A Telemonitoring Solution to Long-Distance Running Coaching

Lucas Serven
Carlos Asuncion
Uma Balakrishnan
Hannah Sarver
Eugene Song

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-100
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-100.html

May 13, 2016

Acknowledgement

Daniel Aranki, University of California, Berkeley
Professor Ruzena Bajcsy, University of California, Berkeley
Professor Ali Javey, University of California, Berkeley
Dr David Liebovitz, MD, University of Chicago Medicine

BERKELEY TELE-MONITORING

A Telemonitoring Solution to Long-Distance Running Coaching

Master of Engineering Capstone Report 2016

Lucas Servén

with Carlos Asuncion, Uma Balakrishnan, Hannah Sarver, and Eugene Song

Prepared for Professors Ruzena Bajcsy and Ali Javey

Contents

List of Figures

Introduction

Our team, advised by Dr. Ruzena Bajcsy and Ph.D candidate Daniel Aranki, is contributing to the existing Berkeley Telemonitoring Project (Aranki et al. 2014; Aranki et al. 2016) that allows doctors and researchers to produce powerful smartphone-based telemonitoring applications. The Berkeley Telemonitoring Project is a framework designed with security, ease of use, and robustness in mind to permit the user to more easily develop state-of-the-art telemonitoring services.

In order to contribute to the framework, we have collaborated with Dr. David Liebovitz of University of Chicago Medicine to develop a smartphone-based application for marathon training. By developing this application, we will necessarily expand the functionality of the framework by creating modules for data collection, analysis, and transmission. Furthermore, by creating a real-world application, we will be testing the framework from end-to-end, providing crucial input, from the perspective of real users, that will shape its future development.

The goal of the marathon training application is to remotely monitor long distance runners as they train and to provide real-time input to improve their performance. The application will gather live data while the runners train including cadence, speed, and heart rate, and offer feedback. Dr. Liebovitz, is advising us on the cues and feedback that the application should deliver in order to have the greatest positive effect on their performance.

1 Background on Telemonitoring and the framework

The American Telemedecine Association defines telemonitoring as "the use of medical information exchanged from one site to another via electronic communications to improve a patient's clinical health status." Conceptually, telemonitoring can be applied to virtually any field in which remote measurements are made, e.g. electricity generation networks, autonomous navigation systems, smart thermostat etc. Telemonitoring can be visualized as a closed-loop system similar to the one presented in Figure 2. This loop consists of five basic components:

1. measurements are made on a remote environment;
2. the raw data is transmitted to a center for processing;
3. the data is processed and analyzed and turned into useful information;
4. the information is sent to an expert, human or autonomous, who reviews it and designs an intervention; and
5. the intervention is delivered to the environment.

When we began contributing to The Berkeley Telemonitoring Project, the framework was approaching maturity. The framework, visualized in Figure 1, already had most of these components in place in its three principal libraries: the client, server, and core libraries. The client library includes the classes for data extraction and estimation, while the server library holds classes for managing, storing, and performing computations on the collected data. The core library is a collection of classes designed to be used in conjunction with both the client and server libraries. This library provide classes that implement the the data structures and secure communication protocol needed to interface the client and server.

The paradigm of the framework includes two types of nodes: client nodes, running on a Android-enabled devices in the environment in question, and server nodes, which run in the cloud and process client data. Clients could estimate heart rate and energy expenditure, establish secure connections with the server, and transmit data while making some guarantees

about preserving data in case the communication failed. Server nodes could store and view the data, as well as send responses to the client.

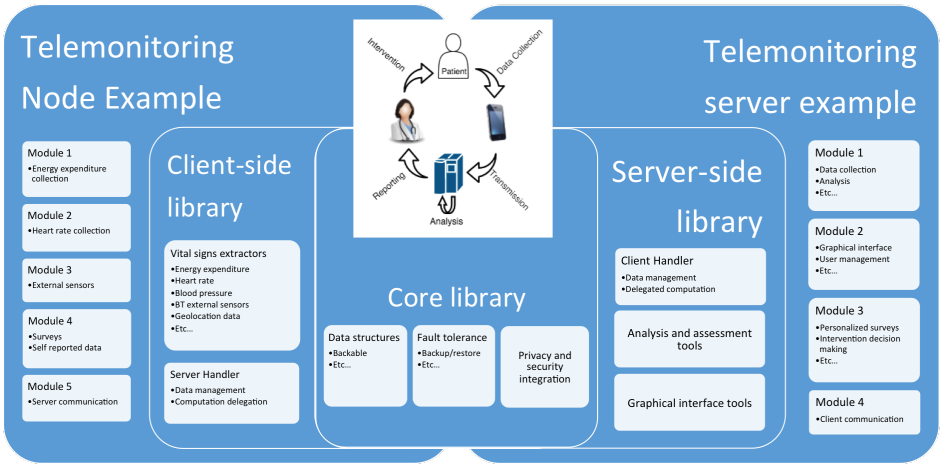


Figure 1: A breakdown of the libraries composing the Berkeley Telemonitoring Project and their modules.

Notably lacking from the framework, however, were data analysis tools, such as a toolkit of regressions and machine learning algorithms to study and interpret the data, modules to design interventions, and modules to intervene in the target environment. Furthermore, the modules for measuring the environment, termed extractors and estimators in the framework based on whether they collect raw sensor data or process data to generate derived estimates, were limited and could not estimate a node's location, walking speed, or cadence.

2 Task Breakdown

To ensure the success of our project, we have distributed the required work in accordance to the strengths and backgrounds of our team members, as shown in Figure 3. Expanding the framework primarily involved developing an intervention methodology, data analysis modules, and new data collection modules. The two former components were relevant to both the server and client nodes of the framework, while the latter was specific to the client, as it involved collecting remote data from client nodes. The data collection portion of the project included developing extractors and estimators for cadence (see Asuncion's paper), running speed, and GPS (see Sarver's paper).This project also required significant client-side

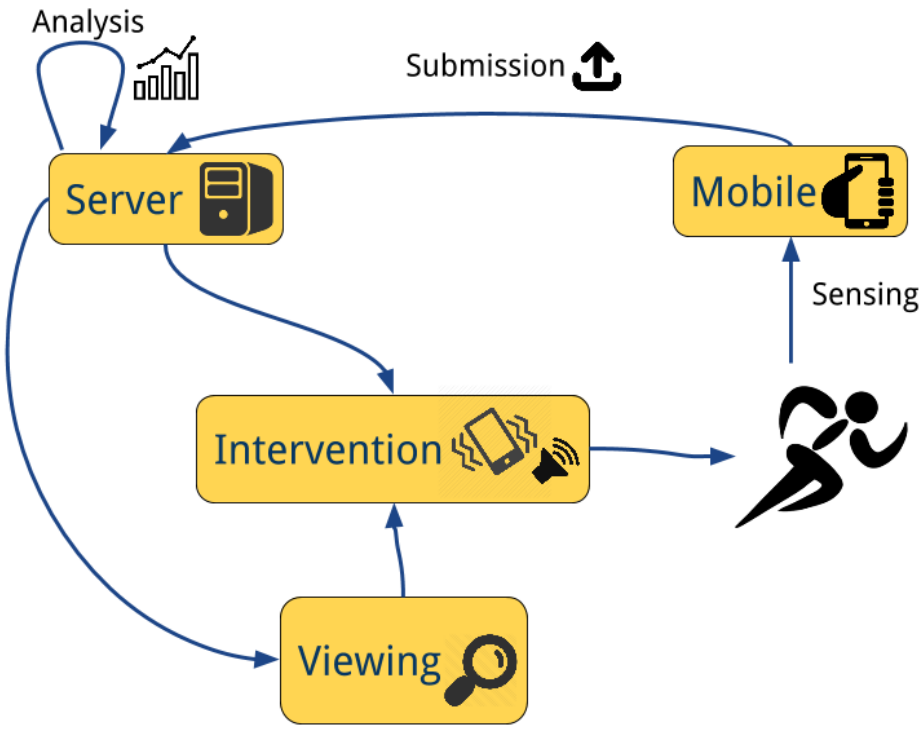


Figure 2: A telemontoring loop.

Android application development to create the actual framework-based app. This report will discuss the contributions made with respect to data extraction and analysis for running speed estimation given that these are the sections on which I have worked the most, as well as the implementation of several of the Android application modules necessary for this estimation.

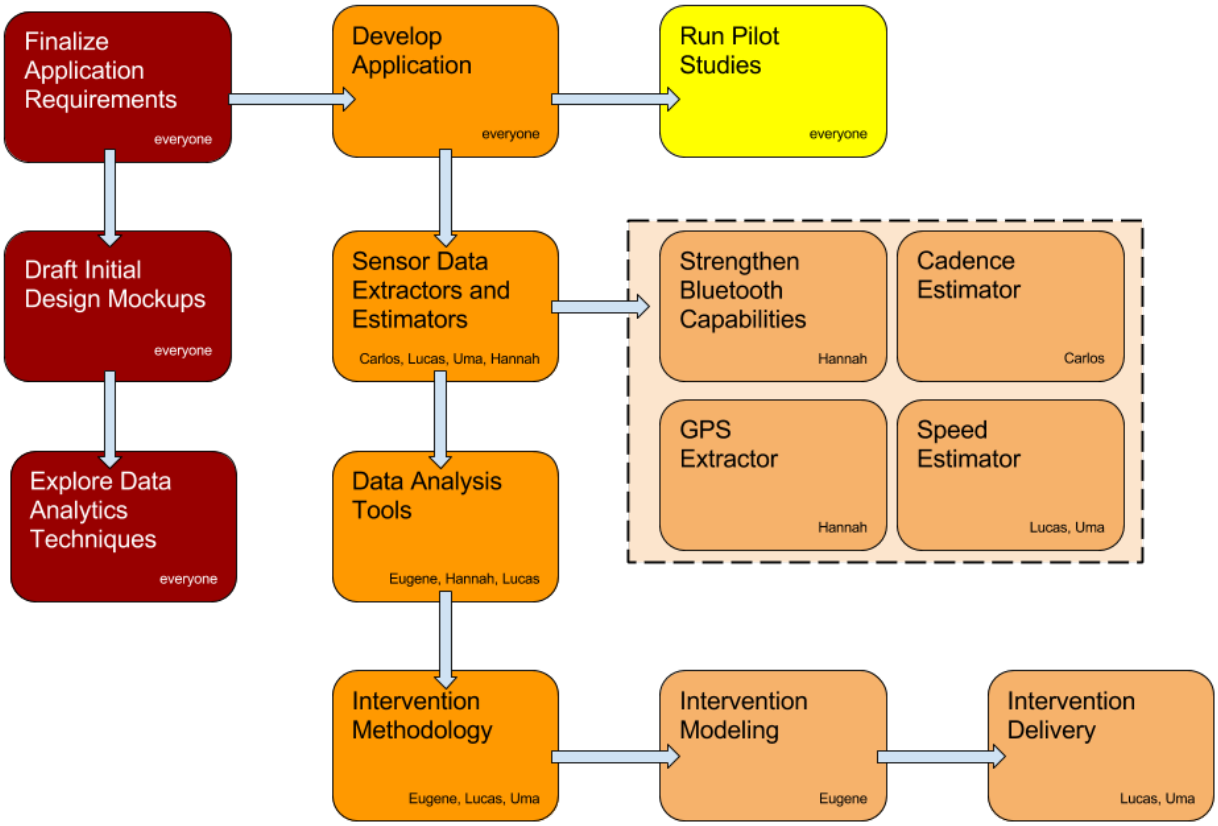


Figure 3: Breakdown of tasks between team members.

Chapter 1

Individual Technical Contributions

1.1 Speed Estimation

1.1.1 Background and Motivation

Preparing for a marathon involves training to safely complete the 26.2 miles of the race and to reduce total race time, effectively increasing the average speed of the runner. Consequently, speed is an important metric in running marathons and as a result it is imperative for our marathon training application to accurately measure it. Unfortunately, measuring the speed of a runner using a smartphone application is not a trivial undertaking and there exist several techniques that seek to solve this problem. Smartphone "app stores" distribute dozens of running application that claim to measure running speed using sensors available on the phone, but there are two chief issues with the majority of these implementations:

- they are inaccurate; many apps, including the popular Fitbit often produce inaccurate estimates for speed and energy expenditure. Fitbit reports to calculate distance, and the resulting instantaneous pace (time/distance), by multiplying a user's steps by her stride length (Fitbit 2016). This is is a nice heuristic, but it is overly simplistic. When running uphill, one's steps become shorter, so the estimated distance will be greater than the observed distance. Furthermore, Fitbit allows users to set a walking stride length and a running stride length, but this simplification ignores the stride lengths in

the range of running speeds between arbitrarily selected walking and running speeds. Several researchers have sought to study the accuracy of Fitbit devices, and while they found that the number of steps and energy expenditure reported was highly accurate (Mammen et al. 2012, Pande et al. 2013, Takacs et al. 2014), they found that the error in estimated distance was about 39% (Takacs et al. 2014); and

- they are not open source: Fitbit and others offer proprietary solutions for estimating cadence, energy expenditure, and speed whose algorithms are not known publicly. This means that although the algorithms can be experimentally validated, their theoretical accuracy, limitations, and general applicability are not necessarily known. This can cause issues for doctors and researchers seeking to include include cadence and running speed in a study.

To address this challenge, Uma Balakrishnan and I performed an extensive literature review of state-of-the-art approaches to smartphone-based running speed estimation.

1.1.2 Literature Review

Our goal while reviewing the existing approaches to running speed estimation was to identify and implement an estimation algorithm that satisfies several constraints:

- it must generate accurate predictions for a broad range of running speeds in order to be useful in real world conditions, where a runner may vary her speed from a fast run down to a brisk walk;
- it must be battery efficient; it is not acceptable for the algorithm to drain the user's battery as this will discourage use and will render the application useless for scenarios involving long runs since the runner's phone will die; and
- it must rely exclusively on sensors typically available to Android smartphones; this requirement is both obvious and nuanced: a marathon running application is not accessible if it requires highly specialized equipment to operate; it should instead rely on

hardware that is already highly available on smartphones. This raises an obvious question: what defines high availability in terms of market penetration of sensor hardware? Should the sensor be available on most phones? Most modern phones?

The papers we reviewed described algorithms that relied principally on acceleration data extracted from a smartphone's accelerometer to estimate the running speed of a user. Accelerometer are powerful devices for this application because their readings very precise, they can be read at very high frequencies, they are readily available in virtually 100% of smartphones, and they are well studied. When sampled at the same frequency, accelerometers can use 75%-85% less power than GPS sensors (Ben Abdesslem, Phillips, and Henderson 2009). Furthermore, the Assisted GPS (A-GPS) modules available on most smartphones provides accuracies within 50 feet outdoors and 160 feet indoors (Bajaj, Ranaweera, and Agrawal 2002), which could be too imprecise to offer useful speed estimates at human running speed magnitudes. While A-GPS is the standard method of estimating the speed of moving car in mapping application such as Google Maps, it is not convenient for continuous speed estimation of a runner because it too imprecise, drains the battery very quickly, and cannot be used reliably indoors.

The accelerometer solves the majority of these concerns, however, it presents a new challenge, chiefly: estimating speed from acceleration data is not straightforward. When a runner takes a stride, the acceleration of their body is not only forward, but also vertical and lateral. This means that acceleration measurements cannot simply be integrated with respect to time to obtain velocity. Furthermore, determining which axis of a smartphone corresponds to the forward direction of motion of a runner is not a simple task, as this axis can change from runner to runner depending on where they place their smartphone on their person and the axis can change throughout a run as the smartphone can move in the runner's belt, pocket, or bag.

The papers proposed a broad range of approaches to determining running speed from accelerometer data. Some of the algorithms relied on kinematic models of running motion (Hu, Sun, and Cheng 2013), others used estimated energy expenditure from acceleration and

then extrapolated running speed from that value (Panagiota, Layal, and Stefan 2012), and others yet relied on machine learning techniques to classify accelerometer samples as data from a particular activity (Altini et al. 2014).

After reviewing several published articles outlining different algorithms, we selected one that performs well and is suitable for runners (Park et al. 2012). In this paper, Park et al. present a machine learning-based algorithm that depends on a single accelerometer located at the subject's hip. This algorithm estimated gait speed with an error rate of less than 15% for a range of walking speeds in the presented leave-one-out generalization study. Furthermore, the algorithm benefited from being agnostic to sensor orientation. Finally, the algorithm outperformed other approaches considered to be state of the art, including the algorithms described in (Weinberg 2002) and (Cho et al. 2010).

1.1.3 Speed Estimation Algorithm

The algorithm's pipeline works as follows (Park et al. 2012):

1. The accelerometer is sampled at 100Hz. Readings are put into windows of 512 samples with 50% overlap between windows;

2. Take the L2 norm of every accelerometer reading. This step makes the readings agnostic to the orientation of the device;

3. Perform a 512-point Discrete Fourier Transformation (DFT) on the window of readings. This DFT produces a vector of coefficients of the magnitudes of different frequencies, $M(f_j)$. We only consider the first 60 values from this transformation, effectively applying a low-pass filter with a cutoff frequency of about 12Hz on the data. The motivation behind this filter is that human running motion occurs below 12Hz, so higher frequency data will typically be mostly noise;

4. Estimate the energy for the window by summing the square of the acceleration magni-

tudes from the DFT vector:

$$x_E = \sum_{j=1}^{60} |M(f_j)|^2$$

This energy will be the first feature for the sample.

5. Generate a feature for every element in the DFT vector by normalizing the magnitude of acceleration by the window energy:

$$x_M^\top = \left(\frac{|M(f_1)|}{\sqrt{x_E}}, \cdots, \frac{|M(f_{60})|}{\sqrt{x_E}} \right)$$

6. Combine these 61 vectors into a feature vector for the sample and predict a value using a regularized least squares estimation with a custom Radial Basis Function (RBF) kernel:

$$K_s(x, x') = exp\left(-\frac{\|x_M - x'_M\|^2}{2\sigma_M^2} \right) + exp\left(-\frac{(x_E - x'_E)^2}{2\sigma_E^2} \right)$$

Where σ_E and σ_M are half the median pairwise distance between x_E and x_M respectively for the training set.

Among the benefits of this algorithm is the fact that no subject-specific calibration is required after the model described is trained during the implementation phase. This training does, however, require a large number of samples in order to achieve high prediction accuracy: Park et al. reported using 2853 samples to train their estimator. Depending on the precision and compression of this data, the file size can reach the 5MB. Maintaining all of these data in memory can prove to be a limiting factor on smartphones and impossible on some embedded devices. This challenge and others are described in further detail in section 1.1.5.

1.1.4 Algorithm Prototype

Having selected an algorithm, Uma Balakrishnan and I created prototypes of of the model using Matlab and Python respectively. These prototypes were designed to serve as proof-of-concepts for the algorithm described by Park et al. To validate the prototype, we needed

to test it on hundreds of training samples and train the model at a broad range of running speeds. To do so, we have executed a pilot study on ourselves in which we collected acceleration data for walking and running using a phone located at our hip. We estimated the ground truth running speed and are comparing it to the speeds predicted using the algorithm to test its accuracy. This data collection procedure is described in section 1.1.5 and the prototyping and testing task is described in more detail in Balakrishnan's report.

In order to incorporate the model into the telemonitoring framework, we needed to translate the prototype into the Java programming language. This requirement is derived from the fact that the entire telemonitoring framework is written in Java. This constraint allows applications to be easily created for the Android smartphone environment, which exclusively runs Java code. The goal is to contribute a package of Java code to the framework's client library, so that client nodes can predict running speeds, and to the core library, so that both client and server nodes can interpret speed data.

1.1.5 Algorithm Training Methods

For the purpose of training the algorithm, we needed to supply the model with a large training set of accelerometer readings paired with their labeled observed running speed. In order for the training data to be accurate, the runners had to travel at a constant speed throughout the run. This is a difficult proposition over large distances. Park et al. make no mention as to how they collected their training data and maintained constant speed throughout each sample. Our current training data set includes 527 samples collected by four of the five teammates, which pales in comparison to the set amassed by Park et al.

We devised a data collection strategy that would allow us to collect accelerometer readings from subjects running at fairly constant speeds: the subjects would run short, predetermined, measured distances between 8 meters and 20 meters depending on the running speed, and use two Android applications to make the necessary measurements, the details of which can be found in Uma Balakrishnan's report. These samples were collected on an Olympic-regulation track. The subjects would start the accelerometer in the first application and

place the phone in their pockets. This sensor would log time-stamped accelerometer values. The runners would then start running a few meters before the beginning of the marked distance and then click a button on the application in the second phone application, which logged the times of the clicks, when they entered the marked section, allowing them to reach their before starting the sample window. They would then click the button after exiting the marked distance and run a few meters past the mark to ensure they did not slow down before the end of the sample. Using the times of the clicks from second phone, we could extract the relevant accelerometer data from the first phone.

This data collection process was cumbersome and as a result it took large amounts of time to coordinate the collection of only a few sample runs. In addition, the output files from each sample run still required a non-trivial amount of massaging to determine the observed running speed. Furthermore, each sample run included some non-negligible error: there is some natural human delay between when the runner passes the line and when they actually press the button on the app; and it is not guaranteed that the runner maintained a constant speed throughout the entire run and we rely on the short distance to minimize any variance in speed.

1.1.6 Java Implementation

I encountered several technical challenges while translating the speed estimation prototype into Java and the telemonitoring framework. These were principally focused around the added complexity of writing general purpose mathematical models in Java, which unlike MATLAB or NumPy, was not designed for mathematical computing. This means that a lot of the programming idioms and conventions that arise naturally in these two languages do not have obvious analogs in Java and must be supplemented with additional libraries. For example, the Fast Fourier Transform, a crucial algorithm for analyzing the frequency domain properties of a signal, can be executed trivially on matrices of large dimensions in both Matlab and Python, but requires considerable extra care to implement in Java.

Furthermore, Java is a strongly typed language, meaning that code written in that lan-

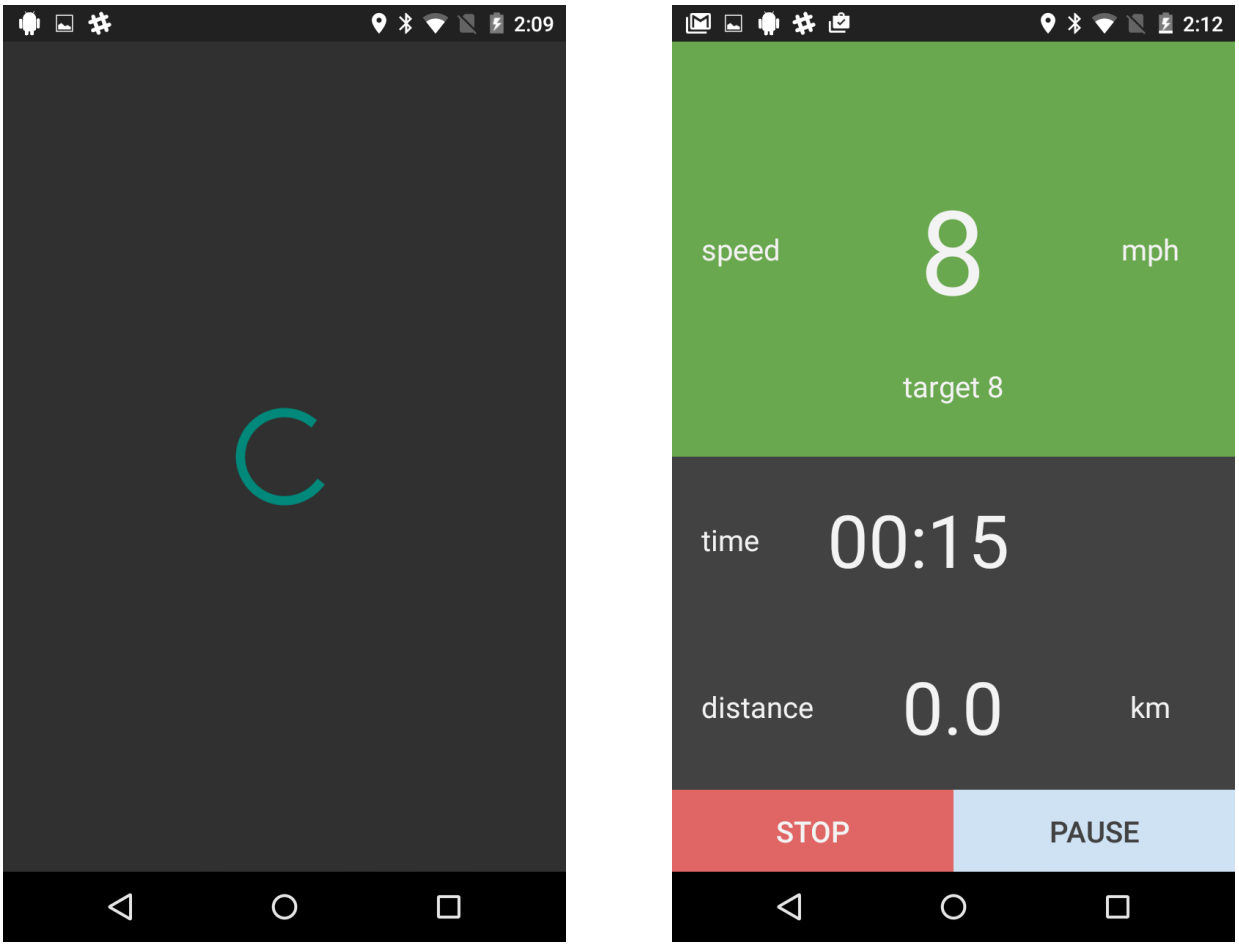
guage must explicitly declare the types of variables that are being computed a priori. This resulted in additional overhead when considering the translation of the speed estimation prototype since the algorithm deals with variables of subtly different types, including: time-valued arrays; acceleration-valued arrays, of which each entry is a three-element array; and two dimensional matrices.

To begin porting the algorithm to Java, I developed a framework estimator that would collect accelerometer values whenever new readings were available. These readings were placed into a general purpose encapsulator for accelerometer values that, when full with the necessary amount of accelerometer data, would re-sample the readings to 100Hz and generate a new instance of the feature vector class.

I continued by writing wrapper classes to implement this feature vector and each of the remaining basic types for the speed estimation algorithm namely: a class representing a speed sample of accelerometer values and a class for the predictor. These classes were given methods to process the sampled data, moving it further along the algorithm's pipeline. The processing methods take care of several tasks that are critical to the functioning of the model: resampling, normalizing, gravity elimination, and fast Fourier transformation. These elements will be required by the application in order to process real-time data that is collected while the runner trains so that it can be fed to the actual prediction model. The feature vector class described an object that could consumed by the predictor class to estimate the current running speed. The predictor class was modeled as a singleton to avoid the cost of loading the training data set into memory multiple times, or worse, for every prediction.

While writing these classes, I discovered an additional complication: the integration of disparate existing classes for sampled data in the framework. Simply put: over the last two years, students contributing to the framework have developed several classes that fulfill nearly identical roles, but represent slightly different data types. The result is that the framework has grown more complex and there is redundant code, making it more difficult for future users of the framework to develop efficient applications. To tackle this issue, I

will have to consolidate existing classes into the lowest common denominator that fulfills all the needed functionality. This effort may result in the development of a more general math package for the framework.



(a) (b)

Figure 1.1: In-app configuration of cadence trajectory. (a) A loading screen conditionally rendered only if the training data has not finished loading. (b) The run screen shown when a user starts a run

The final steps involved writing the class that could load training data, located somewhere in non-volatile memory, and generate a feature matrix to operate the model. This was a less than trivial undertaking because of the size of the data set. Our current data set, a comma-separated file containing only 527 samples, resulted in a non-compressed filesize of 1.5MB. If read into memory in a blocking manner, it would stop the user interface for several seconds, resulting in an unpleasant experience for the user. To overcome this issue, I load the data in a separate thread and use a series of callbacks to notify the rendering thread

when the loading is complete. This procedure allows us to load the training data ahead of time to ensure that the user does not need to wait before starting a run. If the application user begins a new run very quickly before the data has finished loading, then they will see the screen shown in Figure 1.1a, however if the data has finished loading, they will directly see the run activity screen shown in Figure 1.1b.

Apart from the unpleasant delay in rendering, this training data file-size resulted in a second issue: decreased application portability. If the training data filesize it too great, the Android application file generated from the source code would be very large as well, making it more costly to download and distribute. To workaround this issue, I modified the helper class tasked with loading training data to first try to load a training data file from the smartphone's SD card, and if the file is not present, the class will default to using a smaller training dataset shipped with the application code. This allows the application implementor to supply their own dataset if they wish, giving them more control over the accuracy of the estimator at the speed ranges they expect to see, and also allows them to use our data if they do not want to deal will supplying a dataset. Furthermore, I modified the helper class to directly load the feature matrix from a serialized Java object, rather than load the raw dataset in the form of accelerometer values and generate the feature vector from them. This modification had the effect of dramatically reducing the filesize of the training dataset to approximately 100KB.

1.1.7 Cross Validation

To improve the prediction capabilities of the running speed estimator, I decided to design a cross-validator class to select the best values for the hyperparameters. At the moment, this class is instantiated with the parameters necessary to conduct k-fold cross validation on the speed estimator, that is with a matrix of speed data, a vector of labeled speeds, values for σ_M, σ_E, and λ, and a k value for the validation. However, it will not be hard to abstract the class to use any model that satisfies instead a given interface with specified methods. Once the class is further abstracted, the cross-validator will be added to the library of data

analysis tools for the server.

In Park et al. 2012, the authors write that the method they used to select the hyper-parameters σ_M, σ_E, and λ was chosen to avoid the need for a costly search for these three values. I decided to move forward with the process by first calculating the values for σ_M and σ_E that Park et al. suggested, that is by calculating half of the median pairwise distance for x_M and x_E respectively. I used these as initial values around which to begin the search for the hyperparameters. Using the class I created, I instantiated the cross-validator with 729 different combinations of hyperparameter values across several orders magnitudes and performed an initial 10-fold cross validation on each set. The combination of hyperparameters that achieved the lowest error rate was: $\sigma_M = 1.0$, $\sigma_E = 10e4$, and $\lambda = 0.1$. The results can be found in 1.2. Note that only the results with $\lambda = 0.1$ are shown in order fit all the results in a three-dimensional plot. With this combination of values, the speed estimator achieved an RMSE of 11.9%.

This search took several hours to complete, but was able to refine the search space for the hyperparameters to a smaller region. Once this search finished, I conducted a second round of cross validation to better tune the selection of the parameters. Once again, I used 729 combinations of hyperparameters, but this time over a smaller search space with the results from the first round of cross validation used as initial values. This time, the best selection of hyperparameters was: $\sigma_M = 0.2$, $\sigma_E = 10e6$, and $\lambda = 0.01$. This selection of hyperparameters produced a cross validation RMSE of 4.7%. Figure 1.3 shows a 3D plot of the results from this cross validation. Again, to reduce the dimensions of the plot, only results with $\lambda = 0.01$ are shown.

1.1.8 Results

To test the prediction capability of the speed estimator we created, we conducted a second study on ourselves where we we collected accelerometer readings taken while running at various speeds on a track and recorded the ground truth running speeds as before. This leave-one-participant out test dataset contained 238 vectors, roughly half the size of the

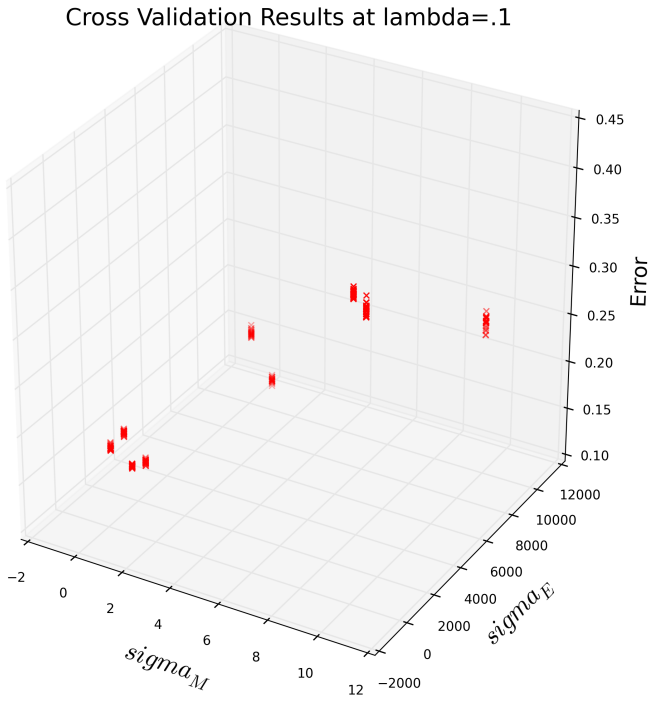


Figure 1.2: Results from 10-fold cross validation of the speed estimator using 729 combinations of hyperparameter values.

training set, from a runner whose data was used exclusively for testing and not for training. We tested the speed estimator using the hyperparameters we found using the cross-validator and found that our implementation produced an RMSE of 10.3%. It is worth noting that this accuracy exceeds the results reported by Park et al. who found RMSEs between 12%-15-%. We suspect that this reduction in the error rate is largely due to the improved selection of hyperparameters.

In the study by Park et al., the training and testing was conducted on data retrieved from different participants using the same accelerometer. In our study, each participant used their own LG Nexus 4 Android-enabled phone. It would be valuable to investigate whether using different phones with different accelerometer chips adversely affects the performance of the estimator. If this is indeed the case, then the training set may need to be diversified with

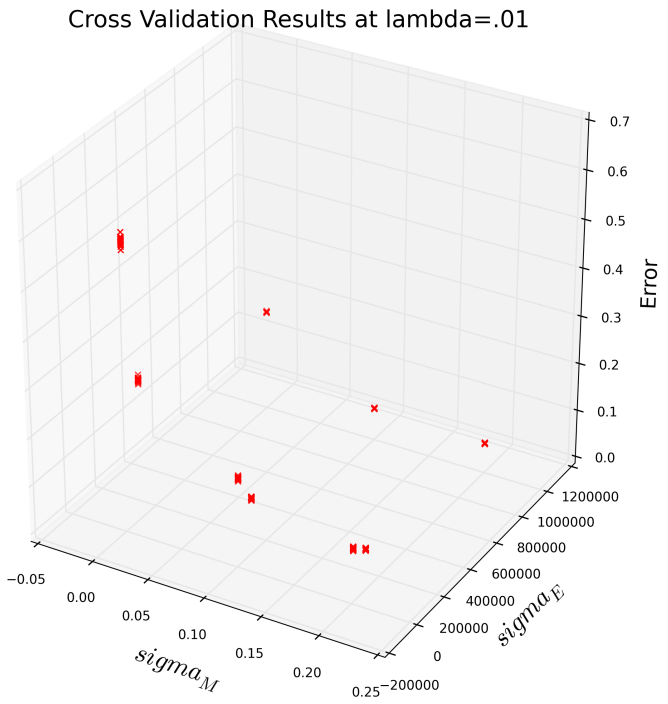


Figure 1.3: Results from the second round of 10-fold cross validation of the speed estimator using 729 combinations of hyperparameter values.

data collected both from many different devices as well as many different runners. We expect that as the size of the training dataset is increased, as made feasible thanks to the reduced filesize of the serialized data, the estimation accuracy of the estimator will improve.

1.2 Conclusion

The work done over the past months has added a host of robust features to the Berkeley Telemonitoring Project "The Berkeley Telemonitoring Project - Privacy-Aware Health Telemonitoring" 2016. We have added a powerful new data extraction and estimation tools that will allow users to leverage the framework to build a whole new range of telemonitoring applications. In the months to come, the Berkeley Telemonitoring Project will execute a

pilot study that the team designed and use the results to study and improve the framework. The pilot study will be performed on self-described long-distance runners while they continue their normal training regimen. The runners will use the application we created on their Android-enabled smartphones to collect the data they choose using the extractors we implemented, i.e. they will have the options to collect and send cadence, speed, heart-beat rate, and distance data, along with qualitative feedback about the run and our application.

In order to execute the study, we submitted a protocol to the University of California Internal Review Board (IRB). As of April 2016, the IRB has approved the study, allowing us to begin execution as soon as we like. We hope that the data collected from the study will help us to improve the application and ultimately the telemonitoring framework on which it is built. It is our aspiration that furthering the Berkeley Telemonitoring Project will promote telemedicine and positively impact health-care.

Chapter 2

Engineering Leadership

2.1 Industry and Market Analysis Overview

While there exist several smartphone based platforms that can be used to create applications for various purposes, such as Canvas, Appery.io and Mobile Rodie (Smith 2013) most of these offer limited functionality and access to sensors. Additionally, these products lack the ability to easily build predictive models for automated generation of personalized interventions. More generally, there are no such platforms that cater to the issue of telehealth. Our telemonitoring framework, targeted towards doctors and coaches, addresses this unmet need.

To guide the expansion of the framework, we will consider the design and implementation of new features in the context of a commercial application that would be used by marathon trainees. To this end, it is useful to perform a market and industry analysis on existing fitness tracking technology. By examining consumer behavior and industry offerings, we can better understand what functionality is missing and what features athletes desire.

2.2 Market Analysis

According to surveys (Running USA 2015) in 2014 there were more than 1200 marathon events held within the US, with a total of 550,637 finishers. These are both all-time high

statistics, with the number of marathon finishers growing about 1.8% from 2013 to 2014. A survey of marathon runners showed that 74% of them relied on wearable technology for training and 88% of them relied on said technology for motivation (Freescale 2014). Between 2014 and 2015, the number of wearables purchased is said to have nearly tripled from 17.6 million to 51.2 million (Gfk 2015). Of Internet users who exercise between the ages of 18-34, 38% of males and 21% of females use wearable fitness trackers (Mintel 2014). Wearable technology has clearly entered into the mainstream, especially in the area of fitness training with fitness trackers. Marathon runners are no exception. With their ubiquity and proclivity for training technology, they represent an acceptable target market for our application.

2.3 Porter's Five Forces Analysis

We will now conduct a Porter's Five Forces analysis of our mobile application for marathon runners to contextualize it in the industry and develop a strategy for differentiating and promoting it (Porter 2008).

2.3.1 Bargaining Power of Buyers

Buyers have strong bargaining power only when they are consolidated. Consumers of fitness tracking products are numerous, but diffuse in their buying patterns. Buyers are many and demand is great, weakening the bargaining power of buyers.

2.3.2 Bargaining Power of Suppliers

The power of suppliers refers to the power of businesses that provide materials to another business to demand increased prices for materials (Porter 2008). The application is developed for the Android platform, and cell phones have become a commodity, indicating a weak bargaining power.

2.3.3 Threat of New Entrants

New entrants have the potential to bring new ideas to a market. The market of activity monitors poses few barriers and connected fitness trackers are projected to grow from $2 billion to $5.4 billion from 2014 to 2019 (Parks Associates 2015). With the burgeoning of the Internet of Things, it is expected that there will be new players in many applications of telemonitoring. Thus, the threat of new entrants is perceived to be strong.

2.3.4 Threat of Substitutes

A product from a different industry that offers the same benefits to consumers is referred to as a substitute product. Substitute products pose a threat because there is possibility of replacement for a company's product (Porter 2008). One substitute product for runners training for marathons is meeting one-on-one or in small groups with dedicated professional trainers and coaches. There is an approximately $1.5 billion industry existing in intensive personal athletic training in the United States (Witter 2015). This includes firms and independent individuals who provide services granting personalized attention to athletes training for sports seasons or upcoming events such as marathons. However, human trainers conducting in-person training generate problems not seen in the activity monitor/trainer application. For example, scheduling is a factor for this substitute, as the athlete would need to train according to the trainer's schedule and location. Having a human trainer is also significantly more expensive than using an activity monitor. The application does not come with these added cost and conditions. For these reasons we believe that the threat of substitutes is weak.

2.3.5 Rivalry Amongst Existing Competitors

Rivalry can pose a great threat when the rivals compete over price instead of features. The market for tracking and training of fitness, including endurance running, is a crowded one. In this market, our application will be competing with a variety of technologies, such as smartphone apps and specialized fitness tracking hardware. We will need to ensure that

our feature offerings are differentiated in order to avoid significant threat from price-based rivalry.

Wearable fitness tracking devices have seen widespread adoption among runners and other athletes. There are several subcategories of device functionality in this area, ranging in metrics measured, accuracy of these metrics, and price. These include step counters such as the Fitbit One or Nike+ FuelBand at the lower end of functionality and price, GPS-based speed and distance monitors like the Garmin ForeRunner 620 or TomTom Runner at the higher end, and multi-functional devices like smartwatches, such as the Apple Watch or Pebble Time that have some built-in fitness features (Carter 2013).

Other competing fitness devices include specialized peripheral hardware, such as chest straps to monitor heart rate, shoe inserts to track impact and step duration, and devices that help athletes recover from training in terms of bloodflow and muscle relaxation, such as the Firefly Recovery System (Alger 2014). These products are more targeted at health monitoring and feedback for runners, which we can compete with by providing without specialized hardware outside of the mobile phone itself.

Additionally, given the demand for personal training, new products which provide personalized feedback, such as the Moov, have already begun to appear. Moov's successful crowdfunding campaign indicates a demand for fitness trackers that can provide this type of feedback (Colao 2014). Major players are pushing for greater personalization. For example, FitBit, a key player in wearable fitness tracking, acquired the startup FitStar in 2015 (Lawler 2015) which provides users with personalized instructional videos. Finally, our application will be competing with a host of other smartphone fitness applications. A huge market for personal fitness tracking exists in the app stores of the smartphones that so many Americans already carry with them daily. A study (Stanfy 2013) estimated that in 2012 there were over 40 thousand health and fitness applications available for mobile phones, reaching over 500,000 active users, and that number has only increased in the past few years. A wide variety of fitness and run tracking, goal-setting and socially competitive, and motivational applications are available. Some of the most popular apps specifically targeted at runners

are RunKeeper, MapMyRun, and Runtastic (Carter 2013). On the more creative side are apps like Zombies, Run which provides audio motivation in a narrative form, taking a runner through customizable missions in a fictional environment.

Given the great number of players in this industry, the threat of existing rivals is strong. However, given the still largely unexplored area of personalized coaching within the crowded space of fitness tracking technology, we believe that this rivalry will primarily be features-based.

2.4 Technology Strategy

Considering our market research and Porter's Five Forces analysis, we have developed a strategy for our product in order to minimize the threats posed to our product. Our strategy revolves around marketing to customers based on the features offered by our product, particularly focusing on measurement and real-time feedback regarding performance metrics, such as speed and cadence. For instance, despite its importance, many fitness tracking solutions do not measure cadence. In addition, the products that do are typically not transparent about the estimation algorithms used and their accuracies. Even for those that do report accuracy, the algorithms used are still unpublished, and the accuracy of specific metrics, such as cadence, are conspicuously missing (Garmin 2016). Our application uses algorithms backed by published scientific literature, and the accuracy of our implementation will be further measured and published. Furthermore, the framework on which the application is built includes a fault-tolerant client-server protocol for secure and convenient data syncing, and a wide library of well-tested data collection and analytics functionality to support our application's features and ensure they remain reliable and easy to use. Raising the standards of information transparency, estimation accuracy, and application reliability would not only allow our application to gain traction in the market if we were to actively promote it, but would also impose barriers to new entrants.

Bibliography

Alger, Kieran. 2014. "Training for a marathon: Tech for half marathon, marathons and beyond". http://www.t3.com/features/training-for-a-marathon-half-marathon-ultra-and-beyond.

Altini, Marco, Ruud Vullers, Chris Van Hoof, Marijn van Dort, and Oliver Amft. 2014. "Self-calibration of walking speed estimations using smartphone sensors". In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, 10–18. IEEE.

Aranki, D., G. Kurillo, P. Yan, D. M. Liebovitz, and R. Bajcsy. 2016. "Real-time Tele-monitoring of Patients with Chronic Heart-Failure Using a Smartphone: Lessons Learned". *IEEE Transactions on Affective Computing* PP (99): 1–1. ISSN: 1949-3045. doi:10.1109/TAFFC.2016.2554118.

Aranki, Daniel, Gregorij Kurillo, Posu Yan, David M. Liebovitz, and Ruzena Bajcsy. 2014. "Continuous, Real-time, Tele-monitoring of Patients with Chronic Heart-failure: Lessons Learned from a Pilot Study". In *Proceedings of the 9th International Conference on Body Area Networks*, 135–141. BodyNets '14. London, United Kingdom: ICST (Institute for Computer Sciences, Social-Informatics / Telecommunications Engineering). ISBN: 978-1-63190-047-1. doi:10.4108/icst.bodynets.2014.257036. http://dx.doi.org/10.4108/icst.bodynets.2014.257036.

Bajaj, R., S. L. Ranaweera, and D. P. Agrawal. 2002. "GPS: location-tracking technology". *Computer* 35, no. 4 (): 92–94. ISSN: 0018-9162. doi:10.1109/MC.2002.993780.

Ben Abdesslem, Fehmi, Andrew Phillips, and Tristan Henderson. 2009. "Less is more: energy-efficient mobile sensing with senseless". In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, 61–62. ACM.

Carter, Jamie. 2013. "10 best running gadgets: the top tech for training". http://www.techradar.com/us/news/world-of-tech/roundup/10-best-running-gadgets-the-top-tech-for-training-1157180/1.

Cho, Dae-Ki, Min Mun, Uichin Lee, William J Kaiser, and Mario Gerla. 2010. "Autogait: A mobile platform that accurately estimates the distance walked". In *Pervasive computing and communications (PerCom), 2010 IEEE international conference on*, 116–124. IEEE.

Colao, J.J. 2014. "Who Needs Kickstarter? Exercise Sensor Moov Raises $1 Million In 15 Days". http://www.forbes.com/sites/jjcolao/2014/03/14/exercise-sensor-moov-raises-1-million-in-15-days-without-kickstarter/#704c414614e3.

Fitbit. 2016. "How do I measure and adjust my stride length?" https://help.fitbit.com/articles/en_us/help_article/how-do-i-measure-and-adjust-my-stride-length.

Freescale. 2014. "The next evolution in running". https://web.archive.org/web/20141223192158/http://blogs.freescale.com/iot/2014/12/wearables-next-evolution-in-running-marathon/.

Garmin. 2016. "How accurate are the Running Dynamics from the HRM-Run and how was the accuracy tested?" https://support.garmin.com/support/searchSupport/case.faces?caseId=%7B435cc8c0-e2e1-11e3-47b1-000000000000%7D.

Gfk. 2015. "GfK forecasts 51 million wearables will be bought globally in 2015". gfk.com/news-and-events/press-room/press-releases/pages/gfk-forecasts-51-million-wearables-sold-globally-2015.aspx.

Hu, Jwu-Sheng, Kuan-Chun Sun, and Chi-Yuan Cheng. 2013. "A kinematic human-walking model for the normal-gait-speed estimation using tri-axial acceleration signals at waist location". *Biomedical Engineering, IEEE Transactions on* 60 (8): 2271–2279.

Lawler, Ryan. 2015. "Fitbit Confirms FitStar Acquisition To Bring Training To Its Fitness Portfolio". http://techcrunch.com/2015/03/05/fitbit-confirms-fitstar-acquisition-to-bring-training-to-its-fitness-portfolio/.

Mammen, George, Sarah Gardiner, Arrani Senthinathan, Laura McClemont, Michelle Stone, and Guy Faulkner. 2012. "Is this bit fit? Measuring the quality of the FitBit step-counter". *The Health & Fitness Journal of Canada* 5 (4): 30–39.

Mintel. 2014. *Exercise Trends - US - October 2014*. Market report. Mintel.

Panagiota, Anastasopoulou, Shammas Layal, and Hey Stefan. 2012. "Assessment of human gait speed and energy expenditure using a single triaxial accelerometer". In *Wearable and Implantable Body Sensor Networks (BSN), 2012 Ninth International Conference on*, 184–188. IEEE.

Pande, Amit, Yunze Zeng, Aveek K Das, Prasant Mohapatra, Sheridan Miyamoto, Edmund Seto, Erik K Henricson, and Jay J Han. 2013. "Energy expenditure estimation with smartphone body sensors". In *Proceedings of the 8th International Conference on Body Area Networks*, 8–14. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Park, Jun-geun, Ami Patel, Dorothy Curtis, Seth Teller, and Jonathan Ledlie. 2012. "Online pose classification and walking speed estimation using handheld devices". In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, 113–122. ACM.

Parks Associates. 2015. "Global revenues from connected fitness trackers to exceed $5 billion by 2019". `http://www.parksassociates.com/blog/article/pr-march2015-whcc`.

Porter, Michael E. 2008. "The Five Competitive Forces That Shape Strategy". *Harvard Business Review* 86 (1): 78–93.

Running USA. 2015. "2014 Running USA Annual Marathon Report". `http://www.runningusa.org/marathon-report-2015`.

Smith, Grace. 2013. "10 Excellent Platforms for Building Mobile Apps". `http://mashable.com/2013/12/03/build-mobile-apps/#SXF0URANsqqg`.

Stanfy. 2013. "Fitness in Mobile: Case-study". `http://www.slideshare.net/stanfymobile/fitness-cs-22962137`.

Takacs, Judit, Courtney L Pollock, Jerrad R Guenther, Mohammadreza Bahar, Christopher Napier, and Michael A Hunt. 2014. "Validation of the Fitbit One activity monitor device during treadmill walking". *Journal of Science and Medicine in Sport* 17 (5): 496–500.

"The Berkeley Telemonitoring Project - Privacy-Aware Health Telemonitoring". 2016. `https://telemonitoring.berkeley.edu/`.

Weinberg, Harvey. 2002. "Using the ADXL202 in pedometer and personal navigation applications". *Analog Devices AN-602 application note* 2 (2): 1–6.

Witter, David. 2015. *IBISworld Industry Report 61162: Sports Coaching In The US*. Industry report. IBISworld.