George Thomas

Activity classification using wearable devices

Computer Science Tripos – Part II

Peterhouse

5th May 2015

Proforma

Name:	**George Thomas**
College:	**Peterhouse**
Project Title:	**Activity classification using wearable devices**
Examination:	**Computer Science Tripos – Part II, July 2015**
Word Count:	**11342**
Project Originator:	Dr. Neal Lathia
Supervisor:	Dr. Neal Lathia

Original Aims of the Project

The aims of the project are:

1. to classify activities based on accelerometer recordings from a consumer smartwatch and smartphone; and
2. evaluate to what extent the smartwatch is better at helping to classify activities.

Work Completed

I have built an Android smartphone app and an Android Wear smartwatch app that allows a user to collect accelerometer data from both devices simultaneously. I collect over five hours of accelerometer data over a range of activities. I extracted features from time divisions of the data and used a set of classifiers

to classify the activities. I evaluate the performance of each classifier on the set of activities when given access to only the watch data, only the phone data and both the watch and phone data together.

Special Difficulties

None

Declaration

I, George Thomas of Peterhouse, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date

Contents

List of Figures

Acknowledgements

Many thanks to Dr. Neal Lathia for his help and advice, and also to Prof. Cecilia Mascolo for supporting the project and providing required hardware.

Chapter 1

Introduction

This dissertation describes the implementation and evaluation of an activity classifier using accelerometer data captured simultaneously from a smartphone and a smartwatch.

The aims of the project are:

1. to classify activities based on accelerometer recordings from a consumer smartwatch and smartphone; and
2. evaluate to what extent the smartwatch is better at helping to classify activities.

This project is motivated by the unparalleled sensing ability of recently released commodity smartwatches like the Apple Watch and similar devices. No other device is carried at all times or is worn in exactly the same place against the skin.

The classifier using data from both sources outperforms a classifier using only smartphone data, and the classifier that uses only smartphone data outperforms a classifier using only smartwatch data.

1.1 Motivation

Wearable devices are set to become the next big technology trend. Wrist-worn wearables, including smartwatches, formed the majority of the 21m wearable

devices sold year. Analysts predict the Apple Watch will sell between 20m and 40m in its first nine months [18]. Estimates suggest close to 1m were sold on its first day of release [8].

One of the primary appeals of wearables is their ability to sense. Like smartphones before them, smartwatches will enhance the ability to collect data about people. This data is important to consumers, who purchase specialised wearables to measure activity, sleep patterns and calorific intake. The data's research potential is also laudable — Apple's ResearchKit will allow medical researchers to access data about their patients with greater ease than ever before [14].

Accelerometer data on its own does not reveal much about the user. In order to report on fitness or monitor health, determining a user's activities is typically required. Accurate activity classification therefore has many academic and commercial applications. To be marketable, activity classification solutions must use current consumer devices.

Though activity classification is available on Android smartphones, an approach that utilises simultaneous collection from a smartphone and smartwatch warrants further research. A common opinion is that smartwatches will always be more accurate activity classifiers than smartphones because they are worn on the body. The investigation of this dissertation attempts to quantify this position.

For that reason, this dissertation details the implementation of accelerometer data collection using current consumer devices (an Android smartphone and Android Wear smartwatch), classifies a user's activities and compares this classification accuracy to using only smartphone data and using only smartwatch data.

1.2 Challenges

This project requires knowledge of a variety of disparate areas in computer science.

Writing software for mobile devices requires knowledge of their paradigms and nuances. Mobile devices are also subject to computational power and battery life constraints[13] and particular care must be taken to build a solution that

works in practice. A project that utilises built-in sensors also requires an understanding of the features and limitations of those sensors and good knowledge of the APIs that are provided to access them.

The sensors also output data at a high rate and care must be taken to correctly handle the performance and concurrency issues that may arise. Storage and transfer of large amounts of raw data, especially on a memory-limited device such as a smartwatch, also requires special consideration: many methods and tools were designed for computers with persistent communication connections and unlimited power consumption.

The data processing aspects of the project will require an understanding of digital signal processing, Fourier methods, data mining and machine learning, and statistics.

1.3 Related Work

Activity classification using accelerometer data from body-mounted devices is an active area of research. I highlight three papers and discuss their similarity to this problem. Summaries of their work are found in Table 1.1.

1.3.1 Sensors attached to body

Bao *et al.* [2] detect physical activities using five biaxial accelerometers worn on different parts of the body: hip, wrist, ankle, arm and thigh. They find that accuracy is not significantly reduced when using just thigh and wrist accelerometers. Furthermore, recognition rates for thigh and wrist data resulted in the highest recognition accuracy among all pairs of accelerometers, with over a 25% improvement over the best single accelerometer results. This supports the viability of this project, with the improvement of being able to use triaxial accelerometers found in consumer smartphones and smartwatches.

Long *et al.* [10] use a single triaxial accelerometer placed on the wrist and use it to achieve an 80% activity classification accuracy in five activities. However, only 50% of all cycling is correctly classified. Bao *et al.* achieve an accuracy of $> 92\%$ by using thigh and wrist data. This would suggest that wrist data

alone is not sufficient to accurately classify certain types of activity. Cycling requires periodic leg motion (pedalling) while the hands and wrists move comparably little. Many of the features of motion used in activity classification require frequency domain analysis, and so data that contains periodic motion will be easier to recognise.

Atallah *et al.* [1] focus on two important facets of accelerometer-based activity classification: sensor location and useful features. Much like Bao *et al.* they use seven sensors on the chest, arm, wrist, waist, knee, ankle and ear. Of their analysed features, the averaged entropy over three axes, the mean of the pairwise cross-covariance of axes and the energy of a 0.2 Hz window around the main frequency divided by total energy are all highlighted as being highly ranked for distinguishing activities. However, this study neglects to use a decision tree classifier in its classification, recommended by both Bao *et al.* and Long *et al.*

1.3.2 Smartphone-only activity classification

Which paper(s) would you recommend here Neal?

1.3.3 Smartwatch activity classification

Smartwatch activity classification is a relatively new field because consumer smartwatches are relatively new devices. Papers at WristSense 2015[1], a conference for those working on "wrist worn smart devices", focus primarily on gesture detection or recognition of particular activities for applications such as diet monitoring, detecting self-harming and detection of bad habits.

1.3.4 Comparison to this project

Unlike many previous investigations into this topic, this project differs by:

1. being implemented on consumer hardware;
2. using devices that are not fixed to the body (in the case of the phone);

[1] https://sites.google.com/site/wristsenseworkshop2015/program

	Bao *et al.* [2]	**Long *et al.* [10]**	**Atallah *et al.* [1]**
Activities	Walking, sitting & relaxing, standing, watching TV, running, stretching, scrubbing, folding laundry, brushing teeth, riding elevator, carrying items, computer work, eating or drinking, reading, bicycling, strength-training, vacuuming, lying down, climbing stairs, riding escalator	Walking, running, cycling, driving, sports	Lying down, preparing food, eating and drinking, socialising, reading, getting dressed, corridor walking, treadmill walking, vacuuming, wiping tables, corridor running, treadmill running, cycling, sitting down and getting up, lying down and getting up
Features	Mean, energy, correlation, entropy	Standard deviation, entropy, orientation variation	Mean, variance, root mean square, entropy, correlation, range, energy, primary frequency, skewness, kurtosis
Classifiers	Decision table, nearest neighbour, decision tree, naive Bayes	Decision tree, principle component analysis, naive Bayes	K-nearest neighbours, naive Bayes
Overall accuracy	84%	80%	N/A

Table 1.1: Prior work on accelerometer-based activity classification

3. attempting to classify a broad range of activities using the smartwatch.

The project also differs in the activities it attempts to classify.

Chapter 2

Preparation

This chapter details the work done before the main implementation of the project was started. It details the devices chosen to implement this project and the reasons for choosing them. It then discusses the existing libraries and APIs available for those devices and for the required data processing. The design decisions are supported by some preliminary results on sensor noise. Finally, it describes software engineering techniques used.

2.1 Requirements analysis

The aim of the project is to classify activities based on accelerometer recordings from a consumer smartwatch and smartphone, and evaluate to what extent the smartwatch is better at helping to classify activities. The requirements to accomplish this can be split into two categories: data collection and data processing requirements.

Data collection requirements

1. access tri-axial readings from accelerometer on both the smartwatch and the smartphone;
2. store this accelerometer data temporarily on the internal memory of each device using suitable data structures;

3. transmit this data from the smartwatch to the smartphone using a suitable protocol, because of memory constraints and to enable transfer to a;
4. store the data permanently on the smartphone.

Data processing requirements

1. parse the data into a manipulatable format;
2. preprocess the data, including filtering and splitting into fixed-length bins;
3. extract features from each bin;
4. train classifier(s) on the extracted features;
5. test classifier and record evaluation statistics.

The remainder of this chapter describes work done to ensure these requirements could be fulfilled.

2.2 Hardware devices

The success of this project depends partly on correct selection and understanding of the devices used to collect data. Both the smartwatch and the smartphone are required to contain accelerometers accessible to developers.

Android devices were chosen as Android Wear was the most mature platform for developing with wearable devices at the time. It runs on the widest variety of devices and provides developer access to its sensors.

2.2.1 Smartphone

The smartphone chosen for development was the Google Nexus 5. Smartphone technology has advanced to the point that many Android smartphones are homogeneous with respect to this project — they all contain sufficient processing power, internal memory and an accelerometer capable of recording data.

The Nexus 5 contains a tri-axial accelerometer capable of recording measurements $\pm 2\mathrm{g}$ on each axis, where $\mathrm{g} \approx 9.81 \mathrm{m\,s^{-2}}$. This gives a total possible magnitude of $\sqrt{3 \times (2\mathrm{g})^2} = 2\mathrm{g}\sqrt{3} \approx 34 \mathrm{m\,s^{-2}}$. Many other smartphones are susceptible to this limit and it is not thought that this will be an issue for classification.

Figure 2.1 shows the front face of a Nexus 5 with the axes of the accelerometer labeled.

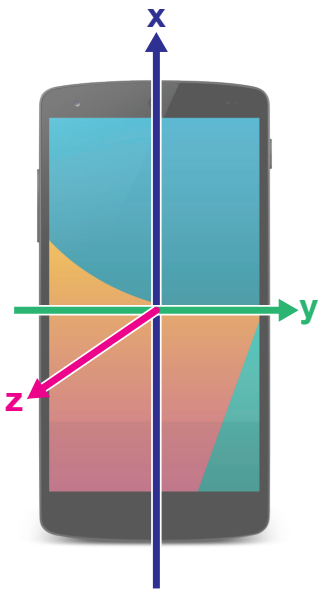


Figure 2.1: A Nexus 5 device, overlaid with the coordinate system used by the Android API. The positive x direction is defined as towards the top of the phone, the positive y direction is defined as towards the right of the phone and the positive z direction comes out of the screen. These directions are all relative to the natural portrait orientation of the device; they do not change when the device is used in horizontal orientation.

2.2.2 Smartwatch

The smartwatch chosen for development was the Samsung Galaxy Gear Live, running Android Wear. It pairs to any device running Android 4.4 or higher and communicates over Bluetooth.

Wearable devices that do not run Android typically run either Tizen, an open-source but not widely adopted operating system, such as the Samsung Galaxy

Gear 2, or a proprietary operating system that does not allow access to the raw accelerometer data, for example the Jawbone Up, and therefore cannot be used to compare classification accuracy with smartphones.

There is more differentiation in smartwatches than there is in smartphones, with them varying not just in screen size but also in screen format (round or rectangular), battery life, charging facilities and sensors. Table 2.1 presents an overview of possible smartwatch devices.

Though the Sony Smartwatch 3 has the best technical stats, it wasn't yet fully released at the time we acquired the smartwatch. The group has had previous success with Samsung devices and the Gear Live met all the requires I had of the smartwatch for the project.

Device	**Samsung Galaxy Gear Live**	**Samsung Galaxy Gear 2**	**LG G Watch**	**Sony Smartwatch 3**
Operating System	Android Wear	Tizen	Android Wear	Android Wear
Processor	1.2 GHz single-core Qualcomm Snapdragon 400	1.0 GHz dual-core Exynos 3250	1.2 GHz single-core Qualcomm Snapdragon 400	1.2 GHz quad-core ARM A7
Memory	512 MB RAM	512 MB RAM	512 MB RAM	512 MB RAM
Storage	4 GB	4 GB	4 GB	4 GB
Sensors	Touchscreen, Accelero-meter, Gyroscope, Compass, Heart Rate Monitor	Touchscreen, Accelero-meter, Gyroscope, Heart Rate Sensor, 2 MP Camera	Touchscreen, Accelero-meter, Gyroscope, Compass	Touchscreen, Accelero-meter, Gyroscope, Compass
Radios	Bluetooth 4.0 Low Energy	Bluetooth 4.0 Low Energy	Bluetooth 4.0 Low Energy	Bluetooth 4.0 Low Energy, GPS, NFC, Wi-Fi
Battery	300 mAh	300 mAh	400 mAh	420 mAh
Notes		Pairs only with Samsung devices		

Table 2.1: An overview of possible smartwatch devices. The Samsung Galaxy Gear Live was the device eventually chosen.

2.3 Working with accelerometer signals

The output from any accelerometer is a time-series representing its acceleration. Effectively extracting information from this time-series is central to the success of this project. Knowledge of signal processing is therefore critical.

It is essential to capture as much of the movement as possible. Conversion from continuous physical acceleration to a discrete time-series requires sampling. The Nyquist-Shannon sampling theorem states that a signal can be exactly reconstructed from its samples if the sample rate is greater than twice the highest frequency of the signal.

The highest frequency of a physical activity is not well defined. The activities I hope to classify will vary in their periodicity. Some, like walking, will be very periodic, while others, like climbing, will have no period at all. Considering common period activities like cycling and walking, I anticipate that the frequencies that best describe movement will be present in the 0–5Hz range, and so will require sampling at a frequency of at least 10Hz.

Frequency domain analysis

Much of the analysis of the accelerometer readings will be done in the frequency domain. A time domain signal can be converted into the frequency domain using a Fourier transform.

The discrete Fourier transform of a sequence of N complex numbers $f_0, f_1, \ldots, f_{N-1}$ is the sequence F_k, defined by:

$$F_k = \sum_{n=0}^{N-1} f_n \cdot \mathrm{e}^{-2\pi \mathrm{i} k n / N}$$

The power spectral density, PSD_k, of a signal describes how power is distributed over different frequencies. One method of estimating the power spectral density is to take the square of the absolute value of the Fourier transform component:

$$PSD_k = \|F_k\|^2$$

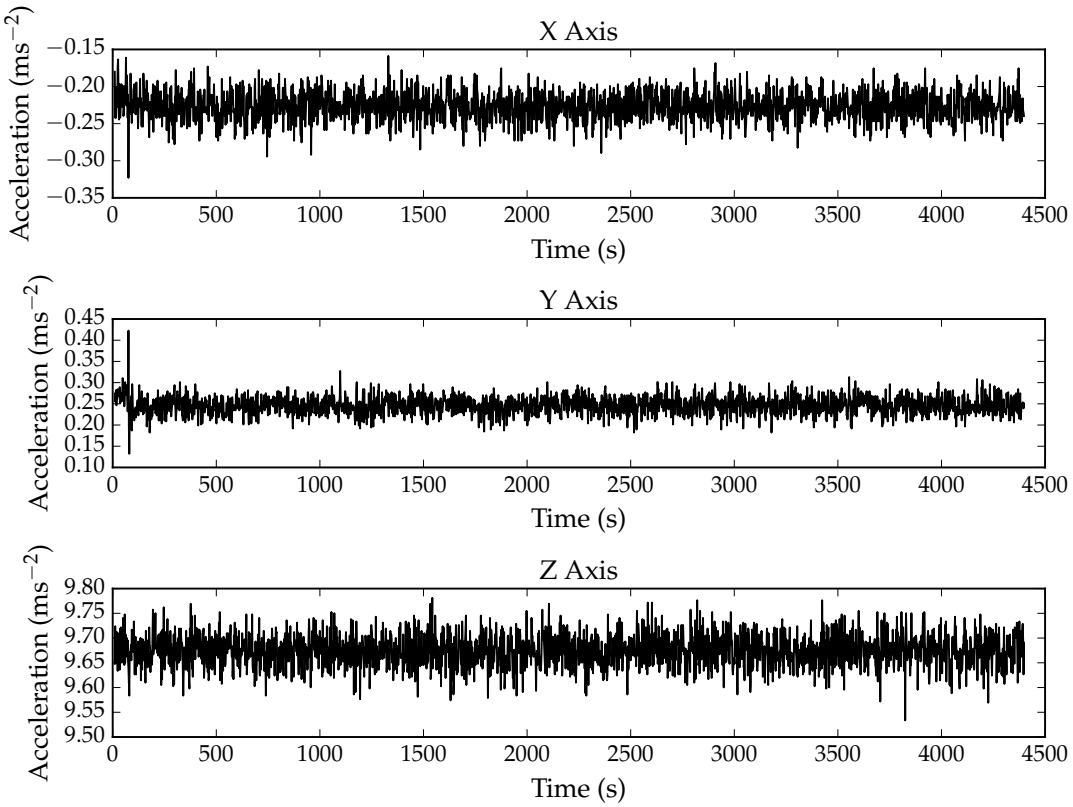


Figure 2.2: The X, Y and Z axis readings from an hour long accelerometer recording of the chosen smartphone laying flat on a table. The readings contain noise.

Noise and filtering

The readings from the accelerometer are subject to noise, exhibited in Figure 2.2, which plots readings from the X, Y, and Z axes during an hour long recording with the chosen smartphone laying flat on a table.

Figure 2.3 plots the distribution of the magnitude of the acceleration, where the magnitude $\|\mathrm{x}\| = \sqrt{x^2 + y^2 + z^2}$. The magnitude, which should be a constant $\mathrm{g} \approx 9.81 \mathrm{m\,s^{-2}}$, is subject to normally distributed noise.

Figure 2.4 gives a normal probability plot of the same magnitude data. Points on a normal probability plot should form a straight line if they are normally distributed. The straight line of best fit exhibits a coefficient of determination, $R^2 = 0.9993$, which is very close to 1. It is very likely that the noise is normally distributed.

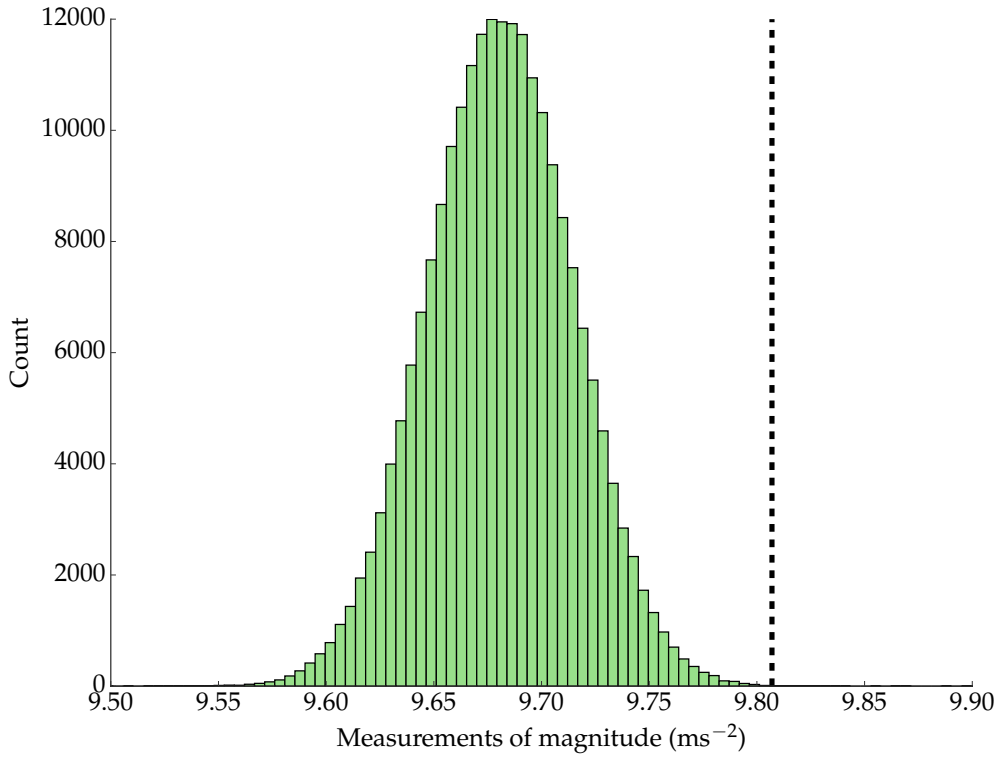


Figure 2.3: Histogram of the magnitude $\|\mathbf{x}\| = \sqrt{x^2 + y^2 + z^2}$ *from the data shown in Figure 2.2. The magnitude should measure* $\mathrm{g} \approx 9.81 \mathrm{m\,s^{-2}}$*, which is marked as the black dashed line on the graph. The noise implies the accelerometer data is imprecise. The mean of the data is less than* g*, which indicates the recording is also inaccurate.*

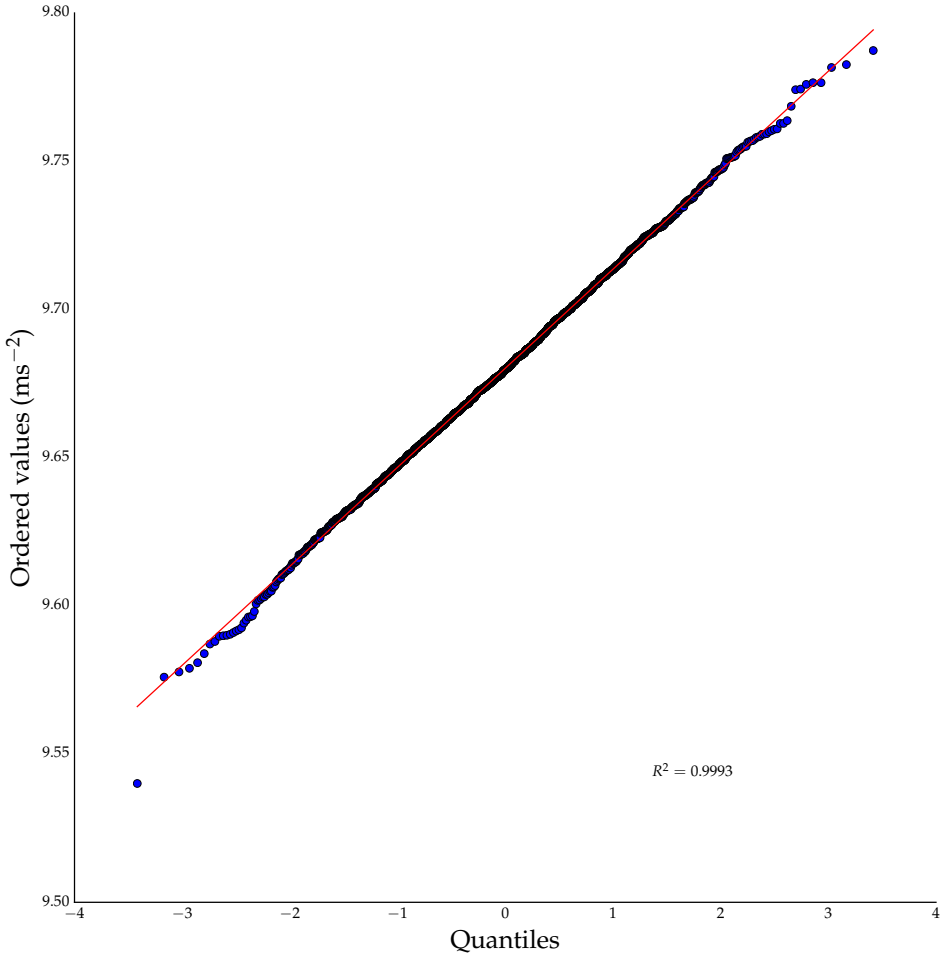


Figure 2.4: A normal probability plot of the magnitude $\|\mathbf{x}\| = \sqrt{x^2 + y^2 + z^2}$ *from the data shown in Figure 2.2. Data that is normally distributed will form a straight line when plotted in this way. This data is very likely to be normally distributed, as indicated by the straight line. ThThe straight line of best fit exhibits a coefficient of determination,* $R^2 = 0.9993$

Noise can be reduced with the application of a low-pass filter. A low-pass filter attenuates signals with a higher frequency than some cutoff, such as the noise exhibited in the signal. More information on low-pass filters in Section 3.3.1.

2.4 Libraries and APIs

This project makes use of existing libraries and APIs for the data collection, data handling and classification aspects of the project. I investigate each library and API early on to ensure I don't encounter any potential show-stopping issues further along.

2.4.1 Android Sensor API

The Android platform Sensor API[6] is implemented using a publisher-subscriber model. Listeners must be registered to a particular sensor and must implement an `onSensorChanged()` method. The `onSensorChanged()` method is called whenever the sensor reports a new value. A `SensorEvent` object is provided, containing a timestamp at which the data was reported together with the new data.

The rate at which `onSensorChanged()` is called is 'user-suggested'; though it can be specified by the user, it can also be altered by the Android system. In practice, this means that the difference in timestamps is not constant but is approximately equal to the specified delay. A histogram of timestamp differences for a particular 1 hour recording is given in figure 2.5.

Android provides both acceleration and linear acceleration sensors, related by

$$\text{acceleration} = \text{linear acceleration} + \text{gravity}$$

They each provide a timestamp represented as a 64-bit integer (i.e. a long) and three 32-bit float values representing the acceleration of each axis in $\mathrm{m\,s^{-2}}$ at that timestamp. Table 2.2 gives a graphical representation of the data returned.

Curiously, the timestamp returned as part of the data is documented only as "The time in nanosecond [sic] at which the event happened" [6]. Further explor-

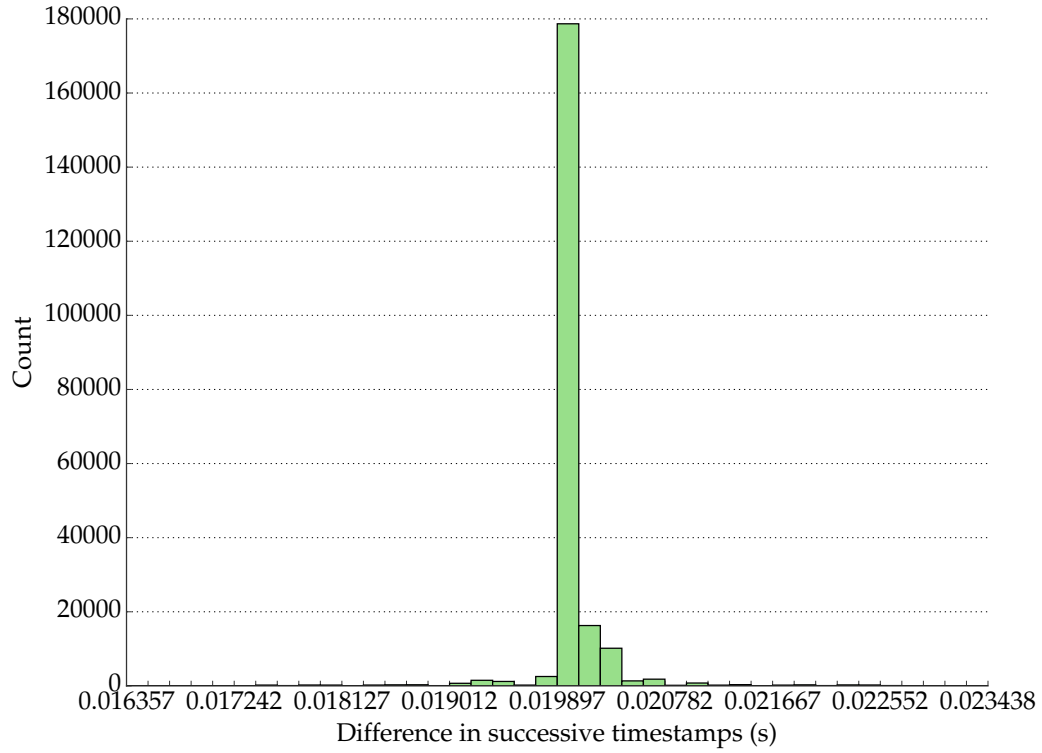


Figure 2.5: Histogram of the differences in successive timestamps from the data shown in Figure 2.2. The sample rate was set to 50 Hz*. 0.02002s accounted for 75% of the differences. Thus the actual sample rate is approximately the user-suggested sample rate.*

ation reveals that the timestamp is not defined against any particular zero-base, but rather the time since the device was powered on [5, 15]. The implication of this for the project is that while the timestamp can be relied on for intervals between measurements, it cannot be used between different sets of recordings or across devices.

2.4.2 Android Wear Data API

As discussed in Section 2.2.2, the only radio present in the Samsung Galaxy Gear Live is Bluetooth. To transfer any recorded data from the watch, it must first be transferred to the paired smartphone. The Android Wearable Data Layer

Timestamp	X acceleration	Y acceleration	Z acceleration
ns	$m\,s^{-2}$	$m\,s^{-2}$	$m\,s^{-2}$
Long	Float	Float	Float
2 bytes	1 byte	1 byte	1 byte

Table 2.2: Data from the accelerometer sensor provided to the `onSensorChanged()` method.

API allows communication between Android handheld and wearable devices. It provides three methods of communication between devices:

Data items provide data storage with automatic syncing;

Messages are good for remote procedure calls but do not carry data;

Asset objects for sending binary blobs of data.

The project makes use of Assets to send the accelerometer data and Messages to indicate that a particular device has started recording. Use of this API is described in Section 3.1.3.

The data layer synchronises data between the handheld and wearable. To do so, the Wearable Data Layer API requires the registration of a listener service, much like the Sensor API. The listener service listens for data layer events, such as the creation of asset objects or when messages are received.

2.5 Choice of tools

2.5.1 Programming languages

Java is the native programming language used on Android. Although it is possible to write code for Android in programming languages other than Java, for example by using the Android Native Development Kit, doing so would add much complexity for marginal performance gain that is not required. Using a language other than Java would lead to rewrites of many of the Android APIs The Android SDK builds on the principles of Java but is complicated by having to manage interactions with the Android operating system.

XML is Android's standard markup language. All user-interface components are written in XML. The project includes a user interface on both the phone and the watch to configure and control the recording of data.

Python 3.4 was chosen as the data processing language due to its ease of use and the strength of its data processing, signal processing and machine learning libraries:

NumPy: a scientific computing library and the basis for the other three libraries below[17].

Pandas: extensions to NumPy that enable easier processing of time-series data[11].

SciPy: signal processing tools and other statistical features[9].

Scikit Learn: machine learning classifiers and utilities to work with them[12].

All of NumPy, SciPy, Pandas and Scikit Learn are open-source and licensed under the BSD license.

2.5.2 Development Environment

Two IDEs, Android Studio and PyCharm were used for the development of the Android app and the Python data pipeline respectively. Android Studio is available for free from Google, while PyCharm is provided free for educational use by JetBrains. Both include advanced debuggers.

Though the Android SDK contains a device emulator, it runs slowly and cannot simulate sensors. Developing the Android apps is therefore done by connecting them to a computer and running new versions of the code. This also enables access to the device's logs from the development environment. I made extensive use of logging to determine that the program was executing as expected.

2.6 Software engineering techniques

2.6.1 Development methodologies

I used a combination of development methodologies for the project. The data collection apps were developed using a waterfall methodology, while the data processing was developed using an Agile methodology.

Waterfall models are excellent when the end goals of the project are known and can be well specified. The goal of the data collection apps can be easily stated: to write apps for the smartphone and smartwatch that will allow user collection of accelerometer data.

The data processing and machine learning elements of the project required an Agile methodology. The goal here is less well defined — to classify activities with the greatest accuracy — and the implementation to achieve the goal is far more experimental.

2.6.2 Version control and backups

I used three separate Git repositories for the data collection code[1], the data processing code[2] and the dissertation[3] respectively. The Git repositories were synced to GitHub at each commit. Version control allowed me to follow a *implement–test–commit* pattern when writing code.

GitHub also served as one method of backup. Each GitHub repository is publicly accessible such that I can continue implementation even if my primary development computer crashed and I was also locked out of my GitHub account. In addition, I backed up periodically to Dropbox and to an external hard drive. The external hard drive backup retained old copies of files when they were updated. This gives four replications of my entire project, with two of these able to access previous versions of the code.

[1] https://github.com/geotho/WearableActivityClassificationApp
[2] https://github.com/geotho/Wearable-Activity-Classifier-Machine-Learning
[3] https://github.com/geotho/Wearable-Activity-Classification-Dissertation

2.7 Activities for classification

Activities can vary in the amount of movement required, their typical body position and the parts of the body which are moving. When picking activities I wanted to attempt to classify, I wanted to select a mix of activities that were similar in some characteristics and that varied in others.

The activities selected fit into three broad categories:

- physical activities requiring whole body motion;
- activities that primarily require arm motion;
- low energy upright activities.

The activities that I hope to classify are as follows:

- Physical activities requiring whole body motion:
  - Climbing
  - Cycling
  - Gym cycling
  - Running
  - Stair climbing
  - Walking
- Activities that primarily require arm motion:
  - Computer use
  - Eating
  - Playing fussball
- Low energy upright activities:
  - Gallery perusal
  - Standing
  - Teeth brushing

2.8 Summary

In this section I presented:

- an overview of digital signal processing;
- information on the smartphone and smartwatch used;
- details of key APIs used including the Android Sensor API and the Android Wear Data API;
- development tools and software engineering techniques;
- activities I seek to classify.

Chapter 3

Implementation

This chapter details the implementation of the three main project areas:

1. data collection;
2. data processing;
3. classification

It also provides a graph of a sample recording of each of the activities classified. This process enabled me to better understand the readings recorded during the activities and implement the most useful features at the end.

3.1 Data collection

This section contains details of the components built to access the accelerometer data and transfer it to a computer.

Because both the smartwatch and the smartphone both run Android, it is possible to create components that are shared between the devices, resulting in less redundancy, less complexity and, ultimately, a more reliable implementation. Both the `AccelerometerListenerService` and the `AccelerometerDataBlob` are shared between both devices.

3.1.1 Accessing the accelerometer

The `AccelerometerListenerService` is responsible for receiving readings from the accelerometer and delivering them to the data structure responsible for storage.

As described in Section 2.4.1, the Sensor API utilises a listener methodology. It is required to create and register a listener that implements `onSensorChanged()`.

Performance considerations

Because the accelerometer can update its values at a rate of over 50Hz, it is vital that any implementation of `onSensorChanged()` is non-blocking and ideally be very quick to execute. Any expensive computation or IO operation has to be moved to a separate thread.

If the execution of `onSensorChanged()` takes longer than $\frac{1}{\text{sample}-\text{rate}}$, requests for `onSensorChanged()` will queue and eventually lead to the exhaustion of memory or dropping of data.

For this reason, the data structure used, discussed in Section 3.1.2, is very lightweight and `onSensorChanged()` is only responsible for passing data to it.

Concurrency considerations

Because `onSensorChanged()` can be called at such a high rate, it is possible that new calls to the method can be made while previous calls are still executing. Data corruption could result from improper handling of asynchronicity.

The documentation for the Sensor API is not explicit about whether calls to `onSensorChanged()` queue on the same thread or whether they can be dispatched asynchronously. For this reason, the `AccelerometerListenerService` was designed to be thread-safe by using Java concurrency primitives.

Power consumption considerations

Typically, Android will power off the display and later the CPU after a period of user-inactivity. Powering off the CPU means that the device will stop recording accelerometer data, and so it is required to maintain a wake-lock which keeps the CPU from powering off. It is also important to remember to release the wake-lock once accelerometer recording is complete. Otherwise, the device's CPU will remain on even when the device appears to be on standby, using battery. It is for this reason that care should be taken to minimise power usage where possible, while still collecting all the required data.

One tradeoff had to be made between collection strategies. One strategy is to record data at a specified sample rate from when the recording is turned on until it is turned off. An alternative strategy is to record a window of data at set intervals and sleep the remainder of the time. For example, one might set the accelerometer to record 10 seconds of data every 50 seconds.

Though the latter strategy saves battery power as the device turns off the accelerometer between recordings, a continous recording approach was taken in this project in order to have as much data as possible. In addition, the battery life was not severly impeded by the continous recording approach.

Sampling rate

In ideal conditions, it would be sensible to sample at the fastest possible rate: the resultant data can always be downsampled afterwards if it is not required. As per the Nyquist-Shannon sampling theory, discussed in Section 2.3, our sample rate should be greater than twice the highest frequency of the signal. Because it isn't possible to know what the highest frequency is going to be, it would be reasonable to sample at a far higher rate.

However, picking a very fast sample rate in this context has two potential downsides: battery life drain and the size of resultant data. I investigated whether either battery life or the size of the resulting data would be a limiting factor of sample rate.

The impact on power consumption when increasing the sample rate was negligible. This may be because the increase in extra work as a result of an increased

sample rate is negligable compared to the cost of keeping the CPU awake at all.

Recall from Table 2.2 that each measurement has a total size of 20 bytes. At a sample rate of 50Hz, data is produced at approximately 1 KBps or 3.6 MB per hour. The most memory-constrained device is the smartwatch, which only has 512 MB of RAM but 4 GB of internal storage. A data structure that stores the accelerometer data to the internal storage rather than to memory is required, but a sample rate of 50Hz produces a storable amount of data on any reasonable-length (i.e. up to one hour) activity recording.

Another potential concern regarding data size is the transfer from the smartwatch to the smartphone. The only connection available is Bluetooth. The Bluetooth connection empirically has a maximum transfer rate of no more than 150 KBps, meaning an hour of activity data will take approximately 30 seconds to transfer.

3.1.2 Storing accelerometer data

The data structure to hold the accelerometer data on both the phone and the watch is required to be:

- **fast** because it will be written to many times per second and cannot block;
- **on-disk** rather than in-memory, because the smartwatch may not have enough free memory to store all the accelerometer data for lengthy recordings;
- **thread-safe** as it is unclear whether calls to `onSensorChanged()` are queued or concurrent.

The data structure decided on was a temporary random-access file with buffered writing. The data is written as bytes through an output buffer. The output buffer is maintained in memory and is flushed when it reaches capacity. The capacity of the output buffer was set to 20000 bytes as data is only written in multiples of 20 bytes and the smartwatch is comfortably able to keep 20 kb in memory. This equates to data being saved to disk approximately every 20 seconds.

	DataItem	Asset
Advantages	• no separate data fetching step • simpler, more reliable receiver code • negligable transmission time	• no hard size limit • can create an Asset from a File without storing it in memory
Disadvantages	• 100 KB size limit • have to insert byte arrays	• some constructors don't seem to work • transmission of large files takes a noticable amount of time • separate data fetching step requires more receiver-side code

Table 3.1: Advantages and disadvantages of using the `DataItem` and `Asset` to transmit data from the smartwatch to the smartphone.

3.1.3 Transmitting accelerometer data

The accelerometer data has to be transmitted from the smartwatch to the smartphone before it can be transferred to a computer. As discussed in Section 2.4.2, there are two relevant methods to transfer data between the smartwatch and the smartphone: a DataItem and an Asset. Their advantages and disadvantages with respect to this project are highlighted in Table 3.1.

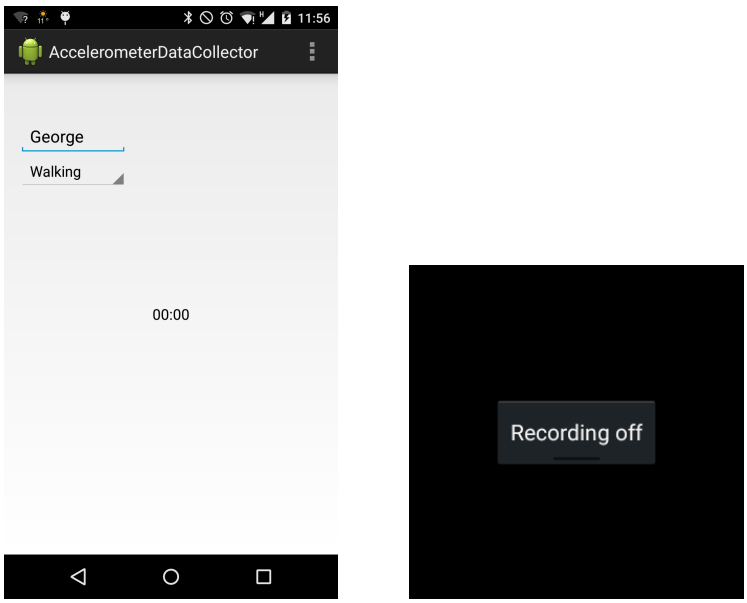
Because the DataItem has a 100 KB limit, an alternate transmission and storage system would have to have been built, where the smartwatch collects 100 KB of data and sends that to the smartphone while it continues to record. It is then reassembled at the smartphone receiver.

I consider this solution inferior to the Asset implementation, which allows transmission of any size of data.

3.1.4 Mobile apps

This section concerns the development of the user-facing components of the application.

Figure 3.1 presents screenshots of both the smartphone and smartwatch apps produced.



(a) The phone app *(b) The watch app*

Figure 3.1: Screenshots of the phone and the watch apps

On the smartphone, configuration options are provided on the smartphone. This includes the ability to type a recorder's name and select the activity that is being recorded, which is then included in the recording's filename. A timer is also present, which displays the duration of the current recording so far.

Activity recording is started and stopped from the smartwatch. This is because the smartwatch is typically more accessible than the smartphone, being worn on the wrist rather than left in a pocket. The user is also free to place the phone anywhere before recording starts. This is the only user activity available from the smartwatch.

Figure 3.2 illustrates the state machine which models the recording cycle. A typical recording cycle is thus

1. The user sets their name and the activity they are recording on the phone.
2. The user begins the recording on the watch.
3. The watch messages to the phone app, telling it to begin recording.
4. The phone begins recording.
5. The user performs the activity.
6. The user ends recording on the watch.
7. The watch app messages the phone app, telling it to end recording.
8. The phone stops recording.
9. The watch sends its data to the phone, which requires a separate lifecycle:
   (a) The watch messages the smartphone indicating there is data to transfer.
   (b) The phone sets up the transfer of the data from the watch.
   (c) The phone saves the data from the watch.
10. The phone also saves its own data.

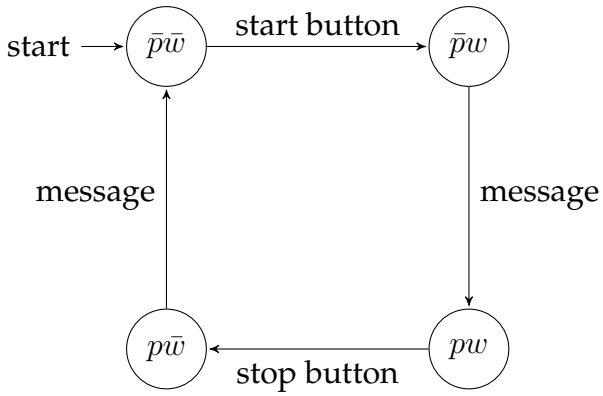


Figure 3.2: A state machine of the recording cycle. p and w indicate that the phone and watch are recording respectively. Message passing is implemented through the Android Wear API.

A decision that had to be made was whether it was required for the recordings to start at precisely the same time, as the message parsing between the watch and the phone to signal the start of the recording is fast but it is not instant.

Recordings starting at exactly the same time mean that it may be easier to correlate movements between devices, but it this doesn't seem to be particularly useful if the message parsing means the records start within microseconds of each other and cross-correlation may be able to help align them completely.

3.2 Activity analysis

The apps were used to collect data over a range of activities. This section provides the graphs of the magnitude and the Fourier transform of the magnitude. I used these graphs to better understand the accelerometer readings resulting from different activities and justify the utility of the features I hope to extract from the data.

For each activity, I graph an arbitrary ten second snippet as recorded by both the phone and watch. The ten second snippet has been low-pass filtered with a critical frequency of 5Hz, as described in Section 3.3.1. The Fourier transform for the same snippet is also provided, so as to display the recording in the frequency domain as well as the time domain.

For all recordings, the watch was worn tight on the non-dominent left hand and the phone was kept in the right side trouser pocket.

Recall from Section 2.7 the activities I picked to classify:

- Physical activities requiring whole body motion:
  - Climbing
  - Cycling
  - Gym cycling
  - Running
  - Stair climbing
  - Walking
- Activities that primarily require arm motion:
  - Computer use

  - Eating
  - Playing fussball
- Low energy upright activities:
  - Gallery perusal
  - Standing
  - Teeth brushing

3.2.1 Climbing

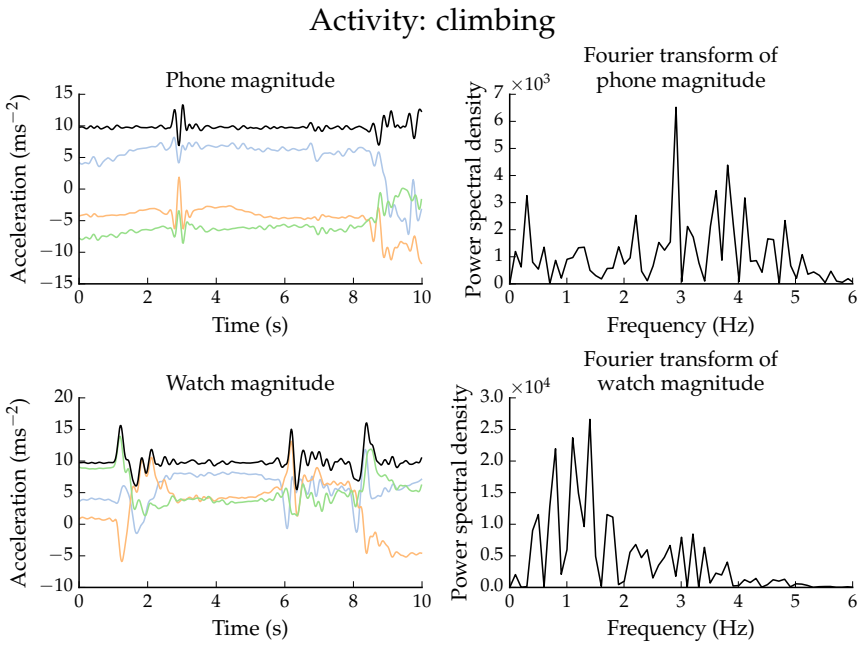


Figure 3.3: Ten seconds of phone and watch data from a climbing activity together with their Fourier transforms.

I recorded a session of indoor bouldering, which is climbing on short walls with no ropes.

Climbing has no period or pattern, as movement is wholly dependent on the routes being climbed. Magnitude of acceleration is unpredicable: moves can be made quickly or slowly.

3.2.2 Computer use

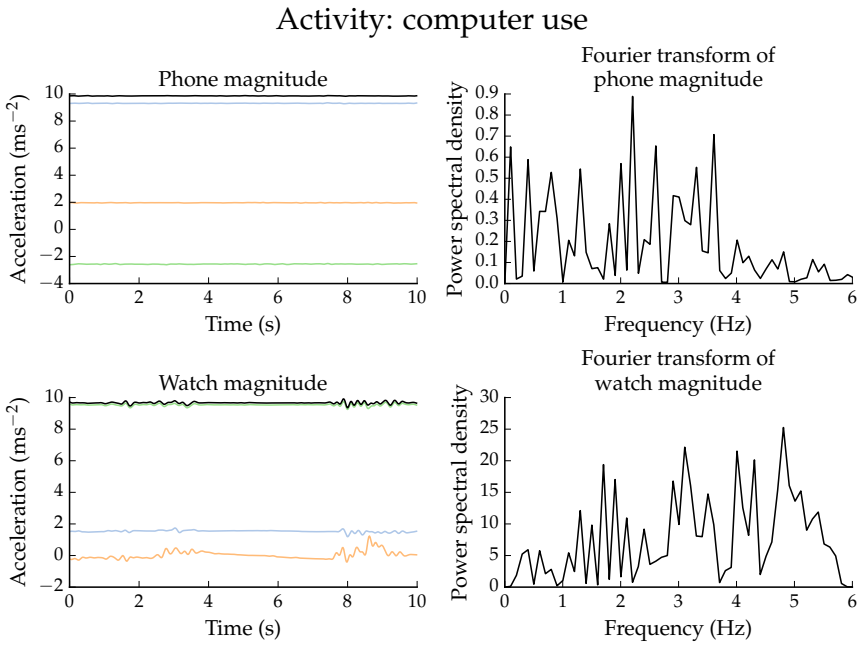


Figure 3.4: Ten seconds of phone and watch data from a computer use activity together with their Fourier transforms.

Computer use is predominently typing or using a laptop trackpad while seated. The watch was worn on the left, non-dominant wrist while the phone was kept in the right trouser pocket. There is very little movement in the phone, as the leg is mostly stationary. The watch exhibits some periodic movement punctuated by periods of inactivity. This is presumably from typing a word and then pausing.

The Fourier transform of the watch magnitude indicates that watch movement is also aperiodic, as one might expect from typing with frequent pauses.

3.2.3 Cycling

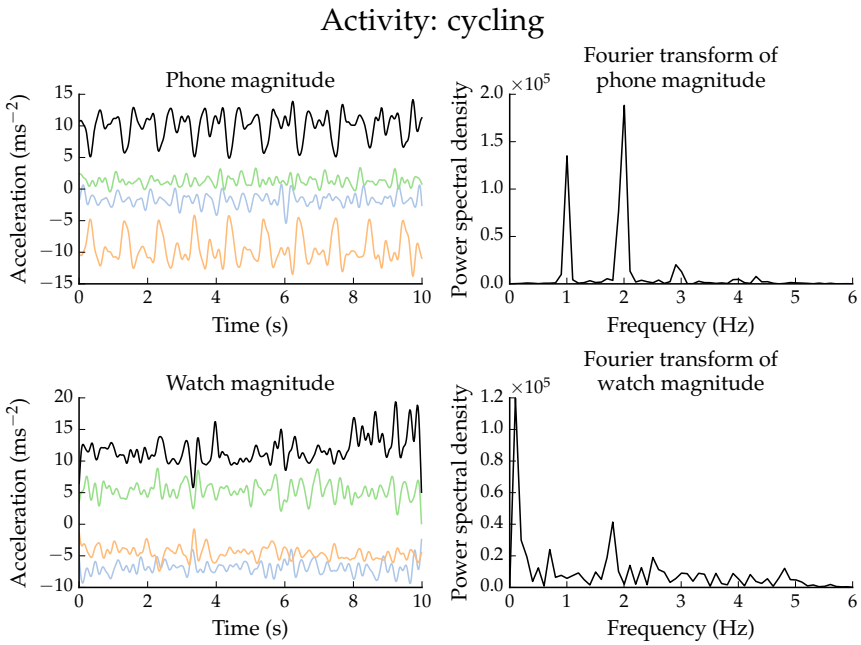


Figure 3.5: Ten seconds of phone and watch data from a cycling activity together with their Fourier transforms.

Cycling is another activity with high periodicity in the phone measurement, but, unlike walking, does not have much periodicity in the watch measurement. This is primarily because of the changing position on the handlebars.

3.2.4 Gym cycling

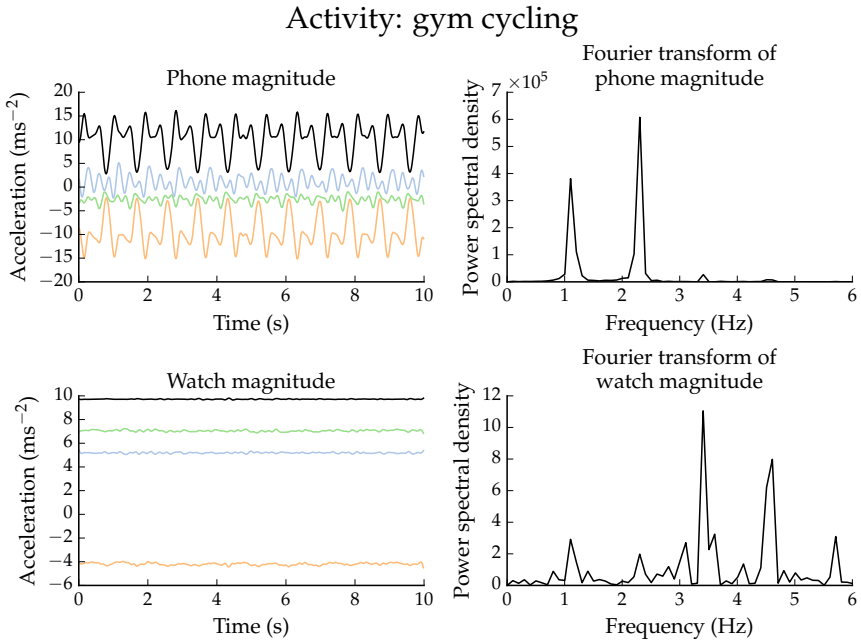


Figure 3.6: Ten seconds of phone and watch data from a gym cycling activity together with their Fourier transforms.

Gym cycling is cycling performed on a fixed cycling machine, as opposed to a bike in the real world.

Compared to cycling outdoors, gym cycling has little wrist movement and is more starkly periodic in the phone measurement, as the peddaling action is much more consistent. There is also a lack of linear acceleration in the gym which is present outdoors. Outdoor cycling is often subject to stopping and starting.

Both types of cycling activity would benefit from anaylsis of the top two frequencies of maximum power, at they both exhibit very strong peaks at 1Hz and 2Hz.

3.2.5 Eating

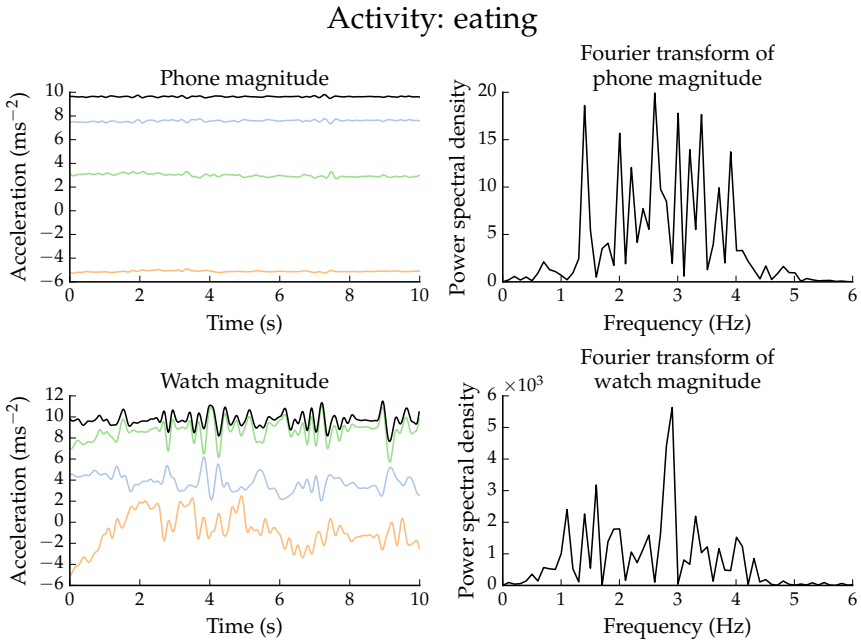


Figure 3.7: Ten seconds of phone and watch data from an eating activity together with their Fourier transforms.

Eating is a seated activity, so shares much of the phone data with something like computer use, again assuming the phone is in the right trouser pocket. The watch data has more energy, but is aperiodic.

A feature that distinguishes amount of movement in the watch should be able to distinguish between eating and computer use activities.

3.2.6 Playing fussball

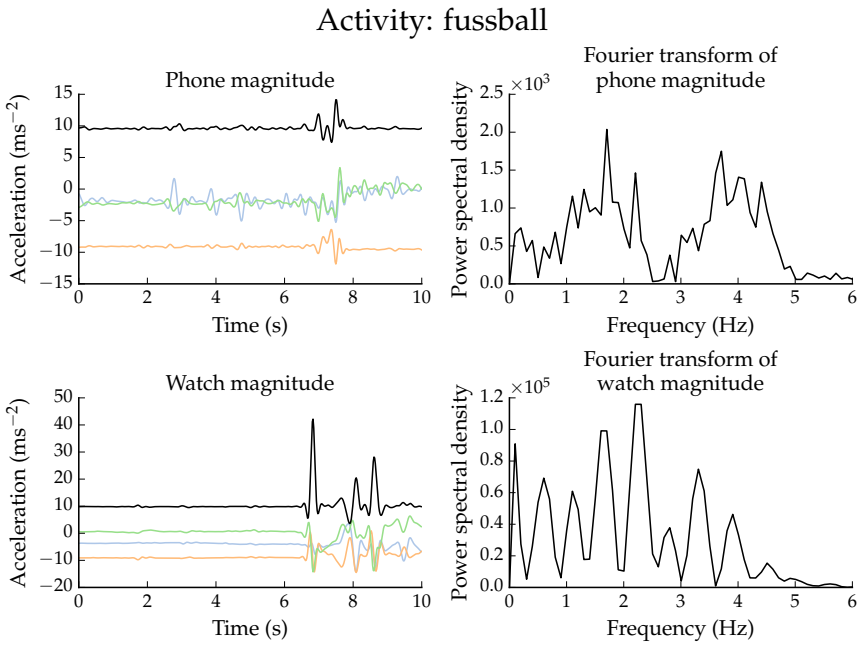


Figure 3.8: Ten seconds of phone and watch data from a fussball activity together with their Fourier transforms.

Fussball (also known as table football) is characterised by periods of inactivity followed by sharp acceleration in the watch measurement. The maximum of the magnitude should in theory differentiate this activity from others.

However, the maximum is by definition a statistic that is highly sensitive to outliers and so other activities may exhibit a similarly high magnitude even if it is not characteristic of that activity. A better metric might be a count of the number of data points that exceed a certain thereshold magnitude e.g. $40\text{m}\,\text{s}^{-2}$

3.2.7 Gallery perusal

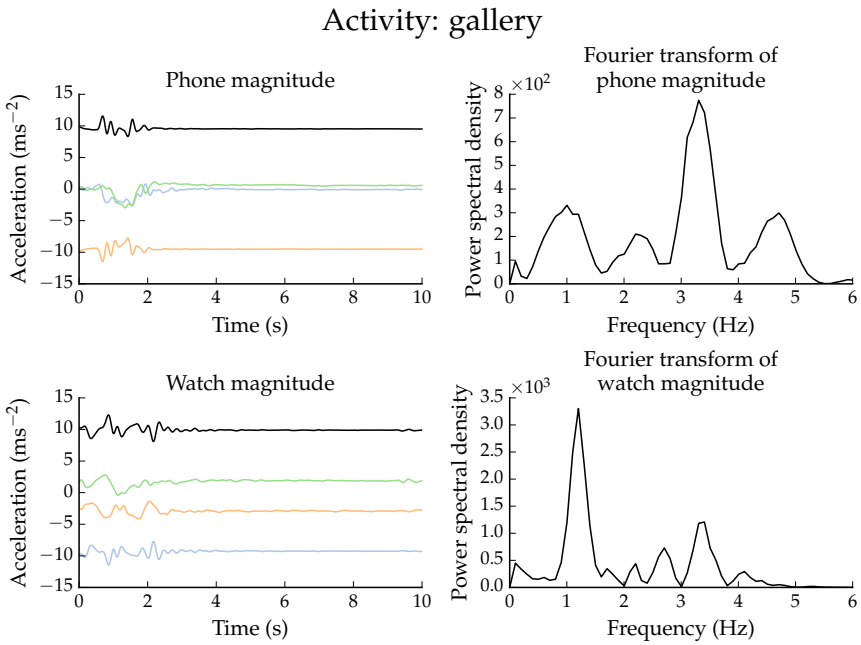


Figure 3.9: Ten seconds of phone and watch data from a gallery perusal activity together with their Fourier transforms.

I recorded data while viewing a gallery exhibition. Gallery perusal presents a unique combination of slow walking and standing.

A good classifier would therefore recognise that both walking and standing were present in the recording and classify it as a gallery viewing activity instead.

3.2.8 Running

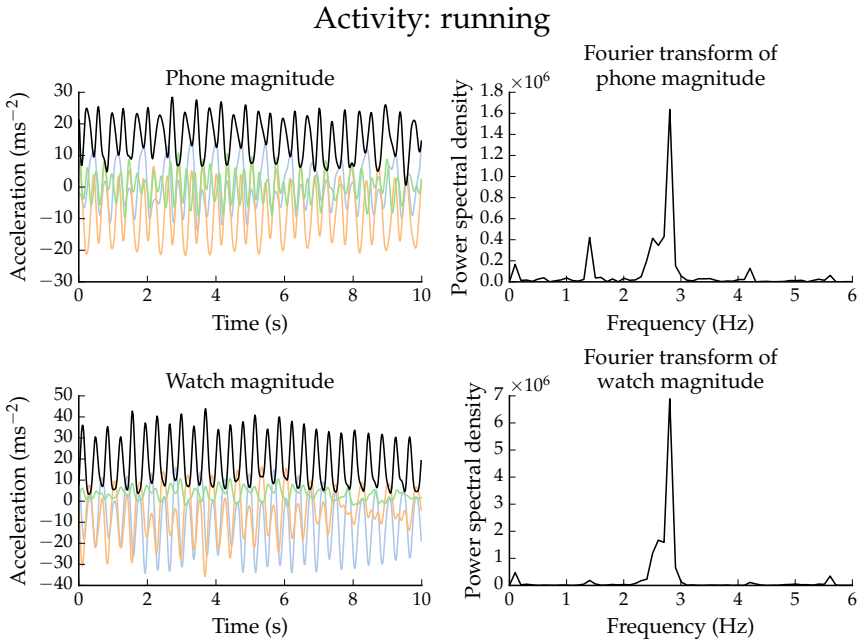


Figure 3.10: Ten seconds of phone and watch data from a running activity together with their Fourier transforms.

Running was performed outdoors. Running has a strong period in both the watch and the phone at a frequency which is slightly higher than of walking.

An analysis of the frequency of maximum amplitude may be suffient to classify this activity, though it might be necessary to deterimine how much bigger the peak is than the rest of the power spectrum.

3.2.9 Stair climbing

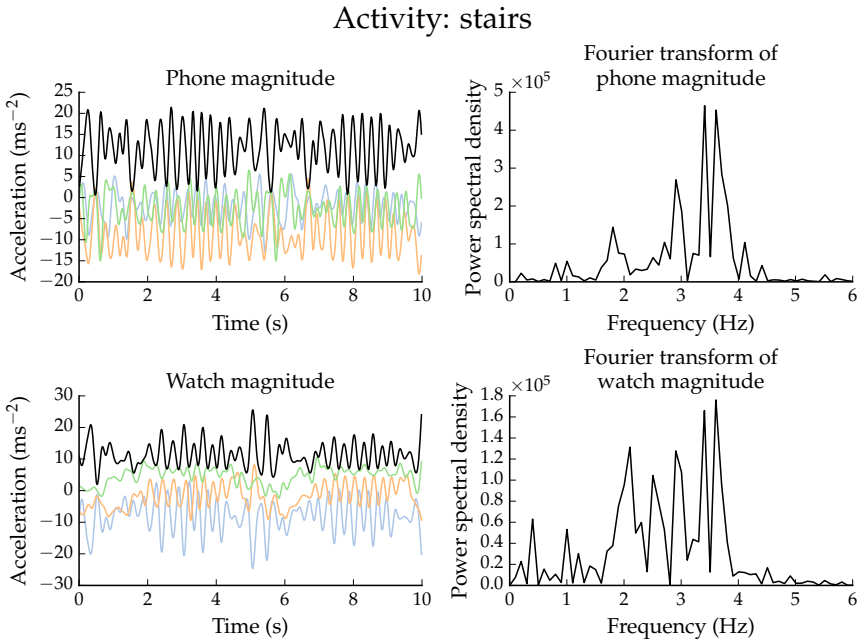


Figure 3.11: Ten seconds of phone and watch data from a stair climbing activity together with their Fourier transforms.

Stairs were climbed in the computer lab. A single recording contains climbing and descending stairs. Stairs were climbed either one at a time or two at a time, with the hand either loose or holding the handrail.

This means that stair climbing exhibits a variety of frequencies — descending stairs is typically done a much quicker pace than climbing stairs, for example. This particular example seems to be drawn from stair descending.

Stairs can be differentiated from walking by noting that though there are still definite periods apparant in the Fourier transform, the peaks are not quite as clear as they are while walking.

3.2.10 Standing

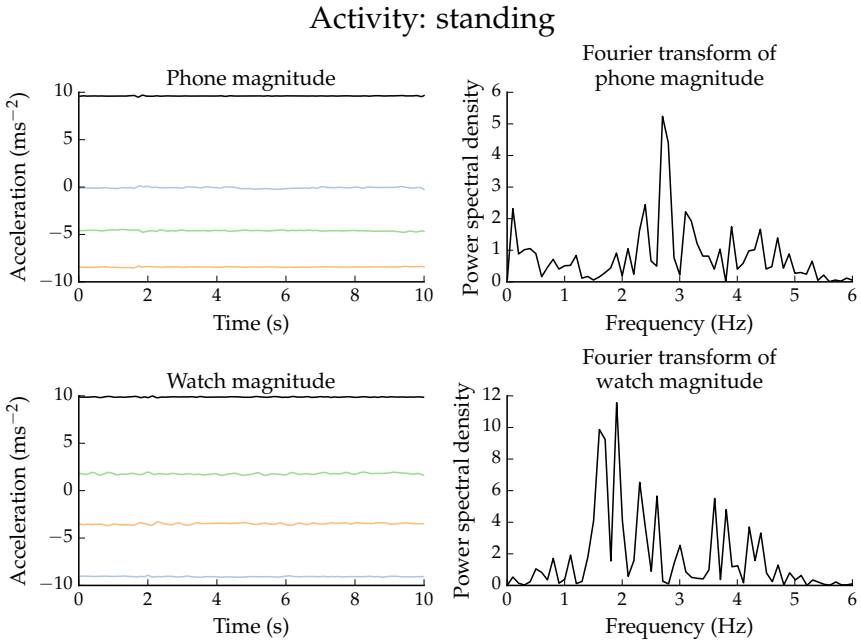


Figure 3.12: Ten seconds of phone and watch data from a standing activity together with their Fourier transforms.

Standing as an activity exists to test differentiation between other upright activities such as toothbrushing and fussball. The phone is largely stationary while the watch moves between certain key standing positions (e.g. arm hanging, in pocket, on hip etc.).

The mean and the standard deviations of the x, y, and z axes of the phone will show standing activities. Standing exhibits less periodicity than teeth brushing and exhibits less magnitude than fussball, but this magnitude may only be seen in the watch.

3.2.11 Teeth brushing

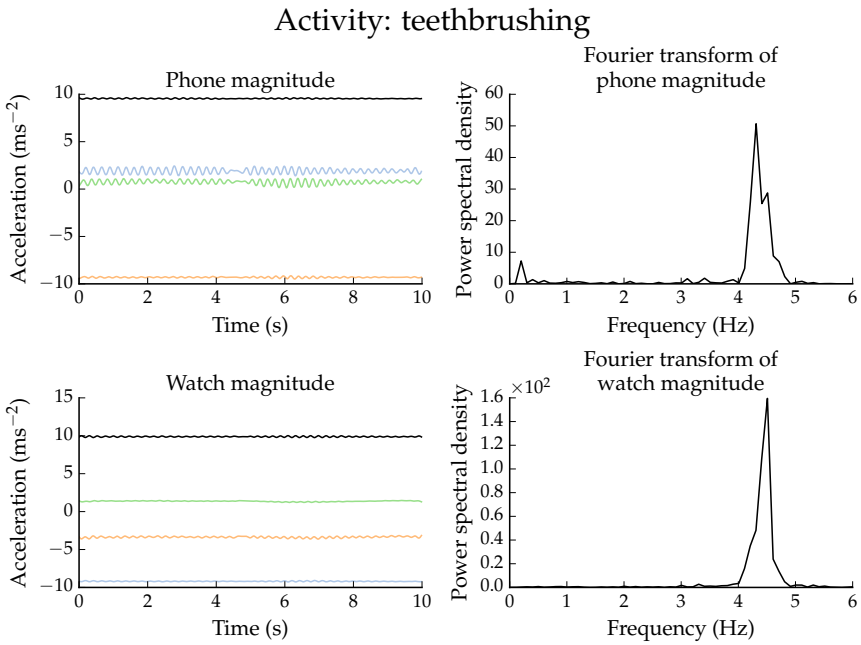


Figure 3.13: Ten seconds of phone and watch data from a toothbrushing activity together with their Fourier transforms.

Teeth brushing is conducted with my dominant right hand, while the watch is worn on the left. The phone remains in the right hip pocket. The left hand is often left hanging or resting on the sink. Nevertheless, quite a clear peak is seen on both the watch and the phone recordings of teeth brushing.

Note that this peak is not likely to appear had the teeth brushing be conducted with an electric toothbrush.

As the only activity with a peak in the $4 \sim 5$Hz range, it should be easy to distinguish from other activities such as standing.

3.2.12 Walking

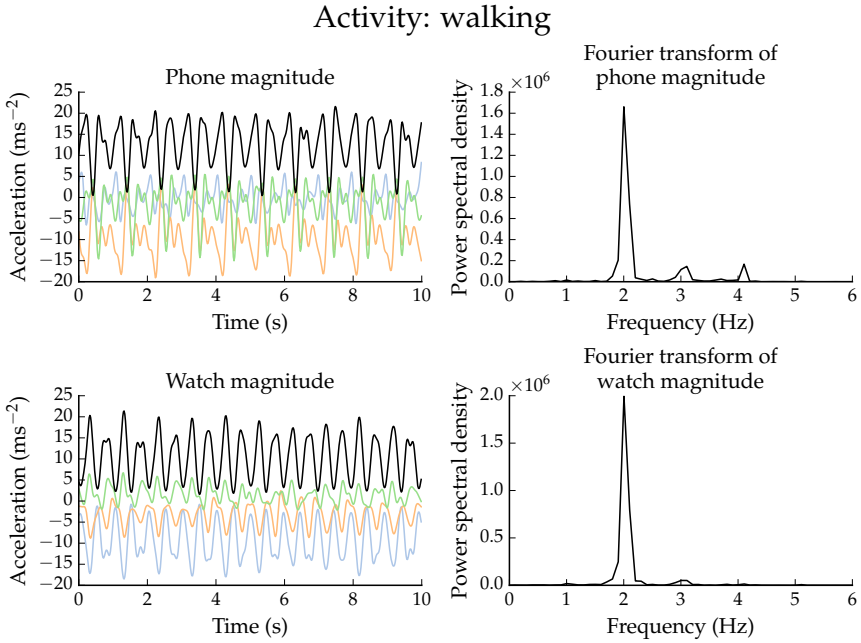


Figure 3.14: Ten seconds of phone and watch data from a walking activity together with their Fourier transforms.

Walking is among the most periodic of all the activities investigated. There is a very strong peak at about 2 Hz – a typical walking pace. There are smaller local maxima at 1, 3 and 4 Hz in the Fourier transform of the phone magnitude, and peaks at 2 Hz and 4 Hz in the watch magnitude. The rest of the Fourier transform is relatively flat.

A useful feature for distinguishing between activies that exhibit strong periodicity is the spectral flatness measure, discussed in Section 3.3.2.

3.3 Data processing

As explained in Section 2.5.1, the data processing pipeline was written in Python, because of the strength of its numerical, statistical and machine learning libraries. The data processing was done on a computer as opposed to directly on the phone for two reasons:

- the computational and memory demand of training a classifier; and
- data processing and extracting features post-hoc permits me to explore different features from the raw data. Feature extraction is a distructive process and so conducting it on-device means I cannot then extract further features in the future. This goes against the agile development methodology.

Extracting features on the watch and the phone would have had one potential upside: the volume of data required to store and to send is greatly reduced.

If this were to be developed for wider use, one might consider using a cloud server for classification.

3.3.1 Importing and preprocessing

Import

Each recording is stored in a separate binary file. The filename is always of the form: `<timestamp>-<recorder>-<device>-<activity>.dat`

NumPy includes methods to specify the types of binary data in a file and create an array from it. These are used to great effect to convert the binary data back into longs and floats.

Data files are then accessed via a SQLite database, using the timestamp as the unique ID. The database allows easier access to individual records and, for example, all recordings of a certain activity.

Preprocessing

Data is preprocessed before feature extraction.

The first step is the drop the first and last 10 seconds of each data recording. This step is justified as these parts of the accelerometer recordings will not actually be representative of the activity to be classified. Rather, they will primarily be recording the starting and ending of an activity.

For each data recording, the magnitude of the acceleration $\|\mathbf{x}\| = \sqrt{x^2 + y^2 + z^2}$ was calculated. Patterns in the magnitude were found to be more distinguishing than any of the features extracted from the three axes individually. The magnitude is orientation-invariant, which gives better results when considering that a wrist may move in the same way but may be oriented slightly differently. In this scenario, periodicity will still be observed in the magnitude but may not be observed in each of the three axes individually.

Data is then filtered. As discussed in Section 2.3, the data recorded by the accelerometer is subject to noise. Reducing the effect of this noise will produce a signal in which it is easier to observe the underlying patterns produced by movement.

A fifth-order Butterworth Filter with a critical frequency of 5 Hz was used in order to achieve this. The Butterworth Filter was chosen because it has no gain ripple in the pass band or the stop band. The slow cutoff is not a problem for the application, as the frequencies of activities concerned are far less than the frequency of the noise. A graph of the frequency responses for several Butterworth Filters is given in Figure 3.15.

The frequency response $G(\omega)$, or gain, of an nth-order Butterworth filter whose critical frequency is 1 Hz is:

$$G(\omega) = \sqrt{(\frac{1}{1 + \omega^{2n}})}$$

Windows

Each data recording is split into 10 second windows, as in Hemminki *et al.*[7]. Features are extracted from each of these bins individually. Splitting into windows allows the production of multiple feature rows from the same recording. In theory, every window for a particular activity should exhibit extracted features that are consistent. A 10 second window was picked as a balance between producing enough feature rows from collected data and ensuring that several cycles of an activity were included.

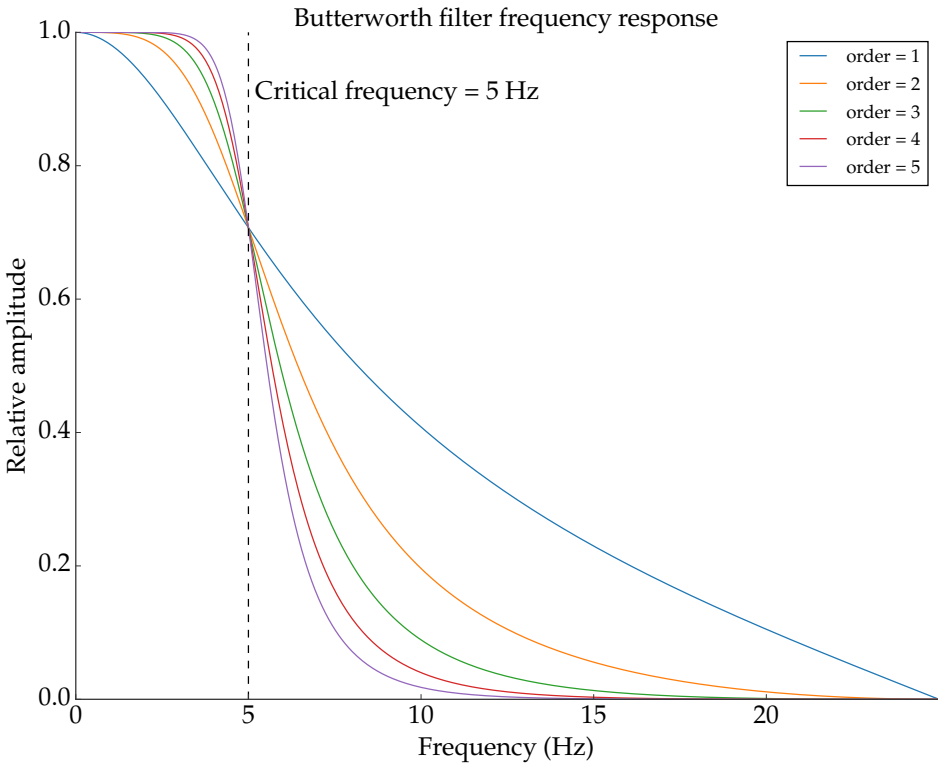


Figure 3.15: Frequency response for Butterworth filters of different orders. The higher the order, the quicker the cutoff between the pass band and the stop band. Each has a critical frequency of 5 Hz*. At the critical frequency, the frequency response is equal to* $\frac{1}{\sqrt{2}}$*. A fifth-order Butterworth filter was used to filter the noise from the accelerometer data.*

No of features	Description of each feature
4	Mean of each axis and magnitude
4	Standard deviation of each axis and magnitude
4	Maximum amplitude of each axis and magnitude
4	Median absolute deviation of each axis and magnitude
3	Pairwise correlation coefficient of each of the three axes
1	Spectral flatness of magnitude
1	Spectral entropy of magnitude
1	Frequency of maximum amplitude in the power spectrum of the magnitude

Table 3.2: A summary of extracted features. A total of 22 features were extracted from the phone data, and another 22 features extracted from the watch data.

3.3.2 Feature extraction

In total, 22 features are extracted from each window. The smartwatch and smartphone data are treated as separate windows for the purposes of feature extraction. This section goes on to describe and justify each of these features.

Mean of each axis and magnitude

The arithmetic mean $\bar{X}$ defined as

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

was calculated for each of the x, y and z axes and also for the magnitude.

The arithmetic mean does not encode a lot of data, but is useful for determining primary orientation during the activity. For example, computer use has a z mean which is close to gravitational acceleration while the others are near zero. This indicates the devices primarily points upward during this activity.

Standard deviation of each axis and magnitude

The standard deviation σ defined as

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\bar{x} - x_i)^2}$$

was calculated for each of the x, y and z axes and also for the magnitude.

The standard deviation gives a measure of how much variation is present in each of the axes, and hence is useful when trying to recognise those activities that consistently have little variation in a particular measure. Gym cycling is one such activity, where the wrist moves comparatively little.

Maximum amplitude of each axis and magnitude

The maximum $x_{\max}$ defined as

$$x_{\max} = \max_i x_i$$

The maximum is a potentially useful figure but has the propensity to vary significantly between instances of the same activity. Another, perhaps more useful measure, would be the number of times the magnitude of the acceleration exceeds a certain threshold.

Median average deviation of each axis and magnitude

The median average deviation x_{mad} is defined as

$$x_{\text{mad}} = \text{median}_i(|x_i - \text{median}_j(x_j)|)$$

Though this measure follows the same trend as the standard deviation, its use of the median ensures it is a robust statistic — one that is resistant to outliers — and offers good performance on data that is not normally distributed. This is a desirable characteristic in activities such as fussball as recorded from the watch, as fussball requires gentle moves interspersed with quick movements with high acceleration.

Pairwise correlation coefficient for each of the three axes

The covariance is a measure of how much two random variables change together. The covariance of two random variables X and Y, $\mathrm{Cov}(X, Y)$ is defined as

$$\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \bar{X})(Y - \bar{Y})]$$

The correlation coefficient, $\mathrm{Cor(X, Y)}$, of two random variables X and Y is the normalised covariance of the two random variables.

$$\mathrm{Cor(X, Y)} = \frac{\mathrm{Cov(X, Y)}}{\sigma_{\mathrm{X}}\sigma_{\mathrm{Y}}}$$

$\mathrm{Cor(X, Y)}$ has a value between -1 and 1, with 1 representing total positive correlation, 0 representing no correlation and -1 representing total negative correlation.

The correlation coefficient was calculated for each pair of axes, producing three features: $\mathrm{Cor(X, Y)}$, $\mathrm{Cor(X, Z)}$, and $\mathrm{Cor(Y, Z)}$. The correlation coefficients give a measure of how much the axes move together during the recording. This encodes some information about the direction of movement.

Spectral flatness of magnitude

Spectral flatness is also known as the tonality coefficient. It is a measure of how noise-like or tone-like a signal is. White noise has spectral flatness approaching 1, while a pure tone has spectral flatness approaching zero.

Spectral flatness is calculated from the power spectrum of the signal. Recall from Section 2.3 that the power spectrum is the squared magnitude of the Fourier transform of the signal.

If x_i represents the magnitude in the power spectrum of bin i, then the spectral flatness of a power spectrum $X = [x_1, x_2, \ldots, x_N]$ is defined as

$$\mathrm{Flatness(X)} = \frac{\sqrt[N]{\prod_{i=1}^{N} x_i}}{\frac{1}{N}\sum_{i=1}^{N} x_i}$$

The geometric mean can be expressed as a summation of logarithms rather than a product, giving an alternative formula for the spectral flatness that does not require a large product or an nth root. As the Fourier transforms in this project are likely to be over 10000 elements long, avoiding the large product or the expensive nth root calculation is desirable. Following conversion to a logarithmic summation, the spectral flatness is:

$$\text{Flatness(X)} = \frac{\exp\left(\frac{1}{N}\sum_{i=1}^{N}\ln x_i\right)}{\frac{1}{N}\sum_{i=1}^{N} x_i}$$

Spectral flatness gives a measure of how periodic a signal is. Activities where there is very high spectral flatness, akin to white noise, are aperiodic. For example, fussball and standing have no associated period, while walking has a clear period when measured through the smartphone.

Spectral entropy of the magnitude

The spectral entropy of a signal is calculated as the entropy of its power spectrum. It is defined as:

$$\text{entropy} = -\sum_{i=1}^{N} x_i \log_2 x_i$$

This feature is used successfully in Hemminki *et al.*[7] as a measure of how much information is present in its power spectrum.

Frequency of maximum amplitude in the power spectrum of the magnitude

The power spectrum of the magnitude shows the distribution of power in the frequencies of a signal. It is defined as the squared magnitude of the Fourier transform of the signal.

Rather than take the Fourier transform of the original signal, the mean of the signal is first subtracted. If the original signal oscillates about some non-zero

offset, the Fourier transform will have a spike at the origin (0 Hz, or the DC component). To avoid incorrectly classing 0 Hz as the maximum, the mean is subtracted from the original signal. The frequency of maximum amplitude of a power spectrum X is then:

$$\mathrm{f}_{\max} = \underset{f}{\operatorname{argmax}}\, X_f$$

The frequency of maximum amplitude gives the principle frequency of the accelerometer data. For example, able-bodied people take between one and two steps in a second, and so we should expect a frequency of maximum amplitude in the $1 \sim 2$Hz range. Indeed, this is empirically what we see.

A summary of the extracted features is given in Table 3.2.

3.3.3 Machine learning

Classification is supervised learning problem. This requires a classifier to be supplied with a set of instances, comprising sets of features, and a set containing a label for each instance. I use the following notation:

- a set of instances $X = \{X_1, X_2, \ldots, X_N\}$ where each X_i is a vector of j features;
- a multiset of labels y where $y_i \in \{1, 2, \ldots, K\}$ is the label for instance X_i;

This set of labelled instances is referred to as the *training set*, which a classifier uses to generate a model. The classifier is then given a set of unlabelled instances, referred to as the *test set*. The true labels of the test set are stored, but remain unknown to the classifier. The classifier generates a prediction for the test set, and a comparison between the predicted labels and the true labels forms the basis of any evaluative technique.

I used four different classifiers:

1. Proportionally stratified random classifier (as a baseline)
2. Naive Bayes classifier
3. Decision Tree classifier

4. Random Forest classifier

I used implementations of these classifiers from Scikit Learn, though I explain the mechanisms below.

Proportionally stratified random classifier

The proportionally stratified random classifier acts as a dummy classifier. It randomly assigns labels to instances based on the proportion of the labels in the training set. This classifier is used to establish a baseline for evaluation.

Thus, this classifier completely ignores all feature information. Given the training set, the classifier generates, for each $k \in \{1, 2, \ldots, K\}$ a count c_k of the number of times label k appears in the set of labels y i.e. the number of instances that are labelled k.

Then during the test phase, the outputted label is in each case randomly selected. The probability of a label k being output is c_k/N, where N is the total number of instances in the training set.

This classifier is only used to provide a baseline measurement against which we can compare other classifiers that do make use of the extracted features.

Naive Bayes classifier

The Naive Bayes classifier makes the naive assumption that all the features in the instance are independent. It then uses Bayesian probability to calculate the most probable class given the instance.

Mathematically we want to find $\mathbb{P}(y_k \mid X_i)$ for each label y_k and each instance $\mathbf{X}_i$.

$$
\begin{aligned}
\mathbb{P}(y_k \mid \mathbf{X}_i) &= \mathbb{P}(y_k \mid x_1, x_2, \ldots, x_j) && X_i \text{ is a vector of features} \\
&= \frac{\mathbb{P}(y_k)\mathbb{P}(x_1, x_2, \ldots, x_j \mid y_k)}{\mathbb{P}(\mathbf{X}_i)} && \text{Bayes' rule} \\
&\propto \mathbb{P}(y_k)\mathbb{P}(x_1, x_2, \ldots, x_j \mid y_k) && \mathbb{P}(X_i) \text{ is constant w.r.t. the label} \\
&= \mathbb{P}(y_k)\mathbb{P}(x_1 \mid y_k) \cdots \mathbb{P}(x_j \mid y_k) && \text{Independence assumption} \\
&= \mathbb{P}(y_k) \prod_{i=1}^{j} \mathbb{P}(x_i \mid y_k)
\end{aligned}
$$

The task then is to find the most probable label given the data:

$$k = \underset{k}{\operatorname{argmax}}\, \mathbb{P}(y_k) \prod_{i=1}^{j} \mathbb{P}(x_i \mid y_k)$$

To calculate the probability of a particular continuous feature value x_i given a label y_k, one can assume that the feature values follow a Gaussian distribution. Then, one can calculate the mean μ and the variance σ^2 for the value of the feature for a particular class.

During the test phase, one can calculate the probability that x_i takes its actual value v given each of the classes using the equation for a normal distribution parameterised by μ and σ^2 for that particular class:

$$\mathbb{P}(x_i = v \mid y_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(v-\mu)^2}{2\sigma^2}\right)$$

Decision Tree classifier

A decision tree classifier forms a tree where each node is a decision about a feature and labels appear as leaves.

The training procedure for a decision tree classifier builds the tree recursively. For a node m, let Q be the set of instance–label pairs in m. Create a possible

split $\theta = (j, t)$ where j is a feature and t is some threshold value. Generate two subsets Q_{left} and Q_{right}, where

$$Q_{\text{left}}(\theta) = \{(\mathbf{X}_i, y_i) \mid x_j <= t\}$$

$$Q_{\text{right}}(\theta) = Q \setminus Q_{\text{left}} = \{(\mathbf{X}_i, y_i) \mid x_j > t\}$$

The impurity G at m is a measure of how far from a perfect dichotomy the split θ is for Q. The impurity G requires an impurity metric H.

$$G(Q, \theta) = \frac{|Q_{\text{left}}(\theta)|}{|Q|} H(Q_{\text{left}}(\theta)) + \frac{|Q_{\text{right}}(\theta)|}{|Q|} H(Q_{\text{right}}(\theta))$$

There are two typical functions for the impurity metric H: the Gini impurity and the information gain.

The Gini impurity is given by:

$$H(\mathbf{X}) = \sum_{i=1}^{K} f_i(1 - f_i) = 1 - \sum_{i=1}^{K} f_i^2$$

The information gain impurity is given by:

$$H(\mathbf{X}) = \sum_{i=1}^{K} f_i \log f_i$$

where f_i is the proportion of instances labelled y_i in $\mathbf{X}$. Note that both the Gini impurity and the information gain impurity reach their minimum value, 0, when $\mathbf{X}$ contains only a single class.

The task then for the training phase of the decision tree classifier is to select the split θ^* that minimises the impurity:

$$\theta^* = \operatorname*{argmin}_{\theta} G(Q, \theta)$$

The classifier then recursively repeats this process for Q_{left} and Q_{right} unless:

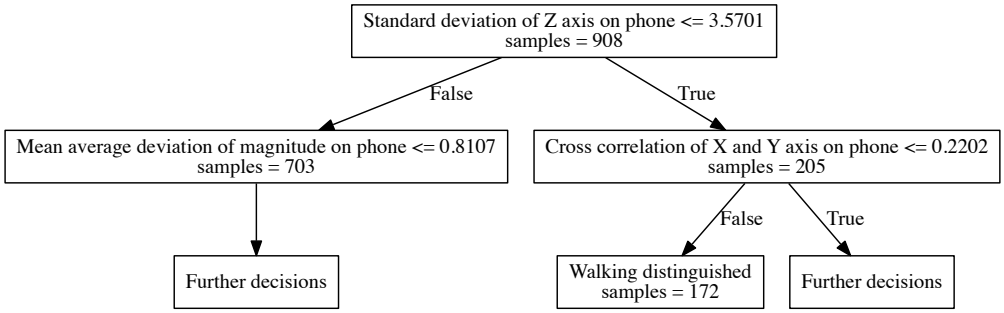


Figure 3.16: An extract of a decision tree trained on phone and watch data. Only two decisions are required to distinguish all walking samples in the training set.

- Q_{left} or Q_{right} contain a single class, at which point they become a leaf node; or
- a pre-specified maximum depth has been reached; or
- the number of samples sent to the child node is less than a pre-specified minimum.

The introduction of a maximum depth and minimum sample size attempts to reduce the tendency for decision tree classifiers to overfit to the training data.

The Scikit-Learn implementation of decision tree classifiers uses an optimised version of the CART (Classification and Regression Trees), developed by Breiman *et al.*[4].

A single CART model is easy to interpret, as it can be illustrated as a set of binary decisions. An extract of a decision tree classifier trained on phone and watch data can be seen in Figure 3.16.

The settings I used for the configuration of the decision tree – known as hyperparameters – I left at default settings. Optimising hyperparameters was ultimately unnecessary for accurate classification, and is dealt with partially by also using a random forest.

Random Forest classifier

The random forest classifier is an example of an ensemble classifier, one that makes use of a set of other classifiers. The random forest classifier that trains a collection of decision tree classifiers. Each decision tree classifier finds the best split from a random subset of features, rather than the absolute best split. The outputs from all the decision tree classifiers are then aggregated by the random forest classifier by outputting the modal label.

Decision tree classifiers are prone to overfitting, especially when the are deep. The random forest classifier reduces variance by excluding different features from the training set, thus is less prone to overfitting.

The cost of a random forest classifier over one single decision tree classifier is the increase in training time and memory requirements and also the loss interpretability of the resulting tree structure. The increased computational time required of a random forest classifier is not such a detriment in this offline processing task, but one might choose either a single decision tree or even a naive Bayes classifier if attempting to classify activities live.

I kept most of the random forest hyperparameters at their default values except for the number of trees to use, which I increased from 10 to 50. More trees increases confidence in results.

3.4 Summary

In this section I have discussed:

- the backend and frontend components necessary to collect accelerometer data from the smartphone and smartwatch
- the activities I hope to be able to classify;
- how those activities have influenced the features I have extracted from the data;
- how four different classifiers use labelled instances of those features to classify new instances.

Chapter 4

Evaluation

4.1 Data collection method and data collected

Table 4.1 details the amount of evaluation data collected.

The data was collected with the smartwatch worn on the left wrist and the phone loose in the right hand trouser pocket.

Activity	Counts	Proportion	Time recorded (minutes)
Walking	354	19.49%	59
Gymcycling	291	16.02%	48
Cycling	286	15.75%	48
Fussball	173	9.53%	29
Eating	172	9.47%	29
Computeruse	165	9.09%	28
Standing	92	5.07%	15
Stairs	76	4.19%	13
Running	62	3.41%	10
Gallery	59	3.25%	10
Teethbrushing	49	2.70%	8
Climbing	37	2.04%	6
Total	1816	100.00%	303

Table 4.1: A summary of data collected.

4.2 Evaluation process

Once the data was gathered, it was processed and features were extracted according to Section 3.3.

The data set was then split into training set and a testing set. This was done with a stratified shuffling splitter. This means that the proportional distribution of true labels in the testing set follows that of the training set. Whether an instance is placed in the training set or the testing set is chosen at random.

The split was performed 10 times, with elements chosen at random each time. The splitter does not guarantee that the same split will not be made on subsequent splits, but such an event is unlikely given the size of the dataset.

The data set was split in the ratio 50:50 training:testing.

Once the data had been split, the labels were stripped from the testing data and set aside. Then, three separate instances of each of the four classifiers was trained on the training set. Each of the three instances only had access to features extracted from the phone accelerometer data, features extracted from

the watch accelerometer data and both the phone-extracted and the watch-extracted features respectively.

Each of the three instances were then tested on the test set and their evaluation metrics — confusion matrix and F_1 measure — were calculated by comparing the true labels to the classified labels.

4.3 Evaluation metrics

Two primary methods of evaluation are used in this project: F_1 measure and the confusion matrix. This section details these methods of evaluation. These metrics are discussed in Sokolova *et al.*[16]

4.3.1 F1 measure

In order to define the F_1 measure, it is necessary to first define precision and recall. In a

Precision , defined as $\frac{\text{True positives}}{\text{True positives}+\text{False positives}}$, is the proportion of instances of a particular label in which the classifier's labels agree with the true labels.

Recall , defined as $\frac{\text{True positives}}{\text{True positives}+\text{False negatives}}$, is the proportion of all the instances with a particular label which are labelled as such.

The F_1 measure is then defined as the harmonic mean of precision and recall:

$$\mathrm{F}_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

We use the F_1 measure rather than precision or recall in isolation because neither provides sufficient information. It is trivial to maximise recall in isolation: simply label everything. A precision of 1 indicates everything we have returned is correct, but does not take into account how many instances we have missed.

As opposed to accuracy — the proportion of instances that were correctly classified — the F_1 measure is more informative when the instances are rare com-

pared to the size of the dataset. Consider a binary classification problem with two labels: χ and $\bar{\chi}$. If χ occurs just 1% of the time, a classifier that always returns $\bar{\chi}$ will by 99% accurate. However, its F_1 measure for the χ class will be 0. Because no class makes up more than 20% of the dataset (see Table 4.1), F_1 is a better evaluation measure in this case.

The F_1 measure reaches its best value, 1, when both precision and recall equal 1. That is when all the classifier's labels agree with all the true labels and none of the true instances have been mislabeled.

Precision and recall, and thus the F_1 measure, are only defined for a single class (i.e. a binary classification problem). As this is a multi-class classification problem, the F_1 measure is reported for each class separately.

The results from the classifiers are nondeterministic for two reasons:

1. the stratified shuffling splitter splits the dataset into a training set and a testing set proportionally at random; and
2. the decision tree classifier and random forest classifier are nondeterministic in their operation.

Because of this nondeterminism, multiple trials with different splits are conducted. Because multiple F_1 measures are calculated through these independent splits, the mean of the F_1 measure is given together with its standard error. The standard error is given by:

$$\mathrm{SE}_{\mathrm{F}_1} = \frac{\sigma}{\sqrt{n}}$$

where σ is the standard deviation of all measurements of the F_1 measure and n is the number of trials. Ten trials were used in this project.

4.3.2 Confusion matrix

A confusion matrix lists the true labels as rows and the classified labels as columns. Any given cell c_{ij} is a count of the number of instances with a true label of i that were classified as label j. Confusion matrices display all classification s and misclassifications.

The confusion matrices presented in this project are the additions of confusion matrices from separate trials.

4.4 Phone-only measurements

The following results have been obtained by training each of the four classifiers on features extracted only from phone-collected accelerometer data.

Figure 4.1 gives the F_1 measures for each activity resulting from classification using each of the four classifiers. The random forest classifier performs best of the four classifiers, outperforming others in 75% of activities. All three proper classifiers significantly outperform the baseline dummy random classifier.

Figure 4.2 averages the F_1 measures given in Figure 4.1. The random forest classifier outperforms the other three classifiers in average F_1 measure. The decision tree and naive Bayes classifiers perform at the same level. Again, all three score significantly higher than the baseline dummy random classifier.

Table 4.2 presents a cumulative confusion matrix for the random forest classifier trained on phone-only features. The random forest classifier was chosen as the best performing classifier. The matrix is the addition of indvidiual confusion matrices from ten independent trials.

From the confusion matrix, we see some of the misclassifications we would expect from phone-only classification:

- Computer use misclassified as eating, and vice versa. Both these activities are the only seated activities and the phone should struggle to differentiate between them.
- Gym cycling misclassified as cycling. Both of these activities exhibit the same pedaling motion in the leg, which phone data alone may struggle to differentiate.
- Many standing activities have been misclassified as each other, such as fussball, teethbrushing, standing and gallery perusal. Again, the phone data struggles to differentiate these static upright activities from each other.

Note that the classifier has almost perfect precision on those activities that require highly periodic leg movements: gym cycling, running and walking. All (bar one) of the instances classified as those activities were correct.

Classified as →	B	U	C	E	F	G	Y	R	S	D	T	W
B = Climbing	110	0	32	0	45	0	0	0	0	3	0	0
U = Computer use	0	810	0	16	0	0	0	0	0	0	0	0
C = Cycling	27	0	1325	12	26	7	1	0	12	16	4	0
E = Eating	0	7	0	853	0	0	0	0	0	0	0	0
F = Fussball	6	0	15	0	833	9	0	0	0	4	3	0
G = Gallery	0	0	0	0	33	212	0	0	0	42	3	0
Y = Gym cycling	0	0	38	0	0	0	1412	0	0	0	0	0
R = Running	0	0	6	0	0	0	0	303	1	0	0	0
S = Stairs	0	0	13	0	0	0	0	0	367	0	0	0
D = Standing	1	0	0	0	5	35	0	0	0	416	3	0
T = Teethbrushing	0	0	0	6	13	10	0	0	0	0	215	0
W = Walking	0	0	19	0	5	1	0	0	3	3	0	1739

Table 4.2: Cumulative confusion matrix from ten trials of the random forest classifier, the best performing of all the classifiers, trained on phone-only features.

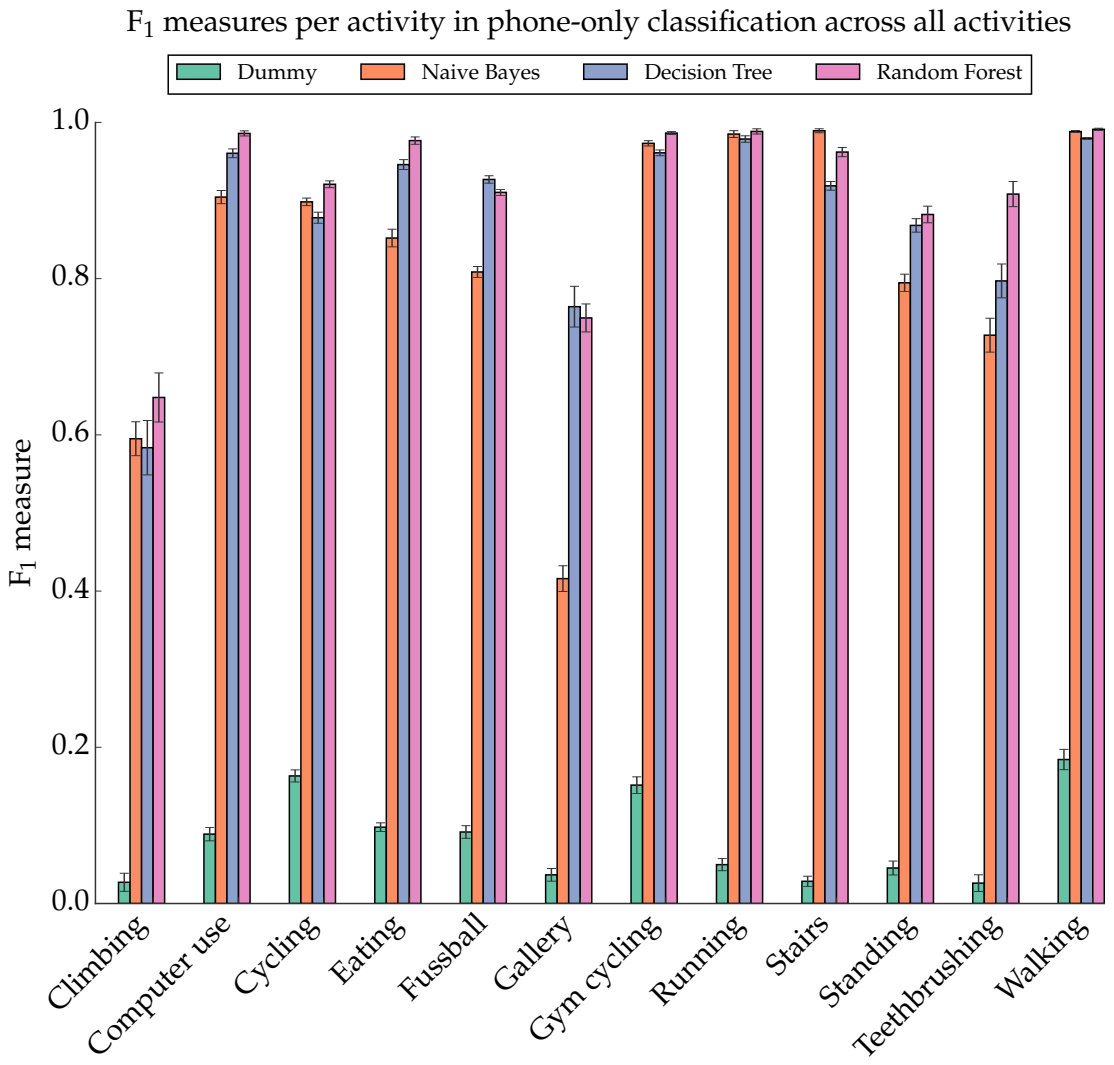


Figure 4.1: F_1 *measures for each activity for each of the four classifiers trained on phone-only features. Best is 1, worst is 0.*

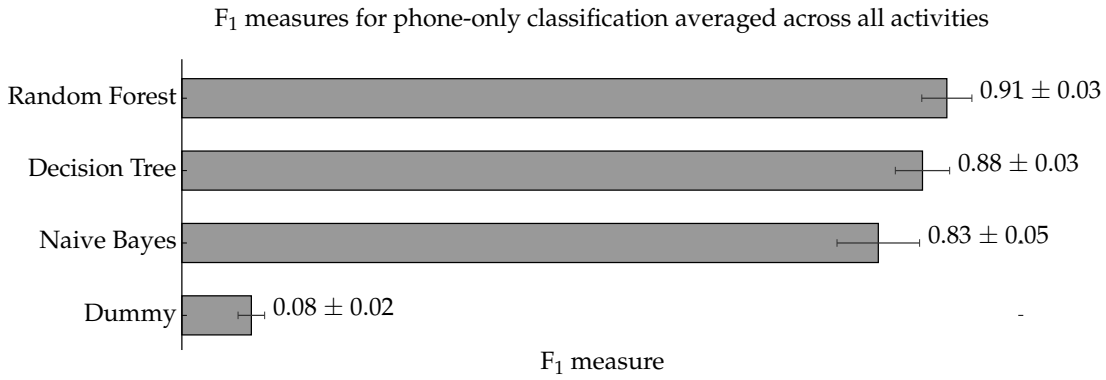


Figure 4.2: Average F_1 measures across all activities for each of the four classifiers trained on phone-only features. Error bars are calculated as the standard error in the mean.

4.5 Watch-only measurements

The following results have been obtained by training each of the four classifiers on features extracted only from watch-collected accelerometer data.

Figure 4.3 gives the F_1 measures for each activity resulting from classification using each of the four classifiers. Like in phone-only classification, the random forest classifier performs best of the four classifiers, outperforming others in 83% of activities.

Figure 4.4 averages the F_1 measures given in Figure 4.3. The random forest classifier outperforms the other three classifiers in average F_1 measure. The decision tree performs marginally better than the naive Bayes classifier. Again, all three score significantly higher than the baseline dummy random classifier.

Table **??** presents a cumulative confusion matrix for the random forest classifier trained on watch-only features. Again, the random forest classifier was chosen as the best performing classifier. The matrix is the addition of indvidiual confusion matrices from ten independent trials.

Compared to phone-only classification, the watch-only classifications exhibit less precision in those leg-period activities, such as running and walking. However, some periodicity is still present in the wrist movement and these activities are still classified accurately. A more interesting case is that of gym cycling, which is highly periodic in the leg movement but completely aperiodic in its wrist movement. As a result, its misclassification rate suffers.

Standing activities also suffer from higher misclassification when using watch-only features.

Unexpectedly, computer use and eating also are subject to a higher rate of misclassification than when using the phone-only features. Fussball, however, is better classified using the watch as opposed to the phone.

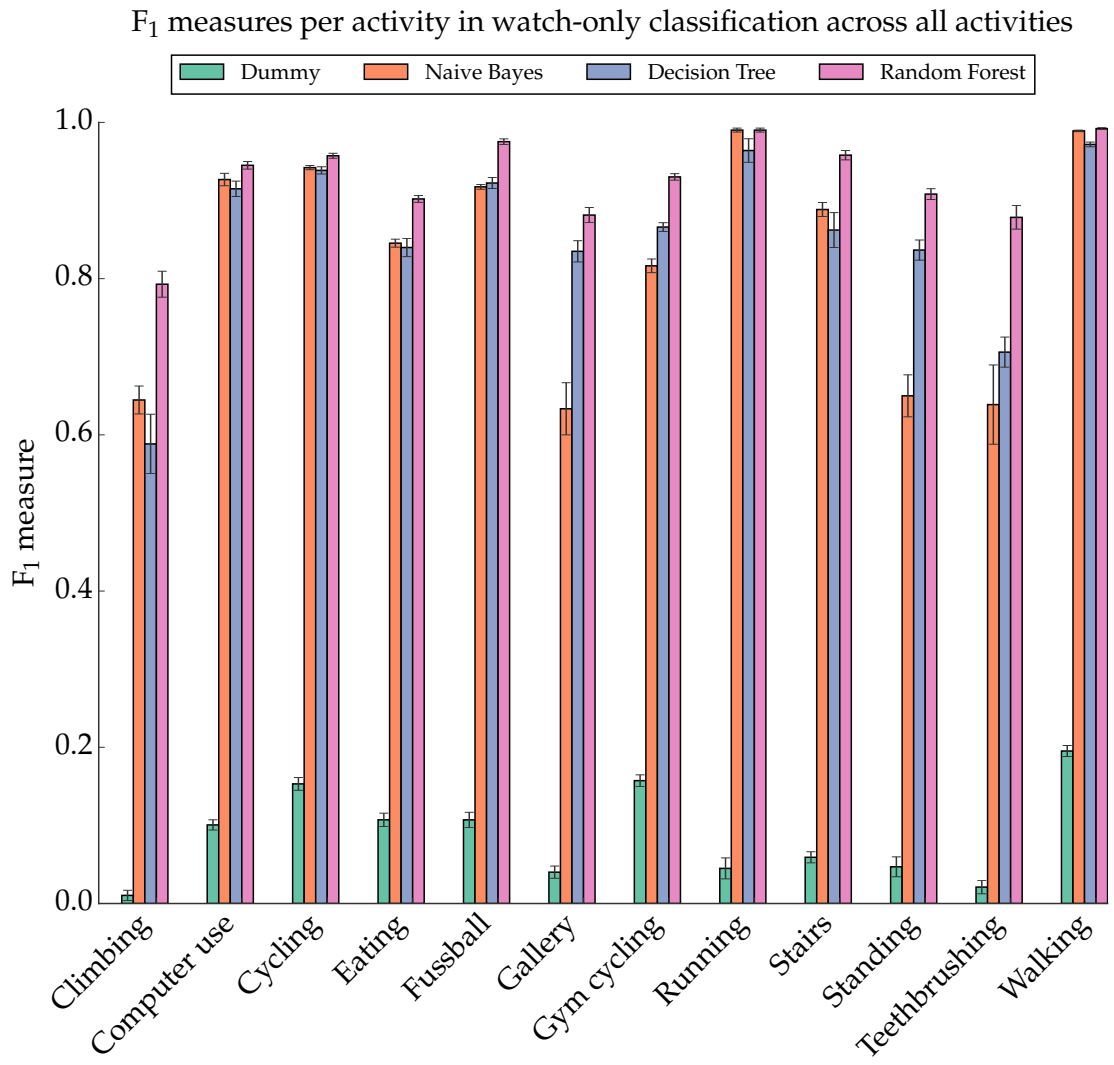


Figure 4.3: F_1 *measures for each activity for each of the four classifiers trained on watch-only features. Best is 1, worst is 0.*

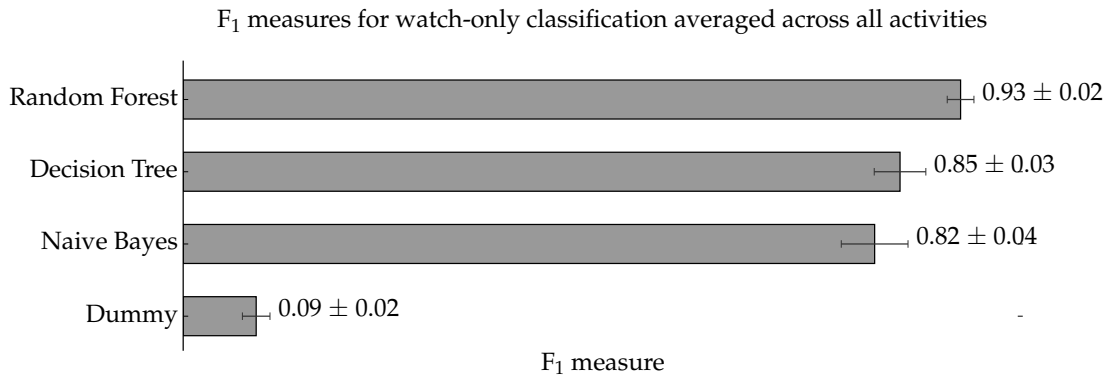


Figure 4.4: Average F_1 measures across all activities for each of the four classifiers trained on watch-only features. Error bars are calculated as the standard error in the mean. The random forest classifier again performs best overall.

Classified as →	B	U	C	E	F	G	Y	R	S	D	T	W
B = Climbing	133	0	6	3	0	4	36	0	8	0	0	0
U = Computer use	0	767	0	39	0	0	15	0	0	5	0	0
C = Cycling	3	0	1376	0	12	13	0	0	4	22	0	0
E = Eating	1	19	0	812	0	0	25	0	0	3	0	0
F = Fussball	0	0	7	1	851	3	4	0	3	0	0	1
G = Gallery	0	0	19	0	0	261	0	0	0	5	4	1
Y = Gym cycling	7	11	2	66	6	0	1352	0	3	3	0	0
R = Running	0	0	2	0	0	0	0	304	0	0	0	4
S = Stairs	0	0	0	5	0	0	1	0	374	0	0	0
D = Standing	0	0	12	0	1	12	17	0	0	417	1	0
T = Teethbrushing	0	0	12	14	4	9	6	0	0	3	196	0
W = Walking	1	0	9	1	1	0	1	0	9	0	0	1748

Table 4.3: Cumulative confusion matrix from ten trials of the random forest classifier, the best performing of all the classifiers, trained on watch-only features.

4.6 Phone and watch measurements

The following results have been obtained by training each of the four classifiers on features extracted from both phone and watch accelerometer data.

Figure 4.5 gives the F_1 measures for each activity resulting from classification using each of the four classifiers trained on phone and watch features. Like in phone-only and watch-only classification, the random forest classifier performs best of the four classifiers, outperforming others in 67% of activities. This percentage however is the lowest of the three feature-set cases. The naive Bayes classifier performs better in more types of activities than the decision tree classifier. This is particularly evident in those activities identified to be periodic: cycling, gym cycling, running, stairs, teethbrushing and walking.

Figure 4.6 averages the F_1 measures given in Figure 4.5. The random forest classifier outperforms the other three classifiers in average F_1 measure. The decision tree and naive Bayes classifiers both score equally at the F_1 measure. Again, all three score significantly higher than the baseline dummy random classifier.

Table 4.4 presents a cumulative confusion matrix for the random forest classifier trained on both phone and watch features. Again, the random forest classifier was chosen as the best performing classifier.

Using both phone and watch features reduces the effect of the broad categories of uncertainty that were present in phone-only and watch-only classification, though in many cases the difference is negligible. The only activity in which using phone and watch features together significantly outperforms either phone or watch classification on its own is in the climbing activitiy.

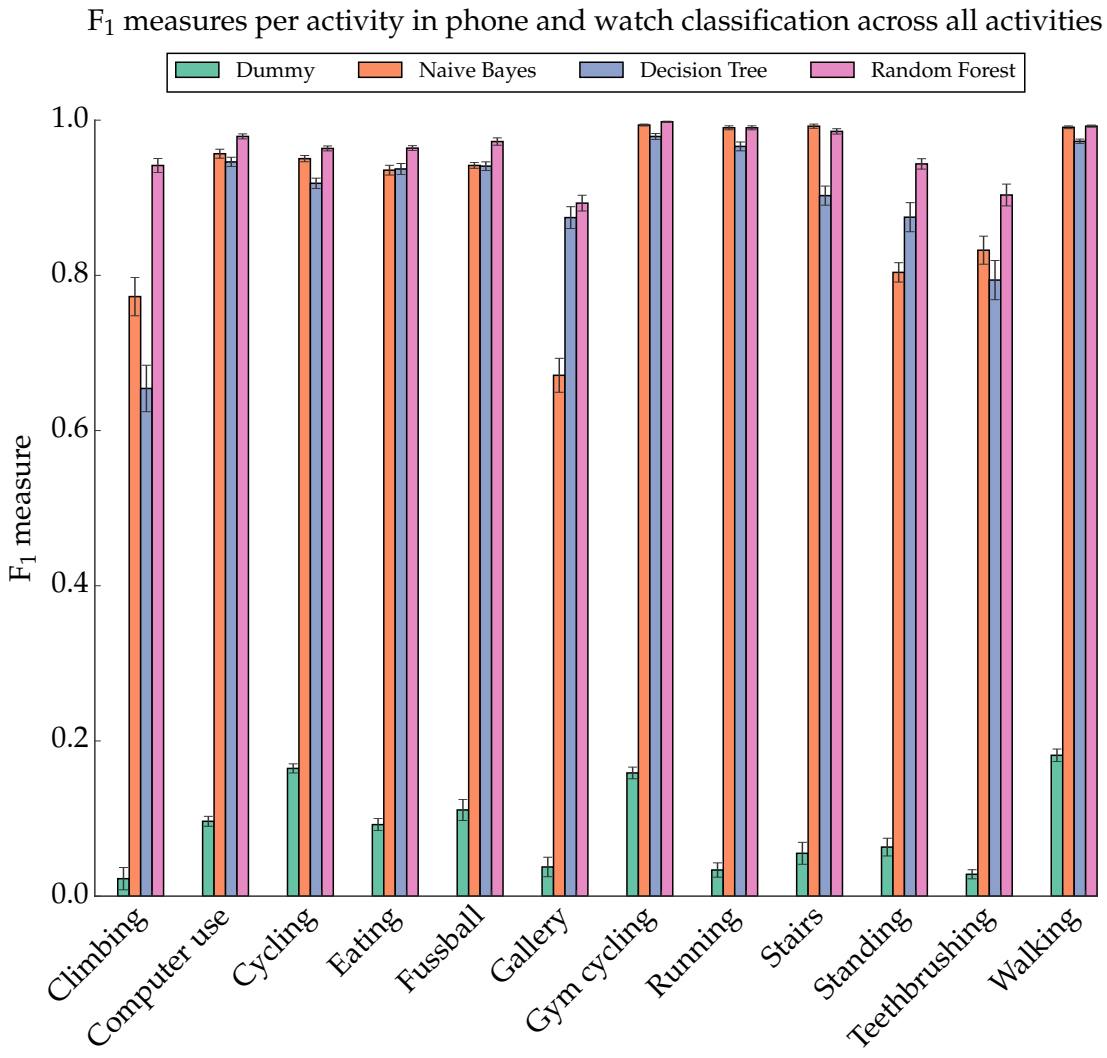


Figure 4.5: F_1 *measures for each activity for each of the four classifiers trained on both phone and watch features. Best is 1, worst is 0.*

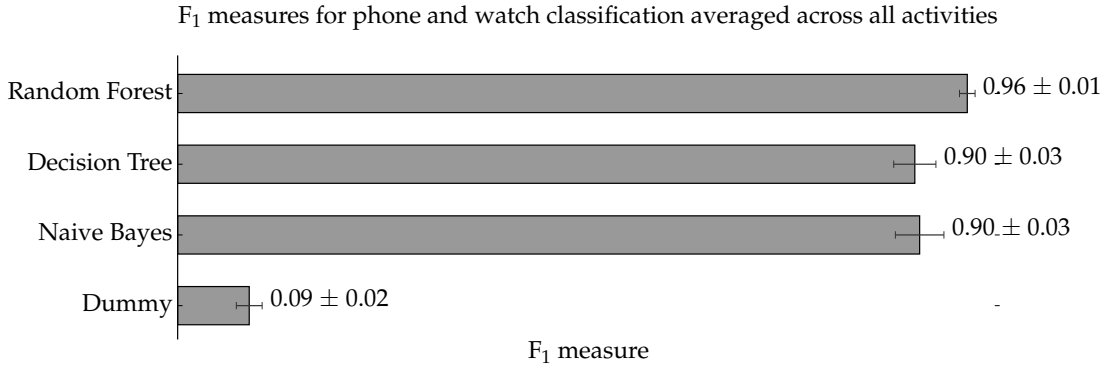


Figure 4.6: Average F_1 measures across all activities for each of the four classifiers trained on both phone and watch features. Error bars are calculated as the standard error in the mean.

Classified as →	B	U	C	E	F	G	Y	R	S	D	T	W
B = Climbing	177	0	5	1	1	4	0	0	1	0	1	0
U = Computer use	0	799	0	27	0	0	0	0	0	0	0	0
C = Cycling	2	0	1375	9	13	12	0	0	0	16	3	0
E = Eating	0	7	0	853	0	0	0	0	0	0	0	0
F = Fussball	0	0	9	0	860	0	0	0	0	0	1	0
G = Gallery	0	0	5	0	10	271	0	0	0	3	1	0
Y = Gym cycling	4	0	2	0	0	0	1444	0	0	0	0	0
R = Running	0	0	5	0	0	0	0	304	0	0	0	1
S = Stairs	0	0	5	0	0	0	0	0	374	0	0	1
D = Standing	2	0	1	1	9	17	0	0	0	430	0	0
T = Teethbrushing	1	0	4	19	2	9	0	0	0	2	207	0
W = Walking	0	0	13	0	4	4	0	0	4	0	0	1745

Table 4.4: Cumulative confusion matrix from ten trials of the random forest classifier, the best performing of all the classifiers, trained on both phone and watch features.

4.7 Comparison

This section presents graphs that directly compare the F_1 measures as calculated by classifiers trained with phone-only, watch-only and phone and watch feature sets.

Figure 4.7 gives F_1 measures for each activity using the random forest classifier trained on each of the three feature sets. On average, the random forest classifier performed best and so is discussed primarily in this evaluation.

Climbing is the only activity in which using both phone and watch feature sets significantly outperforms either feature set on its own. In all other trials, using both the phone and watch feature sets was better but not significantly so. This is not necessarily because using both feature sets performed badly, but because both the phone-only and watch-only feature sets performed well.

Figure 4.8 averages F_1 measures across all activities grouped by classifier and feature set. In all three of the classifiers, the phone and watch outperform phone-only features and watch-only features. On average, the phone-only and the watch-only features perform equally well.

A particularly interesting result is that phone-only classification performs best in computer use and eating classification; introducing features extracted from watch data actually reduces the accuracy of the classifier. It is paradoxical that adding more data would make classification less accurate, especially as most of the information to be gained from computer use and eating activities should come from the watch rather than the phone.

One possible reason for this misclassification is the lack of other seated activities. Computer use and eating are the only two seated activities. Contrast this to semi-stationary standing activities such as fussball, standing, gallery perusal or teethbrushing. Watch-only classification consistently outperforms phone-only classification in these activities, while phone-only classification scores lower in each case than it does for computer use or for eating. I'd argue that one's ability to classify a certain activity depends very much on the other activities present in the dataset and how fine the nuances are between them.

The second of the two overall aims of the project was to evaluate to what extent the smartwatch is better at helping to classify activities. Both figures here show the the smartwatch is just as good as the smartphone, outperforming just the

smartwatch in some activities and performing marginally better on average, though it is not statistically significant. Using both phone and watch features outperforms both on average and on a per-device level.

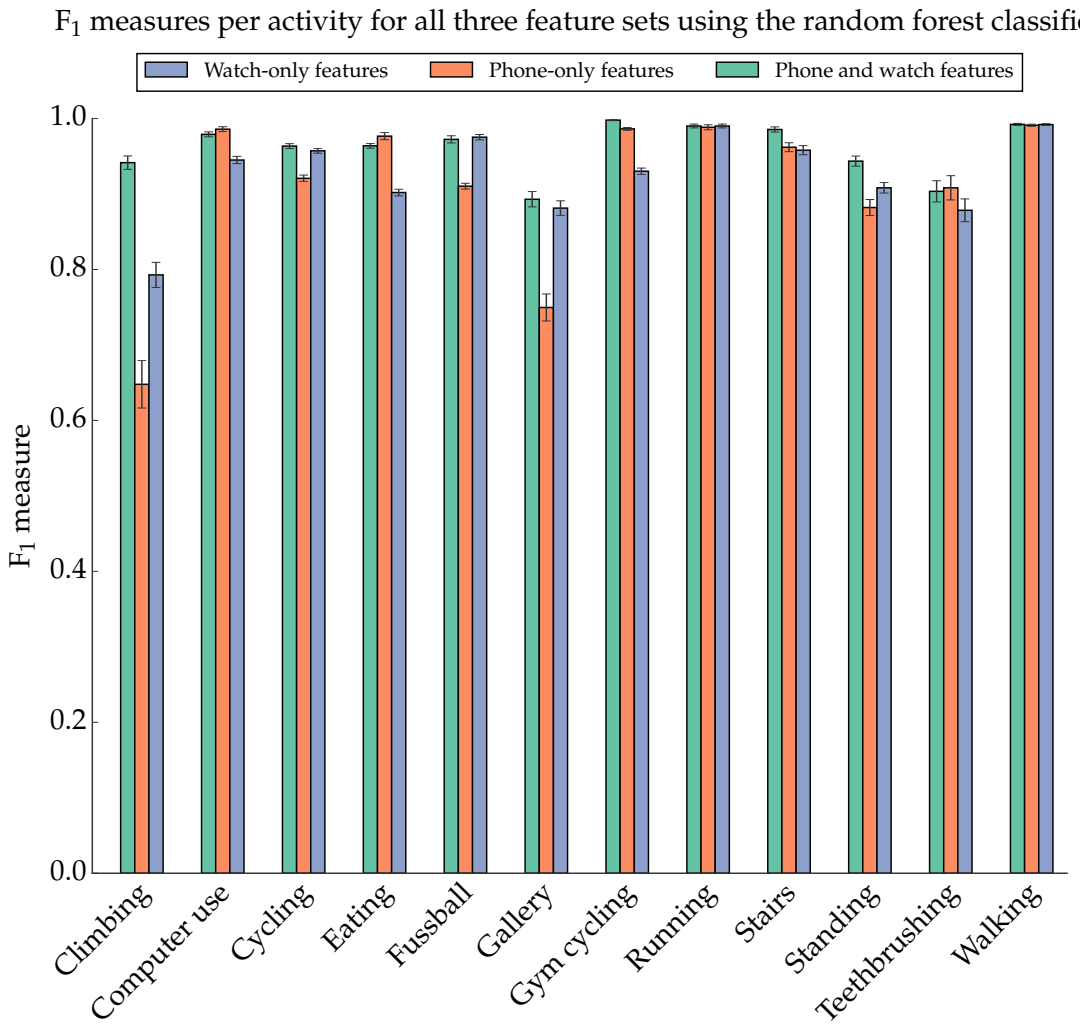


Figure 4.7: F_1 *measures for each activity using the random forest classifier trained on phone-only, wear-only and both phone and wear features.*

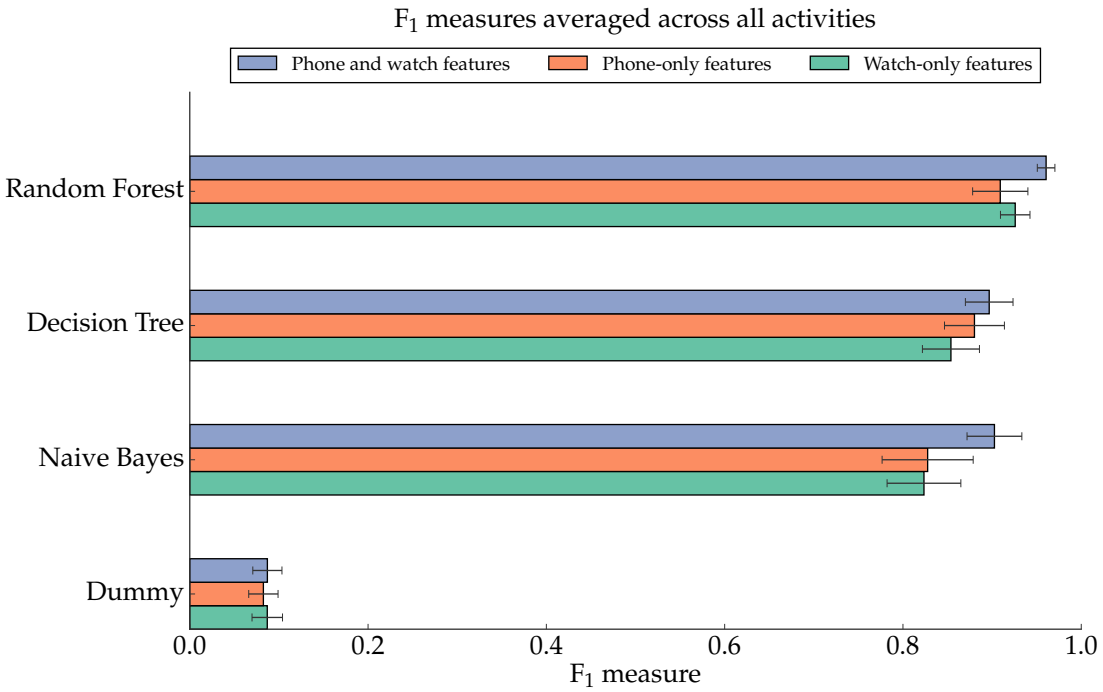


Figure 4.8: Average F_1 *measures for each activity from all classifiers, trained on phone-only, wear-only and both phone and wear features.*

4.8 Feature importances

Feature importances can be calculated when using decision trees and random forests. Feature importances, also known as Gini importance, is the normalised total reduction in impurity brought by that feature[3]. A feature that can split the whole dataset into exactly two labels would have a feature importance of 1.

In the case of classification using both phone and watch features, we would expect to see a mix of features from both devices.

Figure 4.9 presents the top five most important features from phone-only, watch-only and both phone and watch classification. The feature importances were averaged from 50 component decision trees of a random forest classifier. As expected, phone and watch features are equally important in the phone and watch classification task.

This method is also useful to evaluate feature selection. Of all features calculated, three general classes of feature stand out as being important:

1. correlation between axes;
2. spectral flatness, a measure of periodicity; and
3. peak frequency.

These three classes of feature can be collected just as easily on the watch as on the phone, and the quality of data seems to be comparable.

Figure 4.10 gives cumulative feature importances for each activity. A random forest classifier was given both phone and watch features but was trained using sets of one vs. rest binary labels. That is, for each activity χ, the labels were converted into having two possible values: χ and $\bar{\chi}$. This allows extraction of feature importances per activity.

Feature importances for some activities are understandable: phone features are more important when classifying stairs, standing and walking. Some, however, make less sense. As discussed earlier, computer use and eating also assign more importance to phone features, while cycling is the activity that assigns the most importance to watch features.

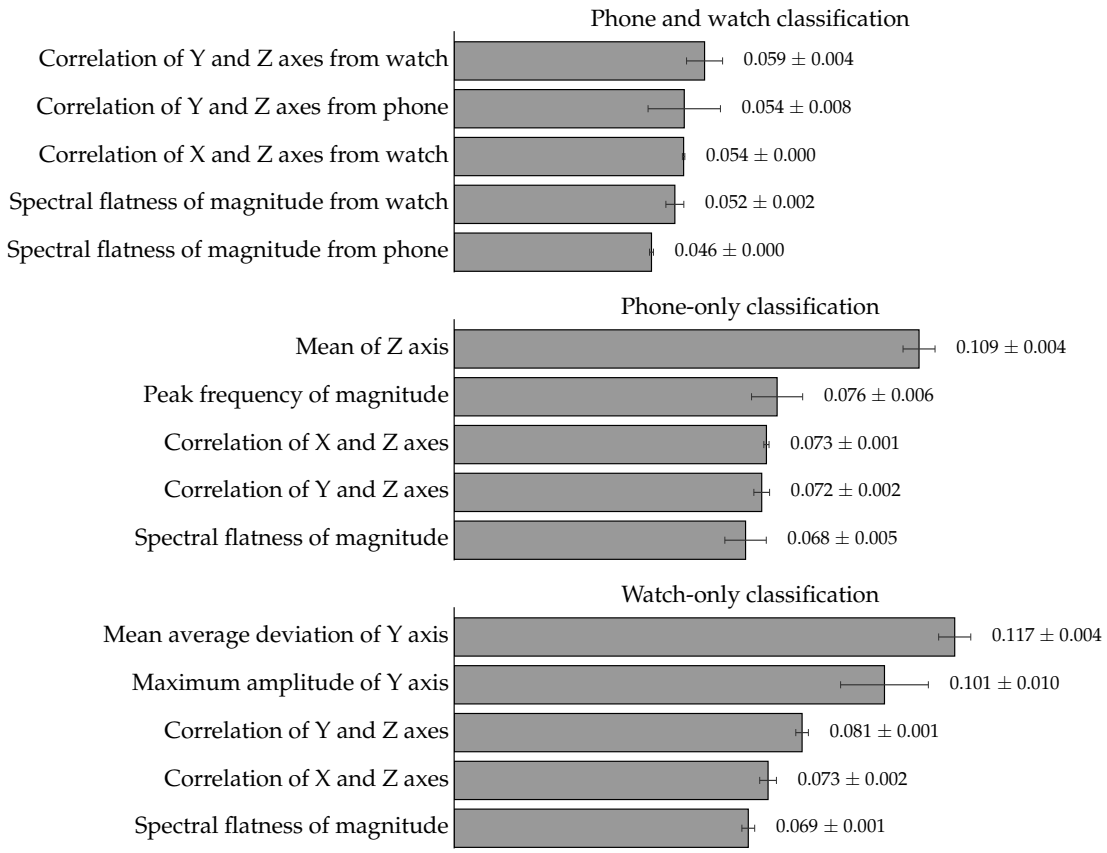


Figure 4.9: Feature importances of the top five most important features averaged over 50 decision trees in a random forest classifier trained with the three feature sets. Recall that data from the phone and the watch each produce 22 features, and so phone and watch classification has 44 features from which to pick.

One possible reason for this observation is that while phone features are good enough to place cycling into a general class of leg-periodic activities, differentiating them requires more information than is available through the phone alone. Though the most characteristic movements could come from one accelerometer, these could be shared with other activities. A second accelerometer that does not necessarily follow those characteristic movements, such as the watch while cycling, could potentially be used to nuance the results and differentiate cycling from, say, cycling in the gym.

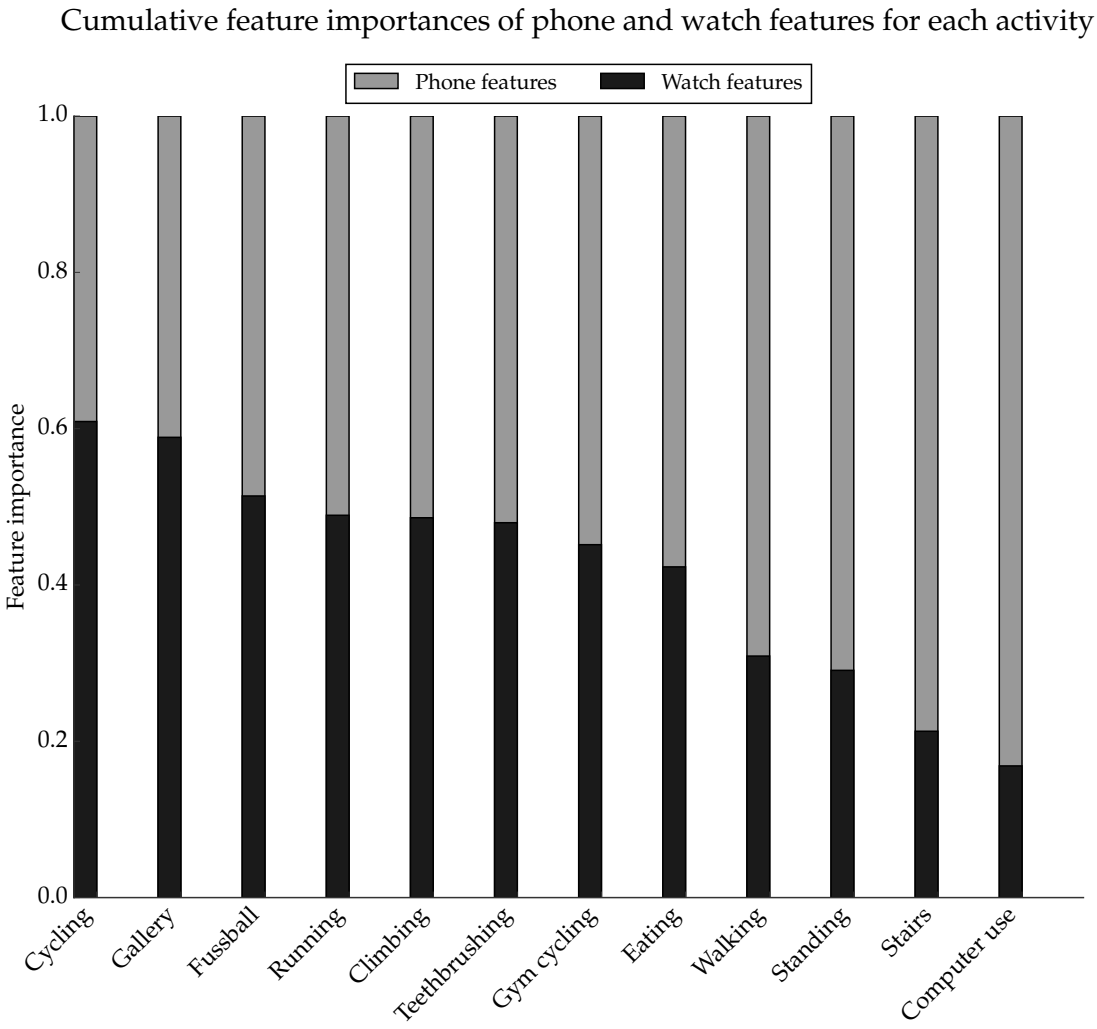


Figure 4.10: Cumulative feature importances for each activity. A random forest classifier was trained with one vs. rest labels and both phone and watch features. The importances of all the phone features and of all the watch features were totaled separately. The average total watch feature importance, ≈ 0.42 is marked as the dotted line on the graph.

4.9 Relation to original goals

The goals of the project were:

1. to classify activities based on accelerometer recordings from a consumer smartwatch and smartphone; and
2. evaluate to what extent the smartwatch is better at helping to classify activities.

The confusion matrices presented in each of the three sections show correct classification of between 80% and 90% of activities, which is between 60% and 70% better than a random baseline. Evaluation using F_1 measures show similar improvement.

The data does not show the smartwatch to be better at classifying activities on average than the smartphone. However, using data from both devices performs better than either device individually. Certain devices are better for classification of particular activities.

4.10 Summary

In this section I compare F_1 measures per activity and on average over all activities between phone-only features, watch-only features and both phone and watch features classification. I also present confusion matrices for each of these three feature sets.

In terms of F_1 measure, phone and watch classification outperforms either device individually. Individually neither device can be said to outperform the other on average, but certain devices are better for classification of particular activities. The range of F_1 measures between the three sets of classification is not large, but this is primarily because there is not much scope to improve on using either device individually: they both individually score >0.9 on the F_1 measure on average across all activities.

I also use feature importance from decision trees to evaluate phone and watch features. This method calculates that watch features have an average total im-

portance of 0.42 compared to the phone's 0.58. Some activities a total importance as high as 0.6 to watch features, while others are as low as 0.2.

Chapter 5

Conclusion

This dissertation has described the implementation and evaluation of a system to record accelerometer data from commodity consumer smartwatches and smartphones and use that data to classify what activity a user was performing.

5.1 Achievements

The project has met its two original aims: to classify activities based on accelerometer recordings from a consumer smartwatch and smartphone; and evaluate to what extent the smartwatch is better at helping to classify activities.

5.2 Lessons learnt

This dissertation has been an excellent place to implement concepts learning in Digital Signal Processing, Mobile and Sensor Systems, Information Retrieval and other Computer Science Tripos courses. I have also developed a range of technical skills that are not explicitly covered in the Computer Science Tripos.

The machine learning aspects of the project have been highly informative. I have investigated the mechanisms of some key classifiers, discovered some of

their nuances and learnt out to avoid many of their failings, such as overfitting.

The process of writing the dissertation has also been educational. In particular, writing an evaluation and producing legible, informative graphs from the volume of data available was a challenging process. Deciding what to leave out was often more challenging than deciding what to put in.

5.3 Future work

There are a number of possible directions in which one could expand on the work in this dissertation:

- A larger user study would add weight to the conclusions made in this dissertation. People may cycle or walk in different ways, are different shapes and sizes, and may carry the phone or watch in different places. Accurate classification of activities across all people that is invariant to these differences is a harder problem.
- Investigating the extent to which activities recorded from one person can classify the activities of another. A larger user study would help in this instance also. Studies in this area would make progress towards existence to 'eigenmotions', shared components of activities that are present in everyone.
- Real time classification of activities, locally or through a cloud server, could enable users to get instant feedback on what the device classifies their current activity as. This could be used to implement a reinforcement learning system, where users can say whether or not the classification is accurate, leading to better classification.
- Using additional contextual information available on smartphones and smartwatches could further help to classify activities. Some activities are only performed in certain places, and so geolocation could make classification more accurate. One could also use gyroscopic sensors or the microphone of the device.

Bibliography

[1] Louis Atallah et al. "Sensor placement for activity detection using wearable accelerometers". In: *Body Sensor Networks (BSN), 2010 International Conference on*. IEEE. 2010, pp. 24–29.

[2] Ling Bao and Stephen S Intille. "Activity recognition from user-annotated acceleration data". In: *Pervasive computing*. Springer, 2004, pp. 1–17.

[3] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[4] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984.

[5] *Documentation Enhancement: SensorEvent timestamp*. 26th Apr. 2013. URL: `https://code.google.com/p/android/issues/detail?id=7981` (visited on 28/03/2015).

[6] Google. *SensorEvent — Android Developers*. URL: `http://developer.android.com/reference/android/hardware/SensorEvent.html` (visited on 28/03/2015).

[7] Samuli Hemminki, Petteri Nurmi and Sasu Tarkoma. "Accelerometer-based transportation mode detection on smartphones". In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2013, p. 13.

[8] Slice Intelligence Jaimee Minney. *First Apple Watch data: one just isn't enough*. 12th Apr. 2015. URL: `http://intelligence.slice.com/first-apple-watch-data-one-just-isnt-enough/` (visited on 04/05/2015).

[9] Eric Jones, Travis Oliphant, Pearu Peterson et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 2015-05-04]. 2001–. URL: `http://www.scipy.org/`.

[10] Xi Long, Bin Yin and Ronald M Aarts. "Single-accelerometer-based daily physical activity classification". In: *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*. IEEE. 2009, pp. 6107–6110.

[11] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51 –56.

[12] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[13] Kiran K Rachuri et al. "Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing". In: *Proceedings of the 17th annual international conference on Mobile computing and networking*. ACM. 2011, pp. 73–84.

[14] *ResearchKit for Developers*. 23rd Mar. 2015. URL: `https://developer.apple.com/researchkit/`.

[15] *SensorEvent timestamp field incorrectly populated on Nexus 4 devices*. 13th June 2013. URL: `https://code.google.com/p/android/issues/detail?id=56561` (visited on 28/03/2015).

[16] Marina Sokolova and Guy Lapalme. "A systematic analysis of performance measures for classification tasks". In: *Information Processing & Management* 45.4 (2009), pp. 427–437.

[17] Stefan Van Der Walt, S Chris Colbert and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation". In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[18] "Wearable technology: The wear, why and how". In: *The Economist* (14th Mar. 2015). URL: `http://www.economist.com/news/business/21646225-smartwatches-and-other-wearable-devices-become-mainstream-products-will-take-more`.

Part II Project Proposal

Part II project proposal - George Thomas - gt319 - Peterhouse

Activity classification using wearable devices

Introduction, motivation and specification of the work

This project involves the expansion of existing sensor management libraries to wearable devices. These libraries will then be used to augment sensor data collected from smartphone.

The new wearable libraries will be used to answer the question “how much better is activity classification with both wearable and smartphone data in comparison to smartphone data on its own?” The definition of ‘better’ is expanded on in the evaluation section, but essentially means increased accuracy in classification and a wider range of activities able to be classified.

Tracking the motion of a wrist enables distinguishing between various activities where smartphone sensor data is not sufficient. Examples include distinguishing typing from eating.

The recent explosion in available wearable devices (Apple Watch, Moto 360 etc.) has lead to wearable sensor data becoming hot topic in mobile systems research. While it remains to be seen whether wearables will become as ubiquitous as smartphones, the wrist-mounted, sensor-rich smartwatch creates valuable potential for insight.

Resources required

- Smartwatch running Android Wear. Group is buying Samsung Gear Live and a Samsung Gear 2.
- An Android smartphone. Group is buying Nexus 5 and Samsung Galaxy S5.

Starting point

No Android development knowledge. No sensor system knowledge.
Existing sensor management library for Android[1].
DetectedActivity exists in the Android SDK[2], but is quite coarse in the activities it recognises and has no support for Android Wear.

[1] "xsenselabs/SensorManager · GitHub." 2014. 23 Oct. 2014 <https://github.com/xsenselabs/SensorManager>
[2] "DetectedActivity | Android Developers." 2013. 23 Oct. 2014 <https://developer.android.com/reference/com/google/android/gms/location/DetectedActivity.html>

Substance and structure of the project

The project has six main parts:

1. A study of working with sensing systems.
2. A study of developing with Android and Android Wear.
3. Implement an Android Wear app to collect and export sensor data.
4. Implement a classifier external to the phone and to classify activity from the app.
5. Evaluating the Android Wear activity classification against only using smartphone classification.
6. Write the dissertation.

The software engineering complexity comes in part from writing an Android Wear app to collect the sensor data, but primarily from writing a classifier for the data once it has been collected. The classifier will require machine-learning techniques using labelled sample data.

The project will therefore contain three significant pieces of software:

1. An Android phone application that collects accelerometer data from sensors from the phone. It also serves as control for the Android Wear application. This app must also be able to export the phone and watch sensor data as a CSV.
2. An Android Wear application that collects accelerometer data from the smartwatch and copies it to the smartphone.
3. A classifier running externally to the phone which will take the CSVs of data and attempt to classify them. The classifier will be trained using machine learning techniques. The classifier comprises the bulk of the software engineering effort. The classifier will not run on the phone but instead be run offline. Running the classifier on the phone would be computationally intensive and would have very little benefit to running a classifier offline. One can think of the offline model as an approximation to having the data upload to an external server (‘the cloud’) for processing.

Timeline

1. 24/10/14 – Proposal final submission.
2. 31/10/14 – Research into Android app and Android Wear app construction.
3. 06/11/14 – Research into sensor systems and classifiers.
4. 20/11/14 – Build Android Wear app to collect data.
5. 19/12/14 – Build classifier to classify Wear data.
6. 09/01/15 – Write activity report.
7. 23/01/15 – Begin evaluation. Record training data.
8. 20/02/15 – Record and classify data for evaluation.
9. 06/03/15 – Produce evaluation with graphs
10. 01/05/15 – Write dissertation and hand in by beginning of May.

Success criteria

- Android Wear app that collects sensor data.
- Classifier for Wearable data.
- Extension: train classification engine to recognise new types of activity only detectable using a smartwatch.
- Another possible extension: allow users to train their classifier to their own movements for more accurate person-specific classification.

Evaluation

The evaluation will take the form of a user study. In order to have an unbiased evaluation, any user study should take place on a variety of people and ideally these people will not have had involvement with the project beforehand. Having involvement with the project may influence the user to change their movements to make them more recognisable to the classifier.

All evaluation will require the user to wear the device and carry a smartphone for a period while it gathers data. During this data gathering period, users will have to document what activities they are doing and when. The data will be run through the classifier and its classifications compared to the actual activities to get an accuracy of classification measurement.

Labelled sample data may be available from the research group by the time I perform the evaluation, but I have assumed that it won't be and I will have to gather my own labelled sample data to train the classifier.

Other opportunities for evaluation include:

- Comparing the classification from the smartphone and from the wearable separately, and compute which is more accurate for which types of activity.
- Attempt to use both sets of data in the same classifier and compare accuracy of classification compared to smartphone sensors alone.

Possible problems

- Hardware
    - Battery life
        - Solution: we have multiple devices we can swap out, and each device will last at the very least a few hours - enough time to gather some activity data.
    - Sensors not sufficient in current Wearables
        - Sensors in Galaxy Gear 2 sufficient for sensing. Galaxy Gear Live is a newer version.
- Software
    - Developer APIs do not provide sufficient access