CSE 118, 218. Fall 2017. Class projects.

Your class project application

In your project, you will design, implement, and evaluate an application that uses context-recognition. Your application can use other tools/devices as well. An elegant way would be to implement an Android phone application (which reads the label prediction files delivered by the ExtraSensory App on the same phone), but you may also use some other platform and copy the saved label prediction files from the phone to use them offline.

1. Your application should include:
   - **Graphical visualization**. Summarize the recognized context in an informative and appealing way to the user. You can aggregate label-probabilities over minutes (e.g. average probabilities over a day in order to estimate the portion of time spent sitting, or in a car, or eating, etc.). You can perform time-segmentation: deciding how to segment the sequence of recorded minutes into distinct segments of time. Visually you can use bar plots, pie charts, numeric tables/lists, interactive interfaces, or whatever creative idea you have to convey the summarized information to the user in a useful, friendly, and fun way.
     ExtraSensory App can also save location coordinates for every minute (if GPS was available), so you can also incorporate a map view and display the path of the user on the map, perhaps make it interactive, etc.
     Any original ideas for visualization are welcome.
   - **Added value**. Your app should provide some "service" – something helpful/useful/desirable/entertaining (otherwise, why would anyone use it). The context-recognition component should have a role in the overall application.
     The added value can be done explicitly within the code of your app (e.g. when your app detects some drastic change of context, it can activate some function or alert the user). It can also be done implicitly: for example the user can use the recognized (and visualized) context to self-monitor their "progress", while another component/device/tool of your project is used to affect the user's behavior.

2. You should **evaluate** your application, by having people actually use it. One basic purpose of evaluation is to make sure that your software/hardware solutions work smoothly without any bugs. In addition, you should report about the user-experience of your application and how it makes an effect/service/utility to the users. Of course, during the course, there will be not enough time for a thorough evaluation, so you can test your application with few users, for a short time period, possibly with simulations of the relevant situations, and provide preliminary evaluation as a very basic proof of concept. You can even start with synthetic context-recognition: pretend that ExtraSensory provided relevant predictions and test how your app would react to such predictions (only later, try to use actual ExtraSensory predictions in real scenarios). If you manage to get more users, longer test-periods (e.g. over a whole week), or interesting results in the effect of your application, your project will be more impressive.

When developing your application, you can assume that ExtraSensory works perfectly (of course, in reality it has some noise and errors in recognition), and that it provides you with per-minute probability values for the following context-labels:

1. Lying down
2. Sitting
3. Standing
4. Walking
5. Running
6. Bicycling
7. Sleeping
8. Lab work
9. In class
10. In a meeting
11. At work
12. Indoors
13. Outside
14. In a car
15. On a bus
16. Drive – I'm the driver
17. Driver – I'm the passenger
18. At home
19. At school
20. At a restaurant
21. Exercising
22. Cooking
23. Shopping
24. Strolling
25. Drinking (alcohol)
26. Bathing – shower
27. Cleaning
28. Doing laundry
29. Washing dishes
30. Watching TV
31. Surfing the internet
32. At a party
33. At a bar
34. At the beach
35. Singing
36. Talking
37. Computer work
38. Eating
39. Toilet
40. Grooming
41. Dressing
42. At the gym
43. Stairs – going up
44. Stairs – going down
45. Elevator
46. Phone in pocket
47. Phone in hand
48. Phone in bag
49. Phone on table
50. With co-workers
51. With friends

Your particular application may focus on a subset of these labels, for example if it is relevant only in an office environment, or mostly at home, or if it mainly concerns body posture/movement states, etc.

Guiding questions to select/define a project application:

- What does the application do?
- What is the added value of the application? Why would anyone want/need to use it?
- Who will use the application? (it doesn't have to be for everyone)
- When will they use the application? (it can be dedicated to certain times of the day, in certain environments, etc.)
- What are the relevant contexts for this application?
- How will it work? How will it utilize the recognized contexts to provide the added value?

The rest of this document is dedicated to give you more details about the ExtraSensory App, how to use it (basically it should be running in the background and provide you per-minute context-recognition, but there are some technical things you should know), and how to read the context-recognition data from ExtraSensory into your own application.

The ExtraSensory App for real-time context-recognition

ExtraSensory is an Android-phone app that uses the phone's (and possibly a watch's) sensors to recognize the user's behavioral context (where they are, who they're with, what they're doing, etc.). You will use ExtraSensory as a black box tool and integrate its context-recognition service in your project's application.

1. **What ExtraSensory does**:

The ExtraSensory App is a mobile app for Android phones. When it runs in the background, it is scheduled to record a 20-second window of sensor-measurements every minute, and send them to a server. The sensors include accelerometer, gyroscope, location (based on GPS, cellular, and WiFi), audio, and phone-state indicators (app-state, WiFi connectivity, time-of-day, etc.). The server analyses the sensor-measurements, performs "context-recognition", and responds back to the phone (within a few seconds) with behavioral context predictions: it assigns a probability to each of 51 diverse context-labels (sitting, walking, washing dishes, sleeping, eating, at home, at school, etc.)

2. **Running ExtraSensory**:

Install the apk file that the course will supply. Simply open ExtraSensory.
Notice: even if you close the app's interface it will still run in the background, so if you want it to stop collecting data every minute, use the "data collection" switch on the home page (switch it to "off"), or force-close the app through the phone's settings. Every minute, during the 20-second recording window, a red circle will appear on the app's top action bar to indicate "recording now".

3. **Internet connectivity**:

If your application needs to use the context-recognition throughout the day (e.g., every minute, every 20 minutes), you need the ExtraSensory App to stay connected to the internet to communicate with the server. By default, ExtraSensory is set to only communicate via WiFi. This may be sufficient, if your application is designed for home or office (where you assume WiFi will be available). If the user has an unlimited cellular data-plan, you can switch on "allow cellular communication" on ExtraSensory's settings page, and then the app can communicate with the server even when the user is outside, commuting, etc.
If your application only uses the recognized context at the end of the day (or the end of a week), then you don't have to worry much about connectivity during the day – in that case, the measurements will be stored on the phone, until connectivity is available. The home page indicates how many examples (minutes) are waiting to be sent, and there's a button you can press when you are ready to send them (when you have internet connectivity).

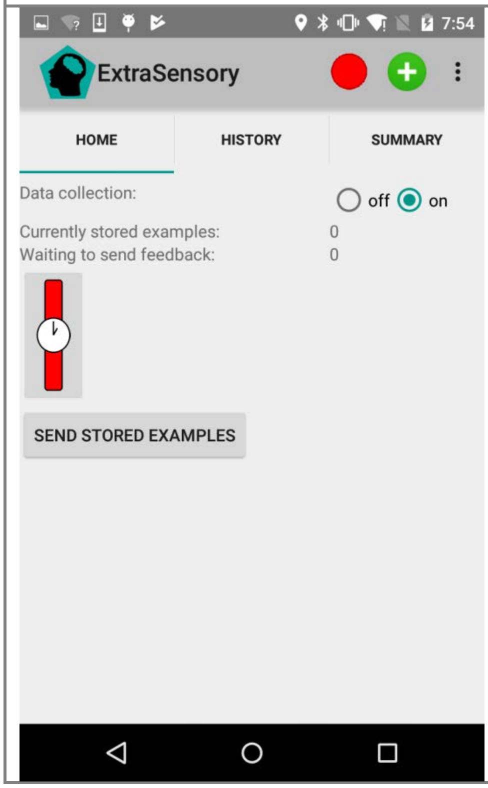
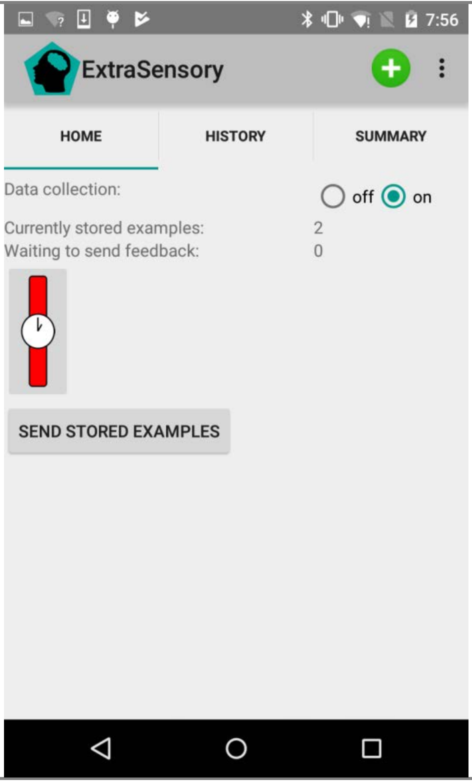
4. **Adding watch sensors (optional)**:

ExtraSensory integrates with a Pebble smartwatch to acquire accelerometer measurements from the wrist and improve recognition of many contexts (walking, computer work, shower, washing dishes, etc.). If you wish to use the watch as well:

- Get a Pebble watch. ☺ (we will supply a watch if you don't have one)
- Sign up to Cloud Pebble (https://cloudpebble.net).

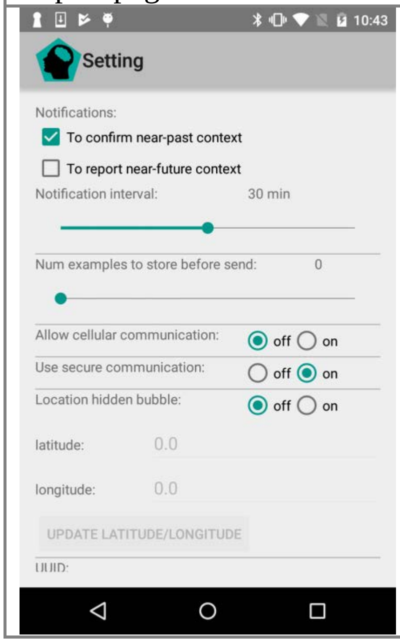
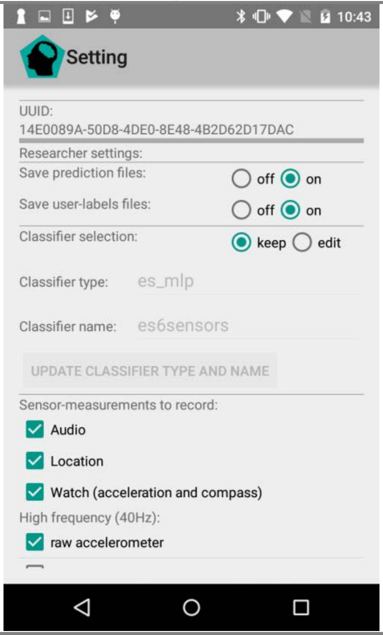
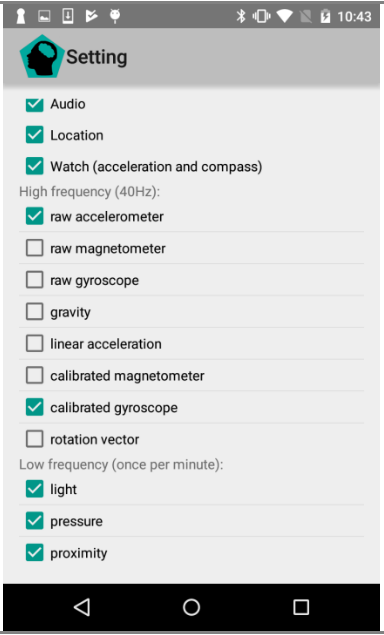
- On the user’s Android phone install the Pebble app from Google Play store. It is free. This is what enables communication (Bluetooth) between the phone and the watch.
- Enable Bluetooth on both the phone and watch.
- Log in to your Pebble account on the Pebble app on the phone.
- Pair the phone with the watch: you can pair it through the Pebble app.
- We will supply you with a pbw file – open it on the phone, to load the Pebble-side app (named ESW) on the watch.
- On ExtraSensory’s settings page, make sure to check the box for recording sensors from the watch. Also make sure the classifier name is “es6sensors” (to make sure the server-side uses a classifier that uses also the watch and not only the phone sensors).
- Whenever reopening ExtraSensory, or when switching on “data collection”, the ESW app should be launched on the watch. In addition, the watch icon on ExtraSensory’s home page is a pressable button that re-launches ESW on the watch, if needed.
- During every 20-second recording window, the word “REC” should appear on the bottom half of the watch face.

5. **The home page**:

Currently in 20-second recording window (red circle appears).	Not recording right now (no red circle), but will start recording within less than a minute. No internet connectivity, so storing recorded examples.
 7:54 ExtraSensory HOME HISTORY SUMMARY SEND STORED EXAMPLES Data collection: ○ off ● on Currently stored examples: 0 Waiting to send feedback: 0	 7:56 ExtraSensory HOME HISTORY SUMMARY SEND STORED EXAMPLES Data collection: ○ off ● on Currently stored examples: 2 Waiting to send feedback: 0

- By default, the "data collection" switch (radio buttons) is "on".
- The "currently stored examples" counter indicates how many minutes have sensor-measurements waiting to be sent to the server. When internet connectivity becomes available, the app may start automatically to send examples, or the user can press the "send stored examples" button.
- The watch icon appears gray when the phone is not paired with a watch and red when it is. It is not enough to have the watch-icon appear red – you also need ESW to be open on the watch, and sometimes you need to refresh the phone-watch connectivity by pressing the red watch-icon.
- The ExtraSensory App has additional features (the history tab, the summary tab, and the green plus button) that are useful for studies that collect self-reported labels from the users. These features are not so relevant for the class project (unless you suggest some brilliant idea for a project that involves users correcting the server predictions and providing their own "ground truth" context-labels).
- The top action bar has a menu (the three dots symbol) through which you can reach the settings page.

6. **The settings page**:

Top of page	Middle of page	Bottom of page
 10:43 Setting Notifications: ☑ To confirm near-past context ☐ To report near-future context Notification interval: 30 min Num examples to store before send: 0 Allow cellular communication: ● off ○ on Use secure communication: ○ off ● on Location hidden bubble: ● off ○ on latitude: 0.0 longitude: 0.0 UPDATE LATITUDE/LONGITUDE UUID:	 10:43 Setting UUID: 14E0089A-50D8-4DE0-8E48-4B2D62D17DAC Researcher settings: Save prediction files: ○ off ● on Save user-labels files: ○ off ● on Classifier selection: ● keep ○ edit Classifier type: es_mlp Classifier name: es6sensors UPDATE CLASSIFIER TYPE AND NAME Sensor-measurements to record: ☑ Audio ☑ Location ☑ Watch (acceleration and compass) High frequency (40Hz): ☑ raw accelerometer	 10:43 Setting ☑ Audio ☑ Location ☑ Watch (acceleration and compass) High frequency (40Hz): ☑ raw accelerometer ☐ raw magnetometer ☐ raw gyroscope ☐ gravity ☐ linear acceleration ☐ calibrated magnetometer ☑ calibrated gyroscope ☐ rotation vector Low frequency (once per minute): ☑ light ☑ pressure ☑ proximity

After installing the ExtraSensory App on a user's phone, make sure the settings fit your particular project-application:

- Notifications: uncheck both types of notifications. These notifications are designed for studies that collect self-reported labels from the users. In the class, we prefer to focus on using ExtraSensory only in the background as an automatic context-recognition service, so we don't want ExtraSensory to nag the user (although you may decide that the application that you design will alert/nag the user as you see fit).
- Allow cellular communication: if the user has unlimited cellular data-plan, you may turn this on (so that ExtraSensory gets real-time server predictions even when WiFi is not available). Otherwise, keep cellular communication off.
- Use secure communication: keep this on. This will use https protocol and the communication with the server will be encrypted.
- UUID (middle of page): this is a unique user identifier that is randomly generated when you first open the app on the user's phone. ExtraSensory will save the label predictions on files on the phone in a directory that has the first 8 characters of the UUID, so you need to look at this UUID, to know how to locate the label files.
- Save prediction files: keep this on. This will save the server-predicted labels on the phone's storage in a file-per-minute that your newly designed application can read.
- Save user-labels files: this is only relevant if you plan to have the users correct the predictions and provide their own labels, and to have your application read the user-corrected labels.
- Classifier selection: if you use also the watch sensors, make sure the "classifier name" field has "es6sensors". This will indicate to the server to apply a context-recognition classifier that uses both phone and watch sensors. This classifier can handle also cases where some sensors may be missing (watch temporarily removed, location services temporarily blocked, etc.). In case you don't plan to use the watch at all, you can change

the classifier name to “es5sensors” – this will indicate to the server to use a classifier that only uses phone-sensor measurements.

- Sensors to record: check the relevant sensors. Keep: Audio, Location, Watch (unless you don’t use the watch), raw accelerometer, and calibrated gyroscope. If, for your particular application or particular user, you want to avoid any sensor (e.g. protect privacy by not recording audio or location), you can uncheck some sensors, but this will cause the recognition to be less accurate.

7. **The predicted labels**:

ExtraSensory describes context with a multi-label setting (multiple labels can be relevant at the same time) and provides real-time “soft recognition” (probability values) for the following context-labels:

52. Lying down
53. Sitting
54. Standing
55. Walking
56. Running
57. Bicycling
58. Sleeping
59. Lab work
60. In class
61. In a meeting
62. At work
63. Indoors
64. Outside
65. In a car
66. On a bus
67. Drive – I’m the driver
68. Driver – I’m the passenger
69. At home
70. At school
71. At a restaurant
72. Exercising
73. Cooking
74. Shopping
75. Strolling
76. Drinking (alcohol)
77. Bathing – shower
78. Cleaning
79. Doing laundry
80. Washing dishes
81. Watching TV
82. Surfing the internet
83. At a party
84. At a bar
85. At the beach
86. Singing
87. Talking
88. Computer work
89. Eating
90. Toilet
91. Grooming
92. Dressing
93. At the gym
94. Stairs – going up
95. Stairs – going down
96. Elevator
97. Phone in pocket
98. Phone in hand
99. Phone in bag
100. Phone on table
101. With co-workers
102. With friends

8. **Using the predicted labels in your own Android app**:

Your project will involve design, implementation, and evaluation of some application (typically, a mobile Android-phone application) that utilizes ExtraSensory’s context-recognition service. ExtraSensory saves the server-predictions to files that are accessible to everyone (other apps on the phone or a computer connected with USB). It will create a designated directory in the device’s storage (from ExtraSensory’s perspective it is external storage, but it may be referred to as “internal storage” when viewed in a file explorer from a computer). The created directory will be “Android/data/edu.ucsd.calab.extrasensory/files/Documents/extrasensory.labels.[uuid_prefix]/”, where “[uuid_prefix]” represents the first 8 characters of the UUID of the ExtraSensory user. Within this directory ExtraSensory will save a separate JSON file for each example (minute), named with the timestamp of the example (standard “seconds since the epoch” timestamp). Each file will hold the list of labels and list of corresponding probabilities that were predicted by the server for this example.

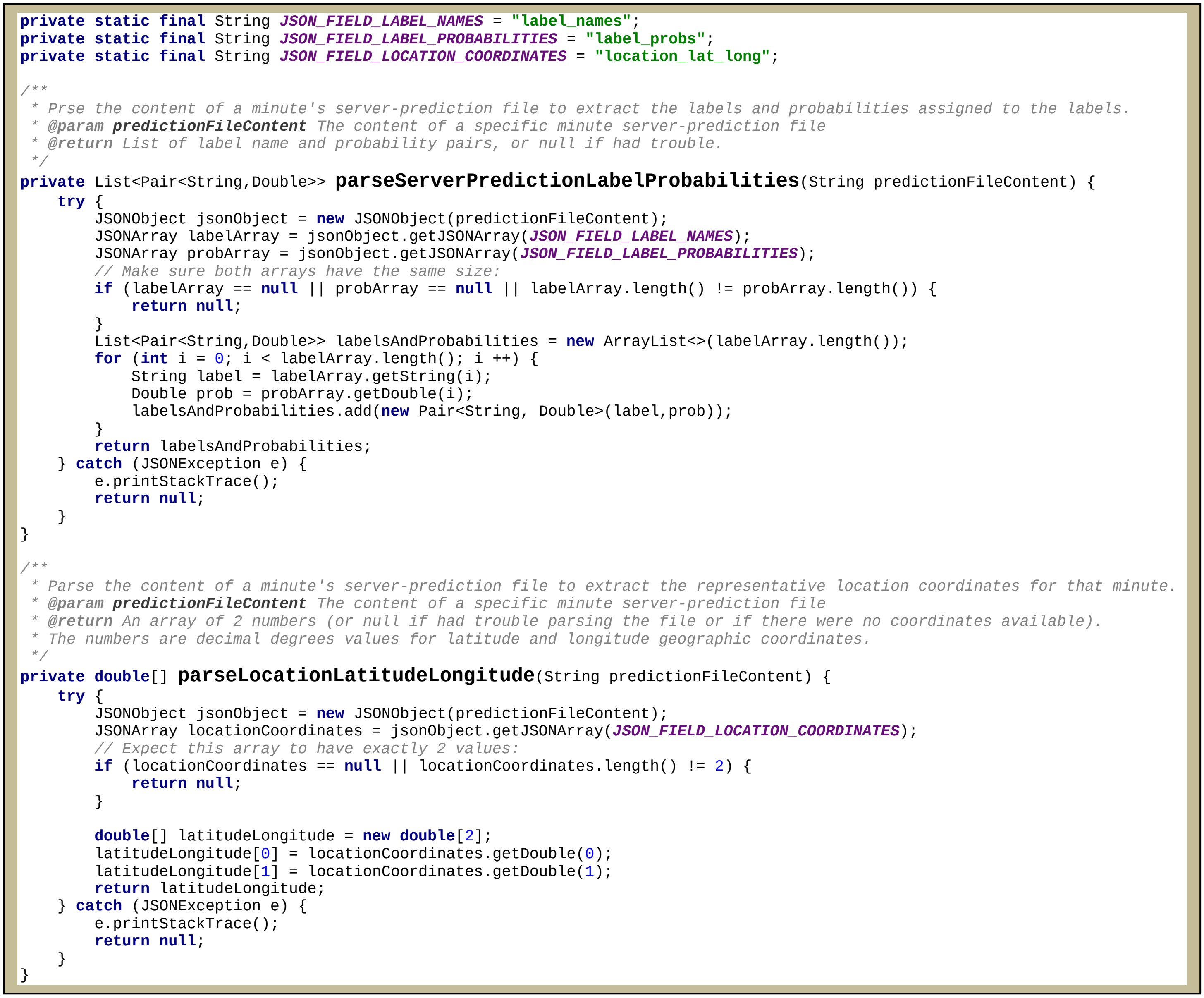
These files can be viewed from a USB connected computer, as well as from another Android app (like the app that you will design for your project). In your own Android app, you you need to declare permission to read external files – in your app's manifest, include the line:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

In the Java code of your app, you can read the files saved by ExtraSensory, by using functions like the following code:

```java
private static final String SERVER_PREDICTIONS_FILE_SUFFIX = ".server_predictions.json";
private static final String USER_REPORTED_LABELS_FILE_SUFFIX = ".user_reported_labels.json";

/**
 * Return the directory, where a user's ExtraSensory-App label files should be
 * @param uuidPrefix The prefix (8 characters) of the user's UUID
 * @return The user's files' directory
 * @throws PackageManager.NameNotFoundException
 */
private File getUserFilesDirectory(String uuidPrefix) throws PackageManager.NameNotFoundException {
    // Locate the ESA saved files directory, and the specific minute-example's file:
    Context extraSensoryAppContext = getApplicationContext().createPackageContext("edu.ucsd.calab.extrasensory",0);
    File esaFilesDir = new
File(extraSensoryAppContext.getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS),"extrasensory.labels." + uuidPrefix);
    if (!esaFilesDir.exists()) {
        return null;
    }
    return esaFilesDir;
}

/**
 * Get the list of timestamps, for which this user has saved files from ExtraSensory App.
 * @param uuidPrefix The prefix (8 characters) of the user's UUID
 * @return List of timestamps (strings), each representing a minute that has a file for this user.
 * The list will be sorted from earliest to latest.
 * In case the user's directory was not found, null will be returned.
 */
private List<String> getTimestampsForUser(String uuidPrefix) {
    try {
        File esaFilesDir = getUserFilesDirectory(uuidPrefix);
        if (esaFilesDir == null) {
            return null;
        }
        String[] filenames = esaFilesDir.list(new FilenameFilter() {
            @Override
            public boolean accept(File file, String s) {
                return s.endsWith(SERVER_PREDICTIONS_FILE_SUFFIX) || s.endsWith(USER_REPORTED_LABELS_FILE_SUFFIX);
            }
        });

        SortedSet<String> userTimestampsSet = new TreeSet<>();
        for (String filename : filenames) {
            String timestamp = filename.substring(0,10); // The timestamps always occupy 10 characters
            userTimestampsSet.add(timestamp);
        }

        List<String> userTimestamps = new ArrayList<>(userTimestampsSet);
        return userTimestamps;
    }
    catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

```java
/**
 * Read text from the label file saved by ExtraSensory App, for a particualr minute-example.
 * @param uuidPrefix The prefix (8 characters) of the user's UUID
 * @param timestamp The timestamp of the desired minute example
 * @param serverOrUser Read the server-predictions if true, and the user-reported labels if false
 * @return The text inside the file, or null if had trouble finding or reading the file
 */
private String readESALabelsFileForMinute(String uuidPrefix,String timestamp,boolean serverOrUser) {
    try {
        File esaFilesDir = getUserFilesDirectory(uuidPrefix);
        if (esaFilesDir == null) {
            // Cannot find the directory where the label files should be
            return null;
        }
        String fileSuffix = serverOrUser ? SERVER_PREDICTIONS_FILE_SUFFIX : USER_REPORTED_LABELS_FILE_SUFFIX;
        File minuteLabelsFile = new File(esaFilesDir,timestamp + fileSuffix);

        // Read the file:
        StringBuilder text = new StringBuilder();
        BufferedReader bufferedReader = new BufferedReader(new FileReader(minuteLabelsFile));
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            text.append(line);
            text.append('\n');
        }
        bufferedReader.close() ;
        return text.toString();

    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```



```java
private static final String JSON_FIELD_LABEL_NAMES = "label_names";
private static final String JSON_FIELD_LABEL_PROBABILITIES = "label_probs";
private static final String JSON_FIELD_LOCATION_COORDINATES = "location_lat_long";

/**
 * Prse the content of a minute's server-prediction file to extract the labels and probabilities assigned to the labels.
 * @param predictionFileContent The content of a specific minute server-prediction file
 * @return List of label name and probability pairs, or null if had trouble.
 */
private List<Pair<String,Double>> parseServerPredictionLabelProbabilities(String predictionFileContent) {
    try {
        JSONObject jsonObject = new JSONObject(predictionFileContent);
        JSONArray labelArray = jsonObject.getJSONArray(JSON_FIELD_LABEL_NAMES);
        JSONArray probArray = jsonObject.getJSONArray(JSON_FIELD_LABEL_PROBABILITIES);
        // Make sure both arrays have the same size:
        if (labelArray == null || probArray == null || labelArray.length() != probArray.length()) {
            return null;
        }
        List<Pair<String,Double>> labelsAndProbabilities = new ArrayList<>(labelArray.length());
        for (int i = 0; i < labelArray.length(); i ++) {
            String label = labelArray.getString(i);
            Double prob = probArray.getDouble(i);
            labelsAndProbabilities.add(new Pair<String, Double>(label,prob));
        }
        return labelsAndProbabilities;
    } catch (JSONException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * Parse the content of a minute's server-prediction file to extract the representative location coordinates for that minute.
 * @param predictionFileContent The content of a specific minute server-prediction file
 * @return An array of 2 numbers (or null if had trouble parsing the file or if there were no coordinates available).
 * The numbers are decimal degrees values for latitude and longitude geographic coordinates.
 */
private double[] parseLocationLatitudeLongitude(String predictionFileContent) {
    try {
        JSONObject jsonObject = new JSONObject(predictionFileContent);
        JSONArray locationCoordinates = jsonObject.getJSONArray(JSON_FIELD_LOCATION_COORDINATES);
        // Expect this array to have exactly 2 values:
        if (locationCoordinates == null || locationCoordinates.length() != 2) {
            return null;
        }

        double[] latitudeLongitude = new double[2];
        latitudeLongitude[0] = locationCoordinates.getDouble(0);
        latitudeLongitude[1] = locationCoordinates.getDouble(1);
        return latitudeLongitude;
    } catch (JSONException e) {
        e.printStackTrace();
        return null;
    }
}
```

In addition, ExtraSensory App sends broadcast messages whenever it saves server-prediction files, so you can register a broadcast receiver in your new app, to listen to the specific intent-action from ExtraSensory: **`"edu.ucsd.calab.extrasensory.broadcast.saved_prediction_file"`**. This can be useful in case your application is designed to detect behavioral changes in real time (within approximately a minute). You can use code similar to the following:

```
private static final String LOG_TAG = "[Using ESA]";
public static final String ESA_BROADCAST_SAVED_PRED_FILE = "edu.ucsd.calab.extrasensory.broadcast.saved_prediction_file";
public static final String ESA_BROADCAST_EXTRA_KEY_TIMESTAMP = "timestamp";

private BroadcastReceiver _broadcastReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (ESA_BROADCAST_SAVED_PRED_FILE.equals(intent.getAction())) {
            String newTimestamp = intent.getStringExtra(ESA_BROADCAST_EXTRA_KEY_TIMESTAMP);
            Log.d(LOG_TAG,"Caught broadcast for new timestamp: " + newTimestamp);
            // Do whatever you want with this new information…
        }
    }
};

@Override
public void onResume() {
    super.onResume();
    Log.d(LOG_TAG,"registring for broadcast: " + ESA_BROADCAST_SAVED_PRED_FILE);
    this.registerReceiver(_broadcastReceiver,new IntentFilter(ESA_BROADCAST_SAVED_PRED_FILE));
}

@Override
public void onPause() {
    this.unregisterReceiver(_broadcastReceiver);
    Log.d(LOG_TAG,"Unregistered broadcast: " + ESA_BROADCAST_SAVED_PRED_FILE);
    super.onPause();
}
```