

April 2016

Smoking Gesture Detection in Natural Settings

Anthony A. Romeo
Worcester Polytechnic Institute

Steven Patrick Ireland
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

Repository Citation

Romeo, A. A., & Ireland, S. P. (2016). *Smoking Gesture Detection in Natural Settings*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/1776>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.

Smoking Gesture Detection in Natural Settings

A Major Qualifying Project
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree in Bachelor Science
in
Computer Science
By

Anthony Romeo

Steven Ireland

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see <http://www.wpi.edu/academics/ugradstudies/project-learning.html>

Date: April, 2016

Project Advisor:

Prof. Emmanuel Agu, Advisor

Abstract

Since 1964, more than 20 million Americans have died as a result of smoking. Being able to reflect on smoking habits gives smokers a greater chance to quit. A worn smartwatch can record accelerometer and gyroscope data from a smoker's wrist, allowing the extraction of features significant to the identification of smoking gestures. This project explores techniques for identifying these gestures which would enable the implementation of passive reflective tools later. Our smoking gesture detection approach used machine learning to detect patterns from various features including max speed, median speed, mean speed, variance of speed, and net, median, and max roll velocity. The machine learning algorithm then classified which gesture was being performed based on training data of 30 people performing four distinct gestures (smoking, drinking, phone call and crossing arms) 30 times each. The WEKA data mining library tested multiple classification algorithms including Naive Bayesian Network, J48 Decision Tree, and the most accurate Random Forest. Successful detection of these smoking gestures can help smokers quit smoking by providing real time interventions.

Table of Contents

Abstract.....	ii
Table of Figures.....	v
Table of Tables	vi
Table of Equations.....	vii
1. Introduction.....	1
1.1. Harms of Smoking.....	1
1.2. Who Smokes	1
1.3. Effects on non-Smokers.....	2
1.3.1. General Population.....	2
1.3.2. Babies.....	3
1.3.3. Children.....	4
1.4. Wanting to Quit.....	4
1.4.1. Quitting Techniques.....	5
1.5. Detection and Smartphones.....	5
1.6. The Goal of this MQP.....	6
2. Related Work.....	7
2.1. RISQ.....	7
2.1.1. How RISQ Tackles the Problem.....	8
2.1.2. Comparison to RISQ.....	8
2.2. Automated Detection of Puffing and Smoking with Wrist Accelerometers.....	8
2.2.1. How Automated Detection of Puffing and Smoking with Wrist Accelerometers Tackles the Problem.....	9
2.2.2. Comparison to Automated Detection of Puffing and Smoking with Wrist Accelerometers.....	9
3. Methodology.....	10
3.1. Watch Selection.....	10
3.2. Considerations.....	11
3.2.1. Confounding Effects.....	12
3.2.1.1. Drinking Beverage.....	12
3.2.1.2. Eating a Meal.....	13

3.2.1.3	Talking.....	13
3.2.1.4	Using a Phone.....	14
3.2.2	Multiple Resting Positions.....	14
3.2.3	Pauses.....	14
3.2.4	Opposite Hand.....	15
3.3	Participant Selection.....	15
3.4	Data Gathering Study.....	15
4.	Implementation.....	21
4.1	Data Collection.....	22
4.2	Feature Extraction.....	23
4.3	Classification.....	26
5.	Results.....	28
6.	Discussion.....	30
6.1	Lessons Learned.....	30
6.2	Limitations of Wrist Watch Sensing.....	30
6.3	Implications of Smoking Gesture Recognition.....	31
7.	Conclusion	32
7.1	Future Work.....	32
	References.....	35
	Appendix.....	36

Table of Figures

Figure 1 – System Diagram.....	10
Figure 2 – Watch comparison.....	11
Figure 3 - Cigarette Simulation (Resting).....	16
Figure 4 - Cigarette Simulation (Active).....	17
Figure 5 - Drinking Simulation (Resting).....	17
Figure 6 - Drinking Simulation (Active).....	18
Figure 7 - Phone Simulation (Resting).....	18
Figure 8 - Phone Simulation (Active).....	19
Figure 9 - Arms Cross Simulation (Resting).....	19
Figure 10 - Arms Cross Simulation (Active).....	20
Figure 11 - Android app interface.....	21
Figure 12 - Basic Watch Screen.....	21
Figure 13 - Initialization of Sensors on Watch.....	22
Figure 14 - Code to Detect the Start of a Gesture.....	23
Figure 15 - Accelerometer Data of Hands at Rest.....	24
Figure 16 - Accelerometer Data of Start of Smoking Gesture.....	24
Figure 17 - Accelerometer Data of Start of Drinking Gesture.....	24
Figure 18 - Accelerometer Data of Start of Drinking Gesture.....	26
Figure 19 - Code to Create a WEKA-Compatible Data Instance.....	26
Figure 20 - WEKA Random Forest Classification.....	27
Figure 21 – Smoking Tracker.....	34

Table of Tables

Table 1 - Percent of Smokers in 2014.....	2
Table 2 - Quitting Statistics.....	5
Table 3 – Participant Selection.....	15
Table 4 - Features extracted.....	25
Table 5 - Confusion Matrix.....	28
Table 6 - Experimental Results.....	28
Table 7 - Other Classifiers.....	29

Table of Equations

Equation 1- Quaternion.....	7
Equation 2- Rotation.....	7
Equation 3- Quaternion Rotation.....	7

1. Introduction

1.1 Harms of Smoking

Tobacco use is prevalent in modern culture and commonly used around the world and in the United States. Tobacco is currently the leading cause of preventable death and illness in the United States [1]. Tobacco has been known to cause emphysema, bronchitis, heart disease, erectile dysfunction, as well as multiple different cancers [1]. These cancers can occur in the throat, mouth, nasal cavity, esophagus, stomach, pancreas, kidney, bladder, and cervix [1]. Almost 90% of lung cancer cases are caused by smoking [1]. Lung cancer is the leading cause of cancer-related deaths [1]. Men who smoke with prostate cancer are more likely to die from this disease [1]. About 80% of Chronic Obstructive Pulmonary Disease deaths are a result of smoking. Women who smoke are up to 40 times more likely to develop COPD compared to women who have never smoked [1]. Smoking increases the likelihood of developing asthma as a youth, increases a person's risk of getting tuberculosis, and slows down lung growth in children and teens [1]. Smokers are 30% to 40% more likely to develop type two diabetes than non-smokers [1]. The average life expectancy for a smoker is 13-14 years lower than non-smokers [1]. It is clear to see that there are a lot of diseases and conditions that can be caused as a result of smoking. When one chooses to smoke one undergoes a very large risk yet so many people still smoke.

1.2 Who Smokes

In 2014 16.8% of all adults in the United States smoked cigarettes, 18.8% of males and 14.8% of females [2]. Each day more than 3,200 people younger than 18 smoke a cigarette for the first time [2]. In addition to this about 2,100 youths who have been occasional smokers transition to become daily cigarette smokers [2]. These huge numbers result in huge death tolls. More than 16 million people already have at least one disease from smoking [1]. Since 1964, more than 20 million Americans have died as a result of smoking [1]. There are still 8.6 million

people still alive, living with a serious illness caused by smoking [1]. Table 1 shows the breakdown of smokers by ethnicity.

Ethnicity	Percent of Smokers
non-Hispanic American Indians/Alaska Natives	29.2%
non-Hispanic multiple race individuals	27.9%
non-Hispanic Whites	18.2%
non-Hispanic Blacks	17.5%
Hispanics	11.2%
non-Hispanic Asians	9.5%

Table 1 - Percent of Smokers in 2014 [1]

1.3 Effects of Smoking on non-Smokers

Not only does smoking have a huge effect on people who chose to smoke, but also has an enormous effect on those who are in the presence of smokers. Secondhand smoke is a term for inhaling smoke that somebody has recently exhaled. Secondhand smoke contains thousands of chemicals, where hundreds are toxic and roughly seventy are known to cause various types of cancer [4]. The effects of secondhand smoke can be as serious as the smoke from the initial puffs and even result in death. When a person breathes secondhand smoke the smoke starts to interfere with the functionality of the heart, blood, and vascular systems, all resulting in a higher risk of having a heart attack [4]. The lining of blood vessels can be damaged and cause blood platelets to become stickier as a result of second hand smoke [4].

1.3.1 General Population

Many people are exposed to secondhand smoke every day. 88 million nonsmoking Americans are exposed to secondhand smoke [1]. The Surgeon General's report that since 1964, 2.5 million nonsmoking adults have died as a result of inhaling secondhand smoke [4]. Each year

about 7,300 nonsmoking Americans die of lung cancer as a result of inhaling secondhand smoke [4]. It is terribly unfortunate that so many Americans that don't smoke die as a result of those who make the choice to smoke. The exposure to secondhand smoke can increase one's risk of getting lung cancer by 20-30% [1]. In addition to this, the risk heart disease increases by 25-30% [1]. As a consequence of others choosing to smoking in a public area one can drastically increase others chance of obtaining a horribly life threatening disease. In the United States alone, over than 33,000 nonsmokers die every year from coronary heart disease that was caused by some form of exposure to secondhand smoke [1].

1.3.2 Babies

Babies undergo a huge risk to the exposure of secondhand smoke. Their bodies are less developed than adults and they don't have the ability to leave an area if they somebody is smoking let alone the knowledge that inhaling secondhand smoke can be lethal. SIDS, Sudden Infant Death Syndrome, is a sudden, unexplained, unexpected death of an infant within its first year of life [4]. It is currently the number one cause of death in seemingly healthy infants [4]. Over 100,000 babies have died over the last 50 years from SIDS as a result of being exposed to secondhand smoke [1]. This is because secondhand smoke increase the risk of SIDS [4]. Infants who die of SIDS have high concentrations of nicotine in their lungs and a higher level of cotinine than infants who die from other causes [4]. The chemicals in secondhand smoke seem to affect the brain in a way that interferes with the breathing regulation of infants [4]. With secondhand smoking being so harmful to babies one would think that mothers would be aware of the danger and not smoke. However each year over 400,000 babies born in the United States are exposed to chemicals in cigarette smoke before birth because their mothers smoke [1]. Mothers who smoke during pregnancy also increase the risk of the baby having SIDS [4]. Despite second hand smoke being so harmful to the babies, mothers still do it. In the United States, babies that are 18 months or younger and exposed to secondhand smoke results in 150,000-300,00 cases of bronchitis and pneumonia each year [1]. Secondhand smoke also leads to 7,500-15,000 hospitalizations each year from babies under 18 months old [1]. This is an outrageous amount of babies that have no choice to avoid the second hand smoke that have to go to the hospital. Babies are also more

likely to get ear infections as a result of secondhand smoke [1]. Secondhand smoke is far more harmful to babies who have a less developed immune systems than fully grown adults and they have no way to avoid the smoke.

1.3.3 Children

In addition to babies, younger children also have an increased risk of secondhand smoke exposure. In 2007-2008 53.6% of children ages 3-11 years old were exposed to secondhand smoke [1]. They are also more likely to live with nonsmokers than adults are. While adults have the choice to pick roommates that will not smoke, children are at the mercy of their guardians. In the United States only 5.4% of adult nonsmokers lived with somebody who smoked inside their home from 2007-2008 [1]. Compare that to 18.2% of nonsmoking children who lived with somebody who smoking inside their home [1]. These children that are exposed to secondhand smoke are more likely to get afflicted by certain diseases. They are more susceptible to ear infections, respiratory issues, including coughing, sneezing, and having less air per breath [1]. They are also more likely to have more frequent and more severe asthma attacks [1]. Just like babies they are at an increased risk of SIDS [1]. They can also contract respiratory infections including bronchitis and pneumonia, just like the babies [1]. In addition to all of these diseases their lungs grow less than normal children who do not breathe secondhand smoke [4].

1.4 Wanting to Quit Smoking

There are many smokers who want to try to quit but cannot overcome the addictive chemicals within tobacco. It is not easy to quit and it is more difficult for people who are older to quit. Table 2 shows that many people of all ages attempt to quit and fail, with younger smokers trying to quit more often and older smokers slowly giving up.

Age of Smoker	Percent who stopped for a day to quit
All Adult Smokers	42.7%
18-24 year olds	48.5%
25-44 year olds	46.8%
45-64 year olds	38.8%
65+	34.6%
Highschool smokers	48.0%

Table 2 - Quitting Statistics

1.4.1 Quitting Techniques

There are several techniques that can be used to quit smoking. Nicotine patches or other nicotine substitutions stop smokers from committing the act of smoking but still encourage smokers to take small doses of nicotine [3]. Hypnosis is another way to try to quit smoking by having somebody tell the smoker's subconscious mind that smoking is wrong and help them unlock the will power [3]. Having a partner aid in the act of quitting smoking can also be helpful. A partner can provide real time interjections. It is less likely for a smoker trying to quit to tell a friend they need to smoke than it is for them to habitually light a cigarette without thinking about it [3].

1.5 Detection, Smartphones, and Watches

One tool that could be used to help smokers who want to quit, quit would be a smartphone paired with a smartwatch. Smartphones have grown in a way that allows them to have a lot of computing power for real time data. They have the power that some higher-end PCs had years and years ago. Smart watches come with various sensors that can detect acceleration and tilt. These sensors are the accelerometer and gyroscope respectively. These paired with the GPS sensor in the smartphone can allow us to detect when somebody is smoking and where they are when they are smoking. There has not yet been a smoking detector developed that has high

sensitivity and is easy to wear on a day-to-day basis. Precise gesture detection is vital for success of the app.

1.6 The Goal of this MQP

The objective of this project is to provide smokers who are trying to quit smoking with smoking event detection tools to help them quit. Detecting smoking events and associated context (time of day, location, etc), they can reflect on their smoking patterns, and habits. Once smokers become cognizant of their habits they can start trying to stop them. For example if a smoker knows that they have a tendency to smoke every day on the way home from work, they can make ensure that they don't leave cigarettes in their car.

In addition to providing information regarding smoking habits, the envisioned application will be able to detect a smoking session in real time. Once a smoking session is detected the user will be prompted if they are about to have a smoking session. If the smoker is trying to quit smoking and have a friend, or machine, questioning if they really want to smoke, they are less likely to smoke. It makes it less habitual and makes the smoker consciously declare that they do want to smoke.

- Provide smokers who are trying to quit with self reflection tools to monitor when and where they have been smoking to help them quit
- Detect when the user is making a smoking gesture based on sensor data from their smart watch
- Test different machine learning classifiers to determine the most optimal
- Determine what gyroscope and accelerometer features are the most effective at detecting different gestures

2. Related Work

2.1 RISQ

Parate et al [6] created a similar project where they recognized smoking gestures with inertial sensors on a Invensense MPU-9150. They used a wristband that could measure 9-axis inertial measurement to calculate the position relative to the elbow to predict if the user was smoking. They represented the space that the hand was occupying using quaternions. Quaternions are convenient mathematical entities for representing orientations and rotations of objects in three dimensions [6]. Quaternions are very good for easy calculations for performing various rotations of objects in 3D space.

$$\mathbf{q} = q_s + q_x \hat{i} + q_y \hat{j} + q_z \hat{k}$$

Equation 1 - Quaternion Equation

$$\mathbf{q} = \cos \frac{\theta}{2} - x \sin \frac{\theta}{2} \hat{i} - y \sin \frac{\theta}{2} \hat{j} - z \sin \frac{\theta}{2} \hat{k}$$

Equation 2 - Rotation

$$\mathbf{p}' = \mathbf{q} \cdot \mathbf{p} \cdot \mathbf{q}^{-1}$$

Equation 3- Quaternion Rotation

To detect the smoking patterns they had a machine learning pipeline to process the data and detect smoking gestures in real-time. They identified hand to mouth gestures while also distinguishing confounding effects such as eating and drinking. They managed to have 95.7% accuracy, 91% precision, and 81% recall. They conducted a user study that accurately depicted the number of smoking sessions a user had during the day. They used duration, speed, displacement in the Z axis, displacement in the XY plane, net displacement, roll velocity, roll, and pitch as features.

2.1.1 How RISQ Tackles the Problem

RISQ uses the accelerometer and gyroscope of a smartwatch to map the position of the user's wrist compared to the user's elbow. RISQ does this through the use of quaternions. To detect smoking gestures they use a machine learning pipeline to process the data and provide real time interjections to see if the user is smoking.

RISQ also acknowledges that detecting a smoking gesture could be easily done using an instrumented lighter. Whenever the lighter was used, it could ping one's phone to record a smoking session.

2.1.2 Comparison of our work to RISQ

While it was initially a goal to provide smoking gestures in real time we were not able recognize gestures with an accuracy as high as RISQ. We did try to use the same quaternion mapping but it proved quite challenging. We did use the same type of sensor (accelerometers and gyroscope) on a smartwatch that was only worn on the dominant smoking hand. RISQ had hundreds of hours to train their machine learning classifier while we had far less training. They used displacement and time domain features to determine where the hand was and then determine smoking gestures. This project did not derive these features.

2.2 Automated Detection of Puffing and Smoking with Wrist Accelerometers

Quan Tang from Northeastern university also had a similar project using wrist accelerometers for smoking detection for his Master thesis [7]. They suggest that real time automatic detection of smoking behavior allows for “just-in-time” interventions that are very effective at helping smokers quit. Using machine learning with multiple wrist accelerometers allows them to pinpoint when the user is taking a puff and allows them to make that intervention. They utilized 4 accelerometers placed across the body. They had one on the ankle, one on each wrist, and one on the shoulder of the smoking hand. They implemented a two layer smoking detection model that had both low level time domain features such as inhalation, exhalation and respiration duration,, IE ratio, stretch and high level puff frequency detection such as pitch, roll of lower, and upper arm sensors. They observed 6 individuals performing complex tasks while smoking for 11.8 hours. It was a real-life setting and they managed to get a cross validation F1-

score of .70 for puff detection and .79 for smoking detection overall. They had a mean F1-score of .75 for puffing detection while using user-specific training data. Some of their biggest challenges were distinguishing confounding effects such as drinking beverage, eating a meal, talking, using computer and using phone.

2.2.1 How “Automated Detection of Puffing and Smoking with Wrist Accelerometers” Tackles the Smoking Gesture Recognition Problem

Northeastern used multiple sensors to detect various smoking gestures such as multiple accelerometers placed across the body. They had one on the ankle, one on each wrist, and one on the shoulder of the smoking hand. This allowed them to better distinguish gestures that were confounding but used two hands, for example eating a burger. It also allowed them to get different positions that people would smoke in and allowed them to determine if they were sitting down or standing up.

2.2.2 Comparison of our work to “Automated Detection of Puffing and Smoking with Wrist Accelerometers”

Our project had far less sensors than the above project. While they had multiple sensors on each wrist, we only had a sensor for one wrist. If we had a sensor for each appendage then users would be much less inclined to put on all of the sensors. Unfortunately this means that the sensors may not be able to capture all relevant data, which could help identify a smoking gesture. We also didn't have sensors anywhere else on the body. Using all of the sensors would probably be much more work to have them all interact together however it probably can produce better results. That being said it is also much more cumbersome for a user to wear all of the sensors as opposed to just wearing one Android watch.

3. Methodology

In order to begin to explore and develop the app necessary to combat the issue of smoking, we will have 30 subjects perform smoking and non-smoking gestures while wearing an android wristwatch. Data can be collected about these gestures using the accelerometer and gyroscope onboard the watch and eventually used to train an algorithm to classify the gestures. Below in figure 1 is an overview of our system's architecture.

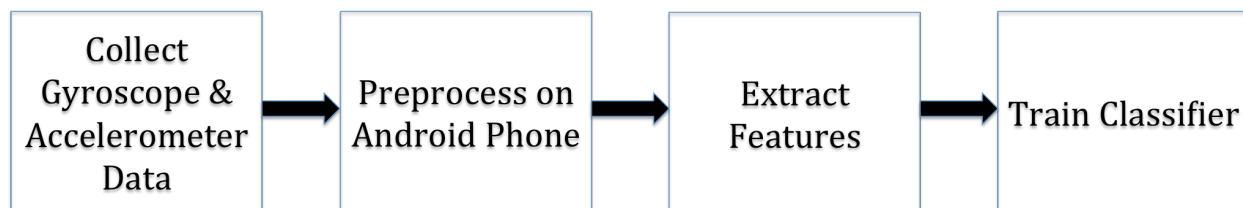


Figure 1 – System Diagram

3.1 Watch Selection

When deciding which smartwatch to use there were a few factors that were taken into account. The first factor for deciding which smartwatch to use was what sensors were available. An elegant way to detect gestures would be to use a smart watch's accelerometer and gyroscope. Pebble Time, Pebble Steel, Lg G Watch R, Sony Smartwatch 3, and the Asus ZenWatch all had the two sensors and were considered.

The cost effectiveness was weighted heavily in the consideration of the watch selection so if it was adopted it could reach a much wider audience. The Asus ZenWatch, shown in figure 1, was substantially cheaper than the competitors and was purchased refurbished for \$80. Some tradeoffs that were sacrificed for cost efficiency were battery power and processing power. Another important factor in watch selection was the watch's compatibility with Android software.

Smart-watches compared

Apple's Watch will be on sale starting next month. Here's how it stacks up to the competition.



	Apple Watch	Huawei Watch	Pebble Time	LG Watch Urbane LTE	Asus ZenWatch
Processor speed	n.a.	1.2 Ghz	100Mhz	1.2 Ghz	1.2 Ghz
Storage	n.a.	4GB	2MB	4GB	4GB
RAM	n.a.	512MB	64KB	1GB	512MB
Heart rate monitor	Yes	Yes	No	Yes	Yes
Touchscreen	Yes	Yes	No	Yes	Yes
NFC	Yes	No	No	Yes	No
Built-in GPS/LTE	No/No	n.a.	No/No	Yes/Yes	No/No
SIM card slot	No	No	No	No	No
Connectivity	Proprietary	Micro USB	No	Micro USB	Micro USB
OS	iOS	Android Wear	Pebble OS	LG WPOS ¹	Android Wear
Camera	No	No	No	No	No
Bluetooth	4.0	4.0	4.0	4.0	4.0
Voice control	Siri	Google Now	Yes	Q Voice	Google Now
Body material	Alum.,SS, or gold	SS, Sapphire crystal	Plastic/SS	Stainless steel	Stainless steel
Swappable band	Yes	Yes	Yes	Yes	Yes
Weight	72g to 125g with band	n.a.	42.5g	na	50g /75g with strap
Water/dust resistance*	Yes**	Yes**	Yes**	IP67	IP55
Size (L x W) in mm	38.6 x 33.3/42 x 35.9	42 (diameter)	47 x 40	46.5 (diameter)	50.6 x 39.8
Display type	OLED	AMOLED	Color, e-paper	Super AMOLED	OLED
Screen size	1.5/1.7 in	1.4 in (diameter)	n.a.	1.3 in (diameter)	1.63 in (diagonal)
Screen resolution (px)	340 x 272/390 x 312	400 x 400	144 x 168	320 x 320	320 x 320
Battery capacity	n.a.	300mAh	n.a.	700mAh	369mAh
Battery life per charge	Up to 18 hours	n.a.	7 to 10 days	~2 days	~1 day
Compatibility	iOS phones	Android phones	iOS and Android	n.a.	Android
Price in the U.S.	Starts at \$349	n.a.	\$199/\$250 (Steel)	n.a.	\$200
Release	April 2015	Mid-2015	July 2015	n.a.	Nov. 2014

Sources: Companies; media reports. *Higher number rating is better. SS=Stainless steel **Rating not available. ¹Wearable Platform OS
C. Inten, 10/03/2015

Figure 2 – Watch Comparison

3.2 Considerations

Due to the limited number of members working on the project in addition to the limited amount of time we had to work on the project, there were a handful of things that could not be addressed in the scope of this project. Should there be people to continue this project moving forward these are some challenges that they could choose to tackle to improve the accuracy of smoking detection.

3.2.1 Confounding Effects

Confounding effects are various gestures and actions that have very similar motion to smoking. For the scope of our project we chose to ignore most confounding effects however we do acknowledge that they exist and that they are a big obstacle. The ability to identify confounding effects would greatly improve the accuracy of smoking detection. When smoking the user moves his hand to his mouth for a puff then moves the hand to a resting position before moving it back to the mouth for another puff. This action is frequently repeated within the smoking session but also similar to other actions which we explore such as drinking and receiving a phone call. These confounding gestures are now discussed in some detail in the following sections.

3.2.1.1 Drinking beverage

If a person were to drink a beverage they would mimic a similar effect to smoking a cigarette. They would probably uncapp the bottle or place a straw in the drink before moving it to their mouth. Once they took a sip, they would bring the cup or bottle back to a resting position until they wanted to take another sip. They would continue this until they were finished with the beverage. This could create false positives when detecting if the user is smoking or not.

Some solutions that could be used to mitigate this could be detecting for some gesture of uncapping a bottle or placing a straw in a cup. If one of these gestures was detected then the algorithm for detecting a puff would know to not send a false positive. When a person is drinking out of a bottle one typically need to tilt one's head back a little bit more and one's wrist needs to travel a further distance to reach one's mouth. While a person is smoking their head is in a resting position and there is no need to tilt one's head back. In both sipping through straws and bottles both typically have the wrist at a further location from the mouth. In the case where one might be holding a cup from the top while drinking through a straw, the orientation of the wrist is very different and could be used to distinguish it from a smoking gesture.

3.2.1.2 Eating a meal

Eating also creates a very similar effect to that of smoking. Whether the user is eating something with their hands or using some sort of utensil, confounding effects could be present. When eating some food such a burger, since the accelerometer and gyroscope are only on one wrist, it would be challenging to distinguish from smoking. Eating a food that required one hand to eat such as chicken nuggets would also be very similar to smoking. While using utensils the orientation of the wrist is a little different and there typically isn't a resting position as one has to bring the utensil to the food and then back to the mouth.

Some solutions to distinguish eating from smoking could be checking the gyroscope orientation to see if the user is using some silverware or utensils to eat. The orientation of the wrist is typically different when using these tools. The algorithm and machine learning agent could also take into account various resting positions. If the user is eating something like spaghetti then they would need to be constantly moving their wrist to get the spaghetti on the fork. Eating something like a stake would require a very interesting orientation of the wrist to hold the steak down with the fork or cutting it with the knife. While eating food with hands such as chicken nuggets the consumer of the food would need to bring the food to their mouth then to a resting position followed by the position of the food before bringing it to their mouth again. These three positions would look different than the two resting positions of smoking. There could also be some use of a GPS feature to see if the user is in a smoke-free restaurant.

3.2.1.3 Talking

When talking or telling a story, people tend to use hand gestures and body language to enhance their communication. Some people may wave their hands when they are excited and if they happened to move them from a resting position to near their mouth this could be confused with a smoking gesture. If someone was to come from the gym and simulate a curling gesture, this could also be confused with a confounding effect.

Some ways to address this could be the time and frequency of the actions. Typically while talking to somebody and using hand gestures there are many different positions and rarely two primary positions. Smoking almost always has a resting position and the position when the user is taking a puff. Talking has much more sporadic gestures and multiple positions.

Talking while smoking however is an even more confounding problem. It would change what the typical smoking gesture looks like as the user may be using their hands to talk and thus not bring them back to a resting position.

3.2.1.4 Using a Phone

When making a phone call the wrist is often near the face as well. If somebody is checking their phone constantly and making a series of phone calls this could confuse the detection service and create a false positive. Typically the average length of a phone call is much longer than the average length of a puff. This could be used to distinguish making a phone call from a smoking gesture.

3.2.2 Multiple Resting Positions

Currently the project has a single resting position and a position where the user is smoking the cigarette. If the user was to have multiple resting positions this would currently interrupt our data. It would be good in the future of this project if position could be used instead of acceleration or velocity. In order to do this a more accurate accelerometer would be needed as there was a lot of noise and fluctuation even at a resting position. While it is almost negligible in terms of acceleration, after two derivations it becomes a significant number. Using position to just locate the hand to the mouth and determine the length of a puff would be more accurate than using acceleration and velocity.

3.2.3 Pauses

For our test cases we are making the assumption that a gesture is either coming from a resting point going to the mouth or another location, or coming from one of those locations to the resting point. This is determined by using acceleration to determine when the gesture has stopped and ended. A positive acceleration followed by a negative acceleration indicates the user started moving and then stopped. If the user were to move their arm half way to the smoking position

and then stop, only to move it all the way to the smoking position afterwards, it would trick the system. A solution to prevent this would be to use position as opposed to acceleration and velocity. This could be achieved with a more accurate accelerometer.

3.2.4 Opposite Hand

Due to the fact that we only have one accelerometer and gyroscope we currently are assuming that the user will smoke with the hand associated with the watch. If the user were to switch hands and smoke with the other hand we currently have no way to detect that they are smoking. To address this either data would be needed for what the off smoking hand looks like or another sensor would be needed.

3.3 Participant Selection

Participants were randomly selected in the WPI campus center with no incentives. We would approach people who seemed not to be busy and ask if they would be interested in helping us with our MQP. Table 3 shows the distribution of genders and ages that were selected for this study. The participants were then shown and described the gestures.

	19 Years Old	20 Years Old	21 Years Old	22 Years Old
Male	2	4	12	9
Female	0	1	0	2

Table 3 - Participant Selection

3.4 Data Gathering Study

Steps:

- 1) Subject wears watch on right hand.
- 2) Subject performs 30 smoking gestures by:
 - a) Subject fixes right elbow on the table palm up.
 - b) Subject holds straw, cut to the length of a cigarette, in hand simulating holding a cigarette.

- c) Subject brings straw to lips to simulate a puff then brings their arm back down to the table.
- 3) Subject performs 30 drinking gestures by:
 - a) Subject holds a cup while keeping their arm on the table.
 - b) Subject raises cup so simulate the start of taking a drink.
 - c) Subject brings cup back to resting position.
- 4) Subject performs 30 phone calls by:
 - a) Subject holds a phone palm up on the table while keeping their arm on the table.
 - b) Subject brings the phone to their ear to simulate taking a phone call.
 - c) Subject brings the phone back to the resting position face up on the table.
- 5) Subject performs 30 arm cross gestures by:
 - a) Subject has arms on table in a natural resting position.
 - b) Subject crosses arms to simulate crossing arms in a natural position.
 - c) Subject moves arms back to resting position.

Below in figures 3-10 are images of the gestures we had participants perform for data collection.



Figure 3 - Cigarette Simulation (Resting)



Figure 4 - Cigarette Simulation (Active)



Figure 5- Drinking Simulation (Resting)



Figure 6 - Drinking Simulation (Active)



Figure 7 - Phone Simulation (Resting)



Figure 8 - Phone Simulation (Active)



Figure 9 - Arms Cross Simulation (Resting)



Figure 10 - Arms Cross Simulation (Active)

4. Implementation

Our implementation is three-fold, with a smartwatch collecting the data, an android phone extracting features from it, and finally classification and machine learning on a computer. We chose this design in order to keep the smartwatch app as lightweight as possible (reading sensor data) while still allowing us the opportunity for real time gesture recognition. Additionally, it allowed us to extract data from watches that may not have an easily accessible file structure or USB port. Below are figure 11 and 12 showcasing the apps we built to collect data.

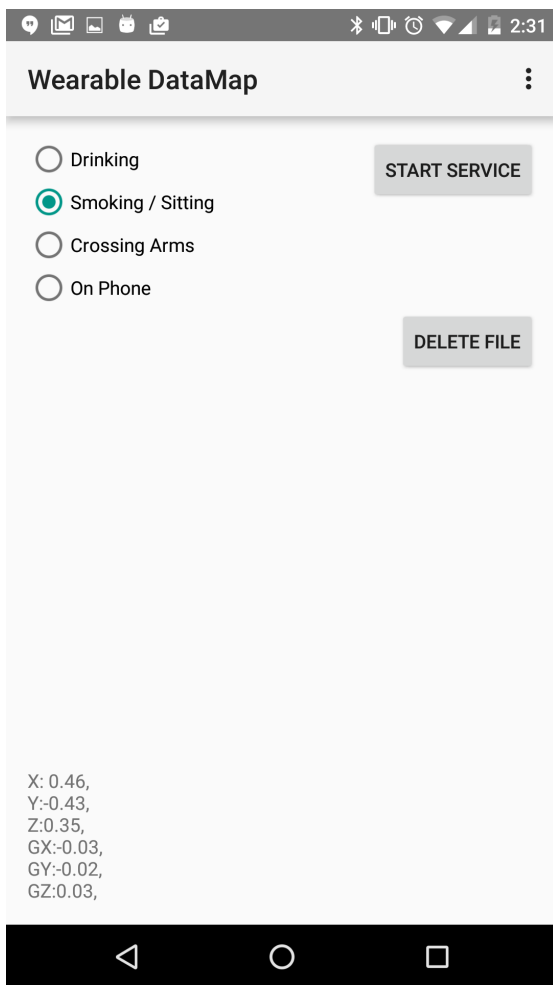


Figure 11 - Android app interface



Figure 12 - Basic Watch Screen

4.1 Data Collection

Data collection was performed on the smartwatch, and sent to the smartphone using Android's DataAPI. We collected gyroscope data and gravity corrected accelerometer readings in order to fuse the two into a more robust quaternion output. The data is sent in blocks of 64 readings, and data is collected as fast as the watch hardware will provide (around 100 hz). No sugaring or filtering is performed on the data; as soon as the buffer fills it is offloaded to the phone for feature extraction. Figure 13 shows the initialization of the Android sensor manager and the accompanying accelerometer and gyroscope.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {

    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

    List<Sensor> sensorList = mSensorManager.getSensorList(Sensor.TYPE_ALL);
    for (Sensor s : sensorList) {
        Log.v("MQP", "Sensor "+s.getName()+", "+s.getType());
    }

    mAccelerometer = mSensorManager
        .getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);

    mGyroscope = mSensorManager
        .getDefaultSensor(Sensor.TYPE_GYROSCOPE);

    mRotationVector = mSensorManager
        .getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);

    mSensorManager.registerListener(this, mAccelerometer,
        SensorManager.SENSOR_DELAY_FASTEST);
    mSensorManager.registerListener(this, mGyroscope,
        SensorManager.SENSOR_DELAY_FASTEST);
    mSensorManager.registerListener(this, mRotationVector,
        SensorManager.SENSOR_DELAY_FASTEST);

    mServiceTaskType = SERVICE_TASK_TYPE_COLLECT;

    mAsyncTask = new OnSensorChangedTask();
    mAsyncTask.execute();

    return START_NOT_STICKY;
}
```

Figure 13 - Initialization of Sensors on Watch

4.2 Feature Extraction

Once the data is unpacked on the phone, our first step is to determine if a gesture has begun within the given block. To do this, we use a high pass filter on the magnitude of force applied. If the magnitude exceeded a certain threshold, we declare that block as the first chunk of a gesture. Figure 14 shows the code that finds the start of a gesture by iterating over all accelerometer readings and starting a gesture series when the readings exceed 1m/s.

```
else if (gestureNum == 0) {
    // check whether this segment starts a gesture

    float[] x = dataMap.getFloatArray("x");
    float[] y = dataMap.getFloatArray("y");
    float[] z = dataMap.getFloatArray("z");

    String sp = "";

    for (int i = 0; i < x.length; i++) {

        float speed = (float) Math.sqrt(Math.pow(x[i],2)+Math.pow(y[i],2)+Math.pow(z[i],2));
        sp = sp+""+speed+", ";

        if (speed > 1.f) {
            gestureSeries[0] = dataMap;
            gestureNum = 1;

            Log.v("MQP", "Gesture begin with speed" + speed);
        }
    }
}
```

Figure 14 – Android Code to Detect the Start of a Gesture

We decided on the threshold based on observation of the accelerometer data. We found that an average reading for a resting hand position was between 0 and 1 m/s², and the accelerometer readings for a moving hand varied widely between 1 and 2m/s². Thus, we chose 1 for our gesture threshold. Below figures 15, 16, and 17 show sample accelerometer readings from the wristwatch which justify our choice of this threshold. Figure 15 shows two instances of the watch in a resting position, demonstrating the noise generated at rest.

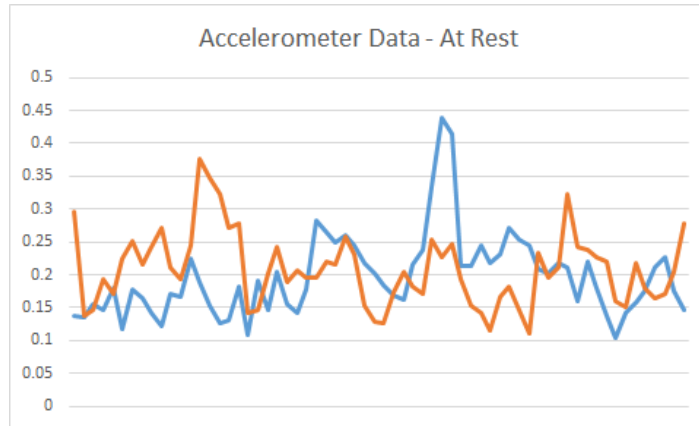


Figure 15 - Accelerometer Data of Hands at Rest

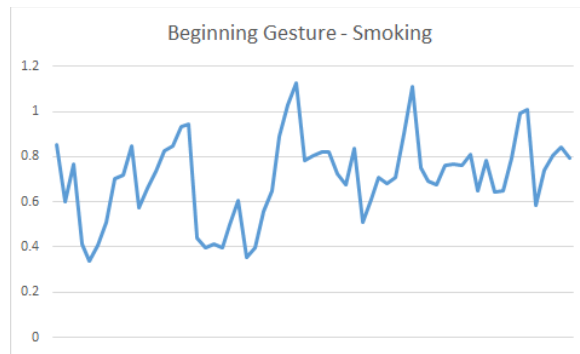


Figure 16 - Accelerometer Data of Start of Smoking Gesture



Figure 17 - Accelerometer Data of Start of Drinking Gesture

The gesture lasts for about 2 seconds, the average length of a smoking puff, with each chunk of data appended for later processing. At the conclusion of the gesture, various features are extracted from the accelerometer and gyroscope data. We encode the gesture into ARFF format and save it into a file for WEKA classification, as shown in figure 19. The features we use are listed below in Table 3, with a sample of a complete gesture in figure 18.

Features	Equation	Use
Max Speed (Accelerometer)	Maximum accelerometer reading for a given gesture	Used to detect quick changes of acceleration such as the start of a gesture
Median Speed (Accelerometer)	The second quartile accelerometer reading	Used in conjunction with mean to detect gestures that start strongly and end softly
Mean Speed (Accelerometer)	$\bar{X} = \frac{\sum X_i}{n}$	Used in conjunction with median to detect gestures that start strongly and end softly
Variance of Speed (Accelerometer)	$s^2 = \frac{\sum (X - \bar{X})^2}{N - 1}$	Used to differentiate gestures with a constant speed or a variable speed
Net roll velocity (Gyroscope)	ΣV	Used to detect how much the wrist has rotated in a given gesture
Median roll velocity (Gyroscope)	The second quartile gyroscope reading	Used to detect gestures that start with a quick rotation and end with little to no rotation
Max roll velocity (Gyroscope)	Maximum gyroscope reading for a given gesture	Used to detect gestures with a sharp changes in rotation

Table 3 - Features extracted

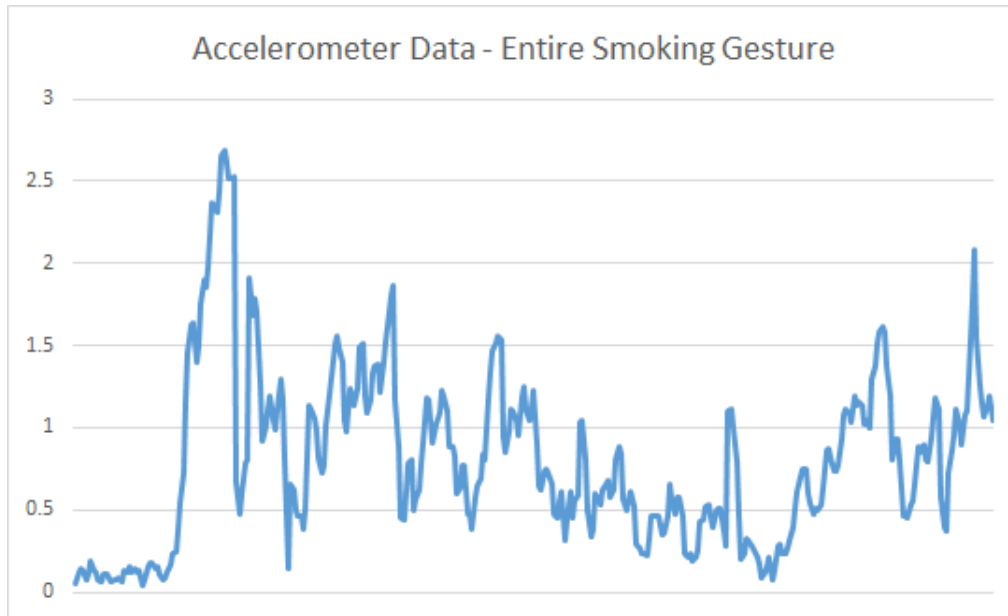


Figure 18 - Accelerometer data of a complete smoking gesture

```
Instance inst = new DenseInstance(attributes.size());
inst.setDataset(mDataset);

inst.setValue(mDataset.attribute("avg_speed"), mean_speed);
inst.setValue(mDataset.attribute("median_speed"), median_speed);
inst.setValue(mDataset.attribute("max_speed"), max_speed);
inst.setValue(mDataset.attribute("variance_speed"), variance_speed);

inst.setValue(mDataset.attribute("net_roll_velocity"), net_roll_velocity);
inst.setValue(mDataset.attribute("median_roll_velocity"), median_roll_velocity);
inst.setValue(mDataset.attribute("max_roll_velocity"), max_roll_velocity);
inst.setValue(mClassAttribute, mClass);

mDataset.add(inst);
```

Figure 19 - Code to Create a WEKA-Compatible Data Instance

4.3 Classification

With the features extracted from the raw data, we encode the results into an ARFF file and then use WEKA to classify each gesture. Using a given machine-learning classifier we were able to build a tree to predict the class recognized gestures. We compared various classifiers in Weka for the task of classifying the gestures performed by our subject. The best we found, random forest, constructs an ensemble of decision trees based on our features and the gesture

data. This ensemble of trees assigns weights based on each feature's effectiveness, and once constructed performs a 10 fold cross validation to output its effectiveness. Figure 20 shows what output WEKA gives from our data.

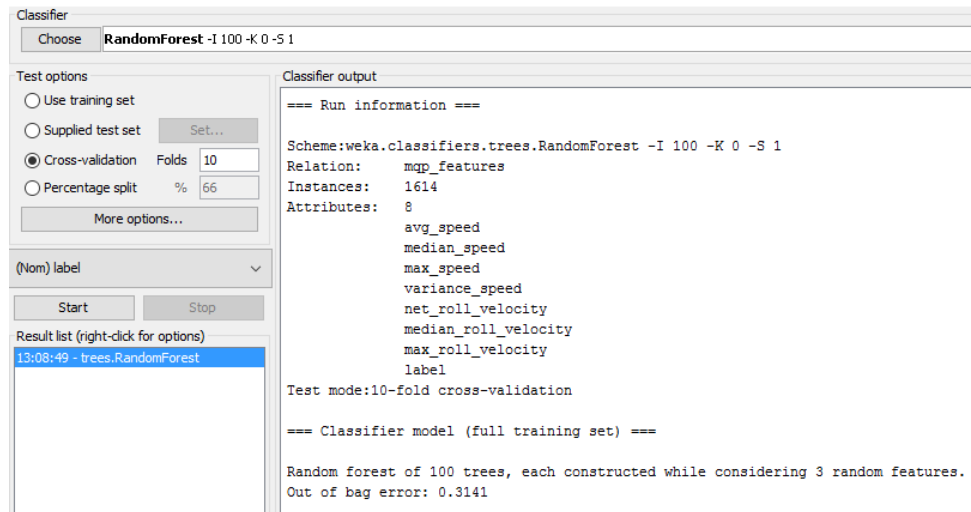


Figure 20 - WEKA Random Forest Classification

5. Results

Using data from all 30 of our participants, we achieved an overall accuracy of 63.7% using a random forest classifier. This means that we are able to, given a gesture detected by our app, classify it among drinking, crossing arms, smoking, or using a phone correctly 63.7% of the time. Below, table 4 shows a confusion matrix illustrating the confounding gestures:

Drinking	Crossing Arms	Smoking	Using Phone	
618	45	60	39	Drinking
45	439	182	160	Crossing Arms
22	123	515	150	Smoking
44	160	152	509	Using Phone

Table 4 - Confusion Matrix

As we can see from this, smoking is most confounded with using a phone due to the similar arc of motion. Depending on the action, we have relatively good TP rates and relatively low FP rates shown in table 5.

Class	TP	FP	Precision	Recall	F-Measure	ROC-Area
Drinking	0.811	0.044	0.848	0.811	0.829	0.955
Crossing	0.531	0.135	0.572	0.531	0.551	0.811
Smoking	0.636	0.161	0.567	0.636	0.600	0.839
Phone	0.588	0.146	0.593	0.588	0.591	0.828

Table 5 - Experimental Results

Interestingly, with this data we can see that we have the lowest TP rate for phone use, which could point out error in our feature selection for the gestures; perhaps rotation is the most important feature and acceleration is weak in comparison. Crossing arms and using a phone should be two of the least confused gestures because they operate on entirely different axes. This shows that we need to capture additional features from our accelerometer data.

Other machine learning classifiers did not produce as impressive results. We also ran our data through J48, JRip, SVM, and Naive Bayes classifiers to see how they fared. Table 6 below contains the overall results of each.

	TP	FP	Precision	Recall	F-Measure	ROC-Area
J48	0.579	0.143	0.580	0.579	0.579	0.745
JRip	0.558	0.154	0.599	0.558	0.557	0.769
SVM	0.599	0.134	0.591	0.599	0.585	0.765
Naive Bayes	0.558	0.146	0.565	0.558	0.539	0.778
Simple Cart	0.619	0.129	0.625	0.619	0.62	0.822

Table 6 – Comparison of Classifiers

6. Discussion

6.1 Lessons Learned

Gyroscope was most distinguishing sensor: While it may seem intuitive that crossing arms should be very distinguishable from smoking, drinking, and taking a phone call, it was not the case. This was a result of the fact that almost all of the gestures had a very similar acceleration and the gyroscope provided the best readings to distinguish gestures. This was a result of not always properly distinguishing various axes of acceleration. As a result crossing arms often fooled the classifier.

Random Forest was the best classifier type: We also learned that the random forest provided the best results. This is due to the fact that many simple trees seem to perform better than a single advanced tree. A tree that takes into account all of the features has to make a hard choice and pick one way or another at a fork in a tree. Taking 100 simple trees that use only some of the features and then weighing the trees on performance removes that hard fork in the tree. Instead the results of the small trees and their corresponding weight are used to calculate the gesture being performed.

6.2 Limitations of Wrist Watch Sensing

Using a smartwatch for aiding smokers in the quest to quit does have its limits. Smokers can easily fool the software by smoking in their non-dominant hand, removing the wristwatch, or even just leaving the cigarette in their mouth and ignoring hand gestures altogether. No matter how many sensors we would have in addition to the watch, users could always find a way around it. We envision the type of people to use this tool would have to work ‘with’ it, so to speak. Users would need to make a conscious effort to quit smoking and not work against the app. If they choose to cheat the app, we are unable to intervene unlike a physical friend could in the real world.

Other limitations of watch technology are that of the battery life and sensor accuracy. Collecting data from the accelerometer and gyroscope constantly is very battery intensive, and with testing we found our app (even though it is as lightweight as possible) would deplete the battery of the watch in around three to four hours. Even though they drained quite a bit of

battery, the sensors were still not that accurate. The noise generated by the watch sensors even at rest was up to 0.5m/s^2 , far too much considering that most gestures took place between 1 and 2m/s^2 . The technology would have to improve more if we want to accurately map acceleration to position

6.3 Implications of Smoking Gesture Recognition

Our app shows that many of the technologies for a final product are well on their way to maturity. Though smartwatches are only in their infancy the technology onboard is enough to extract meaningful features from the accelerometer and gyroscope and process that data in real time. If the techniques we used for data processing were improved upon with additional tinkering, we can see that a functional product could emerge that could provide smokers help in the real world. A companion app utilizing this the gestures that we recognize could help smokers make sense of their habits, and our research shows that smart wristwear technology is on the verge of this dream becoming reality.

7. Conclusion

We successfully developed a system capable of recording users' gestures, extracting features, and applying a machine learning algorithm to classify the gestures. Using data from 30 subjects, we were able to train the algorithm and classify the gestures with an overall accuracy of 63%. There is still much work to do on the topic, and more features will need to be added in order to better differentiate an increased number of gestures. We are proud of the results we were able to get, however, and are confident that technologies like this can be built out to provide a real difference in people's struggle to quit smoking.

7.1 Future Work

If there was more time or resources that could be used to further our app, we could look at a few things to improve our effectiveness at detecting and stopping smoking: additional features, quaternion displacement, and a fully-fledged companion app.

Quaternion representation and pre-processing: The features extracted from our data do not do a great job of differentiating smoking and using a phone, so more features which deal with rotation could help lessen the confusion of the two. Quaternion displacement is a feature we were unable to implement but is present in the RisQ paper, and would also greatly increase our accuracy. Additionally, we process features from the raw data currently which causes noise to introduce error. To get around this, a rolling average or other smoothing techniques could be performed on the data in the data-pre-processing step. Perhaps with this, derived features could also be explored to find new relationships in the data.

Develop companion app: The companion app would also assist users in quitting, as it could provide useful information to users such as a heat map of smoking locations. If a user can know their habits then they are better prepared to prevent their habits. Providing a heat map could allow users to know that they have a tendency to smoke in a car or smoke on their lunch break. This data could be crucial in helping people quit.

GPS Feature: Using a GPS feature on the phone could help the user know their smoking patterns. Once a smoking gesture is registered the phone could check its location using a GPS

feature that are common on most smart phones. The user may think that they smoke when they are stressed out and try to not smoke when stressed. After some smoking sessions are recorded the user may find that based on the GPS locations, they always smoking on their way to work. With the knowledge of where they smoke they can take action to prevent smoking at that location. Maybe they decide to not have a pack of cigarettes in their car. Knowledge and cognizance of smoking locations is a great way to stop the addictive habits.

Real time alerts: Having a real time alert system is very important to provide real time interventions. If the phone can identify that the user is currently smoking based on the data that is being sent from the watch, then they can provide a message to the user. If the user is actively trying to quit smoking and they receive a message asking them if they really want to smoke, they have to actively say they do want to smoke and admit defeat in their attempt to quit. It is a lot tougher to say I want to smoke than just lighting up a cigarette. The real time alert could also be used to determine if the user is in fact smoking or not. If there was a real time alert asking if the user wanted to smoke and they were not smoking, they could let the application know that is just produced a false positive which could help the machine learning agent to correct false positives and better detect smoking patterns in the future. Below figure 2 shows a mock-up of what such an app would look like.










































Figure 2 - Smoking Tracker

References

- [1] Facts About Smoking and Tobacco Use, “Tobacco Facts and Figures,” 2016. [Online]. Available: <http://betobaccofree.hhs.gov/about-tobacco/facts-figures/>. [Accessed 15 February 2016]
- [2] Centers for Disease Control and Prevention, “Fast Facts,” 2014. [Online]. Available: http://www.cdc.gov/tobacco/data_statistics/fact_sheets/fast_facts/. [Accessed 15 February 2016]
- [3] Smokefree.gov, “Find a Quit Method That Works For You,” 2016. [Online]. Available: <http://smokefree.gov/explore-quit-methods>. [Accessed 17 February 2016]
- [4] Centers for Disease Control and Prevention, “Health Effects of Secondhand Smoke.” 2014. [Online]. Available: http://www.cdc.gov/tobacco/data_statistics/fact_sheets/secondhand_smoke/health_effects/. [Accessed 22 February 2016]
- [5] Smokefree.gov, “Quitting Is Hard,” 2016. [Online]. Available: <http://smokefree.gov/why-quitting-is-hard>. [Accessed February 27 2016]
- [6] Parate, Abhinav, Meng-Chieh Chiu, Chaniel Chadowitz, Deepak Ganesan, and Evangelos Kalogerakis. "RisQ." Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '14, 2014.
- [7] Tang, Qu. "Automated Detection of Puffing and Smoking With Wrist Accelerometers." Northeastern University, 2016.

Appendix

Project Structure

- ▼  **display**
 - ▶  manifests
 - ▼  java
 - ▼  com.example.display
 -   MainActivity
 - ▶  com.example.display (androidTest)
 - ▶  res
- ▼  **mobile**
 - ▶  manifests
 - ▼  java
 - ▼  com.androidweardocs.wearableadatamap
 -   Matrix4
 -   MobileDataMapActivity
 -   MobileListenerService
 -   Quaternion
 -   Vector3
 -   Vector4
 - ▶  com.androidweardocs.wearableadatamap (androidTest)
 - ▶  res
- ▼  **wear**
 - ▶  manifests
 - ▼  java
 - ▼  com.androidweardocs.wearableadatamap
 -   SensorService
 -   WatchDataMapActivity
 -   WatchListenerService
 - ▶  res
- ▶   Gradle Scripts

SensorService.java

```
package com.androidweardocs.wearableDatamap;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.os.IBinder;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import android.widget.Toast;

import com.google.android.gms.wearable.DataMap;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ArrayBlockingQueue;

import weka.core.Attribute;
import weka.core.DenseInstance;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ArffSaver;
import weka.core.converters.ConverterUtils.DataSource;

public class SensorService extends Service implements SensorEventListener {

    private LocalBroadcastManager localBroadcastManager;

    private static int ACCELEROMETER_BLOCK_CAPACITY = 64;
    private static int ACCELEROMETER_BUFFER_CAPACITY = 2048;

    private static int mFeatLen = ACCELEROMETER_BLOCK_CAPACITY + 2;

    private static String CLASS_LABEL_KEY = "label";
    private static String CLASS_LABEL_SMOKING = "smoking";
    private static String CLASS_LABEL_PUFFING = "not_smoking";

    private static int SERVICE_TASK_TYPE_COLLECT = 0;
    private static int SERVICE_TASK_TYPE_CLASSIFY = 1;

    private File mFeatureFile;
    private SensorManager mSensorManager;
    private Sensor mAccelerometer;
```

```

private Sensor mGyroscope;
private Sensor mRotationVector;

private int mServiceTaskType;
private String mLabel;
private Instances mDataset;
private Attribute mClassAttribute;
private OnSensorChangedTask mAsyncTask;

private static ArrayBlockingQueue<Float> mAccBufferX;
private static ArrayBlockingQueue<Float> mAccBufferY;
private static ArrayBlockingQueue<Float> mAccBufferZ;

private static ArrayBlockingQueue<Float> mGyroBufferX;
private static ArrayBlockingQueue<Float> mGyroBufferY;
private static ArrayBlockingQueue<Float> mGyroBufferZ;

private static ArrayBlockingQueue<Float> mRotationBufferX;
private static ArrayBlockingQueue<Float> mRotationBufferY;
private static ArrayBlockingQueue<Float> mRotationBufferZ;
private static ArrayBlockingQueue<Float> mRotationBufferW;

public static final DecimalFormat df = new DecimalFormat("#.##");

@Override
public void onCreate() {
    super.onCreate();

    Log.v("MQP", "Sensor Service Started");

    localBroadcastManager = LocalBroadcastManager.getInstance(this);

    mAccBufferX = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);
    mAccBufferY = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);
    mAccBufferZ = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);

    mGyroBufferX = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);
    mGyroBufferY = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);
    mGyroBufferZ = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);

    mRotationBufferX = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);
    mRotationBufferY = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);
    mRotationBufferZ = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);
    mRotationBufferW = new ArrayBlockingQueue<Float>(ACCELEROMETER_BUFFER_CAPACITY);
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {

    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

    List<Sensor> sensorList = mSensorManager.getSensorList(Sensor.TYPE_ALL);
    for (Sensor s : sensorList) {
        Log.v("MQP", "Sensor "+s.getName()+"", "+s.getType());
    }
}

```

```

mAccelerometer = mSensorManager
    .getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);

mGyroscope = mSensorManager
    .getDefaultSensor(Sensor.TYPE_GYROSCOPE);

mRotationVector = mSensorManager
    .getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);

mSensorManager.registerListener(this, mAccelerometer,
    SensorManager.SENSOR_DELAY_FASTEST);
mSensorManager.registerListener(this, mGyroscope,
    SensorManager.SENSOR_DELAY_FASTEST);
mSensorManager.registerListener(this, mRotationVector,
    SensorManager.SENSOR_DELAY_FASTEST);

mServiceTaskType = SERVICE_TASK_TYPE_COLLECT;

mAsyncTask = new OnSensorChangedTask();
mAsyncTask.execute();

return START_NOT_STICKY;
}

@Override
public void onDestroy() {
    mAsyncTask.cancel(true);
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    mSensorManager.unregisterListener(this);
    super.onDestroy();
}

private class OnSensorChangedTask extends AsyncTask<Void, Void, Void> {
    @Override
    protected void doInBackground(Void... arg0) {

        Instance inst = new DenseInstance(mFeatLen);
        inst.setDataset(mDataset);
        int blockSize = 0;
        float[] accBlockX = new float[ACCELEROMETER_BLOCK_CAPACITY];
        float[] accBlockY = new float[ACCELEROMETER_BLOCK_CAPACITY];
        float[] accBlockZ = new float[ACCELEROMETER_BLOCK_CAPACITY];

        float[] gyroBlockX = new float[ACCELEROMETER_BLOCK_CAPACITY];
        float[] gyroBlockY = new float[ACCELEROMETER_BLOCK_CAPACITY];
        float[] gyroBlockZ = new float[ACCELEROMETER_BLOCK_CAPACITY];

        float[] rotationVectorX = new float[ACCELEROMETER_BLOCK_CAPACITY];
        float[] rotationVectorY = new float[ACCELEROMETER_BLOCK_CAPACITY];
        float[] rotationVectorZ = new float[ACCELEROMETER_BLOCK_CAPACITY];
        float[] rotationVectorW = new float[ACCELEROMETER_BLOCK_CAPACITY];

        long time = System.currentTimeMillis();

```

```

while (true) {
    try {
        // need to check if the AsyncTask is cancelled or not in the while loop
        if (isCancelled () == true)
        {
            return null;
        }

        // Dumping buffer
        accBlockX[blockSize] = mAccBufferX.take().floatValue();
        accBlockY[blockSize] = mAccBufferY.take().floatValue();
        accBlockZ[blockSize] = mAccBufferZ.take().floatValue();

        gyroBlockX[blockSize] = mGyroBufferX.take().floatValue();
        gyroBlockY[blockSize] = mGyroBufferY.take().floatValue();
        gyroBlockZ[blockSize] = mGyroBufferZ.take().floatValue();

        rotationVectorX[blockSize] = mRotationBufferX.take().floatValue();
        rotationVectorY[blockSize] = mRotationBufferY.take().floatValue();
        rotationVectorZ[blockSize] = mRotationBufferZ.take().floatValue();
        rotationVectorW[blockSize++] = mRotationBufferW.take().floatValue();

        if (blockSize == ACCELEROMETER_BLOCK_CAPACITY) {
            blockSize = 0;

            Intent messageIntent = new Intent();
            messageIntent.setAction(Intent.ACTION_SENDTO);
            DataMap dataMap = new DataMap();
            dataMap.putFloatArray("x", accBlockX);
            dataMap.putFloatArray("y", accBlockY);
            dataMap.putFloatArray("z", accBlockZ);

            dataMap.putFloatArray("gx", gyroBlockX);
            dataMap.putFloatArray("gy", gyroBlockY);
            dataMap.putFloatArray("gz", gyroBlockZ);

            dataMap.putFloatArray("rx", rotationVectorX);
            dataMap.putFloatArray("ry", rotationVectorY);
            dataMap.putFloatArray("rz", rotationVectorZ);
            dataMap.putFloatArray("rw", rotationVectorW);

            dataMap.putLong("dt", System.currentTimeMillis() - time);

            dataMap.putString("type", "data");

            messageIntent.putExtra("datamap", dataMap.toBundle());
            LocalBroadcastManager.sendBroadcast(messageIntent);

            time = System.currentTimeMillis();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

@Override
protected void onCancelled() {
    Log.v("MQP", "Cancelling SensorService");
    super.onCancelled();
}
}
}

```



```

public void onSensorChanged(SensorEvent event) {

    if (event.sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {

        try {
            mAccBufferX.add(new Float(event.values[0]));
            mAccBufferY.add(new Float(event.values[1]));
            mAccBufferZ.add(new Float(event.values[2]));
        } catch (IllegalStateException e) {

        }

    }
    else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {

        try {
            mGyroBufferX.add(new Float(event.values[0]));
            mGyroBufferY.add(new Float(event.values[1]));
            mGyroBufferZ.add(new Float(event.values[2]));
        } catch (IllegalStateException e) {

        }

    }
    else if (event.sensor.getType() == Sensor.TYPE_ROTATION_VECTOR) {

        try {
            mRotationBufferX.add(new Float(event.values[0]));
            mRotationBufferY.add(new Float(event.values[1]));
            mRotationBufferZ.add(new Float(event.values[2]));
            mRotationBufferW.add(new Float(event.values[3]));
        } catch (IllegalStateException e) {

        }

    }

}

public void onAccuracyChanged(Sensor sensor, int accuracy) {

}

@Override
public IBinder onBind(Intent intent) {
    return null;
}

}

```

WatchDataListener.java

```

package com.androidweardocs.wearable.datamap;

import android.content.Intent;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

import com.google.android.gms.wearable.DataEvent;
import com.google.android.gms.wearable.DataEventBuffer;
import com.google.android.gms.wearable.DataMap;
import com.google.android.gms.wearable.DataMapItem;
import com.google.android.gms.wearable.WearableListenerService;

/**
 * Created by michaelHahn on 1/16/15.
 * Listener service or data events on the data layer
 */
public class WatchListenerService extends
    com.google.android.gms.wearable.WearableListenerService {

```



```

public class WatchDataMapActivity extends WearableActivity implements
    GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener {

    GoogleApiClient googleClient;

    private TextView mTextView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_data_map);
        final WatchViewStub stub = (WatchViewStub) findViewById(R.id.watch_view_stub);
        stub.setOnLayoutInflatedListener(new WatchViewStub.OnLayoutInflatedListener() {
            @Override
            public void onLayoutInflated(WatchViewStub stub) {
                mTextView = (TextView) stub.findViewById(R.id.text);
            }
        });

        setAmbientEnabled();

        // Register the local broadcast receiver
        IntentFilter messageFilter = new IntentFilter(Intent.ACTION_SENDTO);
        MessageReceiver messageReceiver = new MessageReceiver();
        LocalBroadcastManager.getInstance(this).registerReceiver(messageReceiver,
            messageFilter);

        googleClient = new GoogleApiClient.Builder(this)
            .addApi(Wearable.API)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .build();

        startService(new Intent(this, WatchListenerService.class));
        startService(new Intent(this, SensorService.class));
    }

    // Connect to the data layer when the Activity starts
    @Override
    protected void onStart() {
        super.onStart();
        googleClient.connect();
    }

    @Override
    public void onConnected(Bundle connectionHint) {

    }

    // Disconnect from the data layer when the Activity stops
    @Override
    protected void onStop() {
        if (null != googleClient && googleClient.isConnected()) {
            googleClient.disconnect();
        }
        super.onStop();
    }

    // Placeholders for required connection callbacks
    @Override
    public void onConnectionSuspended(int cause) { }

```

```

@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
    Log.v("MQP", connectionResult.getErrorMessage());
}

public void sendDataToPhone(Bundle data) {
    String MOBILE_DATA_PATH = "/mobile_data";

    new SendToDataLayerThread(MOBILE_DATA_PATH, data).start();
}

public class MessageReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle data = intent.getBundleExtra("datamap");

        if (data.getString("type").equals("status")) {
            final TextView tv = (TextView) findViewById(R.id.statustext);
            tv.setText(data.getString("content"));
        } else if (data.getString("type").equals("data")) {
            sendDataToPhone(data);
        }
    }
}

class SendToDataLayerThread extends Thread {
    String path;
    Bundle data;

    // Constructor for sending data objects to the data layer
    SendToDataLayerThread(String p, Bundle data) {
        path = p;
        this.data = data;
    }

    public void run() {

        DataMap dataMap = new DataMap();
        dataMap.putFloatArray("x", data.getFloatArray("x"));
        dataMap.putFloatArray("y", data.getFloatArray("y"));
        dataMap.putFloatArray("z", data.getFloatArray("z"));

        dataMap.putFloatArray("gx", data.getFloatArray("gx"));
        dataMap.putFloatArray("gy", data.getFloatArray("gy"));
        dataMap.putFloatArray("gz", data.getFloatArray("gz"));

        dataMap.putFloatArray("rx", data.getFloatArray("rx"));
        dataMap.putFloatArray("ry", data.getFloatArray("ry"));
        dataMap.putFloatArray("rz", data.getFloatArray("rz"));
        dataMap.putFloatArray("rw", data.getFloatArray("rw"));

        dataMap.putLong("dt", data.getLong("dt"));

        dataMap.putString("type", "data");
        dataMap.putLong("timestamp", System.currentTimeMillis());

        // Construct a DataRequest and send over the data layer

```

```

        PutDataMapRequest putDMR = PutDataMapRequest.create(path);
        putDMR.getDataMap().putAll(dataMap);
        PutDataRequest request = putDMR.asPutDataRequest();
        DataApi.DataItemResult result = Wearable.DataApi.putDataItem(googleClient,
request).await();
        if (result.getStatus().isSuccess()) {
            Log.v("MQP", "DataMap: " + dataMap + " sent successfully to data layer ");
        } else {
            // Log an error
        }
    }
}
}
}

```

MobileListenerService.java

```

package com.androidweardocs.wearable.datamap;

import android.Manifest;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.hardware.SensorManager;
import android.net.Uri;
import android.os.Environment;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

import com.google.android.gms.wearable.DataEvent;
import com.google.android.gms.wearable.DataEventBuffer;
import com.google.android.gms.wearable.DataMap;
import com.google.android.gms.wearable.DataMapItem;
import com.google.android.gms.wearable.WearableListenerService;

import org.w3c.dom.Attr;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.channels.FileChannel;
import java.util.ArrayList;
import java.util.Arrays;

import weka.core.Attribute;
import weka.core.DenseInstance;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ArffSaver;
import weka.core.converters.ConverterUtils.DataSource;

/**
 * Created by michaelHahn on 1/16/15.
 * Listener service or data events on the data layer
 */
public class MobileListenerService extends WearableListenerService{

    private static final String MOBILE_DATA_PATH = "/mobile_data";

    Activity thisActivity;

```

```

File outputFile;
ArrayList<Attribute> attributes;
ArrayList<String> labelItems;

Attribute mClassAttribute;
Instances mDataset;

Quaternion quaternion;
Vector3 position;

String mClass;

DataMap[] gestureSeries; // 6 long for about 2s of data
int gestureNum = 0;

@Override
public int onStartCommand(Intent i, int flags, int startId) {
    int result = super.onStartCommand(i, flags, startId);

    if (i.getExtras() != null) {

        Log.v("MQP", "PHONE SERVICE STARTED");

        attributes = new ArrayList<>();
        attributes.add(new Attribute("avg_speed"));
        attributes.add(new Attribute("median_speed"));
        attributes.add(new Attribute("max_speed"));
        attributes.add(new Attribute("variance_speed"));
        attributes.add(new Attribute("net_roll_velocity"));
        attributes.add(new Attribute("median_roll_velocity"));
        attributes.add(new Attribute("max_roll_velocity"));

        labelItems = new ArrayList<>();
        labelItems.add("drinking");
        labelItems.add("crossing_arms");
        labelItems.add("sitting_smoking");
        labelItems.add("on_phone");

        mClassAttribute = new Attribute("label", labelItems);
        attributes.add(mClassAttribute);

        mDataset = new Instances("mqp_features", attributes, 1000);
        mDataset.setClassIndex(mDataset.numAttributes() - 1);

        //attributes.add(new Attribute("vertical_displacement"));

        quaternion = new Quaternion(new Vector3(0.f, 0.f, 0.f), 0.f);
        position = new Vector3(0.f, 0.f, 0.f);

        mClass = i.getExtras().getString("class");

        gestureSeries = new DataMap[6];

        outputFile = new File(getApplicationContext().getFilesDir(), "mqpdata.arff");
        if (outputFile.exists()) {
            try {
                DataSource dataSource = new DataSource(new FileInputStream(outputFile));
                Instances oldData = dataSource.getDataSet();
                mDataset.setClassIndex(mDataset.numAttributes() - 1);

                int a = 0;
            }
        }
    }
}

```

```

        for (Instance inst : oldData) {
            Log.v("MQP", "Loaded inst " + a + ": " + inst.toString());
            a++;

            mDataset.add(inst);
        }

        outputFile.delete();

        Log.v("MQP", "File loaded");
    } catch (Exception e) {
        Log.v("MQP", "Exception", e);
    }

    } else {
        Log.v("MQP", "File not Found, making new");

        try {
            outputFile.createNewFile();
        } catch (Exception e) {

        }
    }
}

return result;
}

@Override
public void onDataChanged(DataEventBuffer dataEvents) {

    DataMap dataMap;
    for (DataEvent event : dataEvents) {
        //Log.v("MQP", "DataMap received on phone: " +
        DataMapItem.fromDataItem(event.getDataItem()).getDataMap());
        // Check the data type
        if (event.getType() == DataEvent.TYPE_CHANGED) {
            // Check the data path
            String path = event.getDataItem().getUri().getPath();
            if (path.equals(MOBILE_DATA_PATH)) {
                dataMap = DataMapItem.fromDataItem(event.getDataItem()).getDataMap();

                // Broadcast DataMap contents to wearable activity for display
                // The content has the golf hole number and distances to the front,
                // middle and back pin placements.

                Intent messageIntent = new Intent();
                messageIntent.setAction(Intent.ACTION_SEND);
                messageIntent.putExtra("datamap", dataMap.toBundle());
                LocalBroadcastManager.getInstance(this).sendBroadcast(messageIntent);

                logData(dataMap);
            }
        }
    }
}

public void logData(DataMap dataMap) {

    if (gestureNum < 0) {

```

```

        // skip this set, as the last one was just gesture end. Unlikely they take puffs
        in such quick succession.
        gestureNum++;
        Log.v("MQP", "Skipping set");
    }
    else if (gestureNum == 0) {
        // check whether this segment starts a gesture

        float[] x = dataMap.getFloatArray("x");
        float[] y = dataMap.getFloatArray("y");
        float[] z = dataMap.getFloatArray("z");

        String sp = "";

        for (int i = 0; i < x.length; i++) {

            float speed = (float)
Math.sqrt(Math.pow(x[i],2)+Math.pow(y[i],2)+Math.pow(z[i],2));
            sp = sp+""+speed+",";

            if (speed > 1.f) {
                gestureSeries[0] = dataMap;
                gestureNum = 1;

                Log.v("MQP", "Gesture begin with speed" + speed);

            }

        }
        Log.v("MQP", sp);
    }
    else if (gestureNum >= 1 && gestureNum < 6) {
        // add the intermediate datamap to the list

        gestureSeries[gestureNum] = dataMap;
        gestureNum++;
    }
    else {
        // the gesture is complete. Calculate features.

        float[] max_speeds = new float[6];
        float[] median_speeds = new float[6];
        float[] mean_speeds = new float[6];
        float[] variance_speeds = new float[6];
        float[] max_roll_velocitys = new float[6];
        float[] median_roll_velocitys = new float[6];
        float[] net_roll_velocitys = new float[6];

        Log.v("MQP", "Whole gesture: ");
        String gesture = "";

        for (int i=0; i<6; i++) {
            DataMap data = gestureSeries[i];

            float[] x = data.getFloatArray("x");
            float[] y = data.getFloatArray("y");
            float[] z = data.getFloatArray("z");

            float[] speeds = new float[x.length];
            float totalSpeed = 0;

            for (int v = 0; v < x.length; v++) {

```



```

        speeds[v] = (float) Math.sqrt(Math.pow(x[v],2) + Math.pow(y[v],2) +
Math.pow(z[v],2));
        totalSpeed+=speeds[v];
        gesture+=speeds[v]+" ";
    }

    Arrays.sort(speeds);

    max_speeds[i] = speeds[speeds.length-1];
    median_speeds[i] = speeds[speeds.length/2];
    mean_speeds[i] = totalSpeed / speeds.length;
    variance_speeds[i] = 0;

    for (float a : speeds) {
        variance_speeds[i] += (mean_speeds[i]-a) * (mean_speeds[i]-a);
    }
    variance_speeds[i] = variance_speeds[i] / speeds.length;

    float[] rx = data.getFloatArray("rx");
    float[] ry = data.getFloatArray("ry");
    float[] rz = data.getFloatArray("rz");
    float[] rw = data.getFloatArray("rw");

    float[] gx = data.getFloatArray("gx");
    float[] gy = data.getFloatArray("gy");
    float[] gz = data.getFloatArray("gz");

    float[] roll_velocities = new float[gx.length];
    float net_roll_velocity = 0;

    for (int v = 0; v < gx.length; v++) {
        float a = gx[v];
        float b = gy[v];
        float c = gz[v];

        roll_velocities[v] = (float) Math.sqrt(Math.pow(a,2) + Math.pow(b,2) +
Math.pow(c,2));
        net_roll_velocity+= roll_velocities[v];
    }

    Arrays.sort(roll_velocities);

    max_roll_velocitys[i] = roll_velocities[roll_velocities.length - 1];
    median_roll_velocitys[i] = roll_velocities[roll_velocities.length/2];
    net_roll_velocitys[i] = net_roll_velocity;

    quaternion = new Quaternion(new Vector3(rx[rx.length-1], ry[ry.length-1],
rz[rz.length-1]), rw[rw.length-1], true);

}

// calculate net stats

float mean_speed = 0;
float median_speed = 0;
float max_speed = 0;
float variance_speed = 0;

```

```

float net_roll_velocity = 0;
float median_roll_velocity = 0;
float max_roll_velocity = 0;

for (int i = 0; i < 6; i++) {
    mean_speed+=mean_speeds[i];
    median_speed+=median_speeds[i];

    if (max_speed < max_speeds[i])
        max_speed = max_speeds[i];

    net_roll_velocity+=net_roll_velocitys[i];
    median_roll_velocity+=median_roll_velocitys[i];

    if (max_roll_velocity < max_roll_velocitys[i])
        max_roll_velocity = max_roll_velocitys[i];
}

mean_speed = mean_speed / 6;
median_speed = median_speed / 6;
median_roll_velocity = median_roll_velocity / 6;

for (int i = 0; i < 6; i++) {
    variance_speed += (variance_speeds[i] + Math.pow(mean_speeds[i] - mean_speed,
2));
}
variance_speed = variance_speed / 6;

Instance inst = new DenseInstance(attributes.size());
inst.setDataset(mDataset);

inst.setValue(mDataset.attribute("avg_speed"), mean_speed);
inst.setValue(mDataset.attribute("median_speed"), median_speed);
inst.setValue(mDataset.attribute("max_speed"), max_speed);
inst.setValue(mDataset.attribute("variance_speed"), variance_speed);

inst.setValue(mDataset.attribute("net_roll_velocity"), net_roll_velocity);
inst.setValue(mDataset.attribute("median_roll_velocity"), median_roll_velocity);
inst.setValue(mDataset.attribute("max_roll_velocity"), max_roll_velocity);
inst.setValue(mClassAttribute, mClass);

mDataset.add(inst);

gestureNum = -2; // end gesture.

Log.v("MQP", gesture);
Log.v("MQP", "Gesture end");

}

}

@Override
public void onDestroy() {

    if (mDataset != null) {
        ArffSaver saver = new ArffSaver();
        saver.setInstances(mDataset);
    }
}

```

```

        Log.v("MQP", "FILE " + outputFile.getAbsolutePath());
        try {
            saver.setFile(outputFile);
            saver.writeBatch();

            File newFile = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS),
"mqpdata" + System.currentTimeMillis()+".arff");
            copyFile(outputFile, newFile);

            Intent intent =
                new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
            intent.setData(Uri.fromFile(newFile));
            sendBroadcast(intent);
        } catch (IOException e) {
            Log.e("MQP", "error saving");
            e.printStackTrace();
        }

    } else {
        Log.v("MQP", "Dataset NULL");
    }
}

public static void copyFile(File src, File dst) throws IOException
{
    FileChannel inChannel = new FileInputStream(src).getChannel();
    FileChannel outChannel = new FileOutputStream(dst).getChannel();
    try
    {
        inChannel.transferTo(0, inChannel.size(), outChannel);
    }
    finally
    {
        if (inChannel != null)
            inChannel.close();
        if (outChannel != null)
            outChannel.close();
    }
}
}

```

MobileDataMapActivity.java

```

package com.androidweardocs.wearableadatamap;

import android.Manifest;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v7.app.AppCompatActivity;
import android.text.format.Time;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;

```

```

import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.TextView;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.wearable.DataApi;
import com.google.android.gms.wearable.DataMap;
import com.google.android.gms.wearable.Node;
import com.google.android.gms.wearable.NodeApi;
import com.google.android.gms.wearable.PutDataMapRequest;
import com.google.android.gms.wearable.PutDataRequest;
import com.google.android.gms.wearable.Wearable;

import org.w3c.dom.Text;

import java.io.File;
import java.text.DecimalFormat;
import java.util.Calendar;
import java.util.Date;

public class MobileDataMapActivity extends AppCompatActivity implements
    GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener{

    GoogleApiClient googleClient;

    int hasPermission = 0;

    boolean serviceStarted = false;

    String mClass = "standing_smoking";

    Intent serviceIntent;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_data_map);

        // Register the local broadcast receiver
        IntentFilter messageFilter = new IntentFilter(Intent.ACTION_SEND);
        MessageReceiver messageReceiver = new MessageReceiver();
        LocalBroadcastManager.getInstance(this).registerReceiver(messageReceiver,
messageFilter);

        // Build a new GoogleApiClient for the the Wearable API
        googleClient = new GoogleApiClient.Builder(this)
            .addApi(Wearable.API)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .build();

        int permissionCheck = ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE);

        if (permissionCheck != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,

```

```

        new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
Manifest.permission.READ_EXTERNAL_STORAGE},
        hasPermission);
    }

    serviceIntent = new Intent(this, MobileListenerService.class);
    startService(serviceIntent);

    final Button button = (Button) findViewById(R.id.sendButton);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (serviceStarted) {
                stopService(serviceIntent);
                button.setText("Start Service");
                serviceStarted = false;
            } else {
                serviceIntent.putExtra("class", mClass);
                startService(serviceIntent);
                button.setText("Stop Service");
                serviceStarted = true;
            }
        }
    });

    final RadioButton r1 = (RadioButton) findViewById(R.id.radioButton);
    r1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mClass = "drinking";
            Log.v("MQP", "Set class to "+mClass);
        }
    });

    final RadioButton r2 = (RadioButton) findViewById(R.id.radioButton2);
    r2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mClass = "sitting_smoking";
            Log.v("MQP", "Set class to "+mClass);
        }
    });

    final RadioButton r3 = (RadioButton) findViewById(R.id.radioButton3);
    r3.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mClass = "crossing_arms";
            Log.v("MQP", "Set class to "+mClass);
        }
    });

    final RadioButton r4 = (RadioButton) findViewById(R.id.radioButton4);
    r4.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mClass = "on_phone";
            Log.v("MQP", "Set class to "+mClass);
        }
    });

    final Button deleteFile = (Button) findViewById(R.id.deleteFile);
    deleteFile.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

```

```

        new File(getApplicationContext().getFilesDir(), "mqpdata.arff").delete();
    });
}

// Connect to the data layer when the Activity starts
@Override
protected void onStart() {
    super.onStart();
    googleClient.connect();
}

@Override
public void onConnected(Bundle connectionHint) {

}

// Disconnect from the data layer when the Activity stops
@Override
protected void onStop() {
    if (null != googleClient && googleClient.isConnected()) {
        googleClient.disconnect();
    }
    super.onStop();
}

// Placeholders for required connection callbacks
@Override
public void onConnectionSuspended(int cause) { }

@Override
public void onConnectionFailed(ConnectionResult connectionResult) { }

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_data_map, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}

public void sendToWearable(String content) {
    String WEARABLE_DATA_PATH = "/wearable_data";

    DataMap dataMap = new DataMap();
    dataMap.putString("content", content);
    dataMap.putString("type", "status");
}

```

```

        dataMap.putLong("timestamp", System.currentTimeMillis());
        new SendToDataLayerThread(WEARABLE_DATA_PATH, dataMap).start();
    }

    class SendToDataLayerThread extends Thread {
        String path;
        DataMap dataMap;

        // Constructor for sending data objects to the data layer
        SendToDataLayerThread(String p, DataMap data) {
            path = p;
            dataMap = data;
        }

        public void run() {
            // Construct a DataRequest and send over the data layer
            PutDataMapRequest putDMR = PutDataMapRequest.create(path);
            putDMR.getDataMap().putAll(dataMap);
            PutDataRequest request = putDMR.asPutDataRequest();
            DataApi.DataItemResult result = Wearable.DataApi.putDataItem(googleClient,
request).await();
            if (result.getStatus().isSuccess()) {
                Log.v("MQP", "DataMap: " + dataMap + " sent successfully to data layer ");
            } else {
                // Log an error
                Log.v("MQP", "ERROR: failed to send DataMap to data layer");
            }
        }
    }

    public class MessageReceiver extends BroadcastReceiver {

        public final DecimalFormat df = new DecimalFormat("#.##");

        @Override
        public void onReceive(Context context, Intent intent) {
            Bundle data = intent.getBundleExtra("datamap");
            if (data.getString("type").equals("status")) {
                //
            } else if (data.getString("type").equals("data")) {
                String output = "X: ";
                output+=df.format(data.getFloatArray("x")[0])+" ";

                output+="\nY: ";
                output+=df.format(data.getFloatArray("y")[0])+" ";

                output+="\nZ: ";
                output+=df.format(data.getFloatArray("z")[0])+" ";

                output+="\nGX: ";
                output+=df.format(data.getFloatArray("gx")[0])+" ";

                output+="\nGY: ";
                output+=df.format(data.getFloatArray("gy")[0])+" ";

                output+="\nGZ: ";
                output+=df.format(data.getFloatArray("gz")[0])+" ";

                TextView tv = (TextView) findViewById(R.id.output);
                tv.setText(output);
            }
        }
    }
}

```

Sample accelerometer data

Smoking	Drinking	Rest	Rest	Entire Smoking
0.850686	1.204557	0.137204	0.296186	0.052169
0.59856	1.120746	0.135231	0.138498	0.117795
0.76611	1.401392	0.154828	0.146715	0.138873
0.415961	1.290384	0.146218	0.193265	0.132213
0.340746	1.182716	0.178941	0.170757	0.080285
0.405751	0.168174	0.118137	0.225339	0.107539
0.510539	0.122964	0.178357	0.251267	0.194218
0.701473	0.147014	0.163858	0.216584	0.135478
0.720719	0.179834	0.142384	0.242308	0.11498
0.845056	0.158422	0.122513	0.271133	0.079001
0.5731	0.052473	0.169992	0.212189	0.068378
0.659201	0.113449	0.166637	0.193013	0.107093
0.733901	0.122326	0.224843	0.244191	0.10621
0.828133	0.170243	0.189137	0.376537	0.090681
0.846127	0.199341	0.152472	0.347676	0.062366
0.931198	0.138025	0.126026	0.323658	0.069139
0.944889	0.153667	0.130011	0.271347	0.070788
0.438324	0.113408	0.181109	0.278859	0.087164
0.398837	0.099647	0.10733	0.14234	0.058746
0.413983	0.181927	0.190676	0.145275	0.137052
0.398037	0.192413	0.146507	0.202169	0.121926
0.497355	0.211662	0.205162	0.242393	0.157047
0.604705	0.19864	0.155745	0.187879	0.118063

0.355727	0.17793	0.141314	0.207417	0.147428
0.399882	0.129464	0.177829	0.194611	0.125204
0.558224	0.290245	0.283001	0.195276	0.134515
0.647876	0.334333	0.265207	0.219199	0.039222
0.890154	0.387572	0.248879	0.216124	0.079672
1.029796	0.335836	0.260906	0.257214	0.158612
1.12458	0.308039	0.244462	0.230992	0.177753
0.783566	0.236538	0.216776	0.152013	0.179725
0.806552	0.176319	0.201712	0.128054	0.137662
0.820743	0.275754	0.18344	0.125585	0.159363
0.819467	0.30076	0.167618	0.174101	0.108874
0.723091	0.278254	0.161034	0.205396	0.08038
0.677816	0.18468	0.214788	0.182998	0.08803
0.837807	0.152763	0.237862	0.170182	0.11727
0.509015	0.112846	0.334826	0.254183	0.167733
0.613812	0.069762	0.437936	0.226597	0.233544
0.705835	0.127763	0.41512	0.247635	0.244908
0.682565	0.223825	0.21366	0.193698	0.384078
0.709641	0.232096	0.214266	0.152324	0.547234
0.898711	0.155785	0.243893	0.141542	0.728073
1.107588	0.36827	0.218608	0.114617	1.159341
0.750189	0.47468	0.232228	0.165419	1.459003
0.693011	0.533159	0.271638	0.183225	1.631245
0.675405	0.638333	0.253448	0.145353	1.634065
0.7629	0.639178	0.244933	0.109674	1.399104
0.768607	0.569691	0.208191	0.232823	1.498337

0.760935	0.179969	0.201176	0.195923	1.745919
0.807377	0.229886	0.218951	0.212012	1.894114
0.651482	0.337837	0.211276	0.322615	1.857711
0.783605	0.554931	0.159177	0.242509	1.981108
0.642436	0.764687	0.221042	0.237626	2.367457
0.648426	0.93664	0.179399	0.227847	2.350871
0.792998	0.838522	0.137853	0.219033	2.310313
0.992648	2.555586	0.103891	0.1596	2.440385
1.007881	2.243414	0.141847	0.150429	2.654907
0.586507	1.304025	0.157687	0.21777	2.689051
0.738383	1.316755	0.175343	0.180111	2.602295
0.807038	0.541038	0.211477	0.164541	2.513883
0.841806	1.066217	0.227708	0.171367	2.512585
0.791984	0.936439	0.175514	0.204756	2.524624
	1.149514	0.145782	0.277614	0.667777
				0.470628
				0.600609
				0.786874
				0.806851
				1.907791
				1.679003
				1.789835
				1.703563
				1.257534
				0.914619
				1.013913

				1.109702
				1.194641
				1.049836
				0.9862
				1.107229
				1.295299
				1.168453
				0.805549
				0.13973
				0.651257
				0.627156
				0.498102
				0.467098
				0.459802
				0.377147
				0.507851
				1.136917
				1.107286
				1.058548
				0.987719
				0.818717
				0.727094
				0.766631
				1.013398
				1.208476
				1.305489

				1.517075
				1.559677
				1.490923
				1.397171
				1.046987
				0.981175
				1.235729
				1.199204
				1.130717
				1.233099
				1.486952
				1.507699
				1.189987
				1.088734
				1.173044
				1.323827
				1.379839
				1.389164
				1.2191
				1.396598
				1.524301
				1.610931
				1.81051
				1.860191
				1.16808
				0.88966

				0.450546
				0.444169
				0.611184
				0.786874
				0.806851
				0.498721
				0.567809
				0.617252
				0.778946
				0.911093
				1.183365
				1.171531
				0.911162
				0.967663
				1.01872
				1.091341
				1.226483
				1.188079
				1.101537
				0.885875
				0.88545
				0.831691
				0.599021
				0.630312
				0.768387
				0.770779

				0.472586
				0.476035
				0.378356
				0.578474
				0.650258
				0.689854
				0.839932
				0.799305
				1.181574
				1.336907
				1.471311
				1.515655
				1.557511
				1.535033
				0.945496
				0.847897
				0.977171
				1.111594
				1.101158
				1.038936
				0.954082
				1.195223
				1.252208
				1.116766
				1.03961
				1.120182

				1.231605
				0.890218
				0.64391
				0.616863
				0.732342
				0.747691
				0.693106
				0.656512
				0.471193
				0.448786
				0.55708
				0.608821
				0.308563
				0.444169
				0.611184
				0.451065
				0.554159
				0.583732
				1.031887
				1.048903
				0.80085
				0.495109
				0.335597
				0.380345
				0.602768
				0.566017

				0.530775
				0.626063
				0.653232
				0.6781
				0.572918
				0.622185
				0.806991
				0.880185
				0.840667
				0.565885
				0.497656
				0.562062
				0.614042
				0.519245
				0.29556
				0.270739
				0.230748
				0.229543
				0.223101
				0.327653
				0.463928
				0.45993
				0.460866
				0.46458
				0.348299
				0.356562

				0.465201
				0.653234
				0.561361
				0.474211
				0.572785
				0.57393
				0.460869
				0.236904
				0.216688
				0.232679
				0.190853
				0.206891
				0.261408
				0.424661
				0.44358
				0.517216
				0.531656
				0.45329
				0.397237
				0.497671
				0.507667
				0.503843
				0.36602
				0.285441
				1.097356
				1.118102

				0.996398
				0.792192
				0.463389
				0.197608
				0.238531
				0.328768
				0.312309
				0.280778
				0.2573
				0.211152
				0.170983
				0.087188
				0.120427
				0.153587
				0.205997
				0.07996
				0.132922
				0.282732
				0.289611
				0.23675
				0.239862
				0.270033
				0.314392
				0.393603
				0.508696
				0.614538

				0.705435
				0.743425
				0.74591
				0.602062
				0.544774
				0.478
				0.504225
				0.500616
				0.52544
				0.635669
				0.856042
				0.874867
				0.82019
				0.739322
				0.737557
				0.7739
				0.932016
				1.083005
				1.111819
				1.095461
				1.037599
				1.187657
				1.138688
				1.157027
				1.133799
				1.015708

				1.049561
				0.99293
				1.294254
				1.376046
				1.513708
				1.578494
				1.609956
				1.582513
				1.380506
				1.192307
				0.800922
				0.930678
				0.926437
				0.792192
				0.463389
				0.47169
				0.449369
				0.528102
				0.553956
				0.657962
				0.886819
				0.85536
				0.900189
				0.800501
				0.796401
				0.936174

				1.070141
				1.181374
				1.107995
				0.580098
				0.395822
				0.372823
				0.721513
				0.858482
				0.95738
				1.11743
				1.027032
				0.8955
				1.073254
				1.097392
				1.317669
				1.777717
				2.084692
				1.550948
				1.24501
				1.150037
				1.072429
				1.131762
				1.186797
				1.0498