Recognizing and Rewarding Intentionality in Personal Fitness Tracker Usage

Caroline Jaffe
B.S. Candidate in EECS

May 2, 2013

Advisors:

Brian Scassellati, Yale University, Department of Computer Science
Roman Kuc, Yale University, Department of Electrical Engineering

Abstract

The purpose of this project is to deliver intelligent, personalized fitness feedback in an accessible and engaging form. Though many fitness trackers allow users to view their data, it is often presented through web applications that require the user to not only proactively access them, but also undertake their own analysis of the data. In this project, I extract time series fitness data from FitBit, a portable personal fitness tracker, and analyze the data with Machine Learning techniques to determine which activities are planned or intentional fitness efforts, and which are the incidental results of the user's daily experiences. I report these insights via an embodied personal robot, which delivers personalized coaching feedback to the user in a way that aims to congratulate and encourage them.

Contents

1 Introduction

1.1 Overview

The purpose of this project is to provide intelligent, personalized fitness feedback in an accessible and engaging form. Many personal fitness trackers have some limited data analysis functionality. They might stream data to a web application and present some rudimentary graphs that show the user's activity over the past week. While these web applications offer a valuable start, this project aims to make better use of the data by understanding what forms of coaching or encouragement give rise to such data and using that understanding to influence the user's fitness behavior.

Broadly, this project uses Machine Learning techniques to distinguish between intentional and incidental fitness activities, as recorded by FitBit, a portable and personalized fitness tracker. Intentional activites include those where the user has decided to go on a run or undertake a period of sustained exercise. These periods are usually marked by consistently high step and calorie counts. Incidental activity is that where the user is walking to work, class or around the home, and is marked by lower step counts and more eratic activity patterns. The user's activity is compared to their activity history; the system then selects a behavior such as "congratulate" or "reprimand" from a library of actions, and sends those actions to an embodied robot. For this project, I used the Aldebaran Nao robot, because it was easily programmable and of an appropriate size for a personal embodied robot. Though the scope of my project did not include human testing, the objective of my project was to create a project that would effectively engage users and successfully influence their behavior so that they exercised more.

1.2 Background

This project combines a number of different research techniques and integrates notable ideas from several different fields. This project obviously builds upon the existing personal tracker software in that it analyzes and reports a user's fitness data. However, unlike these existing software applications, this project also seeks to incorporate robots to influence human behaviors in a meaningful way. Human-robot interaction (HRI) is an open and active area of research, and the factors that contribute to robot influence–the content of it's speech, tone, pose, etc–have been a subject of particular interest lately at HRI labs around the country including the Yale Social Robotics Lab, Willow Garage, and the Personal Robots group at the MIT Media Lab.

This project is loosely based on a robotics project called Autom from the MIT Media Lab. Autom is an embodied physical fitness coach that learns about its owner's fitness and eating habits. However, Autom operates by having the user input data and then interpreting it, instead of automatically receiving the data and reporting it to the user. To build upon the ideas of Autom, my project sought to automatically communicate the fitness data to the robot, so that the

user does not need to physically input any information or data on a regular basis. My system also aims to present the data in a more contextualized and personalized fashion.

My project builds upon the idea–developed by researchers at Yale and other universities–that the embodied physical presence of a robot is incredibly important in establishing a trusting and durable relationship between the user and the robot. One study from the Yale Social Robotics Lab, for example, showed that the embodied presence of a robotic tutor helped increase students' cognitive learning gains while learning a simple task.

Finally, and on a personal note, this project combines many areas of study and interest. I have been an avid FitBit user for 9 months; I have worked in the Yale Social Robotics lab for almost two years; I am currently enrolled in a Machine Learning class where I learned techniques used in this project; I have worked on and enjoyed a number of projects that require combining a number of moving parts into a working system. By tying together all of these pursuits and interests, I felt this project would be suitable to conclude my studies at Yale.

2 Equipment, Tools and Project Components

- 1 FitBit One Wireless Activity and Sleep Tracker
- Access to Partner Level FitBit API (by special request)
- Use of Google Apps Script API and Google Data API to fetch FitBit data and download it to personal machine
- R script implementing Machine Learning classification algorithm
- 1 Aldebaran Nao humanoid robot, and accompanying software

3 Procedure

This project required the integration of a number of different, complex systems. As is often the case, I spent a significant amount of time making sure that the different pieces of my project worked together smoothly in a way that could be easily understood by a user. The program can be run from a single Python script, `main_script.py`, that instructs the user, at certain points, to facilitate data syncing and authenticate access to the Google spreadsheet. Once the FitBit data has successfully synced to the computer, it is downloaded to a Google spreadsheet via a customized Google Apps Script. The main Python script, `main_script.py`, then makes calls to R for the Machine Learning procedures and sends behaviors to Nao robot using the Aldebaran SDK. The project is best conceptualized in terms of the following three stages: data collection, data analysis, and data reporting.

3.1 Data Collection

FitBit and Partner API

The data I used for my project was collected from the FitBit One Wireless Activity and Sleep Tracker. The FitBit One is a small device that can be clipped onto a user's clothes and uses an accelerometer to count and report how many calories the user has burned, how many steps taken, how many miles traversed, and how many flights of stairs climbed, among other personal statistics. This information is synced wirelessly to the user's online FitBit profile when the user comes within 20 ft. of a wireless base station.

FitBit also provides an API so that developers can access user data by making an HTTP POST request to FitBit's servers. The main FitBit API provides data with daily granularity, but a Partner API provides access to intraday data with minute-by-minute granularity, and allows the developer to request information about certain time ranges of activity. I was granted special permission to work with this Partner API so that I could collect users' minute-by-minute daily activity data. While the regular API provides day-by-day access to a wide range of statistics, including sleep times, food logs, and body weight, the intraday requests are only valid for collecting data on calories burned, steps taken, floors climbed, and elevation gained. Despite these restrictions, I felt these data would stil provide useful insights about the user's activity that would be sufficient for the scope of this project. Data was returned in JSON format.

In order to collect data that I could explore with Machine Learning algorithms, I kept a log of the exact times when I had intentionally exercised over the course of a week. In this way, I developed a training dataset where I knew the expected classification (ie, intentional or incidental) of each minute. Realistically, I found that it was easier to detect runs, which are long stretches of sustained, elevated activity versus the short bursts of activity that one might experience at a sports practice.

Google Apps Script and Spreadsheet

I used a customized Google Apps Script that ran on top of a Google spreadsheet to access and organize the FitBit data. Google Apps Script is a cloud-based scripting language that is based on Javascript and gets executed in the Google cloud. This customized script used the OAuth authentication protocol to access a user's FitBit data, and then filled in a spreadsheet with that data. Once I had the FitBit data in an easily accessible and nicely visualized form, I ran another OAuth authentication from within a desktop Python script to download the Google spreadsheet to my machine as a csv and explore the data using Machine Learning techniques.

OAuth is an authentication protocol that allows users to access server resources on behalf of a resource owner (possibly another end-user), or allows an end-user to authorize third-party access to their data on a server without revealing their username and password. In OAuth, you must first generate an application consumer key and secret key that will uniquely identify a certain

application to the server. These keys are passed to the server through an HTTP POST request, which will prompt the user to authenticate the request. If authenticated, the request will return an access token and an access secret, which will allow that application to access the requested resources. OAuth thus prevents the user from having to reveal their real login credentials, and ensures that only a particular application will be able to access a user's server data.

Using the Google Apps script added another layer of complexity to my project, because it requires OAuth authentication in the script to access FitBit data, as well as OAuth authentication from the desktop in order to access what was now Google data. I initially decided to use the Google Apps script because it offered a quick, easy way to access and visualize the FitBit data. Due to time constraints and the fact that the Google Apps script was already integrated into my project, I was unfortunately not able to modify my project to make this step more efficient. This Google Apps script is included in the accompanying code under the name `fitbitdownload.gs`. In order to fetch the FitBit data after a sync, the user must open the Google spreadsheet and wait for it to sync; he or she is prompted to do so at the beginning of the main Python script.

3.2 Data Analysis

Once the FitBit activity data was successfully downloaded to my computer, I used Python to run an R script, `fitbit_script.r`, as a subprocess. The data was in a csv with one column for the date in the form *mm/dd/yyyy*, one with time in the form *hh:mm*, one with step counts, one with calorie counts, one with floor count, and one with elevation. Data had minute-by-minute granularity. The purpose of this R script was to determine and report the number of minutes of intentional exercise per day.

This R script implemented Random Forests, a classification and regression algorithm that I became familiar with in my Machine Learning class. A classification method is one that tries to categorize data observations into two or more classes; in this case, I was trying to distinguish between activity that was intentional exercise, and activity that was incidental. In order to run these Machine Learning algorithms and develop an appropriate model, I needed a set of training data which was already labelled with the correct classifications. I obtained this training set by keeping track of my own exercise over the course of a week. I now have a classifier, `forest.RData`, that is trained to my exercise style and should be able to distinguish between my intentional and incidental exercise.

Classification Using Random Forests

Random forests is a Machine Learning procedure that can be used for both classification and regression, though here it was used exclusively for classification. The procedure is conducted by growing a number of decision trees and outputting the mode classification of all the trees: "decision by committee".

That is, if 450 of 500 trees classify a certain observation as "Class A", then that is the classification assigned to that observation in the algorithm.

Since individual trees can be quite noisy, combining the output of many trees is a useful way to increase accuracy. There are several ways to combine individual trees, including bagging (bootstrap aggregation). Bagging, however, generally results in higher variance, so Random Forests tries to improve upon this procedure by minimizing the correlation between trees. The variance of B independent, identically distributed trees is given as:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

In both Random Forests and bagging, the second term falls away as B grows. Thus, the procedure variance is dependent on the correlation between the trees, ρ. To minimize the correlation, Random Forests choose input variables randomly. Specifically, before each split, $m \leq p$ of the input variables are selected at random as candidates for splitting. Typical values for m are $\sqrt{p}$ or as low as 1.

I used the training data to grow a random forest object, `forest.RData`, which I saved and used to predict classification values for any subsequent data: "test data". To gauge effectiveness of the random forest, I calculated the misclassification error rate, or the proportion of incorrect classificatons. I used the randomForest library in R to implement this tuning procedure and prediction, which can be seen in the accompanying code in `fitbit_script.r`. In order to run the method as a classification, the response vector argument must be entered as a factor.

3.3 Data Reporting

The final portion of this project consisted of reporting these algorithmically-derived insights on exercise habits in a way that encourages users to maintain (or intensify) their exercise regimen. I used the Aldebaran Nao humanoid robot to communicate with the user, because the Nao is a well-documented and appropriately sized robot that offers a wide range of actions. (I had initially planned to use the Dragonbot, but at the time of execution, felt that the Nao would be simplier and equally sufficient for my purposes). It is important that the Nao is a physically embodied robot, instead of a voice or computer program; one of the main premises of this project is that an embodied presence can give rise to more significant changes in the user's behavior.

Behaviors

I developed a library of behaviors for the Nao robot that were meant to encourage users to make positive fitness decisions. If the user exercised more than 20 minutes in a day, or more than the average of the previous week, they were congratulated with phrases randomly selected from the *congratulate* library and a celebratory cheering motion. If the user exercised fewer than 10 minutes, the

robot gently suggested that they get some exercise, with phrases selected from the *reprimand* behavior library. The Nao also would make a scolding motion. If the user broke their personal record, or if it seemed like they had not synced their FitBit in a while, the Nao robot would also comment on that.

Broadly, the behaviors sought to present the robot as a social presence who understood the context of a user's life and exercise habits. One of the greatest barriers to successful personal fitness tracker usage has been engagement; users are initially very excited by the trackers, but their usage trails off steeply as the sheen of the new product wears off. With the embodied presence of a social robot, the hope is that the insights available from the fitness tracker data will continue to seem relevant and that users will stay engaged with the system.

Aldebaran NAO Robot

The Aldebaran Nao is an autonomous, programmable humanoid robot produced by the French company Aldebaran robotics. The robot was available for my use through the Yale Social Robotics Lab. The robot can be programmed through proprietary software called Choreograph, which allows a user to connect wirelessly to the robot and send behaviors to the robot. Aldebaran also provides a Python module, `naoqi`, which allows users to send behaviors to the Nao from Python. I took advantage of both of these pieces of software in writing my project. I used keyframing in Choreograph to develop motions on the Nao which I then exported to and ran from Python. Then, I used the `ALProxy` from `naoqi` to instantiate a speech module and a motion module with the Nao's IP address and appropriate port. These modules allow behaviors to be sent to the Nao through a few lines of code. The general flow for using Nao in this system is as follows:

- Connect laptop to the wireless network that the Nao is on (or will be on)
- Turn the Nao on and listen for its IP address
- Use Choreograph to connect to the appropriate Nao through the *Connect* menu
- Hard-code the Nao's IP address into the NAO_IP variable in `main_script.py`
- Set the NAO flag in `main_script.py` to 1, which runs code that instantiates the Nao behavior modules
- Use Choreograph to enslave the Nao motors (make sure to un-enslave them after script has run so that they do not overheat)
- Run `main_script.py` to interact with the Nao

Unfortunately, the interaction with the Nao is not perfectly automated, which precludes the seamless flow of this system. However, this setup process is part of the reality of current robotics research technology, and I believe approximates the behavior of the system in a way that is comprehensible and illustrative.

4 Results and Analysis

This project successfully integrates a number of moving pieces to create a system that delivers intelligent, personalized fitness feedback in an accessible and engaging form. A user's data is synced to their online FitBit profile when they come within 20 ft. of a base station, which is usually attached to their computer or smartphone. The user can run a Python script, `main_script.py`, which will prompt them to open a Google Spreadsheet and sync their FitBit data. The Python script will prompt the user to authenticate access to the Google spreadsheet and then download, analyze, and report their data. If the user has properly connected to the Nao as described in the above instructions, the Nao will deliver a personalized, sociable interaction to the user that comments on their exercise activity and either commends their behavior or encourages them to change it. Examples of what these personalized interactions might look like can be found in the accompanying video samples.

4.1 Challenges and Limitations

As is often the case in a project that integrates so many unique pieces, developing a system that is easy for the user to interact with was a significant challenge. I had envisioned a system where the entire robotic interaction process would be triggered by the initial FitBit sync. However, both the robotic setup procedure and the OAuth authentication protocol require the participation of the user, which prevents the interaction from starting automatically. I believe that a longer-term implementation could address this issue with different hardware and a different approach to the authentication.

Another issue I encountered was how to define and discover user intentionality. This issue gets at one of the fundamental difficulties in robotics and artificial intelligence: concepts that are very easy for a human to understand can be very hard to formulate in terms that are accessible to a machine. As a user, it is very easy to understand that when you go out on a run, or go to a sports' practice, or go on a bike ride, you are intentionally getting exercise. However, for a computer system, it is much more difficult to distinguish the sprint you do at the end of a sports' practice from the sprint you do to class when you are running late. My system was best at identifying sustained exercise (e.g., runs) instead of bursts of activity (e.g., a sports practice). I think that a more complex dataset that included location data could help address this issue and lend some useful context to the data analysis portion of this project.

5 Conclusion

This purpose of this project was to deliver intelligent, personalized fitness feedback in an accessible and engaging form. My system extracts time series fitness data from the FitBit personal fitness tracker, and analyzes the data with Machine Learning techniques to determine which activities are intentional and

which are the incidental results of the user's daily experiences. I report these insights via the Nao, an embodied personal humanoid robot, which delivers personalized coaching feedback to the user in a way that aims to congratulate and encourage them.

This project was beset by a number of limitations that made it hard fully realize the automated nature of my original vision. The time and system restraints I faced made me very aware of the valuable possibilites of further investigations and inquiry. Nonetheless, I think this project takes an important and novel step by seeking to combine personal fitness data with an embodied robotic presence in an automated coaching environment.

6 Ideas for Further Inquiry

In reflecting on this project, I have many ideas for way to continue its development to make it more accessible and intelligent. A few of these ideas for further investigation follow:

- Use the initial FitBit sync with the user's FitBit profile to trigger running the Python script, fetching and analyzing the FitBit data, and starting the robot interaction. This modification would require using a robotic system that would be perpetually "on" and could just wait to receive behavior instructions from a script.

- Augment the FitBit data with GPS location tracking data. Over time, the system could use location data to supplement its data classification. For example, it would know that intentional activity is more likely to happen at "the gym" and less likely to happen "at work". I would also be very interested in visualizing this annotated or labelled fitness data on a map, which could help users explore their area or visualize their fitness habits.

- There are many other possibilities for a embodied personal robotic coach. The robot could help a user control and track their eating habits, make sure they're taking medicine at the right time, or help a user maintain effective work habits. This project combines personal data collection and social, contextualized robot interaction in an innovative way; I believe this framework could be extended to a number of other interesting and valuable applications.

7 Acknowledgments

I would like to thank Professor Brian Scassellati and Professor Roman Kuc for advising me on this project. Thank you also to Ph.D. candidate Alex Litoiu for spending time brainstorming with me at the beginning of the semester. I would also like to thank Sam Spaulding and Danny Ullman for helping me learn how to use the Nao and film the Nao interactions. Finally, I would like to thank

Pavel Risenberg and the rest of the FitBit API team for allowing me to access the Partner API.

8 Accompanying Materials

- `main_script.py`: Main Python script
- `nao_actions.py`: Python module that contains scripted Nao actions exported from Choreograph
- `fitbitdownload.gs`: Google script that fetches FitBit data and inputs it into a Google spreadsheet
- `fitbit_script.r`: R script that classifies fitness data
- `forest.RData`: Random Forest object that has been trained to classify fitness data
- *Congratulate.MOV*: A video example of what the "congratulate" robot interaction could look like
- *Reprimand.MOV*: A video example of what the "reprimand" robot interaction could look like
- `history.txt`: Text file containing user's exercise activity history