CSC4006 (2020-21)

Software Development Report

Towards ‘Her’: An AI that Understands You

Eoin Lyness

40204917

Contents

1. Introduction

This document provides an overview of the software developed for the project *"Towards Her: An AI that Understands You"*. In this document we describe the external software required to run the developed software, an overview of the core source files and an outline of each of the main components that make up the system, describing their functionality and role within the system. We also briefly discuss some of the other software work that was used as part of our research but is not a part of the final system.

The system can be considered in two distinct parts. The first of these is an Android application for obtaining the necessary data about the user from their device to be used within the system. The other is a number of Python scripts which perform the core functionality of the system: the processing of this data and training of the model. Additionally, this part of the system includes software for generating a synthetic dataset for performing the experiments.

2. Setup

The programming languages used in developing this software are Java and Python. To build and compile the Android source code, Android Studio should be installed which will also install the necessary dependencies. For running the Python components, a number of non-standard Python modules are required which can be installed through a package manager such as pip. These are as follows:

- *pandas*
- *numpy*
- *torch*
- *syft*
- *fastai*
- *sklearn*
- *scipy*

3. Overview of Files

This section provides a brief description of the core files in the software produced. Further details of the different components of the system will be described in the following section. Figure 1 provides the directory structure for the project highlighting these files. Folders are highlighted in bold, and an asterisk denotes files and folders relating to experimental work that is not part of the final system.

ActivityMonitor

This folder contains the source code for the Android application which monitors and records user data from their device.

- **ActivityMonitor**
  - …
- **data**
  - …
- **misc ***
  - **KotlinSyft ***
    - …
  - **PyGrid ***
    - …
  - get_history.py *
  - pygrid_model_setup.py *
- **models**
  - …
- **output**
  - …
- **params**
  - labels.json
  - people.json
  - variables.json
- **templates**
  - …
- cluster.py
- collab.py
- collab.ipynb
- collab_privacy.py
- collab_privacy.ipynb
- combine_logs.py
- format_log_data.py
- generate_log_data.py
- run.bat
- run.sh

** denotes files relating to experimental work that is not a part of the final system*

Fig. 1. Directory overview for the project highlighting key files and folders

data

This folder is the output folder for the synthetic log files produced for the experiments.

misc

This folder contains additional source code used for experimental work which does not form a part of the final system

> ***KotlinSyft*** – This folder contains the KotlinSyft source code which allows PySyft models to be trained on an Android device. It also contains the KotlinSyft demo application for demonstrating the functionality of this library on an Android device.
>
> ***PyGrid*** – This folder contains the PyGrid source code which allows PySyft models to be hosted on a server for carrying out federated learning on user devices.
>
> *get_history.py* – This Python script retrieves and displays the machine's web history from Google Chrome for the past day.
>
> *pygrid_model_setup.py* – This Python script creates and hosts a PySyft model on the local machine using PyGrid.

models

This folder is the output folder for the fully trained non-privacy preserving model from the experiments.

output

This folder is the output folder for the csv files produced by the system, containing the clustered and non-clustered data and the results from training the models

params

This folder contains a number of JSON files indicating the parameters used for generating synthetic data to train the system

> *labels.json* – This file contains plain English descriptions of each of the ID labels used in the data. This file is not used by the system and exists solely for descriptive purposes.
>
> *people.json* – This file contains details of each of the user profiles used for the synthetic data, containing their names, contacts and mood ratings for each activity

variables.json – This file contains details of each of the possible values that may appear for each activity in the synthetic data

templates

This folder contains the template JSON files used for generating the synthetic log files

cluster.py – This Python script performs the clustering algorithm on the data

collab.py – This Python script performs the training of the non-privacy-preserving collaborative filtering model

collab.ipynb – This Jupyter notebook contains the identical Python code as in collab.py allowing different sections of the code to be executed separately

collab_privacy.py – This Python script performs the training of the privacy-preserving collaborative filtering model

collab_privacy.ipynb – This Jupyter notebook contains the identical Python code as in collab_privacy.py allowing different sections of the code to be executed separately

combine_logs.py – This Python script combines each of the log files in the data folder into a single discretised dataset

format_log_data.py – This Python script takes a single log file and organises it by hour

generate_log_data.py – This Python script produces the synthetic data to be used for the experiments

run.bat – This script executes the individual components in turn for running the experiments on a Windows-based system

run.sh – This script performs the identical steps as run.bat for Unix-based systems

4. Overview of System Components

This section elaborates on the software in further detail providing an overview of the core components in the system and their functionality.

4.1 Monitoring Software

Source files: ActivityMonitor

This component has been implemented as an Android application and is the main software that will be run by the user. The application runs unobtrusively in the background on the user's mobile device, continuously recording data about the user to be output to a log file in JSON format, with a separate file for each day. The data recorded includes the user's application usage, internet browsing history, phone calls, texts and GPS location details. The data is obtained using the various tools and services available within the Android SDK.

The code for this application was adapted from [1].

Fig. 2. Structure of the JSON log file produced by the monitoring software for each day

Overview of Log File

The structure of the output log file produced by the software is shown in Figure 2. Here we provide details of each component of the log and the values stored within it:

appInfo

This list contains information about each application on the device that was used by the user during the current day. This information is obtained using Android's usage manager service to query the device's recent application usage. Each time a new application is used it is appended to the list – if the current application is the same as the last one added to the list then it is ignored.

- *time* – Timestamp of when the application was opened in HH:mm:ss format
- *applicationName* – The name of the application i.e "My Application"
- packageName – The package name of the application i.e "com.example.myapplication"

callList

This list contains information about each call made or received on the device during the current day. This information is obtained using Android's content observer service to detect phone calls made and received by the device.

- *callLength* – The length of the call in seconds
- *missed* – Boolean value indicating whether the call was missed
- *name* – The name of the contact if the phone number is stored on the device. If no contact exists then this field contains the phone number
- *number* – The phone number for the call
- *outgoingCall* – Boolean value indicating whether the call is outgoing
- *time* – Timestamp of when the call was made or received in HH:mm:ss format

path

This list contains information about each location visited by the user during the current day obtained from the device's GPS service. The list is updated each time the device detects that the current location has changed

- *latitude* – The latitude coordinate for the current location
- *longitude* – The longitude coordinate for the current location
- *time* – Timestamp of when the location was visited in HH:mm:ss format

places

This list is updating in parallel with the previous list and contains further details about each location visited by the user.

- *address* – List containing the address lines for the closest address corresponding to the coordinates of the current location. This is obtained from Android's geocoding service allowing coordinates to be transformed into a street address.
- *status* – "outdoors" if the user is detected to be currently outdoors and "indoors" if the user is detected to be currently indoors. The value is determined based on the current precision level of the GPS signal – a higher precision is presumed to be outdoors and a lower precision is presumed to be indoors
- *time* – Timestamp of when the location was visited in HH:mm:ss format

smsList

This list contains information about each SMS message received by the user during the current day. This information is obtained using Android's content observer service to detect SMS messages received by the device

- *body* – The contains of the SMS message
- *from* – The phone number from which the SMS was received
- *name* – The name of the contact that the SMS was received from if this is saved on the device, otherwise this field contains the phone number
- *time* – Timestamp of when the SMS was received in HH:mm:ss format

smsOutList

This list contains information about each SMS message sent by the user during the current day. This information is obtained using Android's content observer service to detect SMS messages sent by the device. The fields are identical to that of smsList.

urls

This list contains information about each website visited by the user during the current day. Each time a new website is visited it is appended to the list – if the current website is the same as the last one added to the list then it is ignored. The URL is obtained by using Android's accessibility tools to read the text content of the browser application's address bar.

- *time* – Timestamp of when the website was visited in HH:mm:ss format
- *url* – The URL of the current website

4.2 Log File Segmentation

Source file: format_log_data.py

This Python script takes a log file as input and divides the continuous log file for the day into one-hour blocks based on the timestamp of each of the activities. The output is a new JSON file, shown in figure 3, with keys corresponding to each hour of the day i.e 0...23, and within

Fig. 3. Structure of the formatted JSON log file organised by hour

each key is the same structure of the initial log file but containing only the entries that occurred within the given hour.

4.3 Synthetic Data Generation

Source file: generate_log_data.py

This Python script forms the basis of our experiments and is used to generate synthetic log files to be used as training data to evaluate the system. The synthetic data is generated using a number of template log files combined with a series of parameters. A total of 1000 log files are generated by the program to be used in the experiments.

The template files contain the hourly structure that is produced by *format_log_data.py* with additional fields containing labels for the data. Each hourly block contains the fields *activity* and *mood* with the value of *activity* being an ID value corresponding to the overall activity label for that hour and *mood* containing the user's mood rating for that hour. In addition, each individual activity contained within the hour has a field containing a label. This additional field is *activity* for the entries in appInfo, and urls, and *id* for the remaining lists. For the path and places lists the label contains an ID corresponding to the location such as being at home or going for a walk. For callList, smsList and smsOutList the label contains an ID corresponding to the person the call or text relates to such as whether it is a friend or family member.

To fill out the template files a list of parameters *variables.json* is used which contains the possible values associated with each label that may appear in the generated data. In addition to this, a set of user profiles *people.json* is used containing the details of each synthetic user for which log files will be generated. This contains a list of mood ratings relating to the given user for each of the activity labels as well as details of their contacts to appear in the calls and texts.

For each log file that is generated, a user is selected at random from *people.json* to be the user for which the current log file belongs to. For each of the 24 hour blocks, the structure of the corresponding hour is chosen at random from one of the template files. For each activity contained within the hour block, the label for the activity is used to select a random value from *variables.json* for that ID and this is used to fill the appropriate fields in the log. The user's rating for that activity is retrieved from *people.json* and added to a sum of the ratings for that hour. When the end of the hour is reached the total sum of the ratings is used to calculate the overall rating for that hour, with 1 being assigned if the sum is >=1, -1 if the sum is <= -1 or 0 otherwise. The resultant log file is then output to a JSON file with the naming format *log_<number>_<person>.json*

The identifiers used in labelling the data are as follows:

Mood

- *1 – Happy*
- *0 – Neutral*
- *-1 – Unhappy*

Activity

- *11 – At home*
- *31 – Going for a walk*
- *101 – Call/text to/from friend*
- *102 – Call/text to/from family*
- *110 – Sleeping*
- *1110 - Working*
- *3721 – Shopping*
- *5140 – Phone call*
- *5141 – Text message*
- *7190 – Browsing/taking photos*
- *7231 – Browsing internet*
- *7241 – Social media*
- *8220 – Watching videos/movies*
- *8320 – Listening to music*

The activity identifiers were adapted from the United Kingdom Time Use Survey 2014-2015 [2].

4.4 Processing of Log Files

Source file: combine_logs.py

This Python script is used to combine each of the formatted log files into a single tabular dataset to be used to train the system. The dataset contains the following columns:

- *id* – ID of the row
- *user* – Name of the user
- *activity* – ID label corresponding to the activity for the hour
- *data* – Log data for the hour as a JSON string
- *mood* – Mood rating for the hour

For each log file in the data folder, each one-hour block is extracted, and the details are added as a separate row to the dataset. The resultant dataset is saved in csv format in the output folder.

4.5 Clustering of the Data

Source file: cluster.py

This Python script executes the clustering algorithm used to cluster each of the one-hour blocks contained in the dataset according to their relative similarity. The program uses a k-means clustering algorithm with a custom distance function implemented for comparing the data. The k-means algorithm initially takes a random sample of the data which calculates the initial cluster centres using the distance function. These initial centres are then used to perform the algorithm on the entire dataset. The resulting cluster that each hour belongs to is appended to the dataset as an additional column *cluster*. This updated dataset is also saved in csv format in the output folder.

The distance function compares a one-hour block X to another Y by comparing the labels for each of the activities in turn and keeping a total of how many of them are identical in both X and Y. If several consecutive activities in either X or Y are identical, only one of these is counted towards the total. The comparisons stop when the end of the shorter of X and Y is reached and a percentage of how many activities were identical across X and Y is calculated. This similarity is subtracted from 1 to obtain the distance between X and Y as a value between 0 and 1, with 0 indicating that X and Y are identical, and 1 indicating that there is no similarity between the two.

The code for this script was adapted from [3].

4.6 Training the Collaborative Filtering Model

The clustered synthetic data is used to train two collaborative filtering models for our experiments. One, non-privacy-preserving, implemented using the fastai library and the other, privacy-preserving using the PySyft library. In both cases, the data is split into a training and validation set using an 80/20 training/validation split. Both models are trained for 50 epochs using a batch size of 64, with both training and validation MSE loss calculated after each epoch as well as the training time for each epoch. This data is written to a csv file in the output folder at the end of the 50 epochs.

Non-privacy Preserving Model

Source files: collab.py/collab.ipynb

This Python script executes the training on the non-privacy-preserving fastai model. The model is created using fastai's collaborative filtering module, *collab*, which allows an effective collaborative filtering model to be built and trained very easily in just a few lines of code. This training uses a dynamic learning rate which is increased with each epoch.

Privacy-preserving Model

Source files: collab_privacy.py/collab_privacy.ipynb

This Python script executes the training on the privacy-preserving PySyft model. Where the fastai library benefits from having a built-in collaborative filtering object for creating the model which handles all of the complexities, creating the PySyft model involves significantly more complex code as many of the necessary functions must be written manually.

The model is created as a neural network object which is trained by simulating federated learning. Virtual workers are created corresponding to each user in the dataset which represent what would be each user's local device in a real world scenario. The data is distributed among each of these workers to be used in the training of each worker's local copy of the model. On each training iteration, the global model is sent to each worker to perform local training on their subset of the data. When each worker completes the training iteration, its updated weights are sent back to the central server which averages the weights of the individual workers to update the global model. This training uses a constant learning rate of 0.1.

The code for this script was adapted from [4].

4.7 Running the Experiments

Source files: run.bat/run.sh

Executing this script will run the experiments for the system by executing the individual component scripts in turn. The steps performed are as follows:

1. Generate synthetic log files (generate_log_data.py)
2. Discretise log files into dataset (combine_logs.py)
3. Cluster the dataset (cluster.py)
4. Train the non-privacy-preserving model (collab.py)
5. Train the privacy-preserving model (collab_privacy.py)

Note: To ensure consistency in the synthetic data produced and the experimental results, the random seed 4006 is used within each of the scripts where appropriate

5. Testing

Due to the novelty of the system, the primary approach to testing was through trial and error. Testing of the software carried out continually whilst developing each of the components, removing any bugs that were found and ensuring that the code produced was correct and functioning as intended. One approach to this was through a number of experimental Python scripts which were written and used to test the different components of the system throughout the development. This included isolating particular functions or sections of the code to test their correctness, as well as testing the components with multiple different values for any parameters, where appropriate, in order to obtain optimal results.

As each component of the system is executed in a particular order, the functionality of one part of the system is dependent on the outputs of those which are executed previously. Because of this it was essential that each component was tested extensively to ensure its correctness before moving onto develop further parts of the system. Figure 4 describes a test plan for each completed component of the system to ensure its correctness in terms of functionality and outputs produced.

Test Number	Component	Description of Test	Expected Result	Outcome
1	ActivityMonitor	Run the application on an Android device. Perform various activities on the device such as launching other applications, browsing the internet, sending a text message etc	Log file for the current date is saved on the device. Contents of the log matches the activities performed on the device.	Success
2	format_log_data.py	Run the script using the log file produced by the previous test as input.	Log file is divided into each hour of the day. The activities from the original log file are contained within the appropriate hour.	Success
3	generate_log_data.py	Run the script. No inputs required.	1000 synthetic log files are generated. The logs contain the appropriate details for each activity.	Success

4	combine_logs.py	Run the script after generating the output from the previous test.	A csv dataset is produced containing the details of each hour of each log file as separate rows.	Success
5	cluster.py	Run the script after generating the output from the previous test.	A csv dataset is produced containing the original dataset with an additional column containing the cluster assigned to each row. The data within a particular cluster is similar.	Success
6	collab.py	Run the script after generating the output from the previous test	A csv file is produced containing training and validation loss values as well as the training time for 50 epochs. The loss values on average do not increase over time and follow a similar trend.	Success
7	collab_privacy.py	Run the script after generating the output from the previous test	A csv file is produced containing training and validation loss values as well as the training time for 50 epochs. The loss values on average do not increase over time and follow a similar trend.	Success
8	run.bat/run.sh	Run the script. No inputs required.	Each component is executed in turn producing the same outputs as in the individual tests.	Success

Fig. 4. Test plan for each component of the system

6. Other Technical Work

Throughout the project other software was explored to inform and enhance the research. The main focus of this work was within the PySyft framework which involved exploring different components to gain an understanding of how the initial non-privacy-preserving system may be transformed into one which does preserve privacy. Two PySyft components were explored; PyGrid and KotlinSyft.

PyGrid [5] is the central server architecture within PySyft, allowing PySyft models to be hosted on a server to allow federated learning to be carried out across devices. The model

hosted on PyGrid is the global model which is sent to each of the devices on each training iteration. The individual local updates from each device are sent back to the PyGrid server where they are aggregated to update the global model. A Python script *pygrid_model_setup.py* was produced, exploring the setup and deployment of a PyGrid server on a local machine, as well as creating a PyGrid model and hosting it on this PyGrid server.

KotlinSyft [6] is a Kotlin-based library for Android devices, which allows PySyft models to be accessed via PyGrid and trained on an Android device. The KotlinSyft demo application was explored to gain an understanding of the library and this was used to access and train the model that was deployed on the local PyGrid server.

An additional piece of experimental software produced was the Python script *get_history.py* which was developed during the exploration of how to record data from a user to be used by the system. This function of this script is a means of obtaining and viewing a desktop machine's Google Chrome web browsing history by accessing the SQLite database stored by Chrome on the device. This script did not become a part of the final system as it was instead decided to monitor the user's browsing history via the Android application developed, allowing all of the data to be recorded from a single piece of software, instead of several programs on different devices.

7. Evaluation & Future Work

The main area for future work is in taking the fundamental components that have been explored here and transforming these into a fully privacy-preserving system. To ensure the effectiveness of any privacy-preserving implementation of the system it is vital to adhere to the principles of privacy by design proposed by Cavoukian [7], which state that such a system should have the following attributes:

1. *Proactive not reactive; preventive not remedial*
2. *Privacy as the default*
3. *Privacy embedded into design*
4. *Full functionality – positive-sum, not zero-sum*
5. *End-to-end security – lifecycle protection*
6. *Visibility and transparency*
7. *Respect for user privacy*

From the investigative work carried out it appears that using the PySyft framework to implement this privacy-preserving system would be the most promising approach, although there may be other privacy-preserving machine learning frameworks such as TensorFlow Federated which are worth exploring and comparing to the PySyft framework.

Considering a federated learning architecture for the system, the collaborative filtering model would be hosted on a central server which would then be accessed and trained locally by each user's device using their own data stored on their device. To preserve privacy and ensure that the user's data never needs to leave their device, the clustering algorithm would also need to be hosted and trained in a federated learning manner.

Despite the user's data never leaving their device, it should still be protected within their device in order to prevent a third party from potentially accessing it. An approach to this

would be to encrypt the data on the device and only decrypt it when it must be shown to the user in order to label it. Ideally a future system should be able to learn to classify what activities are being performed by the user, such as through the use of segmentation on the log data produced. Doing so would remove the need for the user to manually label each activity and so the data can remain permanently encrypted.

Another area for future work is in expanding the kinds of data that is recorded from the user in order to further enhance the level of understanding that the system can have about the user and consequently the level of detail and accuracy in its potential recommendations for the user. This additional data could be obtained through the use of APIs for different services used by the user such as email, social media and streaming services.

Further work should also be carried out in relation to how the system learns about its users. This may involve investigating learning through different lengths of time windows instead of only one-hour blocks. For example, larger time windows could be explored in order for the system to learn a longer term picture about the user's life. Additionally, further investigative work should be carried out into the robustness of the system. In this work we focus on a small number of core activities in the training data due to the inability to gather data from real users and indeed creating a more complex synthetic dataset would still require real data as a reference point. Where it is possible to obtain real data, future work should seek to investigate the behaviour and effectiveness of the system using a dataset that is more complex and representative of the vast range of data that may be obtained from real users.

8. References

[1] Miro382, "My Spy Android," *GitHub*, 2019, [Online] Available: https://github.com/Miro382/My-Spy-Android

[2] J. Gershuny and O. Sullivan, "United Kingdom Time Use Survey, 2014-2015," *UK Data Service,* 2017, [Online] Available: http://doi.org/10.5255/UKDA-SN-8128-1

[3] "Is it possible to specify your own distance function using scikit-learn K-Means Clustering?," *Stack Overflow*, 2011, [Online] Available: https://stackoverflow.com/a/5551499

[4] tusharsoni08, "Federated Recommendation System," *GitHub*, 2020, [Online] Available: https://github.com/tusharsoni08/federated-recommendation-system

[5] OpenMined, "PyGrid," *GitHub*, 2018, [Online] Available: https://github.com/OpenMined/PyGrid

[6] OpenMined, "KotlinSyft," *GitHub*, 2020, [Online] Available: https://github.com/OpenMined/KotlinSyft

[7] A. Cavoukian, "Privacy by Design: The 7 Foundational Principles," *Information and Privacy Commissioner of Ontario, Canada*, Toronto, ON, 2009.