Activity Prediction on Mobile Devices

Dominic DeMarco
Clarkson University
Department of Computer Science

Abstract

In the modern era, nearly everyone carries with them devices loaded with sensors and advanced processors, and often boast 24/7 network connectivity. However, most of the time these devices are underutilized, spending most of their lives idling in the pockets or bags of the everyday person. Proposed here is a technique for activity prediction based on a wide array of sensors on a mobile phones, and the ability to request the current activity of it’s user over a network. This is a potentially useful technology for any application that needs the current status of a primary actor, such as Human-Robot interaction problems, or activity logging for health care purposes.

1 Introduction

The smart phone is an ideal platform to base activity recognition off of. It boasts a wide array of sensors, and is often carried most places by its owner. All that remains is to make use of the sensor data provided by these mobile sensor platforms.

The Android platform was chosen as a base for this system due to it’s widespread use in today’s world. An ever increasing majority of the smart phones sold every year run Google’s Android. Not only is Android the vast majority, but it is an open source operating system with a freely available SDK and IDE. All the code displayed in this paper is available at [2].

2 Random Forests

The machine learning algorithm used is the Random Forest. Random forests were first proposed in 2001 by Leo Breiman[1]. A random forest is created by creating an arbitrary amount of decision trees. The trees are created from a random set of attributes contained within the training set, and report either a “true” or a “false” when a data point is passed through a trained version of it. The reason this algorithm was chosen was not only could it produce accurate classifications
of categorical sets of data, but also provide a confidence in that classification. This allows for the possibility of actions based on a classifications confidence, such as defining a new activity or reinforcing an existing activity with new data.

3 Handling Sensor Input

Figure 1: An illustration representing the makeup of a random forest. Each decision tree is created from a random set of attributes of the training set.

Android’s sensors cannot be queried in order to obtain data, they instead must be subscribed to. This ensures that your program will receive new sensor readings whenever the driver that manages them decides to update them. Keeping in mind this data is received with the intent of classification, there needs to be a point of data for each sensor before we can run the classification algorithm. In order to achieve the goal of sensor for each data, a cyclic system is introduced with the purpose of gathering data over a period of time. The system is illustrated in Figure 2, the operation of this system isn’t complex. Sensors are read into a temporary buffer where they are averaged over a period of time. This helps to ensure that data is collected from every sensor. In the event that data is not collected from every sensor (light reading in a pitch black room, for example), previous data is imported to take it’s place. Once the data is collected and averaged, it is fed into a trained Random Forest, and the resulting classification is output on the screen. This cyclic system can accommodate the introduction of additional sensors if need be, allowing classification on a wide range of sensor inputs.

4 Data Gathering

In order to train a machine learning algorithm, training data is needed. For the purposes of this application, providing pre-recorded training data wouldn’t lead to accurate results. Smart phones and other sensor-enabled devices vary in design and construction, meaning that the output of one devices
sensor won’t be identical to another device’s. This variation in hardware means a method of recording sensor data with the goal of classification is necessary. To streamline the process of recording different activities, each activity’s dataset is stored in a separate file. When the SensorRecord activity is called, sensor data is retrieved from the system outlined in Figure 2 and is stored on the non-volatile memory of the device. The file format conforms to the WEKA .arff file format. This allows for further analysis using the WEKA toolkit.

5 Classification

Once data has been recorded by the user, it can be classified by the Random Forest algorithm. Before the forest can be trained, the sensor data needs to be merged. As explained in Section 4, the sensor data for each activity is recorded as a separate file. When the classification activity is initially loaded, it creates...
a new dataset and populates it with the contents of all the training datasets stored on the device, and then proceeds to train the algorithm. Note the three buttons at the top of the activity screen in Figure 4. Once the “start” toggle switch is activated, the program will proceed to gather sensor data from the cyclic listening system and classify the sensor data it gathers. The classification is output to the log of the classification activity.

6 Networking

A primary purpose of this project is to enable outside devices to observe the activities being classified on a mobile device running this application. Due to the network restrictions imposed by mobile network carriers, without a Virtual Private Network or some other form of traffic obfuscation it isn’t feasible to publish this data over a cellular network. Instead we must rely on the much simpler network layers of a local wireless network. When attached to a wireless network, the user can select the “Start Server” toggle switch to activate a simple TCP server that will reply to any received packets with the most recent activity classification.

7 Results

Under optimal conditions, this system is able to accurately and consistently classify the correct action the user is performing. However it does suffer from the two main caveats that plague most classification algorithms. These are the loss of precision with the increase in possible actions. The more possible activities the random forest needs to consider, the less confident it typically is in it’s classification. This is particularly true with activities that involve a lot of motion. Because the sensor data is averaged over a period of time, activities containing a large amount of varied accelerometer data will produce very similar numbers at the end of the sensor cycle. As a result the Random Forest will not only suffer from decreased accuracy given a higher number of activities, it will also have to deal with a useless data point when attempting classification.

Large sample sizes are necessary when classifying on multiple sources of data. Few data points to train on will not only effect the quality of the classification on an individual level, it will cause the activity classifications to be erratic and somewhat unpredictable. The more samples the Forest has to train against, the smoother and more accurate it’s classification of multiple activities will be.

8 Possible Extensions

The code for this project was purposefully left extensible to ease possible implementation in the future. Most of the restrictions were the complexities of Android programming, in the future more customization could be added via selection menus in the Android activities, for example custom sensor selection and sensor cycle duration selection.

Another possibility could be the implemen-
tation of network security. As it stands the web server opens a TCP socket and waits for incoming connections. Once a connection is received it will respond with the last classification. A simple security option would be to query the user to either accept or deny the connection based on the MAC address.

A direction to take that would increase the accuracy of the classification would be the introduction of pre-classification algorithms into the sensor cycle. This would allow notoriously difficult sensor information like accelerometers to be processed into more usable data. For example, running an algorithm to detect the frequency of oscillations in the sensor data would transform a useless attribute into the backbone of a classification.

References

