

The ExtraSensory App

User and researcher guide



Content:

A.	What is the ExtraSensory App.....	3
B.	Credit.....	4
C.	User guide	5
1.	What the app does (in short)	5
2.	What the app does in the background	5
i.	Using the network.....	5
ii.	Location “bubble” around your home	6
3.	Closing the app.....	6
4.	User permissions.....	6
5.	The red dot.....	6
6.	The main page.....	7
7.	The home page	7
8.	The history page.....	8
i.	Today’s context log	8
ii.	Switch day	9
iii.	Edit labels for an event	9
iv.	Split an event to minute-by-minute.....	10
v.	Merge activities.....	12
9.	The summary page.....	14
10.	The “Active Feedback” button	14
11.	The settings page	14
12.	The feedback page	17
13.	The label selection page.....	18
i.	Mark labels for selection.....	19
ii.	Unmark labels for selection	19
iii.	Index.....	19
iv.	Frequently used labels	19
v.	Server guess	20
vi.	“Done” selecting labels from list.....	20
14.	Notifications and alerts	21

i.	2 types of questions.....	21
ii.	Notification when the app is in the background.....	22
iii.	Alert when the app is in the foreground	22
15.	Smartwatch.....	23
D.	Researcher guide	25
1.	The app components	25
2.	ExtraSensory Server (ESS)	25
i.	Installing and running ESS.....	25
ii.	Preparing for secured communication	29
iii.	The collected data.....	31
iv.	Troubleshooting and useful commands:	31
v.	Customizing the classifier	31
3.	ExtraSensory Phone (ESP)	33
i.	Linking to your own server.....	33
ii.	Customize your study.....	34
iii.	Using labels with another app	35
4.	ExtraSensory Watch (ESW)	41
i.	Prerequisites	41
ii.	Installing a compiled version	41
iii.	Cloud Pebble	41

A. What is the ExtraSensory App

The ExtraSensory App is a mobile application designed for research of behavioral context recognition. It was first used to collect the ExtraSensory Dataset (<http://extrasensory.ucsd.edu>), which includes data from over 300k minutes from 60 participants in-the-wild, meaning during their regular life. The app automatically collects sensor-measurements from the various sensors on the phone (and additional sensors on a smartwatch) and sends these measurements to a server for collection. In addition, the mobile app enables the users to provide their own ground truth labels about their context (what they were doing, where they were, who they were with, etc.). In order to maintain the behavior as natural and uninterrupted as possible, the app provides different mechanisms for the users to self-report their context (reporting past context, immediate future context, and more). The server side of the app generates real-time predictions/guesses of the context based on the sensor measurements sent from the phone.

The ExtraSensory App is very flexible can be used for research in many forms, for example:

- Collection of sensor-measurements and context-labels (supervised data), similar to the ExtraSensory Dataset. You can use the ExtraSensory App to collect both sensor-measurements and self-reported context-labels. The app comes with pre-trained classifiers that were trained on the ExtraSensory Dataset – this makes the real-time predictions more useful – they can help the user recall what they were engaged in at different times, and make label-reporting quicker and easier. You can easily customize the app to your study by selecting which sensors to record and which context-labels to target.
- Collection of only sensor-measurements (unsupervised data). The users will simply keep the app running in the background, collecting sensor-measurements, but not labels. This scenario requires less effort from the users, contributing to even more natural behavior. In such a scenario you may perform a behavioral study, and use the sensor-measurements offline to automatically recognize the user's context, as an additional input variable to the study.
- Utilizing real-time predictions. You can design a different application that utilizes the context-predictions of ExtraSensory App. If the phone-app is connected to the internet, it will keep getting predictions from the server for every minute.
- Context-journaling. You can also utilize the user-interface of the ExtraSensory App, only for journaling users' context, as a flexible replacement to end-of-day surveys or other manual methods for self-logging of behavior.

B. Credit

The ExtraSensory App was originally designed and implemented by Yonatan Vaizman and Katherine Ellis, with the advising of Gert Lanckriet. Yonatan Vaizman implemented the Android version for the original data collection effort and made the adjustments to the publicly available version of the app provided here.

If you use the ExtraSensory App (either with the code as it is or by adjusting the code to your own version) for your work and produce publications out of it (papers, applications, etc.), we request that you cite the paper describing the app:

Vaizman, Y., Ellis, K., Lanckriet, G., and Weibel, N. "ExtraSensory App: Data Collection In-the-Wild with Rich User Interface to Self-Report Behavior". *In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA. DOI: <https://doi.org/10.1145/3173574.3174128>.

C. User guide

This section describes the features of the ExtraSensory App and how to use it. By “user” we refer to the user of the Android phone. Typically this would be a participant (subject) in a research study. This section mainly discusses the phone-side of the app: ExtraSensory-Phone (ESP).

1. What the app does (in short)

ExtraSensory is a mobile application designed for data collection for research. It collects data, including measurements from sensors (built in to the smartphone and from a smartwatch, if available) and labels about the user’s behavioral context (activity, environment, body posture, phone position, etc.). The measurements are taken automatically by the app, while it is running (even in the background). The labels are provided by the user through various mechanisms in the app’s user interface. The app communicates with a university server to send the measurements and the labels to it and to receive context predictions from it.

The user can provide labels of their context through three main mechanisms: history, active feedback and notifications. Through the history page (see [The history page](#)) the user can label context in the past (today and yesterday). Through active feedback (see [The “Active Feedback” button](#)) the user can label context in the near future (up to 30 minutes). Notifications (see [Notifications and alerts](#)) automatically pop-up to remind the user to provide labels and they can trigger the feedback page for the user to label either the near past (last 20 minutes) or near future.

2. What the app does in the background

The app is scheduled to do a “recording session” once every minute. In the recording session sensors (from the phone and from the smartwatch, if applicable) are activated and collect measurements for 20 seconds. The measurements are then bundled together into a zip file, which is then sent to the server through the internet. The server processes the measurements in real-time and responds to the app with a prediction/guess of the user’s context. The predictions come in the form of a list of context-labels (e.g. [sitting, walking, computer work, phone in pocket]) and a list of predicted probabilities assigned to the labels (e.g. [0.7, 0.2, 0.8, 0.5]). This procedure occurs whether the app is running, either in the foreground (the user sees one of the app’s pages right now) or in the background (and even when the phone’s screen itself is locked).

i. Using the network

The app has to use the internet in order to send the measurements (in a zip file) to the server (and expect a context-prediction in response) and also to send “feedback” (the user-provided labels) to the server when the user provides it. This communication is done securely using HTTPS protocol, so there is data encryption and server authentication. The app is defined to only use the internet when the phone is connected to WiFi network (in order to avoid using the user’s cellular data plan). This means that whenever the app is collecting data (either sensor measurements or feedback labels from the user) when there is no WiFi available, this data is stored in the phone’s internal storage until WiFi is available. These are the default and recommended settings, but they can be changed in the settings page (see [The settings page](#)) according to the researcher and user preferences.

ii. Location “bubble” around your home

If “location” is one of the used sensors the app will collect geographical location coordinates (latitude and longitude). In order to keep the collected data anonymized the user can opt to hide the exact location of their home by declaring a location “bubble” centered around the home’s location, with a radius of 500 meters (~0.31 miles). This is done through the settings page (see [The settings page](#)), where the user can turn “location bubble” on and input the exact coordinates (in decimal degree values of latitude and longitude) of the desired center of the bubble (typically, the user’s home).

Then, whenever a recording session is done when the user is located somewhere in the bubble (closer than 500 meters to the bubble-center) the location coordinates will not be sent to the server. Other location-based measurements (e.g. altitude, speed) will still be sent even when inside the bubble.

Whenever the user is outside the bubble all the location-based measurements will be sent, including the exact coordinates (latitude and longitude).

3. Closing the app


You should be aware that simply closing the user interface of the app doesn’t close everything. The app still has the repeating scheduled alarm (which fires every minute to trigger a recording session) and unless it is canceled this alarm will still fire after you closed the user interface and trigger more recordings in the background of the phone. So in order to make sure the app is closed and will not record until you open it you should turn off the “data collection” mode – the home page has a switch for data collection (see [The home page](#)). Only after this you should close the user interface of the app.

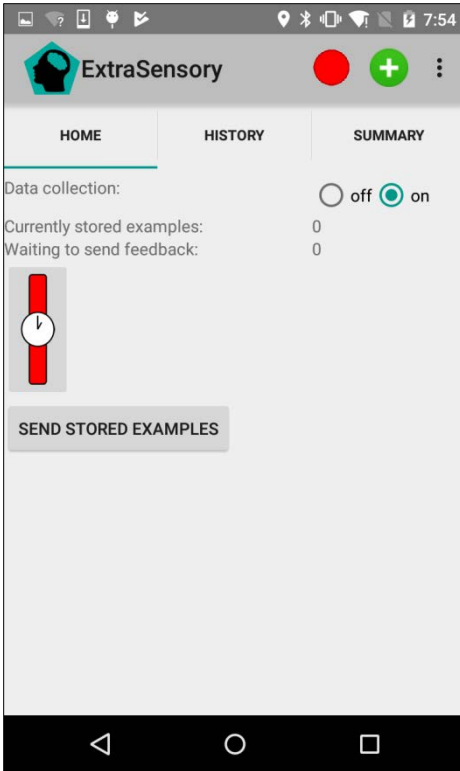
An alternative way is to make sure the app is completely killed by applying “force close” to the app through the phone’s settings->Apps.

4. User permissions

The phone’s operating system may require the user to grant the app permission to do some operations (e.g. use location services when the app is in the background, use the microphone, use Google play services). In the first time running the app there may appear some alert dialogs asking to approve these permissions. The participants in the research are required to approve these permission requests.

5. The red dot

As an indication that the app is “recording right now” the app uses an image of a small red dot (like the red light of a video camera) . It appears in the top action bar.



The red dot appears at the top action bar.

6. The main page

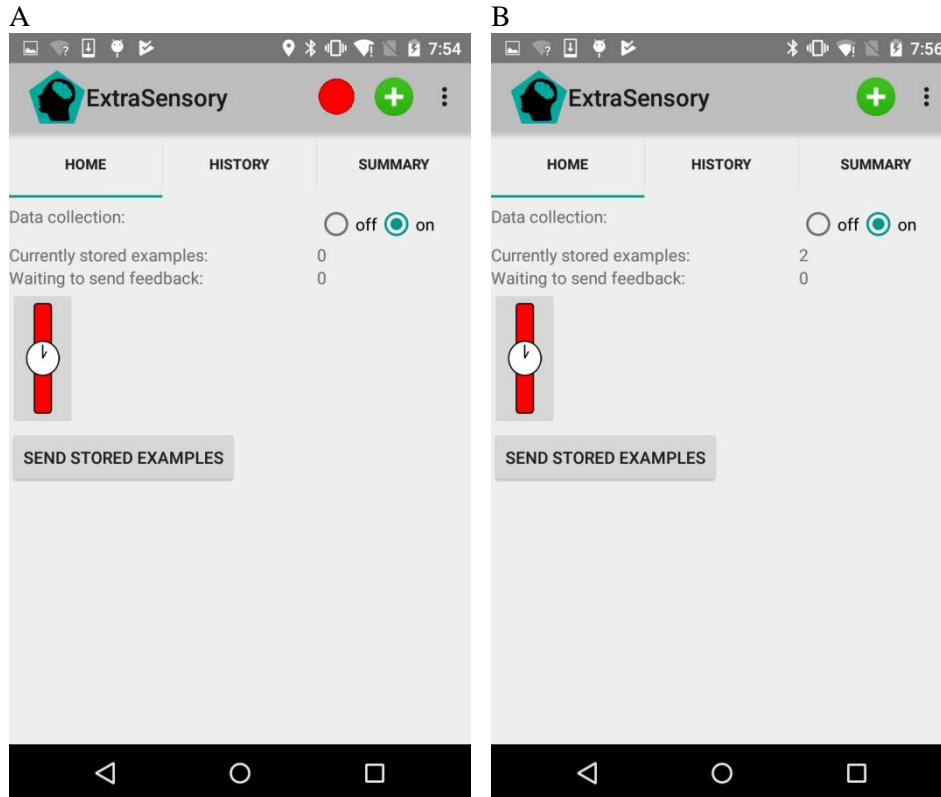
The main page of the app has a tab bar to switch between different pages: the home page, the history page, and the summary page. It also provides a button to quickly perform “active feedback” – where the user initiates reporting their context in the immediate future.

7. The home page

The home tab is the default page that appears when opening the ExtraSensory App. It has a few indicators that give some information to the user:

- **Stored data.** Whenever there are data waiting to be sent to the server there will be an indication on the screen of the home page, either for how many zip files of measurements are waiting to be sent or how many label feedbacks from the user are waiting to be sent (or both).
When WiFi is available these statements should appear on the screen for a brief moment and then disappear. When WiFi is not available the app can accumulate stored files (zip files of measurements or files for feedback) and then these messages can stay on the screen longer and update the number of items waiting to be sent.
- **The data-collection on-off switch,** to allow the user control over when the app records sensors. Turning data-collection off will stop the schedule of recording sensors (once a minute) and also stop the scheduling of notifications. Typically, a researcher can request the participants to keep data-collection on as much as possible and convenient to them.

- Watch icon. This icon's color indicates whether a Pebble watch is paired with the phone. Read [Smartwatch](#).



The home page. A) During recording the red light is on. B) The home page displaying that 2 examples are currently stored and waiting to be sent to the server, because WiFi was not connected yet.

8. The history page

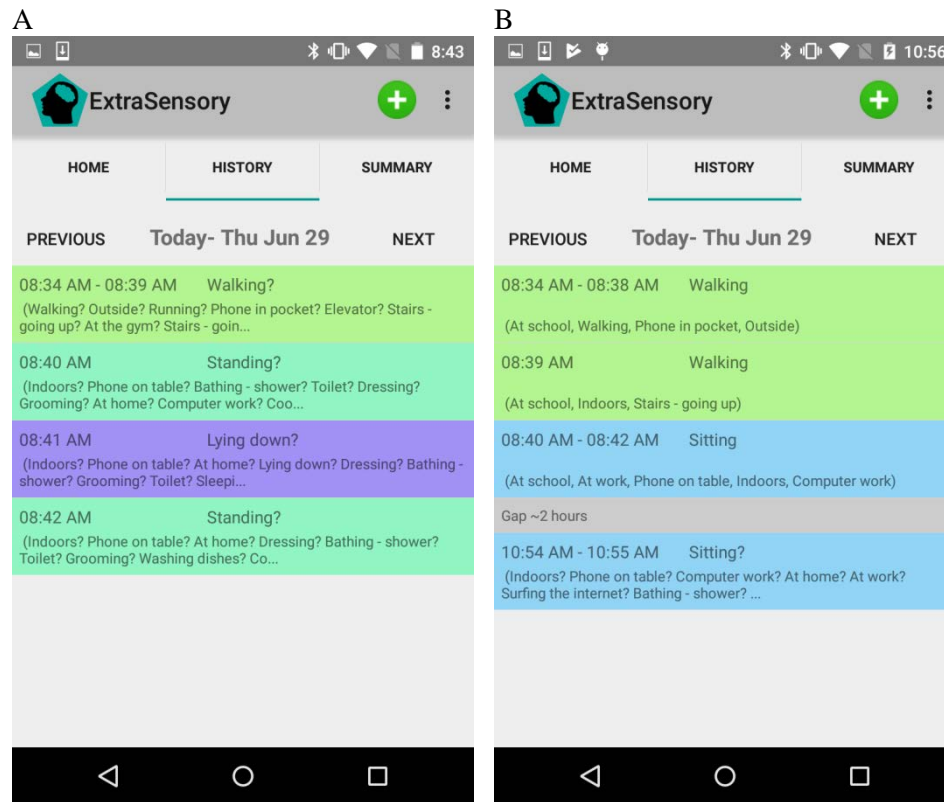
The history page displays a journal of the user's contexts from a single day. This is one of the main places for the user to provide labels – this is where the user can edit the labels of their context in the past.

i. Today's context log

When opened, the history page displays today's log as a list where each "event" (period of time with a constant context) has its own row and they are sorted from earlier (top) to later (bottom). Each row states the time of the events (single minute or a time range), the main activity (mutually exclusive body states) and more specific labels (secondary activities and moods, if applicable). Each row gets a background color that indicates the main activity label to help the user visually distinguish periods of time of sitting vs. walking, running, etc. The list only covers periods of time where the app was running and collecting data. If there were consecutive minutes in the day for which the user assigned exactly the same context-labels (same main activity, same secondary activities and same mood labels) they will be merged together to a single event and represented in a single row with a time range of minutes.

Minutes that had data collection but didn't yet get labeled by the user will appear with the server prediction for the main activity and this will be indicated by the question mark "?" appended to the main

activity. Further context-labels described in the server-prediction will appear also with question marks, sorted according to the probabilities assigned to them (from most probable to least probable). If the user is sitting with the phone for 25 minutes it is reasonable that these 25 minutes will get the same prediction of “Sitting” from the server and will appear as a single event row in the history with a time range of 25 minutes. Minutes that don’t yet have user provided labels and didn’t yet get a server prediction (e.g. the current recording session being done now) appear as just “null”.



The history page. A) Server predictions (guesses) appear with question marks, for both the “main activity” (like walking, standing, etc.) and the finer context-details. The first row shows how consecutive minutes with similar prediction are merged to a single “event”. The colors correspond to the “main activity” to visually help the user. B) Events that were labeled by the user appear without question marks (first 3 rows). Long gaps without recording are represented in “gap rows”.

ii. Switch day

The user can press the “previous” button to switch to the previous day or the “next” button to switch to the next day. Days that are too long ago (before yesterday) are presented only for view and don’t allow editing the labels (to avoid having the user trying to recall what happened 2 days ago). This is unlike the history of today and yesterday, where the user can edit/correct the labels of the events.

iii. Edit labels for an event

When clicking on a row of a specific event the app directs the user to the Feedback page (see [The feedback page](#)) to provide feedback (labels) for this event. These labels will then be assigned (and sent to the server) to the whole time range of the selected event – if it represents a consecutive period of 25

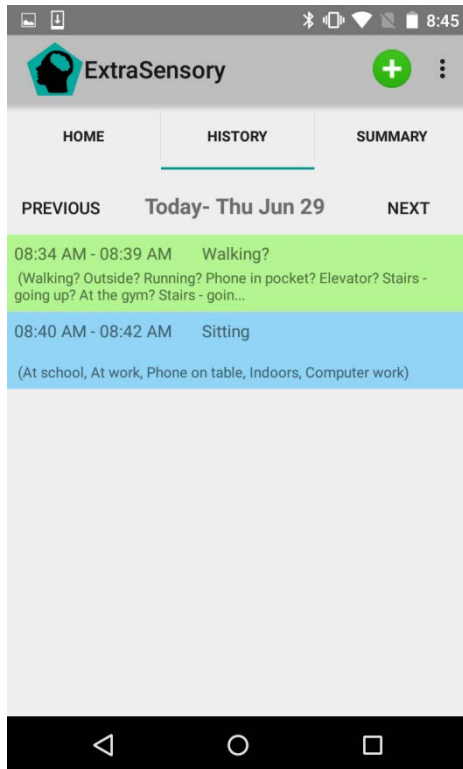
minutes of collected data the provided feedback labels will be applied to all the examples collected during this period (typically 25 examples, 1 for each minute).

iv. Split an event to minute-by-minute

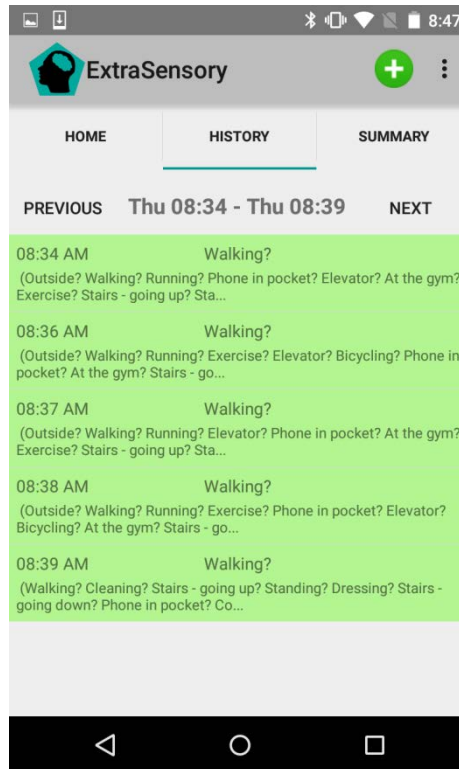
It may occur that the server gives the same prediction to all the minutes in a consecutive period of 60 minutes, and this whole time period appears in the history as a single event (in a single row), even though the user was doing one thing in the first 30 minutes and a different thing in the last 30 minutes. For such cases the app has the flexibility to allow the user to split the activity-row by presenting the 60 minute long “event” in a minute-by-minute way (each minute gets its own row). That way the user can edit the labels of any particular minute (say the 30th minute in that time period) and when returning to the standard history mode those 60 minutes will appear split to several events with different context labels.

To split an event, the user can swipe the finger to the left on the row of the long event (which they want to split). Then the history page will simply show the long event in a minute-by-minute mode – a separate row for every minute in the relevant time range. The user can then edit any of the specific minutes’ labels. To return to the standard (whole day) history mode the user can press either the “previous” or “next” buttons on top.

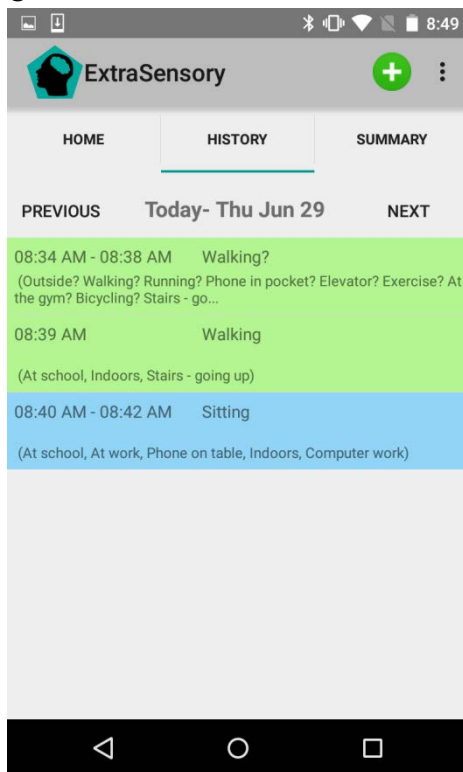
A



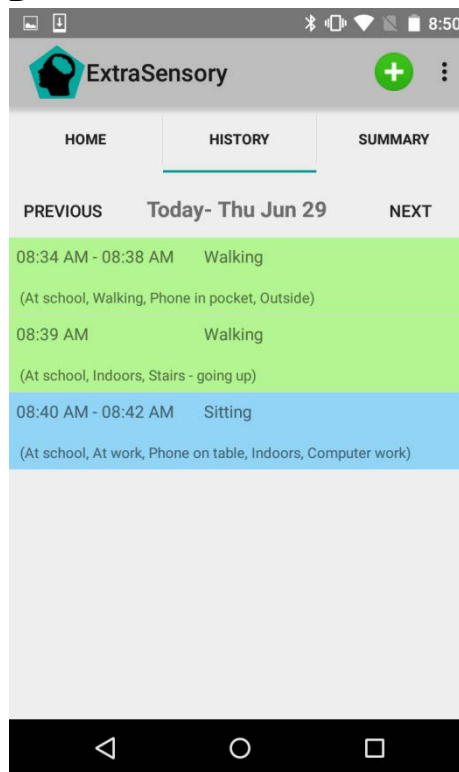
B



C



D




Splitting an event in the history. A) To split the first event (Walking 8:34-8:39) swipe a finger to the left on the event's row. B) Minute-by-minute mode. Now we can click the separate minute-event of 8:39 and provide labels for this minute. C) After providing feedback we are back at the history page, where the singled-out minute is now a separate event with user-labels (at

school, indoors, stairs – going up). We can click the top row to label the rest of the minutes. D) After labeling minutes 8:34-8:38 the history shows two separate events with different context (although both have main activity “Walking”).

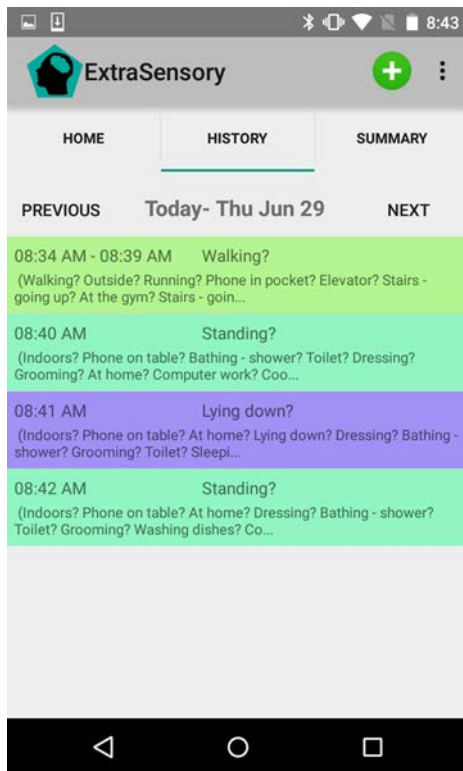
v. Merge activities

The opposite scenario is also possible: where there are long periods of time where the user was doing the same activity, but the server somehow predicted different activities for different minutes. In this case the history page will show many separate rows with different server predictions and the user would like to merge them all together and report the same labels to all these minutes. For such a case the app gives the option to merge different consecutive rows in the history and then report feedback labels once for all of them (which would cause all these minutes to be merged to a single row in the history).

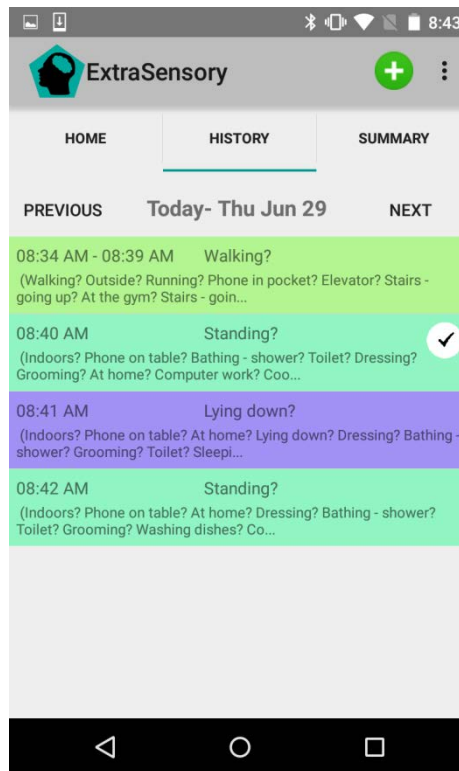
If all the events selected for merging have only a server prediction (and no user-provided labels) the feedback page will display the server prediction of the first (earliest) row in the range as the initial value for the main activity (which the user is free to edit, of course), and for the secondary activities the app will present the average probability (over the minutes in the time range) predicted for each label. If in the selected range there is a single event with user-provided labels, these labels will be used as initial labels in the feedback page – so if the user already labeled an activity and they want to label the minutes before or after it with the same labels, this is a convenient way to do so. The app doesn’t allow merging a range of time that has 2 or more events that already have user-provided labels. This is in order to prevent the user from overwriting their own labels by mistake.

How to merge: in the history page when swiping the finger to the right on a row the row will get a checkmark symbol  on it (symbolizing that it is marked for merging). When swiping to the right another row all the rows in the range between the new row and the already-marked zone will also get marked, so the user can expand the marked range. Then when clicking on any row within the marked range the feedback page will open to provide labels for the entire marked range (which will cause this time range to appear merged in the same row when getting back to the history page). When swiping to the right on a row that is already marked, the whole mark range will clear from all the marked rows.

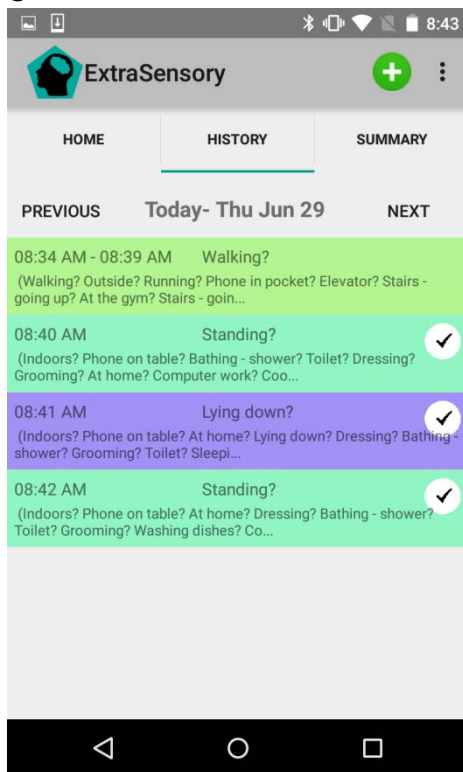
A



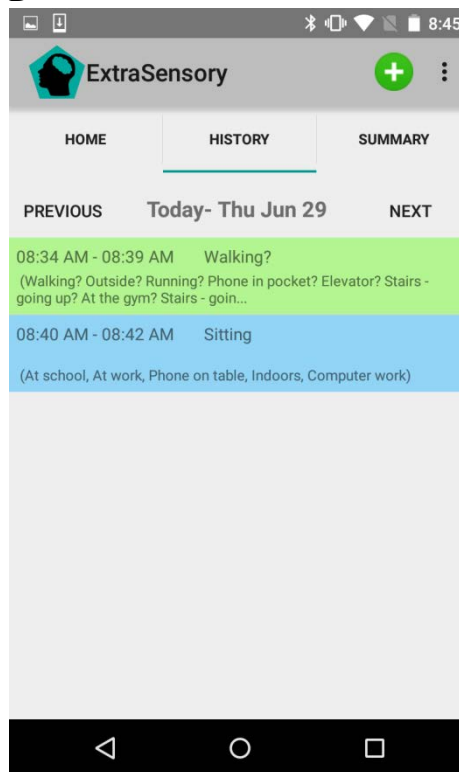
B



C



D



Merging events in the history. A) To provide the same labels for minutes 8:40-8:42 (bottom three rows), swipe finger to the right on the 8:40 row. B) The 8:40 row is now marked for merging with the checkmark symbol. Now swipe finger to the right on the 8:42 row. C) The 8:42 row is now marked, as well as all the rows in between (in this case, only the 8:41 row). Now clicking any

of the marked rows takes you to the feedback page to report labels for all these minutes together. D) After reporting labels for all the merged minutes, they now appear in a single event with user-provided labels (Sitting, 8:40-8:42).


9. The summary page

The summary tab shows basic statistics about the user's behavior – how many minutes was the user engaged in each of the main activities.



The summary page.

10. The “Active Feedback” button

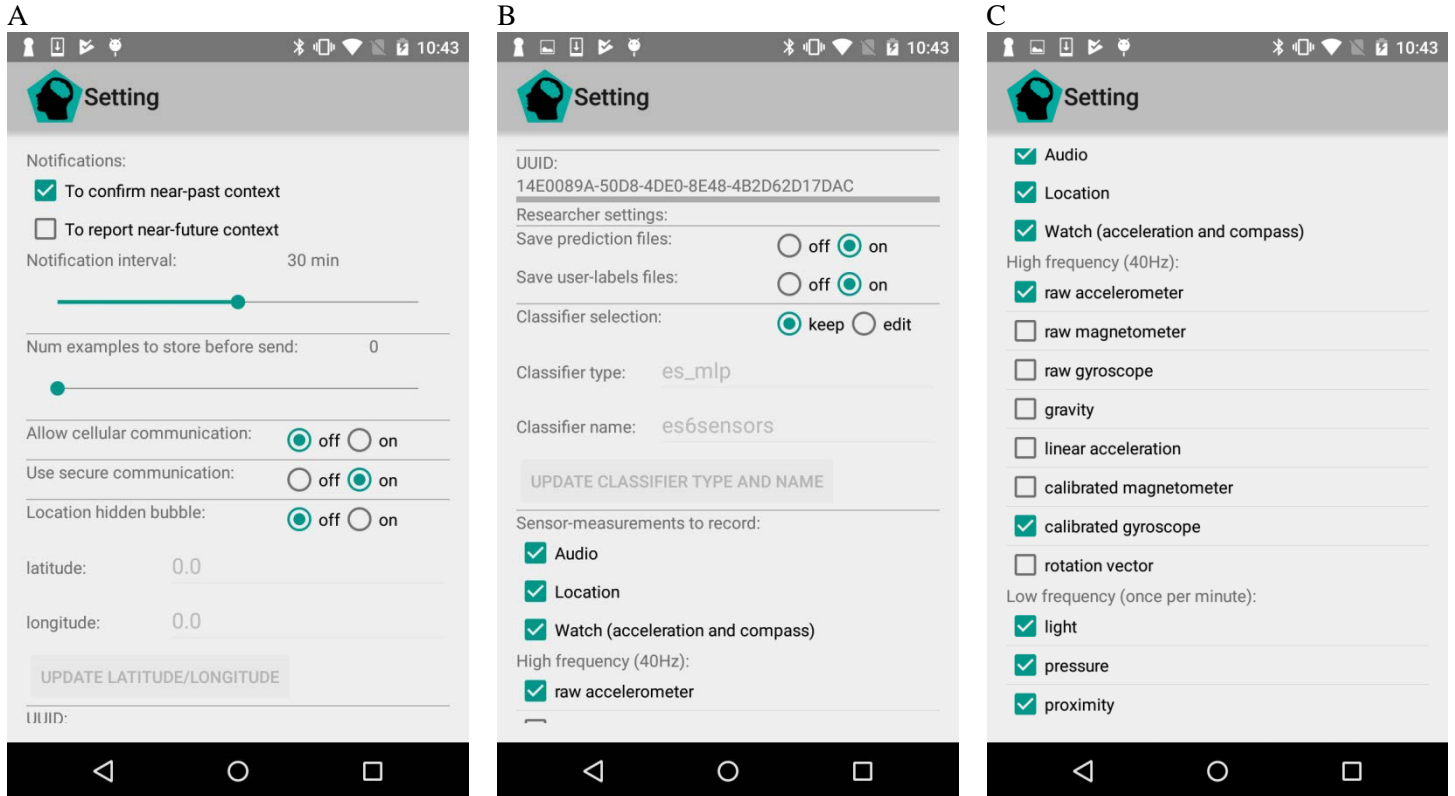
The active feedback button is the main way for the user to provide labels for immediate future context. It appears as a green plus symbol  on the top action bar. When pressing the active feedback button the user is directed to the feedback page (see [The feedback page](#)), in “active feedback” mode – meaning to provide feedback labels for near future context (unlike when opening the feedback page from the history page). Using this button the user can report what they are going to be doing from now for the next X minutes (up to 30 minutes) and that way can be free to actually do the activity without interacting with the app. For example, they can report that in the next 30 minutes their context is going to be “Sitting, Drive – I’m the driver, In a car, With family, Phone in pocket”.

11. The settings page

The settings page allows the user to adjust some preferences. It is accessible through the menu in the action bar, and if the phone has a menu button, through the menu button. The top part specifies preferences that the user can adjust:

- Notifications:
 - Near-past notifications. Should the app pop up near-past notifications/alerts? These are notifications that appear if the user provided labels for the recent 20 minutes, and ask whether or not the context remained the same in the recent past since the last user-report.
 - Near-future notifications. Should the app pop up near-future notifications/alerts? These are notifications that appear if the user did not provide labels for the recent 20 minutes. These notifications simply remind and request the user to report what they are doing in the near-future (like active-feedback).
 - Notification interval: the time interval between notification-checks. In each notification check, the app assesses which type of notification is applicable now (near-past or near-future), and whether or not the user requested to use this type of notification. The default is 30 minutes.
- Num examples stored before send. The default is 0, meaning that every recording session is immediately sent to the server once it is finished (assuming WiFi is available). The user can choose to increase this number and then the app will store the zip files on the phone until it reaches the number of examples, and only then send them. The upside to increasing the number is concentrating the communication with the internet and making it less frequent, which may help conserve battery. The downside is that the user will not get the server predictions in real-time, but rather with a delay.
- Allow cellular communication. By default communication with the server is only done when WiFi is available on the phone. However, by switching this option on ExtraSensory-Phone can send messages to the server also when it is out of WiFi range, by using the user's cellular data plan. Since the amount of data transmitted can be large, it is recommended that this be turned on only if the phone has an unlimited data plan.
- Secure communication. By default whenever turning on the app it is in secure communication "on" mode. This means that the communication with the server (for both sending the zip files of sensor measurements and sending the feedback labels) is done over the secure HTTPS protocol. This switch should stay on and the user should turn it off only if there is some communication problem and they are asked to try it by the researcher to debug the problem. If the data collected in the study is not sensitive, the researcher may also decide to allow unsecured communication (turn this switch "off").
- Hide home location. This switch should be turned on if the user wants to apply a location "hidden bubble" around their home (or some other private or identity-revealing location). (see [Location "bubble" around your home](#)). When turning this switch on, 2 extra fields become available: latitude and longitude. Inside these the user can report the exact decimal-degrees value of the geographical coordinates of the home location. The user has to press the "update latitude/longitude" button after filling the values.

- **UUID.** Universal Unique Identifier. This is just an indication to the user and can't be edited. This is the identifying sequence of characters given to the user of the app. All communication with the server will use this UUID as an identifier (to avoid using name, phone number or other identity-revealing information).
- The rest of the items in the settings page are under the title “researcher settings” to indicate that these are typically things that the researcher decides for the study and the user should not change them.
- **Save prediction files.** If the study involves an external application that uses the server predictions of ExtraSensory App, turn this switch on. This will cause the app to save the server predictions to files that are accessible to other apps.
- **Save user-labels files.** If the study involves an external application that uses the user-reported labels of ExtraSensory App, turn this switch on. This will cause the app to save the user-reported labels to files that are accessible to other apps.
- **Classifier selection.** Switching this switch from “keep” to “edit” enables editing the next two fields – classifier type and classifier name – to indicate which classifier the server should use when performing context-prediction from now on. Read [Customizing the classifier](#) for more details.
- The next section specifies which sensors the app should record. The researcher may decide that some sensors are unnecessary (for example, if the classifier does not use them at all), so unchecking them in this list will save battery (both in recording time and in transmitting the data).



The settings page. A) Top part – user preferences. B) Middle part – researcher settings. C) Bottom part – available sensors list, after selecting to record also Watch.

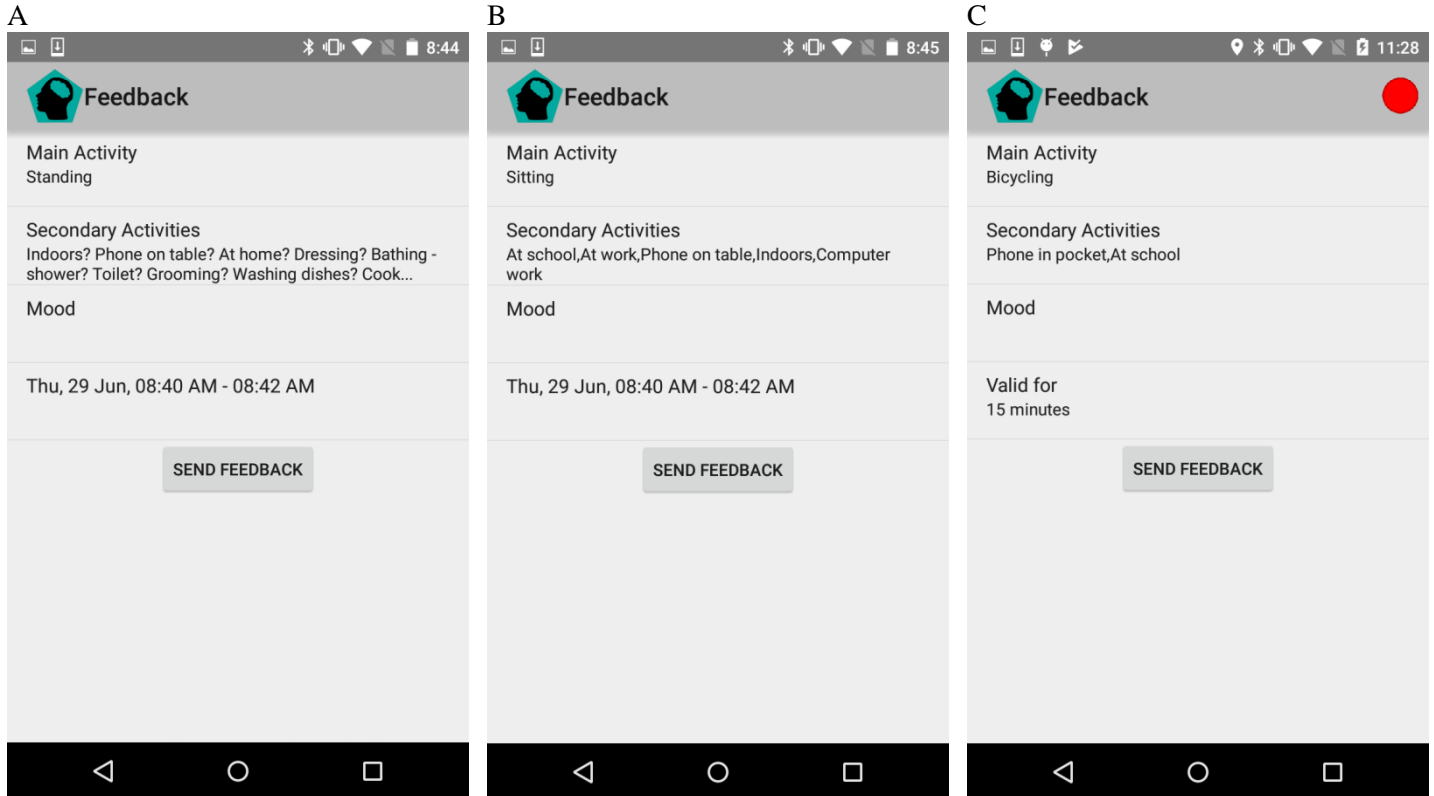
12. The feedback page

The feedback page is the interface of the user to edit context-labels for some event. It can be reached through the history page in “past” mode, to edit labels for past events. It can be reached through the “active feedback” button in “active feedback” mode, to report labels for the near-future context being started now. It can be reached through notifications in either of these two modes (past or future).

The feedback page lets the user fill a label for the main activity (body state), labels for the secondary activities (not necessarily “activities”, but any combination of labels describing the context) and labels for the mood. In active feedback mode these fields start fresh (empty) and there is another field to select for how many minutes this new context is going to be valid (up to 30 minutes). In past mode these fields will appear with the current labels of the edited event (if any) or, if the user didn’t yet provide labels for the edited time range, with the server prediction for the main activity. In past mode there is another field presenting the time period that is being edited now.

In order to fill labels for a field the user simply presses the field and the label selection page (see [The label selection page](#)) is opened with the appropriate selection menu/vocabulary of labels.

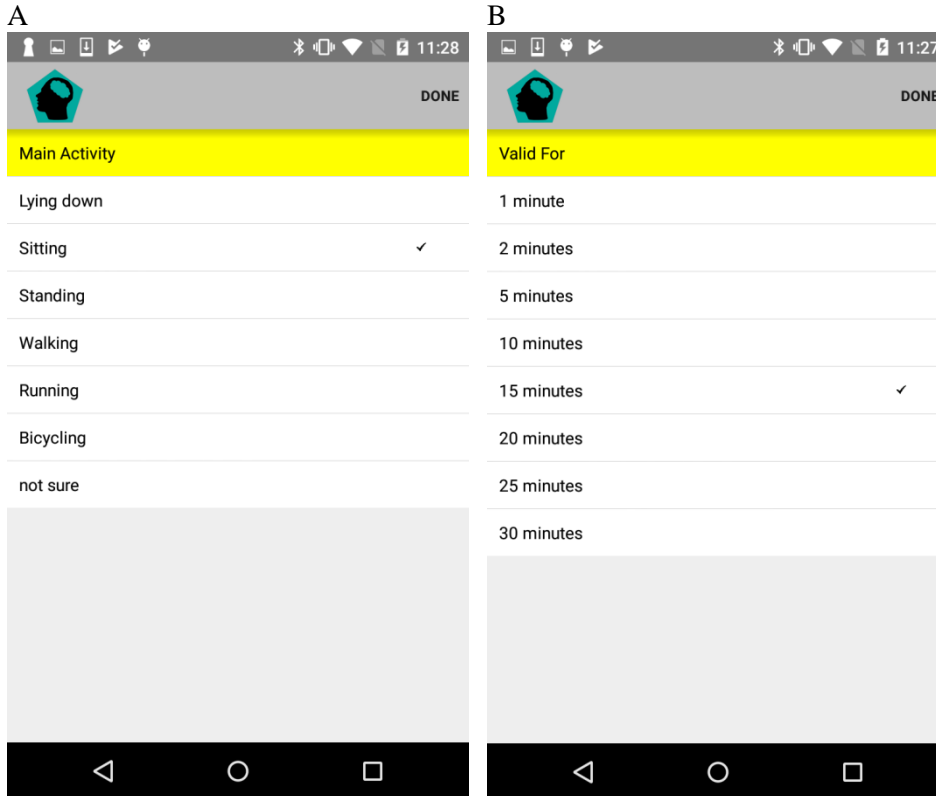
After filling the desired labels in all the fields the user should press the “send feedback” button in order to actually send the labels to the server. Otherwise (if pressing the back or up button) the filled labels will be discarded and no feedback will be done.



The feedback page. A) Feedback from history for the period of time 8:40-8:42 (the edited time period is presented in the bottom row). When entering the feedback page for an unlabeled history-event, the “main activity” field is prepared with the server-guess of main activity (Standing), and the “secondary activities” field presents the server-guesses with question marks sorted by predicted probabilities. By clicking a feedback field (“main activity”, “secondary activities” or “mood”) you can edit the selected labels (through the [The label selection page](#)). B) The feedback page for history event, after selecting the desired labels for “main activity” and “secondary activities”. Now, to report these selected context-labels, press the “send feedback” button. C) The feedback page for “Active feedback”. When pressing the green “+” button on the top action bar, you will reach this page with blank values. After selecting the desired context-labels for your immediate future, you can specify for how long is this context is going to stay valid, and press the “send feedback” button.


13. The label selection page

The label selection page allows to select labels from a menu (or vocabulary) of labels. When pressing “main activity” in the feedback page the label selection page opens in a very simple manner, just selecting one label out of the menu. This is because the “main activity” category is designed for mutually exclusive options – the body posture/movement state. But when pressing “secondary activities” or “moods” in the feedback page, the user may select multiple labels and the label selection page appears with some more features to help the user find the relevant labels quickly.



The label-selection menu for mutually exclusive categories. For these categories, the menu does not allow selecting multiple-labels together. A) Selecting main activity requires selecting exactly one body-state of the menu (or the dummy label “not sure”). B) Selecting “valid for” value for active feedback is also done through the label-selection menu and is limited to 30 minutes in the future.

i. Mark labels for selection

Simply click the label and it will appear as marked (with a checkmark symbol .

ii. Unmark labels for selection

When clicking on a label that is already marked, it is then un-marked.

iii. Index

For the selection of secondary activities or moods there is an index to the right of the list. The index allows for quick access to the user’s label of interest. For the secondary activities the index is according to different topics (e.g. transportation, sports, housework) and clicking an index item makes the list jump to the relevant sub-list of labels. A label may occur in multiple relevant subjects (e.g. the user can reach label “on a bus” from either the “location” index-item or the “transportation” index-item). For the moods the index is alphabetical.

The index may include additional items, such as frequent (see [Frequently used labels](#)) and “selected”, for a sub-list that displays the labels that are already selected (and all appear with checkmarks).

iv. Frequently used labels

This sub-list (that appears for secondary activities and for moods, and has its own index item) shows the labels that were most frequently used by the user in the past, sorted from most used to least used. This section can make it easy for the user to quickly report labels that occur a lot.

v. Server guess

This sub-list (that appears for secondary activities and has its own index item) shows the labels that got server prediction for the edited time-range (this is only available when editing context for the past). These labels are displayed sorted by their average-predicted-probabilities (confidence of server guess). These labels appear with their assigned confidence value also in other sub-lists. High confidence (close to 100%) indicates that the server guesses that this label is relevant in high probability. These predicted probabilities can help the user quickly find the relevant labels, or recall what they were doing at a particular time.

vi. “Done” selecting labels from list

The user must press the “done” button after selecting the desired labels from the list. This will return to the feedback page and the selected labels will appear in the right field in the feedback page. Otherwise (if pressing the “back” or “up” button) the selected labels will not be saved.

A

DONE		
Frequently used	Selected	
At school	51%	✓
Phone on table	19%	
At work		
Indoors	25%	
Computer work	43%	
Server guess (with confidence)		
Walking	82%	
Cleaning	81%	
Stairs - going up	78%	✓
Standing	77%	
Dressing	77%	
Stairs - going down	76%	

B

DONE		
Phone	Selected	
Phone in pocket	25%	
Phone in hand	40%	
Phone in bag	26%	
Phone on table	71%	✓
Phone away from me		
Phone - someone else using it		
Phone strapped		
Physical		
Talking	34%	
Stairs - going up	34%	
Stairs - going down	33%	
Limping		

C

DONE		
Selected labels	Selected	
At school	29%	✓
At work		✓
Phone on table	71%	✓
Indoors	85%	✓
Computer work	53%	✓
Server guess (with confidence)		
Indoors	85%	✓
Phone on table	71%	✓
At home	66%	
Dressing	62%	
Bathing - shower	62%	
Toilet	61%	

The label selection page, for secondary activities (detailed multi-label context). The label menu allows selecting multiple labels simultaneously (multi-label mode). The label menu can be long, so the side-bar index has quick-links for topics to help you find the relevant labels quickly. A) After you report labels, the menu includes the “Frequently used” section (with the index link “Frequent”) that presents the labels that you have used before, sorted by your frequency of usage. When selecting labels for a history-event that was already sent to the server the menu includes the “Server guess (with confidence)” section, presenting the labels from the server prediction sorted by the probability assigned to each label (the server’s confidence). These confidence values are also shown in other sections that these labels appear under. B) By pressing the “Phone” quick-link on the index you jump to the “Phone” topic section of the menu to see labels related to the position of the phone. Similarly you can jump to other topics. C) The “Selected labels” section (indexed by the quick-link “Selected”) presents the labels that are currently selected (so they all have the checkmark symbol).

14. Notifications and alerts

The app is scheduled to generate reminders to the user to provide labels. Every X minutes (defined by the “reminder interval” field in the settings) the app does a checkup to see when the user last reported labels, and according to it, generates a notification with the appropriate question.

i. 2 types of questions

- Notification for near-past activity. If the user provided labels to an event in the past 20 minutes the notification question will ask the user if they are still engaged in the same context in the past Y minutes (e.g. “In the past 11 minutes were you still running (listening to music, on the beach) and feeling energetic?”). The user will then have 3 possible answers to reply (3 buttons to press):
 - Correct. Then the app will apply the exact same labels to all the minutes in the time period from the user-labeled event up to now.

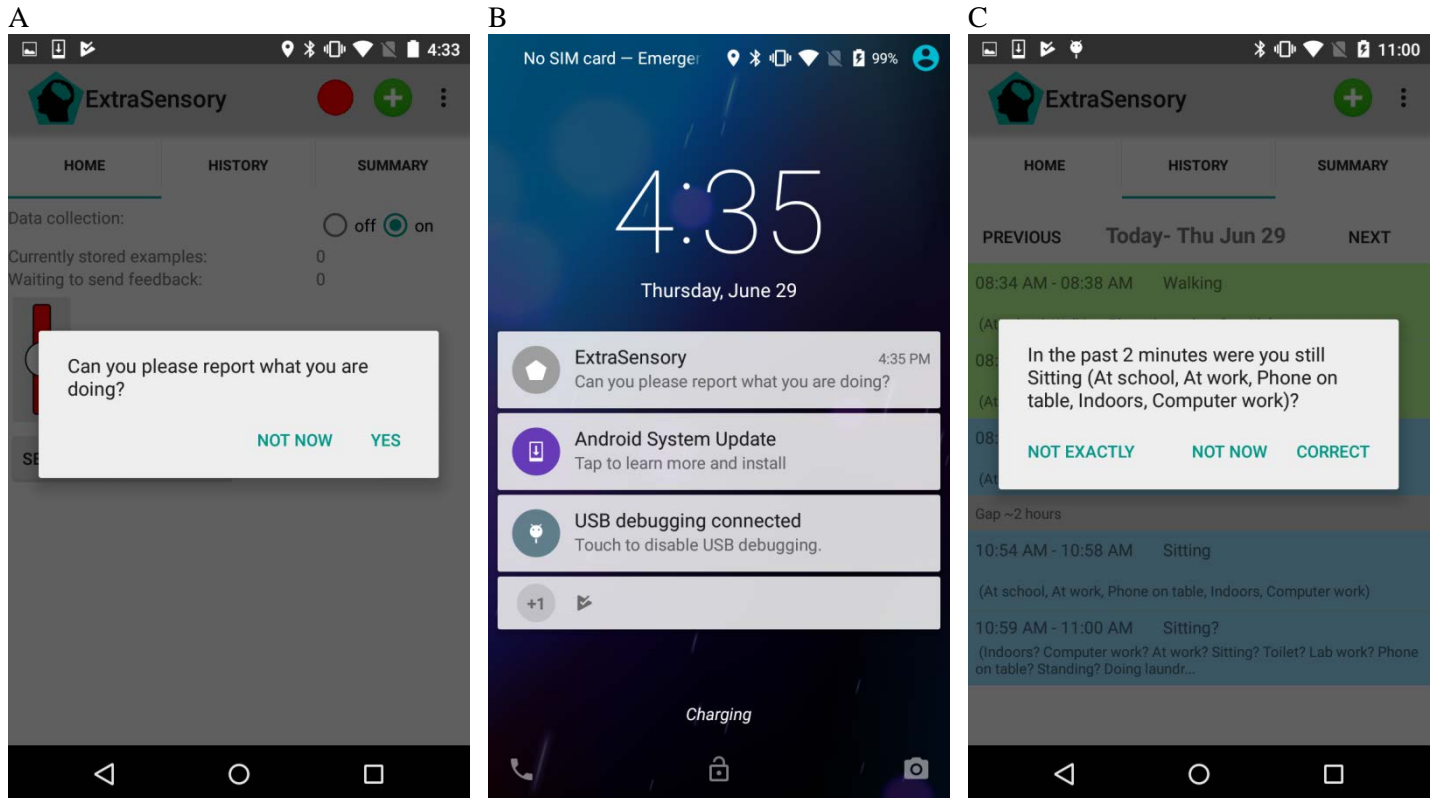
- Not exactly. Then the app will direct the user to the feedback page, in “past” mode, to report labels for the time period from the user-labeled event up to now, with initial labels as that latest labeled event. This is mostly useful if the user wants to change slightly the labels since the last update.
- Not now. This will dismiss the alert without doing anything.
- Notification for active feedback. If the user didn’t provide labels in the past 20 minutes the app will simply ask them to provide fresh new feedback with the question “Can you please describe what you are doing?”. The user will have 2 possible answers to press:
 - Yes. This will direct the user to a fresh new active feedback in the feedback page.
 - Not now. This will dismiss the alert without doing anything.

ii. Notification when the app is in the background

If the notification pops when the app is in the background (when the phone screen is either locked or unlocked) a local notification will pop on the screen with the question, to draw the user’s attention. Then when responding to the notification the app will open (take focus on the screen) and then the alert with the same question will appear and present the possible answers in buttons in the dialog box.

iii. Alert when the app is in the foreground

If the app is already in the foreground, there is no need for a notification to pop, and the alert dialog directly pops on the screen with the question and the possible answers in buttons.



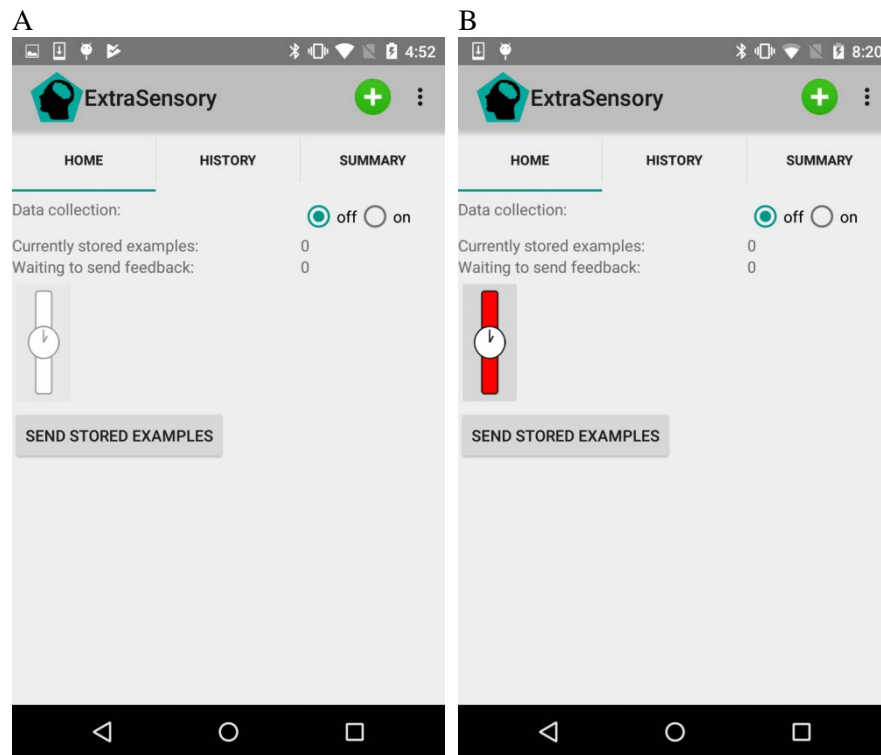
Notifications. A) A blank notification will appear if you did not report any labels in a while. If the app is in the foreground of the phone, an alert dialog will simply pop up. By selecting “YES” you will be directed to the “Active feedback” interface. “NOT NOW” will dismiss this alert. B) If the phone is locked (or open with ExtraSensory App in the background), the notification will appear as a regular phone-notification. You can select it to be directed to the app. C) If you did report some labels recently, the notification will ask you whether your context remained the same in the past few minutes. You can select “CORRECT” to apply the same context-labels to all the minutes the notification is asking about (since your latest label report). If your context has not been exactly the same as in the question, you can select “NOT EXACTLY” and then you’ll be directed to the feedback page and you can adjust the labels for these past few minutes.

15.Smartwatch

The ExtraSensory App has a component for a Pebble smartwatch called ExtraSensory-Watch (ESW) that integrates with the phone component ESP. The user has to install the Pebble app on their mobile phone. It doesn’t have to be running, but needs to be installed. The Pebble app needs to be logged in to a Pebble account (this can even be the researcher’s Pebble account), and the watch-side app – ESW – should remain open on the watch. The watch is used for two things: it collects more measurements (from accelerometer and compass built into the watch) and it provides an easy user interface for the alerts. Whenever a recording session is done and ESW is connected to ESP, the acceleration measurements from the watch will also be sent with the data. Alerts and notifications should pop up on the watch as well as on the phone. For questions about past activity (e.g. “in the past 11 minutes were you still running?”) the upper right button of the watch serves as a reply of “yes” (in which case the user doesn’t have to answer the alert on the phone). The bottom right button of the watch serves as “not now” (in which case the alert/notification will still be present on the phone).

When ESP senses that the pebble watch is paired with the phone, a watch icon will appear on the home page with bold lines and red color filling. When ESP senses that the pebble watch is not paired, the watch

icon will appear with gray lines and blank filling. Notice: even when the icon shows as “active” it is possible that still no measurements come from the watch (for instance, if ESW is not open on the watch). If the user accidentally exited ESW on the watch (or if there is some connectivity problem) they can try to re-launch it by pressing the red watch icon on the home page of ESP. This should launch ESW on the watch and re-establish the connectivity, making ESP and ESW listen to one another.



A) When the Pebble watch is not paired with the phone, a gray watch icon appears on the home page. B) When the Pebble watch is paired with the phone a bold and red watch icon appears on the home page. This icon is then also a button to relaunch the ESW app on the watch and re-establish mutual listening between ESP and ESW.

D. Researcher guide

This section is a guide to the researcher that wishes to use the ExtraSensory App for research. It explains how to deploy the app in your environment (with your own research server), and how to customize it to your research needs.

1. The app components

The full code for the ExtraSensory App (ESA) is available in a single github repository at <https://github.com/cal-ucsd/ExtraSensoryAndroid>.

ESA is comprised of three components that work together:

- ExtraSensory Phone (ESP) – an Android application written in Java. This is the heart of the app (and the bulk of the code) is the phone component, ESP, which runs on an Android phone, records sensor-measurements, communicates them to the server, and has a user interface for self-reporting context-labels.
- ExtraSensory Server (ESS) – a web service written in python. This is the part that collects and stores all the data it receives from ESP: sensor-measurements and context-labels from multiple ESP-users. It also provides real-time predictions of the behavioral context of the user and it sends these predictions back to ESP.
- ExtraSensory Watch (ESW) – a Pebble watch app written in C. This component is not mandatory, but if you have a Pebble watch it can add very helpful functionality: First, you can record measurements from the sensors on the watch (accelerometer and compass). Second, ESW extends the user interface and makes it easier for the user to contribute data.

We now go over each component to describe how you, the researcher, should prepare it for your study.

2. ExtraSensory Server (ESS)

i. Installing and running ESS

All the required code for ESS resides in the ESA repository under “ESA_components/ESS”.

ESS is based on python code. It requires a mechanism to run python code in response to http (or https) requests. Here, we describe a configuration for a Linux server that uses virtualenv, uWSGI, nginx, and systemd. You will need root permissions to setup some of the required components. To setup ESS on our lab’s server, we followed tips from this website: <https://www.digitalocean.com/community/tutorials/how-to-set-up-uwsgi-and-nginx-to-serve-python-apps-on-ubuntu-14-04> with the exception that we had to use systemd instead of Upstart (because of the version of Linux Ubuntu OS that our server had). Virtualenv is a package that enables encapsulating all the required python libraries and tools for the application, without having to install them for the entire server machine. uWSGI is a component that translates between the network API (with http, https protocols) and the python application (function calls). Nginx will act as reverse proxy to redirect the relevant client requests (that come from ESP) to the ESS application through uWSGI (Apache can also have this role, but we used nginx). Systemd is the Linux mechanism to control services that should continuously run in the background.

Required software:

If your server doesn't yet have these programs, install nginx, pip, and virtualenv:

```
sudo apt-get update
sudo apt-get install python-dev python-pip nginx
sudo pip install virtualenv
```

The ESS package:

For these guidelines, let's assume that your server's domain name is "mylab.myuniv.edu" and that you, the researcher/server-administrator, have username "alice".

Place the content of the ESS package in its own directory on the server. This can be place wherever you want, even under the home directory of your personal user (~alice). You will, however, need to adjust the ownership-group of a certain file, to allow nginx (which typically runs as user www-data) to use it:

```
cd ESS
sudo chown alice:www-data start_extrasensory_server.sh
```

Prepare the environment with all the required libraries for the application (including uwsgi). The following command runs a script (only run it once) that uses virtualenv to create this environment (the first line may be necessary to allow permission to execute the script):

```
chmod u+x create_environment.sh
./create_environment.sh
```

Make sure the script runs smoothly and is able to install all the required components into the virtual environment.

***Troubleshooting:** in some systems some installations may fail due to MemoryError Exception. This happened to us when we tried to deploy ESS on an AWS (Amazon Web Server) with 1GB memory and no swap memory – the "pip install scipy" command failed. A fix that helped us in that case was to add the "--no-cache-dir" flag to the command: we edited the create_environment.sh script and changed the relevant line to be "pip --no-cache-dir install scipy". After deleting the ess_env subdirectory and rerunning the create_environment.sh script, it ran smoothly and installed all the required components in the virtual environment.

Create a directory where the uploaded data (sensor measurements and user-reported labels) will be saved. This can be anywhere you want, but you have to make sure to edit the file ess_params.json and specify this directory's path in it (the value for the field "data_dir"). You also have to change the directory's ownership-group to allow the www-data user to write files in it. In this example, we create the data directory as a sub-directory inside ESS:

```
mkdir uploaded_data
sudo chown alice:www-data uploaded_data
```

Running ESS manually:

By this point you can run ESS manually by first entering to the virtual environment that you created, it is called `ess_env`. After the “source” command (activating the virtual environment) the shell prompt should indicate that you are working within the virtual environment. Then, within the `ess_env` environment, invoke the `uwsgi` command, where you instruct uWSGI which network port to listen to and what is the callable entry point of the python application (the python module `ess_wsgi_entry.py` and the specific object inside it named `app`):

```
alice@mylab:~/ESS$ source ess_env/bin/activate
(ess_env)alice@mylab:~/ESS$ uwsgi --socket 0.0.0.0:8080 --protocol=http -w
ess_wsgi_entry:app
```

Then you can do a short test with a browser – try requesting the URL <http://mylab.myuniv.edu:8080> or the URL <http://mylab.myuniv.edu:8080/extrasensory> . Both should present a simple HTML page with a line of text, indicating that you’ve reached the ExtraSensory server. Potentially, the ESS should be ready to work with actual requests from ESP. However, maintaining an open shell terminal, occupied with the `uwsgi` command (entered manually), is not a clever way to keep the server continuously running. To finish the manual running of ESS, hit CTRL+C to stop the `uwsgi` command, and then exit the virtual environment with the `deactivate` command:

```
(ess_env)alice@mylab:~/ESS$ deactivate
alice@mylab:~/ESS$
```

Running ESS automatically:

To keep ESS running in the background continuously, without occupying a shell terminal, you can use `nginx` and `systemd`. `Nginx` will redirect any ExtraSensory-related client-requests to a particular socket and `systemd` will maintain a service running in the background that keeps uWSGI listening to that socket and connecting it to the python application. The package has template configuration files that you need to edit and place in the right directories.

For **nginx**, you need to prepare a configuration file based on the template file `infrastructure_files/nginx/extrasensory.template`, edit it to reflect the correct server domain name and local directories, and place it in the `nginx` directory that specifies the supported websites this server hosts:

/etc/nginx/sites-available/extrasensory:

```
server {
    listen 80;
    # listen 443 ssl;
    # ssl_certificate ssh/ server_certificate.crt;
    # ssl_certificate_key ssh/ mylab.myuniv.edu.private.key;
    server_name mylab.myuniv.edu;

    location /extrasensory {
        include uwsgi_params;
        uwsgi_pass unix:/home/alice/ESS/ess.sock;
    }
}
```

Notice that this template configuration file has 3 lines commented out – these will become relevant when you prepare your server for secure communication. You also have to link this configuration file to the sites-enabled subdirectory, to signal nginx that this website is operational. Then you need to reload the nginx service. Before reloading, you can do a short test with the nginx “configtest” command to make sure the configuration is valid:

```
sudo ln -s /etc/nginx/sites-available/extrasensory /etc/nginx/sites-enabled
sudo service nginx configtest
sudo service nginx restart
```

Make sure that the enabled sites defined in nginx do not have any contradiction (multiple definitions to how to handle URLs). For examples, there may be a default site defined in nginx, with a link, /etc/nginx/sites-enabled/default, to the configuration file /etc/nginx/sites-available/default, and the rules in that file may override the rules that you defined for the “extrasensory” site. If your server only needs to provide the ExtraSensory web service, you can remove the “default” site from the operational sites:

```
sudo rm /etc/nginx/sites-enabled/default
```

***Troubleshooting:** the configtest or restarting of nginx may fail due to server name that is too long. This happened to us in the AWS deployment (where the server hostname was a long string): after failing to restart nginx, we used the command “sudo journalctl -xe” to observe log messages and noticed the message “nginx: [emerg] could not build server_names_hash, you should increase server_names_hash_bucket_size: 64”. To overcome this problem, we edited the main configuration file of nginx, /etc/nginx/nginx.conf, we uncommented the relevant line and made sure to allow sever name longer than 64: “server_names_hash_bucket_size 128;”. Then, configtest was successful and nginx was able to restart.

For **systemd**, you need to prepare a configuration file based on the template file infrastructure_files/systemd/extrasensory.service.template, edit it to reflect the correct directories, and place it in a systemd directory that specifies available services:

/etc/systemd/system/extrasensory.service:

```
[Unit]
Description=uWSGI instance to serve the ExtraSensoryServer app

[Service]
User=alice
Group=www-data

ExecStart=/bin/sh /home/alice/ESS/start_extrasensory_server.sh /home/alice/ESS
```

To keep the service running in the background, invoke it with the systemctl command. You may have to authenticate your user with your own user password:

```
systemctl start extrasensory.service
```

Now, the ESS web service should be available, even when you exit the shell terminal. You can do a short test again with a browser, only now notice we set nginx to listen for the standard http port (80), so you can try the URL <http://mylab.myuniv.edu/extrasensory> and make sure you get the page with single text line.

ii. Preparing for secured communication

ESA can work with unsecured communication (http) between ESP and ESS. For that, you need to make sure that the settings on ESP specify using unsecured communication. However, it is recommended to use secured communication (https), in order to protect the privacy of the users, given that sensitive information is being communicated. The nginx configuration file template we provide already includes the instruction to listen to both the standard http port (80) and to the standard https port (443). In order for the secure communication (SSL) to work, you need to produce for your server a matching pair of private key and public certificate (with public key). You can generate these with the following three commands:

```
openssl genrsa -out mylab.myuniv.edu.private.key 2048

openssl req -new -key mylab.myuniv.edu.private.key -out
mylab.myuniv.edu.public.csr

openssl x509 -req -days 365 -in mylab.myuniv.edu.public.csr -signkey
mylab.myuniv.edu.private.key -out server_certificate.crt
```

The second command will ask you a series of questions; you can fill in some of the details for the certificate if you want. You can leave the password field blank (simply press Enter when it asks you for a challenge password). You can name the generated files what you like, but make sure the filenames match the nginx configuration file you prepared – it has an instruction specifying the private key. The private key should be copied to the appropriate location specified in the nginx configuration (in case of the template we provide, it is a subdirectory “ssh” that you can create under /etc/nginx/). The private key should be kept private – only on the server machine; preferably you should restrict the write permissions only to the root user. The public certificate (server_certificate.crt) should also be copied to the appropriate directory specified in the nginx configuration.

```
sudo chown root mylab.myuniv.edu.private.key
sudo mkdir /etc/nginx/ssh
sudo cp mylab.myuniv.edu.private.key /etc/nginx/ssh
sudo cp server_certificate.crt /etc/nginx/ssh
```

In the nginx configuration file, you now need to uncomment the lines relevant to secure communication (and make sure to edit the correct name of the private key file):

/etc/nginx/sites-available/extrasensory:

```
server {
    listen 80;
    listen 443 ssl;
    ssl_certificate ssh/ server_certificate.crt;
    ssl_certificate_key ssh/ mylab.myuniv.edu.private.key;
    server_name mylab.myuniv.edu;

    location /extrasensory {
        include uwsgi_params;
        uwsgi_pass unix:/home/alice/ESS/ess.sock;
    }
}
```

Don't forget to restart the nginx service:

```
sudo service nginx restart
```

In addition, the public certificate (server_certificate.crt) is the file that should be available to the clients of the server, so you should copy it out of the server machine and place it in a certain position in the Android code of ESP (see details in [ExtraSensory Phone \(ESP\)](#)).

iii. The collected data

Once everything is working, when using ESA, the data directory that you setup (in this example it is the subdirectory to ESS called uploaded_data) will get filled with data. Within it there should be a subdirectory for each ESP user, named by the UUID generated for that user. Within a user's directory there will be a subdirectory for every minute-instance, named with the standard timestamp (seconds since the epoch) of that instance. Every instance directory will hold the zip file (with sensor measurements) that was sent from ESP for that minute. If the user provided labels for this instance, there will also be a file called feedback.

Zip file:

ESS already uses python code to unpack the received zip file for further processing. You can use the same code (call function unpack_data_instance() in the ess_utils.py module) offline, on your own time and possibly on a different machine. The resulted directory will include textual files of the sensor measurements in a more human readable format.

Feedback file:

the file called feedback is a textual file with a JSON structure that was sent from ESP with user-reported labels. If the user sends label-feedback multiple times for the same instance, these multiple feedbacks will be appended and represented in the feedback file as a JSON array, where the last feedback is the most up-to-date.

iv. Troubleshooting and useful commands:

Command	use
<code>systemctl restart extrasensory.service</code>	Restarting the service, if you made changes/fixes to the ESS code.
<code>systemctl status extrasensory.service</code>	Check the status of the service. It should include information about the uwsgi operation, possible problems it had.
<code>cat /var/log/syslog tail -n 200</code>	Systemd directs the output of the services it runs to a log file (typically /var/log/syslog but you can change it). This output is fuller than what you see in the systemctl status command. You can see individual client-requests and responses, including messages printed by the ESS python code. This can help you debug any problems.
<code>sudo service nginx restart</code>	Restarting nginx, if you made changes to nginx configuration.

v. Customizing the classifier

The ESS package comes with a default classifier. It extracts features from six sensors from the phone and watch (same features as extracted for the public ExtraSensory Dataset), and feeds the features into a multiple layer perceptron (MLP) network to produce the probability predictions for 51 target context labels. This is the same multi-label classifier type described in Vaizman, Y., Weibel, N., and Lanckriet, G. "Context Recognition In-the-Wild: Unified Model for Multi-Modal Sensors and Multi-Label Classification". *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, vol. 1, no. 4. December 2017. The model was trained on all the 60 participants' data in the ExtraSensory Dataset. You are free to keep using this default classifier in your deployment.

You are also encouraged to suggest new models and train them on your own, and plug them into ESS. Look at the python code files inside ESS/classifiers. The file (module) `es_mlp.py` is the one that defines the default classifier type. If you want to suggest another type of classifier, you can follow these steps:

1. **Module for classifier type.** Add another module with a similar structure to `es_mlp.py`. This should include:
2. **Python class.** Within your new module you should define a class with the same name as the module that inherits "Classifier".
3. **The classify() function.** Inside the class you should implement the function `classify()` with the same input/output structure as in `es_mlp`. This function gets as input the path to a directory that holds the unpacked sensor-measurement files from an instance, and the timestamp of the instance. The function should return two lists of the same length: a list of label names (strings) and a corresponding list of probabilities (float) that the classifier assigns to the labels.
4. **Temporary measurements directory.** Notice that the input measurements directory is assumed to be just temporary, so do not save in it files that you wish to keep for later. Alternatively, you can change the code in `ess_wsgi_entry` to make it not remove the temporary directory.
5. **Utilities that are already provided.** Similar to `es_mlp`, you are free to utilize the helping functions available in the ESS package. For example, if your new classifier uses similar sensor-features as those in `es_mlp` (the features of the ExtraSensory Dataset), you can call `es_feature_extraction.get_features_from_measurements()`; if you want to use our implementation of MLP you can utilize the classes provided in `neural_networks.py`.
6. **Additional python libraries.** If your code relies on additional python libraries (e.g. `sklearn`), include them in the virtual environment by adding a proper "pip install []" line to the `create_environment.sh` script. Then re-run that script (you may have to delete the `ess_env` subdirectory first, in order for the additions to take effect).
7. **Make your classifier robust.** Keep in mind that not all sensors will be available all the time. It is possible that sometimes the user may block location services (to save battery), that the audio will not be available (while the user is on a phone call), etc. You need to make sure that ESS works smoothly in such cases. So your ESS classifier should not assume that all the sensors' measurements files are full of measurements – they may be

missing or simply have the value 'nan'. The `get_features_from_measurements()` takes care of this, and if some sensor is missing it produces nan for its features. Furthermore, it will be better if you train your model to be resilient to missing sensors, so that not only it will not crash, but it will also produce reasonable predictions in the absence of some sensors.

8. **Trained model.** If your new classifier is a dummy or a rule-based classifier, you may have all the required functionality hard coded in the python module that you add. However, typically, you would use machine learning to train a classifier with data (like the data in ExtraSensory Dataset in case you're using similar sensor-features and targeting similar context-labels). In this case, you should save all your required model parameters in a pickle file. You should create an additional subdirectory under `ESS/classifiers/trained_models/` and within it place your pickle file. Your implementation of function `classify()` should handle reading the content of the appropriate pickle file to get the required information to perform the classification.
9. **Multiple trained models.** You can have multiple trained models for the same classifier type. For example, for the default classifier type `es_mlp`, we have prepared two trained models: `ESS/classifiers/trained_models/es_mlp/es6sensors.model.pickle` is a model that relies on six sensors (including watch accelerometer) and `ESS/classifiers/trained_models/es_mlp/es5sensors.model.pickle` is a similar model that relies only on phone-sensors. The code we implemented for `es_mlp.es_mlp.classify()` can possibly get extra information (in dictionary `extra_data`) indicating the name of the trained model to use (property "classifier_name").
10. **Dynamic classifier-type and trained-model selection.** The ESP interface allows specifying which classifier type (e.g. `es_mlp`) and classifier name (e.g. `es5sensors`) ESS should use when performing prediction. The code in `ess_wsgi_entry` has a mechanism to dynamically (try to) use the classifier type that the ESP-user requests. If there is something wrong in your implementation of the new module, the client-call to upload the zip file should fail with some message. You can also read the systemd syslog to see more details and debug the problem.

3. ExtraSensory Phone (ESP)

ESP is the Android mobile phone app. The full code is available at the ESA repository and it comprises most of the code. The whole ESA package is actually an Android project and you can work on it with AndroidStudio (or Eclipse, or other programming environment).

You can use Android Studio to compile the application and to debug the app with USB connection to an Android device. After compiling the application once you can also use the generated apk file, transfer it (e.g. via email) to any Android phone, and install it without Android Studio. You have to make few adjustments in order to link ESP to your won server. In addition, you can customize the app to your study.

i. Linking to your own server

- You need to tell ESP what is the hostname of your server, where you run ESS. Inside `\app\src\main\res\raw` add a new file called `server_hostname.txt`, and in it a single line with the hostname (or IP address if there is no domain name) of your server.
- For secure communication, you need to provide ESP with your server certificate, so that when it communicates with your server, it can authenticate that it is indeed the server that it trusts. Inside the same directory `\app\src\main\res\raw` add the file `server_certificate.crt` that you generated on the server.

ii. Customize your study

Label menus:

Inside `\app\src\main\res\raw` you'll find text files that define the labels that appear in the label selection menus in ESP. You can edit them to specify your study's labels of interest. Each line has a label. Comma is not allowed in a label. For the secondary labels menu, which was designed as an extensive larger menu, the ESP provides a side-bar index to help quickly find relevant labels according to topics. In order to define these topics, in the `secondary_activities_list.txt` file, each label is followed by pipe (|) and then the relevant topics separated by commas.

Purely multi-label interface:

You can simplify the label-reporting interface. Instead of asking the user to select “main activity” and then select “secondary activities”, you can include all your target labels in the `secondary_activities_list.txt` file and treat your whole study in the multi-label setting. Then simply instruct your participants to only use the “secondary activities” field, where they can find all the relevant labels and mark multiple labels simultaneously. Then, when you arrange the uploaded data, you can simply ignore the values of “main activity” and only regard to what was reported for “secondary activities”. For example, you can add to the secondary activities list the body posture/movement states (like sitting, walking, etc.) and also mood labels.

Sensors to record:

Without changing the ESP code, you can select the sensors you want it to record. This can easily be done in the app's settings page, under the section titled “Researcher settings”. When the ESP starts running on the phone it looks for the available sensors on the phone, to prepare them for recording. This causes the lists of high-frequency and low-frequency sensors in the settings page to automatically reflect only the sensors available on the phone. You can choose which sensors to record according to the needs of the classifier that you use, for example the default classifier `es6sensors` works best when it gets measurements from location, watch, audio, raw accelerometer, and calibrated gyroscope. There are many sensors the `es6sensors` classifier does not use, so you may decide to not record them at all. By limiting the sensors that ESP records you can save power, make it run faster, and reduce the size of the communicated data (which, in turn, also saves power).

In addition, as new devices come to the market, more sensors may become available. You can change the ESP code to look for additional sensors.

Integrate location data:

If location is being recorded, the response from ESS includes the field “`location_lat_long`” that holds a

pair of numbers – these are the estimated centroid location coordinates within the 20-second window, in decimal degrees. If you wish to utilize these location coordinates, you can edit the Android code of ESA. For example: you can add an interface where you display the recent minutes as points on a map – this can help the user remember what they did at different times of the day (using their location as a cue). However, the current version of ESA doesn't have this functionality and you'd have to design and implement it.

Custom classifier:

As mentioned for ESS in [Customizing the classifier](#), you can implement new types of classifiers, and train new models, and plug them into ESS. On the ESP side you do not have to change the code. You can simply use the settings page to edit the classifier type (name of the module, like `es_mlp`) and classifier name (name of the trained model, like `es6sensors`).

iii. Using labels with another app

In case you want another app to use the server-predictions ESP gets from ESS, go to the settings page and enable “Save prediction files”. This will cause ESP to save the server predictions not only to its internal database, but also to files that are accessible to everyone (other apps on the phone or a computer connected with USB). ESP will create a designated directory in the device's storage (from the ESP'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 ESP user. Within this directory ESP will save a separate JSON file for each instance, named with the timestamp of the instance. Each file will hold the list of labels and list of corresponding probabilities that were predicted by the server for this instance.

To view the file's content on a USB connected Windows computer, use the adb tool (which is installed with Android Studio). For example, from a Windows command prompt (cmd):

```
adb -d shell ls -l
storage/emulated/legacy/Android/data/edu.ucsd.calab.extrasensory/files/Documents/extrasensory.labels.2B935B2/

adb -d shell cat
storage/emulated/legacy/Android/data/edu.ucsd.calab.extrasensory/files/Documents/extrasensory.labels.2B935B2/1498164507.server_predictions.json
```

In addition, you may want to allow the user to correct the server predictions and provide their own reported labels. If you wish to access the user-reported labels with another app, turn on the “Save user-labels files” option in the settings page. It will create publicly accessible files in a similar manner to the server-prediction files, and inside each file will be a list of the labels reported by the user (combining

main-activity, secondary-activities, and moods) for the particular minute.

```
adb -d shell cat  
storage/emulated/legacy/Android/data/edu.ucsd.calab.extrasensory/files/Documents/extrasensory.labels.2B935B2/1498164507.user_reported_labels.json
```

Accessing label files from another app:

To access this directory from another Android application (let's call it OtherApp), you need to declare permission to read external files – in the app manifest of OtherApp, include the line:

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

In the java code of OtherApp, in order to read the server-predicted, user-reported labels, or representative location coordinates, for a particular user UUID and minute timestamp, you can use functions like the following:

```

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;
    }
}

```

```

/**
 * Read text from the label file saved by ExtraSensory App, for a particular 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;
    }
}

```

```

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";

/**
 * Parse 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, ESP sends broadcast messages whenever it saves server-prediction files, so you can register a broadcast receiver in your app, OtherApp, to listen to the specific intent-action from ESP:

"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, "registering 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();
}
```


4. ExtraSensory Watch (ESW)

ESW is a small component that runs on the Pebble watch. It enables both collecting recorded measurements from the watch and extending the user interface for convenience. The source code (and a pre-compiled version) for ESW are available on the ESA repository under “ESA_components/ESW”. In order to combine ESW in your deployment follow these instructions.

i. Prerequisites

- Get a Pebble watch. ☺
- 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.

ii. Installing a compiled version

- The ESW package includes a compiled version of ESW. You can transfer this pbw file to the phone (e.g. via email), open it (the phone should identify that this file should be opened with Pebble app), and select to load this app to the watch.

iii. Cloud Pebble

- In your CloudPebble account you can create a new project, and simply add to it the single code file that ESW requires.