Exercise PC Systems Development & Infrastructure	JKU Linz
2012W, Exercise 4	Institut für Pervasive Computing
Return till: Wed, 12.Dec 2012, 12:00	Gerold Hölzl, Michael Matscheko

After having discussed and analyzed several classifiers, we will use them in a real-time scenario. Since Weka is written entirely in Java, most of its classes can be used without any modifications on the Android platform. Some parts, especially everything related to the Weka GUI will not work and needs to be removed. We already prepared a reduced Weka 3.6.6 package (stable version), that can be compiled for Android. All code that was interfering was simple removed, which means some classes will not work as intended or throw unexpected Exceptions. You should not have any problems with the core functions (e.g. loading of ARFF files) and classifiers, but still, you should be aware that this reduced Weka package is incomplete and rather unstable.

1) Using Weka classifiers on Android (15 points)

Create a new Android project for this exercise. Download `weka-3.6.6-android.jar` from the department homepage and put it in a project subfolder called "`lib`". Add `weka-3.6.6-android.jar` to your build path (Project Properties > Java Build Path > Libraries > Add JARs).

Before you continue, you should take a look at the Weka documentation (http://weka.sourceforge.net/doc/):

- `weka.core.Instance`: Class for handling an instance. Only contains plain data.
- `weka.core.Instances`: Class for handling a set of instances. Also contains the instance structure/description.
- `weka.classifiers.Classifier`: Abstract classifier superclass.

Check the slides for examples on how to use these classes. There are also several tutorials online at http://weka.wikispaces.com/.

Training Dataset

To train a classifier on your phone, you will need a training data set. This is just a reduced data set of those instances, that you used in exercise 3 to train your optimal classifier. Load the complete data set, remove all features that you do not need and save them in ARFF format (first Tab of the Weka explorer).

Android Application

Implement a Service / Activity combination similar to the ARFF Recorder in exercise 1: The Activity just provides a GUI to control the Service. The Service does all the work:

- On startup create the classifier you think is working best (one of the 4 classifiers from exercise 3, please do not use others). Then train your classifier with the training data set
- Get raw data from the accelerometer
- Implement a sliding window with the window size you determined to be optimal
- Calculate the features for each window. Do not calculate all the features that we discussed in exercise 2, just those that you need for classification
- Put the calculated features in a feature vector – this is an instance of `weka.core.Instance`
- Use the trained classifier to determine the class of the new instance
- Output the class label somehow, either on the screen, in the log file or via text-to-speech …

In case you cannot compile and deploy your project (some out of memory error), you may need to increase the Eclipse memory limits. Open `eclipse.ini` in your Eclipse program folder and increase all the memory values (e.g. `-Xms…m` and `-Xmx…m`) to something a lot larger (e.g. 1024m).

2) Coordination Server (5 points)

For further evaluation we will collect activity information from all exercise participants at a central location, the coordination server. It simply stores the current activity of each client and also provides an interface to retrieve the state of the whole group.

Download `coordinationclient.zip` from the department homepage and extract all java files to your project folder. The zip file contains more classes, than will be needed; the rest will be discussed in the next exercise.

- CoordinatorClient.java: Connects to the coordination server and transmits your current activity.
- ClassLabel.java: An enumeration of all class labels / activities (walking, standing, sitting). `null` is also valid (null-class)!

You may change the package names, but it should not be necessary to change anything in the classes!

Use the CoordinatorClient to transmit your current activity to the coordination server. The server address is already hard-coded into the client class (netadmin.soft.uni-linz.ac.at:8891). The constructor only expects a client ID to authenticate yourself: use your Matrikelnummer. You will not be able to connect with anything else than a 7-digit ID! The client will immediately try to connect upon instantiation. After that, you can update your current activity on the server by a call to setCurrentActivity(ClassLabel label).

Unless you have a SIM in your phone and data access anyway, you should use the next available WLAN to test your application. Everyone will have to connect during the next lesson, so we recommend testing it at least once with the wireless network of the university.

Make sure the application uses the permission `android.permission.INTERNET` in the manifest; otherwise you will not be able to send data.

To test if the server is receiving your activity updates correctly, start CoordinationServerGui.jar. It will connect to the coordination server and display all participants that have connected at least once and visualize their current state.

Guidelines:

- Zip your eclipse project folder and upload it.
- Your source code has to include comments where appropriate:
  - Before methods explain in 1 to 2 sentences what the method does, unless it is a simple utility method and it is obvious anyway.
  - Inside methods, add short comments (only a few words) before semantically separate statement blocks.
- Add a `readme` file! It should contain your group members, a summary of what you did (just a few sentences) and any problems you had.
- We have a forum at https://www.pervasive.jku.at/Forum/. If you need help, use it!