OVERLAP*MIN_ACTIVITY_TIME)
                                    {
72                                      double s1 = 0, s2 = 0;
73                                      for(double num: accelVector)
74                                      s1 = s1+num;
75                                      for(double num: gyroVector)
76                                      s2 = s2+num;
77
78                                  if( ( (s1*ACC_WEIGHT)+(s2*GYR_WEIGHT) )
                                        /2 >= THRESHOLD ) {
79                                      //send intent
80                                          sendIntent(context);
81                                      break;
82                                  }
83                                  gyroInserts = 0;
84                              }
85                          } catch (Exception e) {
86                              e.printStackTrace();
87                          }
88                          gyrData.flush();
89                      }
90                      line = br.readLine();
91                      line2 = br2.readLine();
92                  }
93              } catch(IOException e) { e.printStackTrace(); }
94          } else {
95              //User is leaving home, I look forward to his homecoming
96              Log.d("TEST","User is leaving home");
97              Toast.makeText(context, "User is leaving home", Toast.
                    LENGTH_LONG).show();
98          }
99      }
100     else {
101         //Received intent not of interest
102         Log.d("TEST","Some other intent received");
103         Toast.makeText(context, "Some other intent received", Toast.
                LENGTH_LONG).show();
104     }
105 }
```

3. Testing

At the end of the development of our application, we tested it using both the data obtained by the smartphone's sensors and with the data taken as input from the *csv* file.
To test our module we developed a simple independent application.
First we verified the correctness of the classifier using a one-hour session of normal activities. During the hour, the user washed his hands 3 times. The application recognized 27 washes as the classifier samples once per second, so the 10 second sliding windows with the constraints explained in the 1.1.7 chapter have been added. We tested the application again and 4 washes were recognized. At this point we have added the constraint for the recognition of subsequent washings: the handwashing counter can only increase if there has been a window that has not recorded the wash. At the end, the application has correctly recognized a total of 3 hand-washes without false positives or false negatives.

3.1 Handwash recognition

In the smartphone, the result can be viewed via the display. If classification is successful, the screen of the smartphone is turned to green, else to red. The two classification vectors (accelerometer and gyroscope) can be seen on the screen as well.

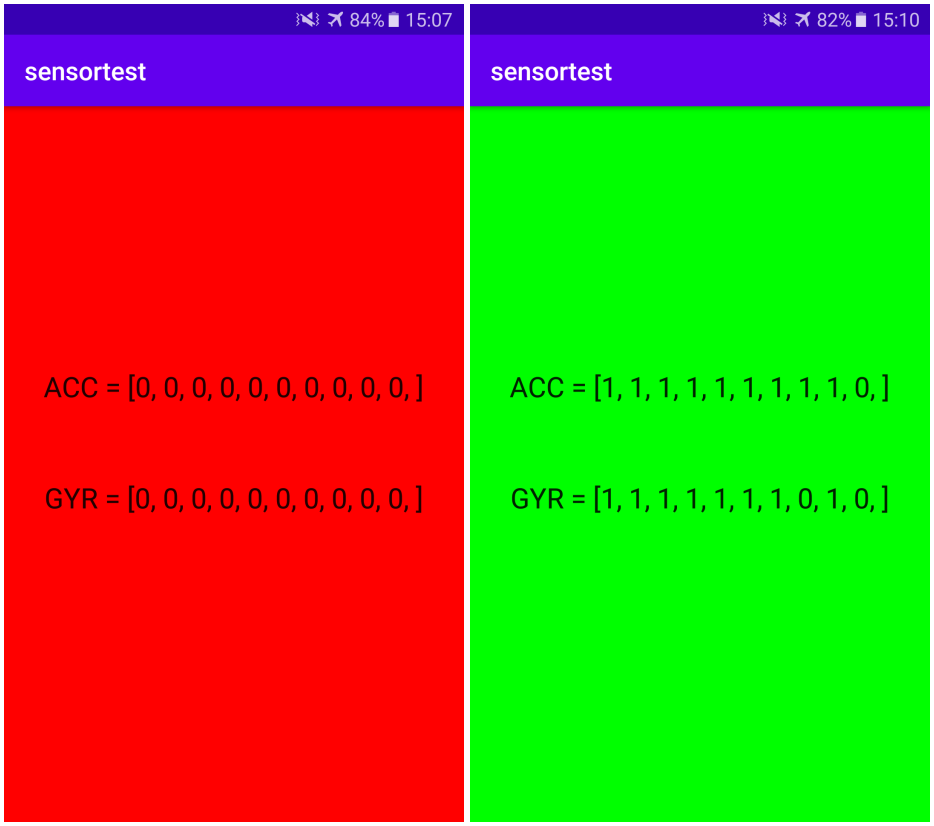

Figure 3.1: No handwashing - Recognized handwashing

3.2 Broadcast Receiver

We subsequently checked the correctness of the Broadcast Receiver implementation by generating an *homeproximity* intent which emulates the user returning home. This time, if classification is successful an *handWashed* intent is sent in broadcast to notify this, otherwise no action is performed.

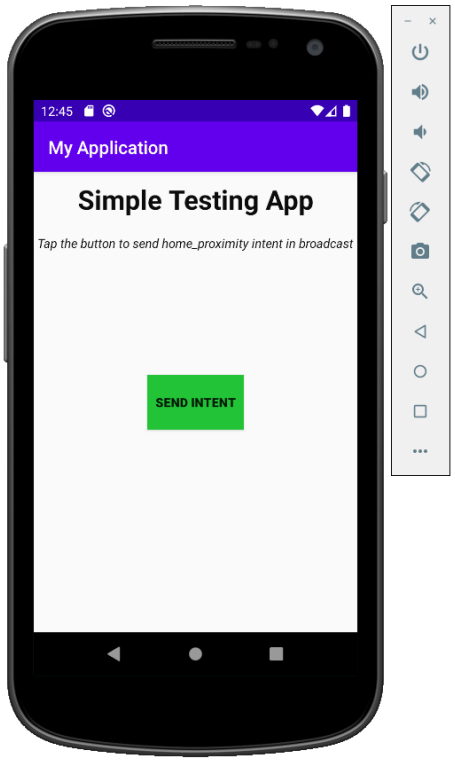

Figure 3.2: Home proximity Intent

4. Conclusion

The application has been tested on a smartphone worn like a smartwatch in the wrist. Moreover, the classificator can be "fooled" by performing fake hand wash motions. This can be avoided by adding audio data as a future work.
Although the application has some limitations, mainly due to the fact that we did not have a smartwatch to run it in, we can be satisfied with the work done since the main goal of the project was achieved with a very good level of accuracy.