Cloud Centered, Smartphone Based Long-term Human Activity Recognition Solution

Lukasz Severyn Kmiecik

June 2013

Acknowledgements

I would like to thank my supervisor, Prof. Duncan Gillies for his guidance, support and invaluable help throughout this project. I would also like to thank my second marker, Prof. Moustafa Ghanem who was able to look at the project from a different perspecive and share his useful remarks. Additionally, I would like to thank Niklas Hambüchen for his patience and helpfulness in explaining answers to my questions, not only thoroughout this project, but throughout the whole course.

Abstract

This report documents the creation of a comprehensive, extremely scalable **modular** solution for quantification of daily activities through use of smartphone accelerometer. The solution is achieved through tackling problems from the areas of **signal processing** as well as **machine learning**. New, very effective signal features that are used to recognise daily activities such as walking, jogging or climbing stairs have been discovered. The solution exploits the newest achievements in the area of cloud based services, bypassing processing limits of the mobile platform by generating classifier models on the **Workstation Module** and later uploading them to the cloud. Every user of the solution is guaranteed to have most precise classifier model, as the **Mobile Module** automatically checks and downloads most up to date classifier model. As opposed to the approach of building the classifier model locally, very little data has to be transferred between the **Cloud Database** and the **Mobile Module** - there is no need to download multiple training samples for the purpose of generating the model (1000 samples = 17MB) and only classifier model has to be downloaded (complete classifier model = 0.0033MB). This allows re-generating the classifier model every day as more training data is gathered, and immediately broadcasting it to thousands of potential users. Additionally, the solution provides the user with motivation to be more active physically active by offering user-friendly **Web Module** where user, after logging in, is presented with daily activity reports, and most dedicated users are awarded through leader board system. The classification, done via custom implementations of Multivariate and Naïve Gaussian Bayes Classifier algorithms, has been thoroughly precise and ready to be used in recognition of daily activities.

Contents

Chapter 1

Introduction

1.1 Context

In 2007, with the release of Apple iPhone, the idea of smartphones as we know them today was born. In 2008, 12 million Android and Apple smartphones were sold globally. (Digithoughts 2012) Four years later, in 2012, 17 million smartphones were sold - every week. (Wilcox). This incredible growth, coupled with more and more tight integration of smartphones into our lives opens new opportunities for research into identifying daily activity patterns of smartphone users, and exploiting this knowledge to improve quality of our lives. Even before the advent of smartphones, research was done in direction of developing human activity recognition systems - often using expensive tri-axial acceleration sensors. Today, every smartphone is shipped with built in accelerometer, making such solutions available to virtually everybody.

1.2 Motivation

This project is an investigation intro creation of a long-term human activity recognition solution. Target user of the application is a person who tries to make their lifestyle more active. Through use of the accelerometer built into the smartphone, the Mobile module of the solution keeps track and identifies activities of the user like walking, running and cycling during the day, even outside specified workout sessions. As user walks or runs, readings of vertical and horizontal acceleration received from the accelerometer differ. By recording such data, application can later try to match new data to the known patterns in order to identify current activity. This allows the user to

keep track of his daily physical activity, even outside of dedicated workout sessions. This particular feature aims to motivate the user to change their habits – for example – cycle to the workplace instead of taking a bus. By saving daily activity distribution and combining it into regular reports, the problem of not seeing progress when exercising is tackled – even if progress is not apparent in users body, it is very visible when looking back at the past week or month.

At its core, the problem tackled by the project is a combination of a signal processing and a classification problem. As each activity is performed, acceleration data gathered (Sampled Data - 2.1) during the activity has a unique set of features (Feature Extraction - 2.2) that can be used for classification (Machine Learning - 2.3).

The report begins with looking at past work in the field of Human Activity Recognition, and then exploring appropriate technical terms. Section 3 describes the specific goals of the project, influenced by researching the background (4.5.5).

After goals are defined, implementation of the solution is described in detail in section 4. The implementations of classification algorithms are described, as well as the workings of mobile application and web interface.

In the end, the solution is evaluated. Methods of the evaluation, together with their outcomes are laid out in section 5. This section focuses on the performance of the solution, as well as decisions influenced by the evaluation

1.3 Contribution

State of the art has been reviewed, identifying approaches to activity recognition and design decisions that have proven to improve precision of classification in the past research. Approach shown in (Jennifer R. Kwapisz 2010) work - discovery of new methods of feature extraction has inspired new developments, discovering and successfully employing new methods which have not been used in past research. The new features have proven to be very effective in the classification process, surpassing (in terms of Information Gain) some of the traditionally used features like mean acceleration or correlations between axes of acceleration.

Additionally, a new incredibly scalable approach to gathering data has been explored, which exploits the cloud services. All of the data in the project can be gathered away from the workstation, from any place on earth with

access to the Internet. This is in stark contrast to past works in the field, which either used Bluetooth (maximum range 10 meters) or relied on manual copying of data from the sensor. This approach allows employing thousands of human subjects for data gathering process, which would be impractical in case of past research.

The cloud-centred approach to the solution allows not-seen-before scalability, with the solution ready to be deployed to thousands of potential users without any change in the architecture. As the generation of classifier model is done on a workstation instead of generating it locally on a phone, solution is not limited by the processing power of the phone or the amount of by data-transfer limits. If classifier was to be built on the phone, not only processing power would be a concern, but also size of the training library. To put things into perspective, 1000 samples take 17MB, while a complete classifier model built from these 1000 samples takes only 0.0033MB.

Quantification of daily activities has proven to be accurate, correctly identifying currently performed activity even when provided with a very small training library. It has been shown that only 192 seconds of sample recording (5.1.5) per activity are needed to achieve 99% reliability in classifications.

Chapter 2

Background

2.1 Sampled Data

2.1.1 Related Work

Accelerometer location and number

Since research into activity recognition systems has begun, many systems, which incorporate triaxial accelerometers, have been developed. Some of these investigated the use of multiple accelerometers attached at locations of subject's body: (D. Minnen 2005);(U. Maurer and Deisher 2006);(M. Ermes and Korhonen 2008). This approach provides higher precision in recognition of performed activities, however it is not feasible for long-term activity monitoring because of practical reasons like daily need of attaching and detaching multiple sensors. This limits the potential for adoption of the solution by general public, which conflicts with the aim of the project.

Use of Gyroscopes

Modern smartphones are equipped with a wide range of sensors, expanding with each new generation. State-of-the-art Samsung Galaxy S4 released in the UK in the month of May 2013 allows developers to access data from barometer and temperature sensors, opening new opportunities for innovative solutions. The sensor that started to be installed after accelerometers became popular was the gyroscope. Gyroscopes measure angular speed, which can be used to calculate rotation rate, which is how fast the device is spinning and

Figure 2.1: Minnen et al. - Sensor locations

the orientation of a body relative to a given frame of reference. As follows from the capabilities of a gyroscope, research into use of activity recognition via gyroscope measurements focused on recognition of activities that revolve around activities involving rotation of the subject, one example being the work of (Thomas Holleczek 2010). No works have been published which incorporated gyroscopes into systems aiming to recognise daily activities - this of course is motivated by questionable usefulness of gyroscope data in context of activities like walking or climbing the stairs.

Sample parameters

In work by Ravi et al. (2005) sample size is of 256, recorded at a sampling frequency of 50Hz. This means that each sample represents time slice of 5.12 seconds. As was shown, considering sample of this size was sufficient to capture cycles in activities such as walking, running, climbing up stairs etc. Additional reason stated for choosing sample of this particular size was that it being a power of 2 enabled fast computation of FFTs used for one of the features. System described in (Dominic Maguire 2009) closely follows from Ravi's (2005) work, with sample size being 256 as well, although the higher sampling frequency of 75Hz made it equal to much shorter time slice of 3.4 seconds.

2.1.2 Technical Research

Some of the considerations on selecting sampling rate are shared between this implementation and related works - specifically requirement of window size

being sufficiently long to capture natural features of data representing the activity and sample size that allows easy FFT computation.

There are however, some additional unique factors that had to be taken into account when deciding on sampling frequency. The sampling frequency could not exceed the capabilities of popular smartphones, to maximise potential for widespread adoption of the solution. Using the data from the article (Harris 2013) published on June 3rd, which compiled a list of most popular handsets in UK, research was conducted into the capabilities of accelerometers built into them. The average maximum sampling rate of built in accelerometers among phones mentioned in the article was found to be in the area of 100Hz, while even lower-mid range phones that have been released over two years ago, like for example Samsung Galaxy S feature accelerometers capable of sampling at 65Hz(Tasse 2012). Because the long-term aim of the solution is to be widely deployed, this information set a limit on sampling rate that could be considered in the implementation.

Additional consideration when designing a long-term activity monitoring system that uses a sensor built into the phone, as opposed to dedicated one has to be battery consumption. Running at maximum sampling frequency would negatively impact the battery life of the phone, therefore an optimal frequency had to be made.

2.2 Feature Extraction

Features are extracted from the raw accelerometer data samples. Choice of features to extract was inspired by an attempt on discovering new features that could prove to be more meaningful as well as past research in activity recognition area.

2.2.1 Related Work

As accelerometers began to be installed in more and more of consumer products, specifically mobile phones and MP3 players, activity recognition gained attention as a research topic.

Many of feature extraction mechanisms employed in the implementation are inspired by past work in the area. Through extensive background research, it became apparent that many papers on the topic (Nishkam Ravi 2005), (Dominic Maguire 2009) focus only on four features per axis, which are:

- **Mean [3]**
- **Standard Deviation [3]**
- **Energy [3]**
- **Correlation [3]**

Both in the paper by Ravi (2005) and the paper by Maguire (2009) the set of activities that are attempted to be recognised includes walking up and down the stairs. This makes precision in recognising climbing stairs a good indicator of how effective using this particular subset of features is, when compared to extended sets. Papers by Maguire(2009) and Ravi(2005) share not only the activity set, but also the location of the sensor on the human subject, which is particularly important if one tries to relate the works in terms of accuracy. Although end result is partially dependent on the method of classification, I have decided to focus on end accuracy evaluation on each of these papers. In the paper by Maguire it is noted that "Walking up and down stairs are shown to be to recognise with the latter only having a precision value of 0.556".

Activity	Classified As							
	Standing	Walking	Running	Stairs Up	Stairs Down	Vacuuming	Brushing	Situps
Standing	**63**	0	0	0	0	0	0	0
Walking	0	**44**	0	1	0	0	0	0
Running	0	0	**17**	16	20	0	0	0
Stairs Up	0	0	0	**9**	12	0	0	0
Stairs Down	0	0	0	19	**0**	0	0	0
Vacuuming	0	0	0	0	0	**45**	0	0
Brushing	18	0	0	0	0	15	**0**	0
Situps	0	0	0	0	0	7	0	**24**

Figure 2.2: Ravi et al. - Confusion matrix

Precision of recognising this particular activity is even more unsatisfactory when the confusion matrix 2.2 from Ravi's(2005) work is considered. As it can be seen, walking up the stairs is more often classified as walking down the stairs (42% accuracy), whereas walking downs the stairs seems not to be recognised correctly at all (0%). These unsatisfactory results pose the question of the need of introducing additional features in order to get closer to precise activity recognition. More recent research (Jennifer R. Kwapisz 2010) tackles this problem, and besides considering standard deviation and mean, it introduces four new features that are extracted from accelerometer data:

- **Average Absolute Difference [3]:** Average absolute difference between the value of each of the readings and the mean value over those values
- **Average Resultant Acceleration [1]:** average of the square roots of the sum of the values of each axis squared
- **Time Between Peaks [3]:** Time in milliseconds between peaks in the sinusoidal waves associated with most activities (for each axis)
- **Binned Distribution [30]:** Range of values for each axis is determined (maximum – minimum), divided into 10 equal sized bins, and then fraction of the values falling within each bin is recorded.

		Predicted Class					
		Walk	Jog	Up	Down	Sit	Stand
Actual Class	Walk	**1543**	5	73	60	1	1
	Jog	3	**1299**	16	3	0	0
	Up	84	24	**335**	98	2	2
	Down	108	10	136	**206**	2	3
	Sit	0	2	4	1	**268**	7
	Stand	1	0	5	4	8	**205**

Figure 2.3: Kwapisz et al. - Confusion matrix

If we consider the confusion matrix that has been produced, 2.3. we can see that both precision of recognising walking up (61.4%) and walking down (62.6%) the stairs has increased when compared to previous papers. In some cases the increase is dramatic - 62% vs. 0% in case of walking down the stairs when compared to work by Ravi(2005). This finding supports the hypothesis that there are more features than just mean, standard deviation, energy and correlation that could be extracted from accelerometer data in order to improve precision of activity recognition.

Another of more recent papers in the field (A. M. Khan 2010) has considered two other, unique features:

- **Autoregressive Coefficients**
- **Signal Magnitude Area**

Unfortunately, authors of the paper do not provide case-by-case accuracy figures on precision of the recognition, therefore it is hard to relate impact of considering the above features to previously mentioned work.

A good analysis of features that are used in activity recognition system has been published in form of work by (Mohd Fikri Azli bin Abdullah 2012), where past research has been aggregated and compared with each other to give a reflection on the current state of this academic field.

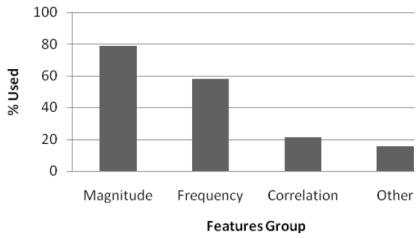

Figure 2.4: Percentage of features group used for activity recognition classification (Mohd Fikri Azli bin Abdullah 2012)

As is found by (Mohd Fikri Azli bin Abdullah 2012), nearly 80% (fig. 2.4) of works use magnitude-based features, reason being the ease of calculation, and only few researchers explore correlation as their feature for classification.

2.2.2 Technical Research

The following section focuses on theoretical background information on main mathematical notions needed to be understood to review the implementation of feature extraction.

Signal Magnitude Area

Signal magnitude area is calculated according to 2.5.

$$SMA = \sum_{i=1}^{N} (|x(i)|) + (|y(i)|) + (|z(i)|)$$

Figure 2.5:

Where x(i), y(i), and z(i) indicate the acceleration signal along x-axis , y-axis, and z-axis respectively. According to research by (A. M. Khan 2010), SMA has been found to be a suitable measure for distinguishing between static vs. dynamic activities using triaxial accelerometer signals and the fact that different activities register different SMA makes it a suitable distinguishing feature.

Fourier Transform

Fourier Transform is one of the deepest insights ever made, widely used in various scientific fields, including research focused around pattern recognition, which encompasses, among many others: speech recognition, face recognition and most importantly activity recognition.

The Fourier Transform is used in the solution in a following way: 1. A subject is asked to perform an activity for one second, as accelerometer records values of acceleration over time in three axes. 3. Values are fed into the Fourier transform method, and as a result a vector of 256 values is returned. 4. The vector of 256 values represents decomposition of the values of acceleration into 128 sine and 128 cosine components that make up the original "wave" of acceleration vs. time data.

Understanding the difference between the resulting vector of Fourier transform and the vector of accelerometer readings is crucial to reaching understanding of the Fourier transform itself. The accelerometer readings vector represents changes in acceleration over time, therefore it is in the time domain. The vector produced by the Fourier transform however, is in frequency domain, as it represents distribution of values over a range of frequencies.

As processing accelerometer readings involves discrete data, focus is put on discrete Fourier transform.

$$F[n] = \sum_{k=0}^{N-1} f[k] e^{-j\frac{2\pi}{N}nk} \quad (n = 0 : N-1)$$

Figure 2.6: F[n] is the Discrete Fourier Transform of the sequence f[k]

Since the operation treats the data as if it were periodic, we evaluate the DFT equation for the fundamental frequency and its harmonics, which is generalised into the formula 2.6.

Exemplary Discrete Fourier Transform result is presented in the figure 2.7 below, and shows particularly well the difference between time and frequency domain.

Autoregressive Coefficients

A signal is often modelled as a zero mean stationary stochastic process, and its main characteristics of interest are described by its second-order statistics:

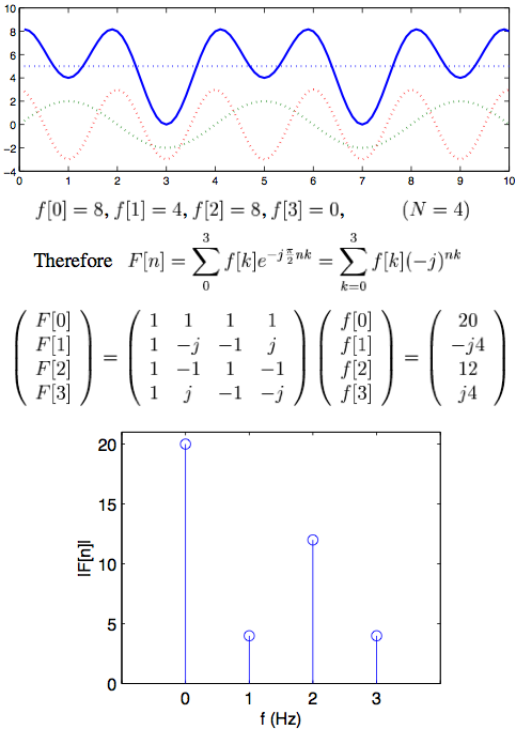

Figure 2.7: In this example, signal was sampled at 4Hz from t=0 to t=3/4. Resultant magnitude of DFT coefficients presented.

the autocorrelation function, or earlier described 2.2.2 Fourier transform.

It can be shown that an given sufficiently high N, the AR signal constructed has the same spectrum as the original signal. Hence, the AR coefficients model a signal that exerts same second-order features as original signal - i.e. is statistically "the same" as the original signal.

An AR process is described by the recursive difference equation: $an = a1xn - 1 + ... + aNxn - M + en$ where en is white noise.

Low Pass Filter

Sampled acceleration values carry information on how acceleration in three axes was changing over sampling period. However, recorded data due to the nature of the sensor is not a perfect representation of the changes, and therefore impact of noise has to be taken into account, especially for features which are particularly sensitive to it, example being **Time Between Peaks** - noise might be incorrectly identified as a peak and resulting feature value will not be reflecting the real feature of the signal. For this purpose Low Pass Filter can be employed.

Filtering describes the act of processing signal in a way that applies different levels of attenuation to different frequencies within the data. A low pass filter will apply minimal attenuation (i.e. leave altitude levels unchanged) for low frequencies, but applies maximum attenuation to high frequencies - smoothing peaks of the signal. Because movement is expressed mostly in low frequencies, this filter allows smoothing of data without significant data loss.

Number of different filtering algorithms are used, however the most popular and possibly the simplest one is the simple infinite impulse response filter implementation. This implementation works by keeping a series of samples and multiplying each of those samples by a fixed coefficient based on the position in the series. The result of each of these multiplications is accumulated and is returned as the output for that sample. The behaviour of the filter is fixed by the selection of the filter coefficients.

The equation for particular sample i in the series can be expressed as 2.8 where RC is the time constant, expressed with the use of smoothing factor α 2.9

$$y_i = \overbrace{x_i\left(\frac{\Delta_T}{RC+\Delta_T}\right)}^{\text{Input contribution}} + \overbrace{y_{i-1}\left(\frac{RC}{RC+\Delta_T}\right)}^{\text{Inertia from previous output}} .$$

Figure 2.8: Low pass filter output for sample i

$$RC = \Delta_T\left(\frac{1-\alpha}{\alpha}\right)$$

Figure 2.9: Time constant

2.3 Machine Learning and Classification

2.3.1 Related Work

The activity recognition is in essence a classification problem, therefore one of it's most important components is the classification algorithm used to classify performed activities as appropriate types. In past works, the algorithm usually is executed either on a desktop machine or smartphone. One of the concerns which has to be taken into account when choosing the algorithm is the capability of the processing platform to execute the algorithm. Performance of the classification algorithm is measured via an evaluation method.

Through research of (Mohd Fikri Azli bin Abdullah 2012) in, which they have aggregated and surveyed classification algorithms used in past solutions, it becomes apparent that a small set of algorithms dominates the field.

"According to the survey, the most popular algorithms are Decision Trees, k-Nearest Neighbour, Naïve Bayes, Support Vector Machine and Neural Network." (Mohd Fikri Azli bin Abdullah 2012)

Interesting point is made in the work (Mohd Fikri Azli bin Abdullah 2012) which points out method of bypassing the limitations of processing power of mobile phones, by generating the classification model on the workstation and uploading it to the phone, so that the only computation phone is required to carry out is the actual classification of the sample. This approach is taken in the works of by (Poovandran 2008) and (S.-I. Yang 2008).

2.3.2 Technical Research

Naïve Gaussian Bayes Classifier

The Naïve Gaussian Bayes Classifier is a classification algorithm based on Bayes rule, that assumes the attributes $X_1 \ldots X_n$* are all conditionally independent of one another, given Y. This assumption greatly simplifies the representation of $P(\boldsymbol{X}/Y)$, and the problem of estimating it from the training data.

Consider an example where we are trying to classify sample having only two features, X_1 and X_n, and are checking confidence of it being of type Y. 2.10

$$\begin{aligned} P(X|Y) &= P(X_1, X_2|Y) \\ &= P(X_1|X_2, Y)P(X_2|Y) \\ &= P(X_1|Y)P(X_2|Y) \end{aligned}$$

Figure 2.10:

Second line follows from a general property of probabilities, and the third line follows directly from the definition of conditional independence. As number of features increases, this can be generalised to the following formula:2.11

$$P(X_1 \ldots X_n|Y) = \prod_{i=1}^{n} P(X_i|Y)$$

Figure 2.11: Gaussian distribution

The values of multiplicands of the form $P(X_i/P)$ for features of continuous nature can be expressed through use of Gaussian probability density function 2.13 if we assume that data follows normal distribution. 2.12.

$$P_{X|Y}(x \mid i) = G(x, \mu_i, \sigma_i)$$

Figure 2.12: Gaussianity of continuous data is assumed

The Naïve Gaussian Classifier picks the prediction that maximises 2.11

Attribute Weighted Naïve Gaussian Bayes Classifier

Weighted Naïve Bayesian classifier relaxes the conditional independence assumption to increase accuracy. Based on Information Gain calculation,

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Figure 2.13: Gaussian probability density function

importance of each feature is analysed and appropriate weight is applied. Features of Information Gain ratio below 1.0 are considered not crucial to the goal of precise classification and are discarded. The impact of weight k on probability calculation can be expressed as 2.14

$$p(x) = k\ \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Figure 2.14: Gaussian probability density function

Multivariate Gaussian Bayes Classifier

The "Naïve" part of the Naïve Gaussian Bayes Classifier algorithm comes from a fact that it ignores relationships between attributes, assuming each attribute is independent and contributes equally to the final hypothesis probability.

The Multivariate Gaussian Bayesian Classification overcomes this drawback by employing the Multivariate Normal Distribution at its heart, with probability density function of 2.15

$$\begin{aligned} p(\mathbf{x}) &= \left(\frac{1}{\sqrt{2\pi}}\right)^d \frac{1}{|\mathbf{C}|^{1/2}}\ e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})} \\ &= (2\pi)^{-d/2}\,|\mathbf{C}|^{-1/2}\ e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})} \end{aligned}$$

Figure 2.15: Multivariate Normal Distribution probability density function

Therefore the class-conditional probabilities are: 4.14

Where:

- $\boldsymbol{x}$ is the vector of feature values of considered sample
- m is the vector of means of features across samples of same type i
- d is the dimension of the covariance matrix $\mathbf{C}$

$$p(\mathbf{x}|\omega_i) \quad = \quad (2\pi)^{-d/2} \, |\mathbf{C}_i|^{-1/2} \quad e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T \mathbf{C}_i^{-1}(\mathbf{x}-\mathbf{m}_i)}$$

Figure 2.16: class-conditional probability for class i function

- C is the covariance matrix between features of the sample of type i. The dimension of the matrix is equal to the number of considered features.

The exponent of the pdf function(4.14) is worth investigating as it is the key component of deciding on the classification. When rewritten to an equivalent form seen in 2.17

$$\|\mathbf{x} - \mathbf{m}_i\|^2_{\mathbf{C}^{-1}}$$

Figure 2.17: exponent

The exponent appears as a squared distance, but the inverse of covariance matrix C acts like a metric (stretching factor) on the space. This is the Mahalonobis distance, which is a descriptive statistic that provides a relative measure of a data point's distance from a common point. Mahalonobis distance takes into account the correlations of the data set, and additionally is scale-invariant. Seeing the importance of the exponent it can be concluded that classification done via the means of multivariate normal distributions is simply a minimum Mahalonobis distance classifier.

2.4 Development Platform Overview

Galaxy Samsung S3 is UK's one of the UK's most popular handsets, surpassing even the hugely popular Apple iPhone 4S and Apple iPhone 5. (Johnson). With over 40 million Galaxy S3 devices sold, developing for this platform allows the solution to reach wide audience. Development for the Android platform is done using extremely versatile and popular language – Java. Familiar language and environment – Eclipse, allowed smooth learning of Android specific syntax. Samsung Galaxy S3 is not only one of the most popular handsets in the UK, it is also one of the most advanced ones. Beside accelerometer sensor, which is crucial to the project, this handset offers extremely powerful, quad-core processor clocked at 1.4GHz, coupled with 1GB of RAM memory. This hardware is of great use for hardware-accelerated calculations. The solution has been developed with Android 4.0+ devices

in mind, with Samsung Galaxy S3 or equivalent being the recommended device. Reasons for this particular choice are: popularity of the device, power, familiarity and quality of built in sensors.

Chapter Summary:

First section (2.1) of the chapter concerned itself with practical issues surrounding recording of accelerometer data. We have looked at number of sensors and chosen recording parameters such as sampling rate or sample size in researchers in the area. Question of gyroscope use was also explored - no works have been published which incorporated gyroscopes in recognising daily activities because of its questionable usefulness in such scenario. In the Evaluation (5) section, this is confirmed to be true, and therefore gyroscope readings are not used in the final solution.

Research was made into capabilities of the most popular smartphones - although solution is developed primarily for Samsung Galaxy S3, it is worth choosing such sample parameters that will allow it to be ran on a wide range of smartphones.

Second section (2.2) of the chapter looks at the signal-processing component of the problem - we look at effectiveness of using different feature sets in past works in the area. It is noted that more extensive feature set can improve the precision of feature extraction. The Technical Research part of the section discusses the theoretical background behind more complex notions used in the implementation of signal feature extraction methods used in this solution - Signal Magnitude area, Discrete Fourier Transform and Autoregressive Coefficients

Section three (2.3), concerns itself with machine learning aspect of the problem. We look at algorithms used for classifying daily activities in past research. Some of the most popular and effective algorithms - Naive and Multivariate Gaussian Bayes Classifiers are described in detail in the Technical Research part.

In the final (2.4) section we introduce the development platform used in the solution - Android operating system and Samsung Galaxy S3 smartphone.

Chapter 3

Specification

3.1 Data Storage

One of the most fundamental choices that had to be made was on how data will be stored and moved around the system. All of the related works have employed either the traditional model of storing samples locally or transmitting them to the workstation via WiFi or Bluetooth. These approaches, while simple to implement are not scalable. As this solution aims for mass adoption, scalability is a major concern.

Research has been undertaken into the direction of storing the data on the workstation, allowing incoming Internet connections from the mobile application to the workstation, enabling sending the samples over the Internet, bypassing drawbacks of WiFi and Bluetooth which requires the subject to be close to the workstation.

The choice of server-hosted database enables potential participation of thousands of users submitting training data from any location in the world, which leads to more robust classification mechanism.

However, having a workstation acting as a server requires it to be constantly on as well as constantly on-line - these requirements could not be guaranteed by equipment at hand, a home workstation, therefore research has been directed into booming area of cloud services. The requirements, which the provider had to satisfy, were as follows:

- Compatibility with Android devices
- Constant uptime

- Free price-plan

In the end the choice has been narrowed down to **mongolab.com**(MongoLab.com) and **iriscouch.com** (irisCouch.com). Both service providers offer free price plans and access via REST services which is convenient for Android mobile applications. However as **iriscouch.com** imposes a read/write number limit on free accounts, **mongolab.com** has been chosen as the service provider.

The cloud database is intended to be used for the following purposes:

- Storage of training samples
- Storage of the classification model
- Storage of individual classification results
- User database for the web module of the solution.

3.2 Sampled Data

This implementation focuses on single accelerometer activity recognition system with main reason being wide adaption of accelerometer-equipped smartphones, and therefore high potential for adaption of the solution by the public. Works of (Nishkam Ravi 2005),(Dominic Maguire 2009), (A. M. Khan 2010) and (Jennifer R. Kwapisz 2010) explored the idea of single accelerometer system.

Both in Ravi (2005) and Kwapisz (2010) system purpose-built accelerometer is attached in pelvic region of the subject. In case of this implementation, this particular sensor location is mimicked by placing the phone in trouser pocket of the subject.

The system in Kwapisz's (2010) work follows from the other two papers, with main differences being the type of sensor used - built into the smartphone vs. purpose-built and exact location - trouser pocket vs. attached to body. These design choices closely line up with this implementation, and follow the considerations of the project.

Each recorded sample consists of 256 raw values of acceleration on x, y and z axes and is recorded via the mobile application, using the accelerometer built into the phone. Samples are recorded with constant sampling frequency of 40Hz that makes each sample represent time slice of 6.4 seconds. The

frequency was chosen based on researching capabilities of popular smartphones on the market. Frequency of 40Hz is only 40% of maximum sampling rate of Galaxy S3.

Similarly to previous works in the field, the implementation follows the pattern of choosing sample size of power of two. This choice of course is dictated by allowing fast computation of FFTs, which are used for some features. Samples are manually labelled with appropriate label corresponding to the activity sample represents. Option exists to switch to the twice-size mode, where each sample will consist of vectors of size 512, which in turn represents 12.8 second time slices.

3.3 Activity Set

The solution will build up on past research (Jennifer R. Kwapisz 2010), (Nishkam Ravi 2005), aiming to improve on recognition precision of following activities:

- **Walking**
- **Fast Walking**
- **Jogging**
- **Walking up the stairs**
- **Walking down the stairs**
- **Sitting**
- **Standing**

Additionally as mentioned in research by (Jennifer R. Kwapisz 2010) - "We plan to improve our activity recognition in several ways. The straightforward improvements involve: 1) learning to recognize additional activities, such as bicycling", the solution will include **Cycling** in the Activity Set.

3.4 Signal Feature Extraction

This implementation follows the direction taken by (Jennifer R. Kwapisz 2010) work in trying to discover new relevant distinguishing features of accelerometer signal in order to increase precision of activity recognition but includes the tried-and-tested features employed in works of (Dominic Maguire 2009) and (Nishkam Ravi 2005) as well. The considered feature set consists of all types

of features pointed out in (Mohd Fikri Azli bin Abdullah 2012) i.e. frequency based features, magnitude based features, correlation based and finally custom features like for example "Average Peak distance" feature.

3.4.1 Feature Set

Features of the sample considered in this implementation are as follows:

- **Mean [3]** - average acceleration on each axis
- **Standard Deviation [3]** - standard deviation of acceleration on each axis
- **Resultant Acceleration [1]** - square root across sum of squared values of three axes
- **Energy [3]** - sum of the squared discrete FFT component magnitudes of the signal. See fig 3.1

$$Energy = \frac{\sum_{i=1}^{|w|} |x_i|^2}{|w|}$$

Figure 3.1: Energy formula

- **Correlation [3]** - three values of correlation: x,y; y,z; x,z. See fig 3.2

$$corr(x, y) = \frac{cov(x,y)}{\sigma_x \sigma_y}$$

Figure 3.2: Correlation formula

- **Signal Magnitude Area [3]** - SMA is calculated for each axis

$$SMA = \sum_{i=1}^{N} (|x(i)|) + (|y(i)|) + (|z(i)|)$$

Figure 3.3: SMA formula

- **Maximum Displacement** [3] - Difference between maximum and minimum value of acceleration on each axis is calculated

- **Average Peak Distance** [3] - Distance between peaks in recorded acceleration values is calculated and average for each axis. To prevent noise and outlier values from affecting the result, signal will be passed through low pass filter before extracting this feature.

- **Acceleration histogram (binned distribution)** [3][10] - An array of 10 values is calculated for each axis, representing the distribution of acceleration values. Histogram for each axis has different range, to maximise relevance of data. Unified range for all values would not be an effective feature as it would result in clusters of values dependent on which axis the distribution represents. Therefore ranges were chosen based on observation of values that acceleration was taking on each axis.
  As x-axis acceleration values were found to be bound in range of -5 m/s^2 to 5 m/s for all of the measured activities, this range was chosen for binned distribution of x values. Similarly for y-axis, range was discovered to be between 5 m/s^2 and 15m/s^2, and this range was chosen. For z-axis, range was selected to be between -8 m/s^2 and 2m/s^2.

- **FFT histogram (binned distribution)** [3][10] - Similarly to the previously described feature, three histograms are generated, one for each axis. Approach to selecting range differs from the previously described feature however, as resultant vector of FFT transformation is in frequency domain, instead of time, which allows us to use uniform range for each axis when binning data. This feature is unique to this implementation and was not considered in past works.

- **Relative crossing count** [3] - crossing count reflects frequency of periodic changes in acceleration associated with measured activities. This feature has not been considered in past works. It is calculated in the following way:

  1. 80th percentile and 20th percentile of data in analysed sequence is retrieved.
  2. If absolute difference between values is lower than set parameter, data is considered to be consisting mainly of noise, and therefore feature calculation is abandoned and the count is set to zero.
  3. Otherwise, "zero" value is set to the mean between 80th and 20th percentile values.

4. Iteration through data is performed, with step set to a parameter, checking if currently considered value has fell below or above the "zero" value, when compared to previously considered value. If that was the case, count is increased.

- **AR Coefficients** [3][3] - Autoregressive Coefficients are computed such that the corresponding AR signal will be of same second order statistics (same spectrum) as original signal.

3.5 Workstation module

The main role of the workstation module of the solution is to provide an interface between the cloud database and classification model generation. The workstation module loads training samples from the cloud database, performs batch feature extraction process and feeds feature values into appropriate classification algorithms. The workstation module also provides functionality of uploading generated modules back into the cloud database, as well as performing classification on samples. The design of the Workstation Module should allow plug-in nature of the classifier algorithms.

3.5.1 Weak

The Weka workbench is a collection of state-of-the-art visualization tools and algorithms for data analysis and predictive modelling, combined with graphical user interface enabling intuitive access to its functionality. One of design considerations of the workstation module of the application is the possibility of easy transfer of list of training samples from the custom 'Accent' format to the Weka `.arff` file format. This is to allow analysis of effectiveness of extracted features in classifying data.

3.5.2 Analysis

Additional feature of the workstation module is capability of evaluating the classification models and presenting the user with appropriate statistics both in terms of percentage of correct classifications as well as in form of confusion matrices.

3.6 Machine Learning and Classification

An approach identified in the work of (Mohd Fikri Azli bin Abdullah 2012) - generating the classification model on the workstation and later uploading it to the phone is chosen in this implementation. This approach combines the advantages of workstation-based classification with convenience of mobile-based. It's main advantage is being very light on mobile processor but at the same time not dependent on the constant availability of the workstation for classification.

The classification model will be generated on the workstation and then uploaded to the cloud database. The mobile application will check for a new model upon start-up, and if such exists, it will replace the locally cached model with the new one. The classification of the sample will be done based on downloaded and cached model. This minimizes data transfer between mobile module and the cloud database, as the only occasion when such has to happen is upon model update.

3.6.1 Classification Algorithms

Naïve Gaussian Bayes Classifier

The Naïve Gaussian Bayes Classifier is one self-implemented classification algorithms in the solution. Described in detail in the **Technical Research** 2.3.2 section, it is a simple yet very popular (described and employed in multiple activity recognition research solutions ((Lau and David 2010), (L. Sun 2011), (T. S. Saponas and Landay), (Yang 2009))) and effective classification algorithm - "In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers."(Wikipedia)

The classification model of this particular classification algorithm is light on data, consisting of $2n$ vectors for n classified types, of magnitude equal to the number of features used in the classification. This is advantageous to the solution, as less data transfer translates to lesser battery consumption, lesser signal requirements as well as smaller data usage charges to the user.

Multivariate Gaussian Bayes Classifier

While Naïve Gaussian Bayes Classifier 2.3.2 has numerous technical advantages to the solution, its main flaw is its "Naivety" - assumption that all features

exerted by the sample are independent from each other. This is why another custom implementation of a classification algorithm is considered in the solution - the Multivariate Gaussian Bayes Classifier. The Multivariate Gaussian Classifier takes into account relationships between feature values through use of covariance matrix. The workings of the algorithm are explored in detail in **Technical Research** 2.3.2 section on the algorithm.

From point of view of viability to the solution, the Algorithm is expected to offer more precise results, but at a cost of:

- Requiring more training data - Multivariate Classifier requires at least as many samples per class as the number of considered features, however it is a bare minimum and usually much more data is needed to allow for accurate results. As (Kozmann G 1991) noted about use of Multivariate Classifier: "(...) it is noted that in 30% of the cases only one, and in 6% of the cases only two parameters could be used for statistical group representation to ensure a reasonable reliability. In 56% of the published cases the sample sizes could not guarantee this reliability even for one feature or parameter."
- Requiring more storage, hence more data needs to be transferred when updating the model. This is caused by the need of storing n covariance matrices of order d where n is the number of activity types and d is the number of features considered in the classification.
- Requiring more computation on the mobile end of the solution - specifically performing multiple matrix transformation operations when executing the probability density function. 2.15

3.7 Mobile Module

3.7.1 Activity Recording

The mobile module of the solution implements methods for recording data samples through use of accelerometer built into the phone. Additionally, it implements functionality for connection the cloud database in order to facilitate easy addition of new training samples.

One of the main requirements of mobile module is to provide a clear interface for previewing the recorded samples as well as easy to use sample-labelling interface.

3.7.2 Activity Recognition

The mobile module retrieves the classification model from the cloud database and caches it locally. The classification can be performed on a single sample from the local library or alternatively by constant classification through use of a sliding window with half-sample size overlap. The results of the classification are sent to the cloud database for **Web Module** 3.8 use. User is to be provided with both textual and graphical representation of probability distribution for each classification. The classification results are cached locally and can be browsed.

3.8 Web Module

The web module role is to provide motivation to the user to perform more physical activities every day. The web module uses classification results from the cloud database to provide the user with graphical and textual breakdown of their daily activity. This is to motivate user to improve their habits and also see how they progress over time, stimulating sense of achievement.

The additional motivational feature exists in form of a daily "leaderboard" displaying first names and photos of users who have spent most time on specific physical activities like running, climbing stairs etc. in the previous day. The day-wide window is chosen to make appearance in the leaderboard possible for every user, which would not have been the case if data from a week was aggregated instead from a day. In case of a weekly window, users who work full time would be disadvantaged against users who have more free time during the week. This makes appearance in the leaderboard possible for everyone who has at least one free day from work.

Additionally the web application provides proof-of-concept functionality of tracking the current activity performed by the mobile module user in real time.

Chapter Summary:

This chapter concerns itself with specific design issues.

First section (3.1) of the chapter explains the decision of going through with clod-based approach to storing data in the solution.

Different providers are compared and decision is made to use mongolab.com as cloud service provider.

After reviewing how accelerometer data was sampled in past works in the area in the previous chapter (2.1, section Sampled Data (3.2) explains how data will be sampled in this solution. Choices are influenced by previous work as well as requirements of feature extraction methods. Sample size is chosen to be of 256 readings, sampled at 40Hz. The sampling frequency was chosen basing on research from section 2.1.

Short section 3.3 specifies which activities the solution will aim to recognise.

Section 2.2 introduces a complete list of features that will be extracted from signal, including new features unique to this solution - the relative crossing count and FFT histogram.

In section 3.5 the goals of the workstation module are specified, them being: acting as an interface between cloud database and classifier model generation, allowing analysis of data with Weka suite and performing evaluation procedures of implemented algorithms.

The Multivariate and Naïve Classifiers are compared from a technical point of view in section 3.6. Concerns are raised about Multivariate variant requiring much more training data than Naïve one.

Two final sections 3.7 and 3.8 describe the design goals of the Web and Mobile modules of the solution, mainly from the point of view of usability, as technical concepts employed in them are described in earlier sections of the chapter.

Chapter 4

Implementation

The solution consists of three modules, centred around the cloud database.

- **Mobile Module**
- **Workstation Module**
- **Web Module**

Implementation of which is described in detail in this section. Full-page diagram 4.1 is ***extremely*** crucial to reaching understanding of the solution.

4.1 Data Flow

Three modules of the solution are interconnected by the **Cloud Database**. The data flows in the solution in a following way:

1. **Mobile Module** records training samples and uploads them to the **Cloud Database**
2. **Workstation Module** downloads training samples from the **Cloud Database**
3. **Workstation Module** builds the classifier model using training samples and uploads it to the **Cloud Database**
4. **Mobile Module** downloads the classifier model from the **Cloud Database**
5. **Mobile Module** performs activity recognition using the downloaded classifier model and uploads the results to the **Cloud Database**

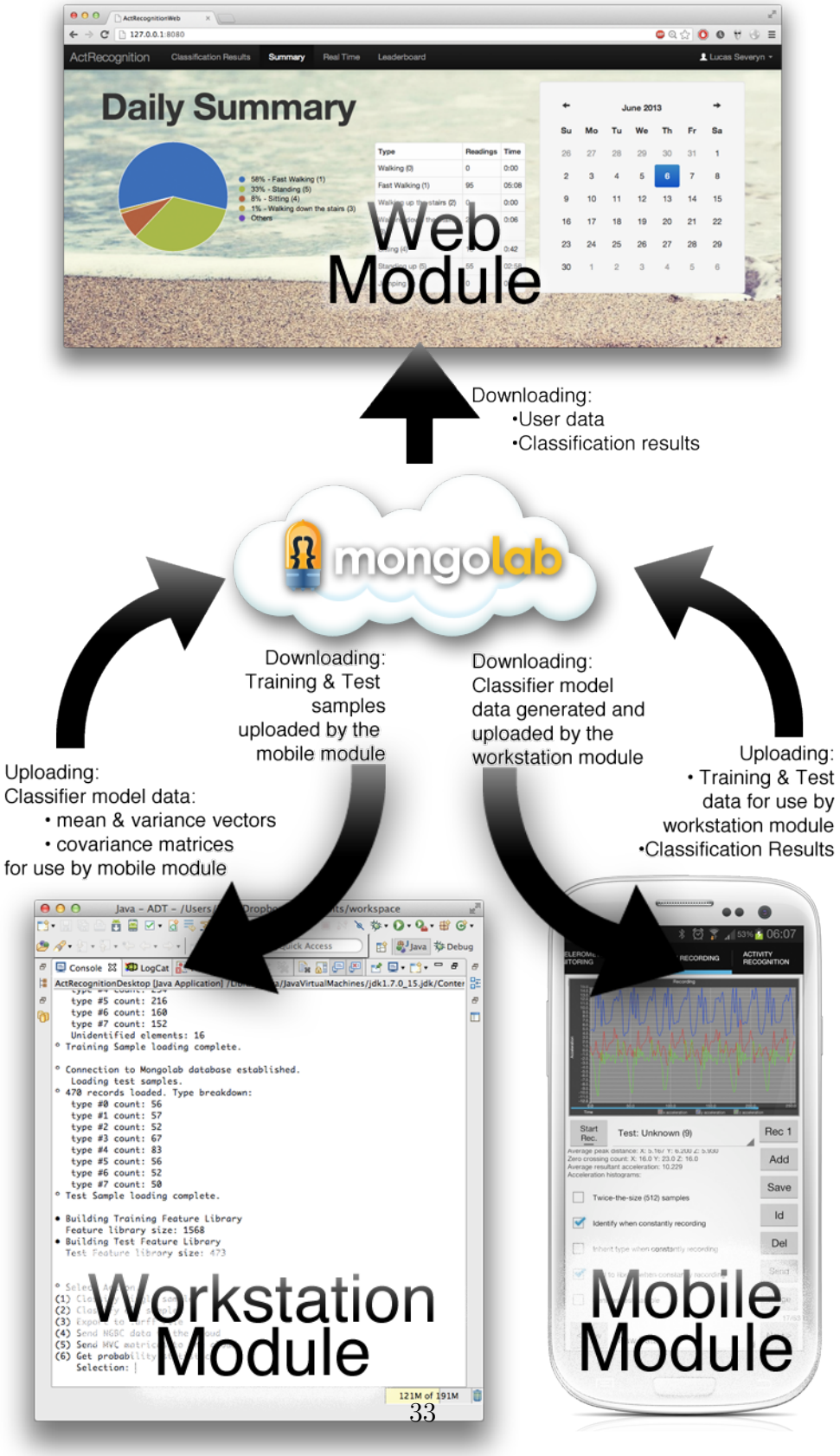

Figure 4.1: Data flow in the solution. Centre: cloud Mongo lab database

6. **Web Module** downloads classification results from the **Cloud Database** to present the user with activity reports, leader boards, etc.

Data flow in the solution is presented on a full-page diagram 4.1.

4.2 Web Module

Web module of the solution has been developed in JavaScript, incorporating angularjs - an open-source JavaScript framework, that assists with developing single-page applications. Angularjs allows clear separation of view from controller - i.e. helps to implement model–view–controller (MVC) design pattern.

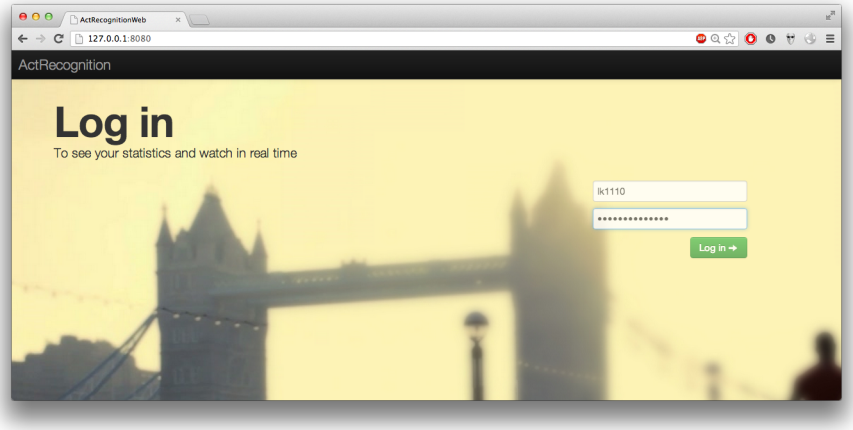

Figure 4.2: Web Module login screen

4.2.1 Database Connection

Communication with the cloud database is done through REST API. As the database uses JSON based storage, there is no need for any kind of conversion in order to use data in the module. It is worth noting that all of the data that has to be loaded into the web application is lightweight, as the only data that has to be downloaded is a numerical label representing the result of classification and the time-stamp of the classification.

4.2.2 Daily Summary

When a date is selected from the calendar date-picker, function 'update-DateFunction' is triggered in the controller which in turn triggers function 'calculateDailyTypeDistribution' and modifies variable representing currently selected date.

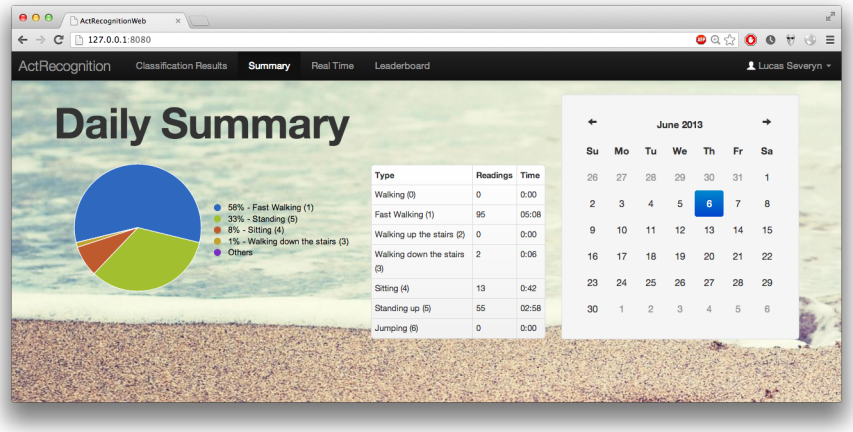

Figure 4.3: Daily Summary screen

'calculateDailyTypeDistribution' changes the contents of 'typeDistribution' array. Directive watching the 'typeDistribution' gets triggered by this change of contents and proceeds to plot a pie chart representing distribution of activities during the day using graphael charting library. Additionally, because of two-way binding capabilities of the angularjs framework, as soon as the 'typeDistribution' changes, the values representing time spent on each activity presented in the table change as well.

4.2.3 Real Time display

Real time display mechanism relies on loading the newest classification result for the logged in user, together with time-stamp. If the time-stamp is older than chosen cut-off of 1 minute, it is assumed that there is currently no broadcast and database is not queried again. However, if such result is found, the database is queried for a new result every 3 seconds.

The implementation of the `loadRealTimeResult` function presented below shows how REST API is employed in retrieving results from the cloud database.

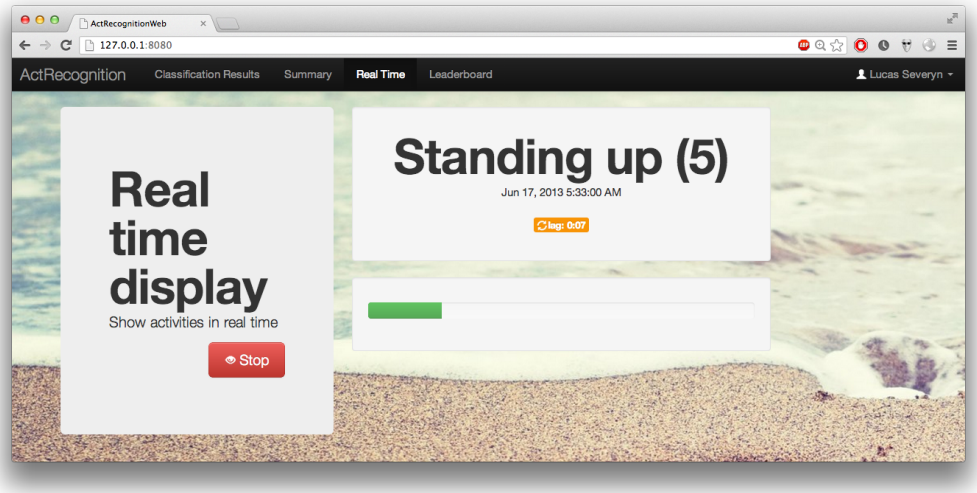

Figure 4.4: Real Time display

```
$scope.loadRealTimeResult = function(){
  $scope.realTimeTickCounter=$scope.realTimeTickCounter+1;
  var value = $scope.realTimeTickCounter;
  $scope.dynamic = (value);
  if(($scope.realTimeTickCounter % 100)===0){
    var url= "https://api.mongolab.com/api/1/databases/activity_recognition/coll
    $http.get(url).success(
      function(data, status, headers, config) {
          var newest = data[0];
          var ageinms=Math.abs(new Date() - dates.convert(newest.date));

          if(ageinms<60000){ //1 minute cutoff
            $scope.realTimeResultAge=$scope.msToTime(ageinms);
            $scope.realTimeResult=$scope.typeToString(newest.result);
            $scope.realTimeTimestamp=newest.date;
          }else{
            $scope.realTimeResult="No broadcast detected";
            $scope.realTimeResultAge="no broadcast";
            $scope.toggleRealTime();
          }
      }
      );
    $scope.realTimeTickCounter=0;
```

```
  }
  mytimeout = $timeout($scope.loadRealTimeResult,30);
};
```

4.2.4 Leaderboard

The leader board shows first names and photos of users who have spent the most time in the past day on a particular activity. The data is aggregated as follows.

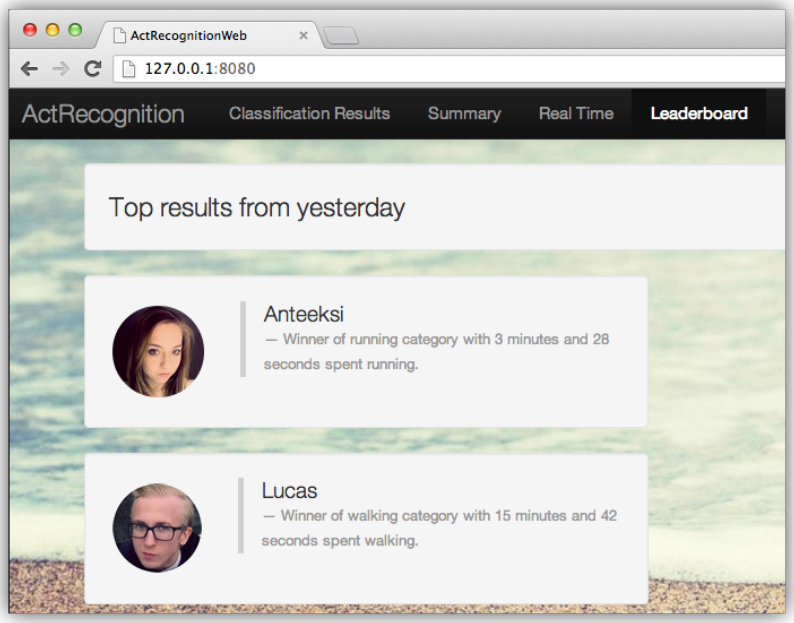

Figure 4.5: Leaderboard view

When user selects the "Leaderboard" tab of the web application, check is performed if top users have already been identified, in case they are not, list of all users' ids is loaded. For each user REST request is made that returns a single number representing count of activities of a particular type that were performed by the user. This approach is extremely efficient in terms of data transfer as there is no need to download whole list of classifications for the user as all of the filtering is done through the properly formatted URI request.

This operation is done for every registered user. Data is aggregated into an array, which is then in turn sorted in descending order, and the top user is selected.

4.3 Mobile Module

The mobile module of the solution has been developed for Android OS, using Java in Eclipse programming environment.

4.3.1 Design

4.3.2 Database connection

Mobile application employs two-way connection to the cloud database. The connection is faciliated through REST API. Data between the mobile application and cloud database is transferred in following cases:

- Training sample being sent to the database
- Classification result bein sent to the database
- Classifier model loaded from the database

In all cases data has to be converted to and from JSON representation. The conversion is done through Google Gson library.

4.3.3 Accelerometer Monitoring

A simple **Accelerometer Monitoring** tab has been implemented for the purpose of visually investigating workings of the accelerometer as well as inspecting the signal for unexpected results. The Visual representation can be toggled to free up computational resources of the smartphone.

4.3.4 Activity Recording

The application allows easy recording of training samples.Each of recorded samples can be reviewed both visually through a graph plot and feature-wise through display of feature extraction 2.2 results.

Once recording starts, a progressbar appears indicating progress in recording of the current sample.

The sampling rate can be easily altered through variable `samplingRate` which then is converted to `sensorDelayMicroseconds` - the format Android uses for it's accelerometer handler.

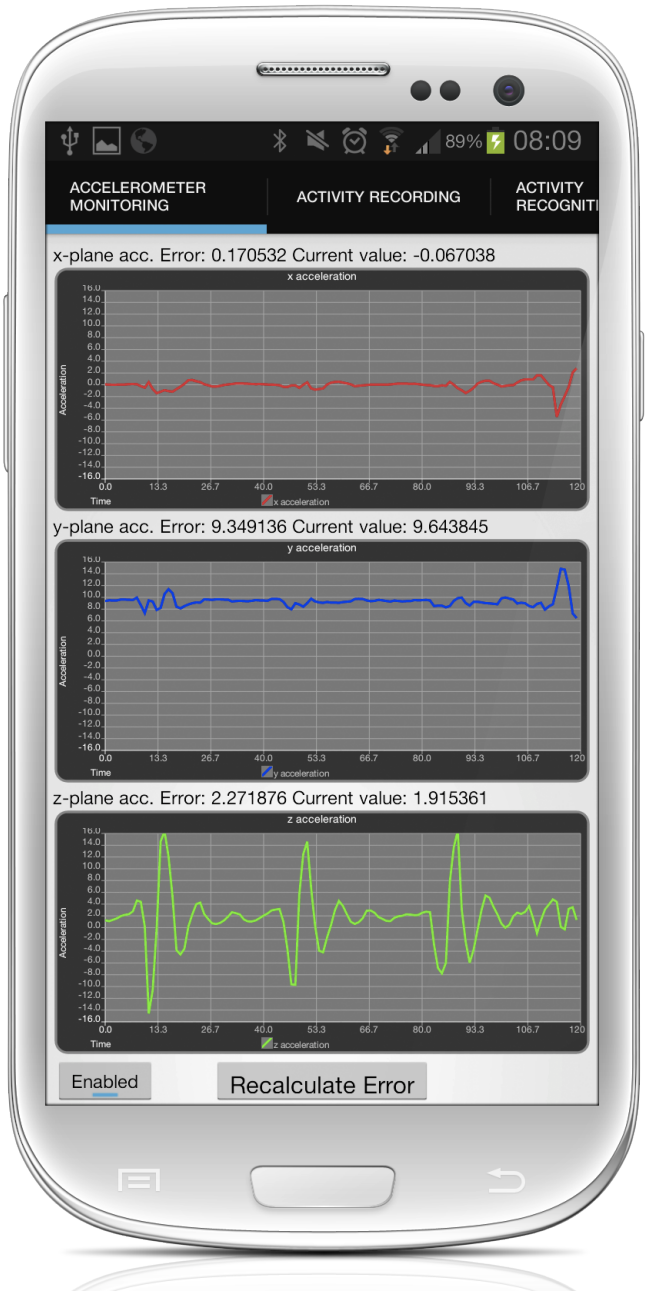

Figure 4.6: Accelerometer Monitoring Tab

```
private double samplingRate = 40; // Hz
int sensorDelayMicroseconds = (int) (Math
    .round(((1 / this.samplingRate) * 1000000.0)));
```

Data Structure - AccData.java

Samples are stored in a form of custom AccData objects of following structure:

```
public class AccData implements Serializable {
  private List<Double> xData;
  private List<Double> yData;
  private List<Double> zData;
  private int type;
```

The three List objects hold the acceleration values, while the type field holds the numerical label for the recording. Appropriate setter/getter methods are implemented.

Recording modes

There are two ways of recording samples:

- Single sample recording (button **Rec 1**) gives the user 5 seconds to put the phone into the pocket, signals that recording will start soon by a vibration alert and then after further 0.5 seconds (so that vibration does not affect recorded acceleration) recording starts. End of recording is signaled by a vibration as well. This recording mode is designed with training sample recording in mind.
- Constant recording (button **Start Rec.**) mode will constantly record samples until it is disabled. This mode is used both in training sample recording stage and recognition stage.

There are several options that can be enabled when recording:

1. **Save to library when constantly recording** - when this option is enabled, each recorded sample will be added to the local library automatically.

Figure 4.7: Activity Recording Tab

2. **Inherit type when recording** - once user sets type label from a drop-down menu, each next recorded activity will inherit that label. This option is especially useful when an opportunity presents itself to record many samples of same activity in a row. Example could be recording of samples by enabling the constant recording mode, together with **Add to library when constantly recoding** and **Inherit type when recording** after choosing **Sitting** [**4**] label from drop-down menu and putting the phone into the pocket for the duration of the lecture. Each recorded sample will be added to library automatically.

3. **Identify when constantly recording** - will aim classify current activity every 3.2 seconds

4. **Twice-the-size (512) samples** - twice as long samples will be recorded. This also changes the frequency of classification to every 6.4 seconds.

Local Storage

After recording a single sample it can be added to the local library using the **Add** button. When **Add to library when constantly recording** option is enabled samples will be added to the local library automatically. Pressing **Save** button will serialize and save the library to phone local storage in form of *.dat file, so that upon restart of the application data from the local library will not perish. The library can be purged by pressing the **Purge** button three times.

Global Storage

Each sample in the library can be sent to the cloud database with a press of **Send** button. Mobile application will convert the currently selected AccData object into JSON format using Google Gson library and after establishing connection with the cloud database, the object will be sent.

Data is stored in JSON format of the following structure (only 3 out of 256 values for each axis shown for clarity)


```
{
    "_id": {
        "$oid": "51a75df0e4b07f1fa5cc0388"
    },
    "zData": [
```

```
            -8.973467826843262,
            -8.96389102935791,
            -8.935160636901855
            ...
        ],
        "xData": [
            3.5625722408294678,
            3.533841609954834,
            3.5625722408294678,
            ...
        ],
        "yData": [
            1.455674648284912,
            1.5610195398330688,
            1.50355863571167
            ...
        ],
        "type": 4
}
```


4.3.5 Activity Recognition

The Activity Recognition tab is used to display output of the classification algorithm used in the solution.

When **Id** button is pressed in the **Activity Recording** tab, currently selected sample from the local library is classified. If classifier model is not downloaded, application will connect to the cloud database and download appropriate classifier model data (mean and variance vectors for Naïve Gaussian Classifier, additional covariance matrices for Multivariate Gaussian Classifier).

If (in **Activity Recording** tab) **Constant Recording** mode is enabled with **Identify when constantly recording** option set to on, classification will be performed through use of a sliding window with 50% overlap. The window size depends on the chosen sample size. If option **Twice-the-size (512) samples** is disabled, window will consider samples of size 256 (6.4 seconds @40Hz) with 50% overlap, otherwise window of size 512 will be considered.

If the **Send results to cloud** option is enabled, each classification will be sent to the cloud database immediately after classifying, otherwise **Send** button needs to be pressed in order to send the result.

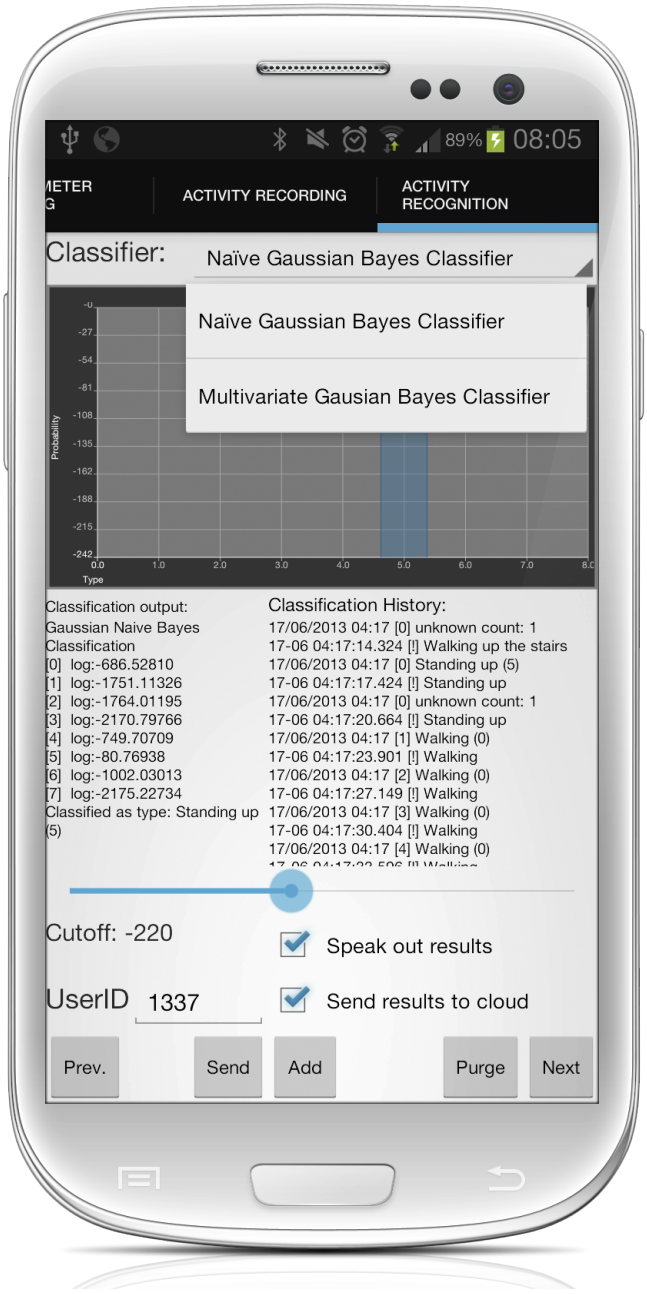

Figure 4.8: Activity Recognition Tab with classifier choice dialog open.

The **Speak out results** option has been developed for evaluation purposes - when enabled, through use of text-to-speech library the result will be read out to the user. This is particularly useful when evaluating the performance of classification in real life situations and is explored in depth in **Evaluation** section of the report.

The **Cutoff** seek bar is used to decide on the cutoff probability value under which classification results will be considered "unknown".

The **Classifier** drop-down menu allows choosing between different classifier algorithms. The classifier that is currently chosen will be used every time classification will be attempted, be it via **Identify when constantly recording** or by pressing the **Id** button in the Recording Tab.

Data Structure - ClassificationResult.java

Classification results are stored in a form of custom ClassificationResult objects of following structure:

```
public class ClassificationResult {
  List<Double> p;
  Date date;
  Integer result;
```

The 'List p' holds the probability distribution for the particular classification result, 'Date date' holds the timestamp on which the classification took place and finally 'Integer result' hold the numerical label representing result of the classification.

The timestamp stored together with the classification result allows local storage of the classification results and submission of them later, without risk that later submission will mean that results get assigned to a different date. This is particularly useful if the user is in low reception area and currently cannot synchronise his results with the database.

The probability distribution is displayed via the means of the graph, allowing intuitive insight into confidence of the classifier.

Local Storage

After classifying a single sample it can be added to the local classification result library using the **Add** button. When **Classify constantly recording**

option is enabled classification results will be added to the local result library automatically. Pressing **Save** button will serialize and save the result library to the phone's local storage in form of *.dat file, so that upon restart of the application data from the local results library will not perish. The library can be purged by pressing the **Purge** button three times.

The classification results can be reviewed and browsed using **Prev** and **Next** buttons.

Global Storage

Each classification result can be sent to the could database with a press of **Send** button. Otherwise, if **Send results to cloud** option is enabled, each classification will be sent to the cloud immediately after classification takes place. When sending, application will convert the currently selected ClassificationResult object into JSON format using Google Gson [@gson] library and after establishing connection with the cloud database, the object will be sent.

Data is stored in JSON format of the following structure


```
{
"_id": {
    "$oid": "51a8d522e4b082fc259fae55"
},
"date": "May 31, 2013 5:51:45 PM",
"p": [
    -848.5727069759712,
    -280.8045515994242,
    -30.86754679639256,
    -75.21763730225379,
    0,
    0
],
"result": 2,
"userid": 1337
}
```

4.4 Feature Extraction

Feature extraction utility class `FeatureExtractors.java` 4.4.1 and `AccFeat.java` 4.4.1 object that stores values of features are used both in the Mobile and Workstation modules of the solution.

4.4.1 AccFeat.java

Values of extracted features of a particular sample are are stored in a form of custom AccFeat object. Each of the fields of AccFeat object corresponds to a unique feature of the considered sample, described in **Feature Set** section above. Structure of AccFeat is as follows:

```
public class AccFeat {
    double[] mean = new double[3];
    double[] sd = new double[3];
    double[] avPeakDistance = new double[3];
    double[] maxDisplacement = new double[3];
    int[][] histogram = new int[3][10];
    int[][] fftHistogram = new int[3][10];
    double[][] AR = new double[3][4];
    int[] crossingCount = new int[3];
    double[] energy = new double[3];
    double[] correlation = new double[3];
    double resultantAcc;
    double SMA;
    int type;
```

Each of features is numerically indexed and it's value can be retrieved via that number, which allows picking of features in the classification algorithms described in sections and .

FeatureExtractors.java

The AccFeat object is built through use of a custom FeatureExtractors class, which consists of self coded methods of calculation of each of mentioned features.

Calling `buildFeatureObject(AccData)` takes previously described (4.3.4) AccData object containing recordings of acceleration values along x, y and z

axes as an argument and after performing Feature Extraction process returns an appropriate AccFeat representation of the sample.

All of the Feature Extraction methods are sample-size independent, which allows for the 256 and 512 sample size recording modes to coexist.

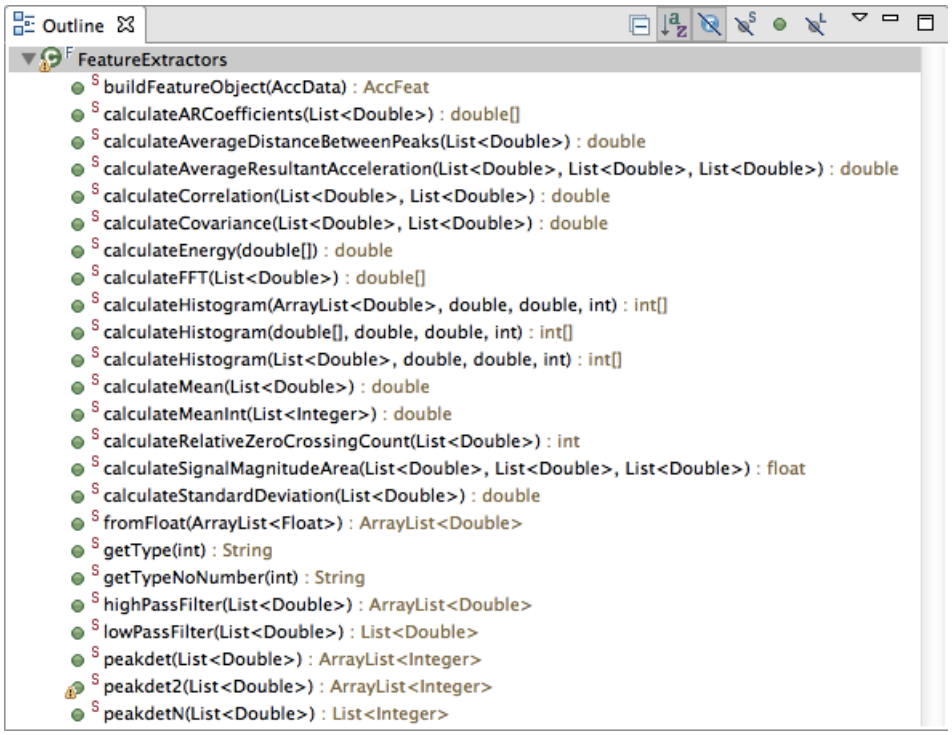

Figure 4.9: FeatureExtractors class outline

4.5 Workstation Module

Workstation application has been developed in order to facilitate creation of classification model for accelerometer data.

Through careful coding, classes:

- **AccData.java** - used for storing raw accelerometer data (4.3.4)
- **AccFeat.java** - used for storing the feature signature of the signal (4.4.1)
- **FeatureExtractors.java** - used to build `AccFeat` from `AccData` through a collection of custom feature extraction methods (4.4.1)

can and are shared between Workstation and Mobile modules of the solution without need for any changes in the code.

The workstation application loads the training samples from the cloud, converting JSON representation into into `AccData` objects and inserting into `List<AccData> accDataLibrary` object.

Batch feature extraction process is performed on the `accDataLibrary`, creating counterpart `accFeatLibrary` which then is passed into classifier objects.

4.5.1 Weka Interface

`WekaFileGenerator` class has been written. It's goal is to provide interface between workstation module of the solution and Weka suite. This is facilitated through `generateFile(List<AccFeat>)` method, which accepts a list of objects representing features extracted from raw accelerometer data samples and returns the '.arff' file, complete with all of the custom feature names, ready to be analysed by Weka.

4.5.2 Classifier design

Both Multivariate and Naïve Gaussian Bayes Classifiers have similar structure, which allowed designing a common interface which they both implement. The UML diagram can be seen in figure 4.11.

4.5.3 Naïve Gaussian Bayes Classifier

Custom Naïve Gaussian Bayes Classifier 2.3.2 implementation was written in a format allowing sharing of the code in unchanged form between mobile and workstation modules of the solution.

The structure of the class allows easy incorporation of outcomes of Weka-based analysis. The classifier will only consider features included in the `attr` array, and will assign weights to features as outlined in the `weights` array. 4.12

Generating classification model - μ and σ vectors

If the class is instantiated with a set of μ vectors and σ vectors, as is the case when the classifier is used on a mobile platform, there is no need to generate

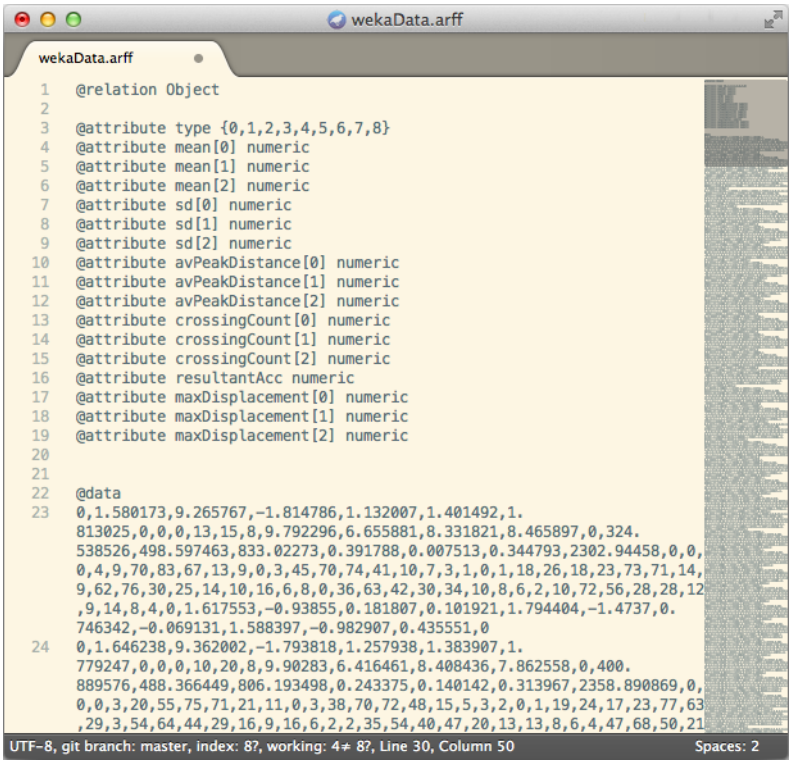

Figure 4.10: Weka file generated by the WekaFileGenerator class from a list of `AccFeat` objects (only 17 out of 96 numerical values shown for clarity)

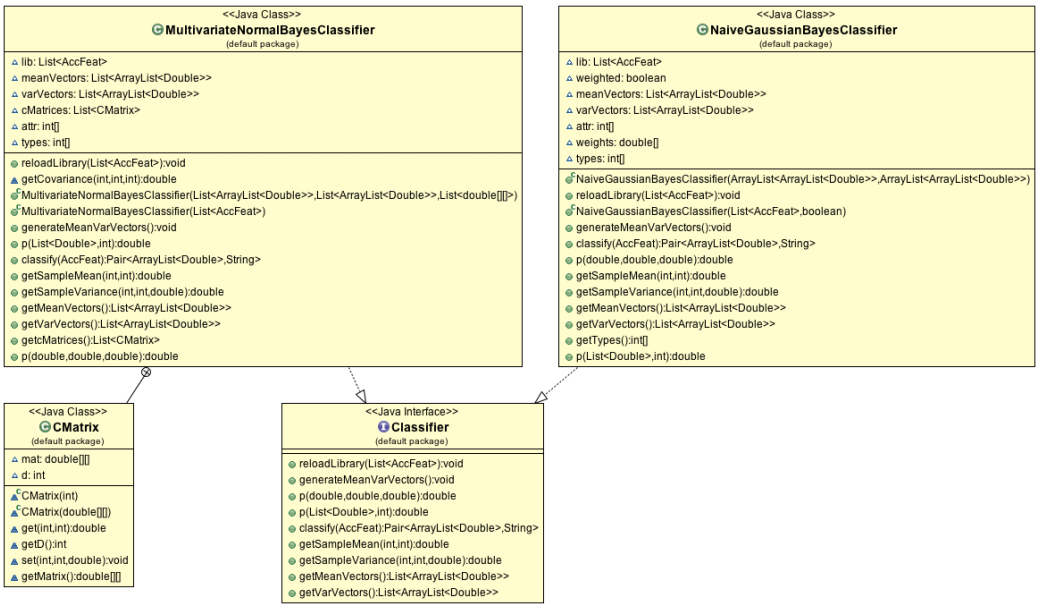

Figure 4.11: Classifier algorithm implementations

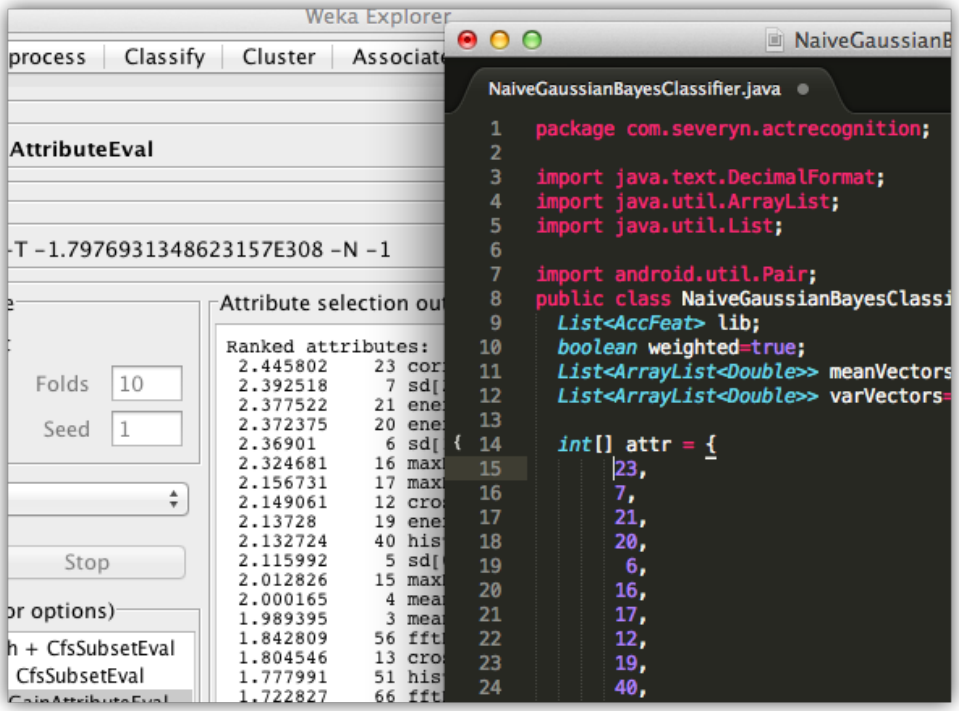

Figure 4.12: Notice how results of attribute ranking can be copied directly to the `attr` array

a classification model and this section is skipped.

If the class is instantiated with a library of AccFeat object as an argument, the classifier will initialise eight μ vectors, which will be filled with mean values of each considered feature for each activity type label. Vectors are of `ArrayList<Double>` type and are put into `List<ArrayList<Double>> meanVectors` object holding all eight vectors.

Initialisation of eight σ vectors which will hold variance for each of calculated attributes follows afterwards. Vectors are of `ArrayList<Double>` type and are put into `List<ArrayList<Double>> varVectors` object holding all eight vectors.

The process of filling vectors with values then ensues, using the following method:

```
public void generateMeanVarVectors() {
  for (int i = 0; i < types.length; i++) {
    for (int k = 0; k < attr.length; k++) {
      double mean = getSampleMean(attr[k], types[i]);
      meanVectors.get(types[i]).add(mean);
      double var = getSampleVariance(attr[k], types[i], mean);
      varVectors.get(types[i]).add(var);
    }
  }
}

 double getSampleMean(int feature, int type) {
     double sum = 0;
     int count = 0;
     for (AccFeat a : lib) {
         if (a.getType() == type) {
             sum += a.getFeature(feature);
             count++;
         }
     }
     return sum / count;
 }

 double getSampleVariance(int feature, int type, double mean) {
     double sum = 0;
     int count = 0;
```

```
        for (AccFeat a : lib) {
            if (a.getType() == type) {
                sum += Math.pow((a.getFeature(feature) - mean), 2);
                count++;
            }
        }
        return sum / count;
    }
```

This results in vectors being filled with appropriate mean and variance values that can be then used in the classification process.

Classification

While aggregating of data and building classification model is intended to be unique to the workstation module, classification is shared between both workstation and mobile modules.

The classification method receives the AccFeat object as an argument, then builds a vector `qf` of features to be considered in the classification.

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Figure 4.13: Probability function used in Naïve Bayes Classifier

Implementation of the probability function 4.13:

```
private double p(double v, double m, double var) {
    double p = ((1 / (Math.sqrt(2 * Math.PI * var))
    * Math.exp(-(Math.pow(v
            - m, 2))
            / (2 * var))));
    return p;
}
```

The calculation of probabilities for each feature value follows from the description of Naïve Bayes Classifier in the **Technical Research** 2.3.2 section of this chapter. Additionally individual feature weights are taken into account if `weights` array is defined, turning the classifier into **Weighted Naïve Bayes Classifier**

```
public void classify(AccFeat q) {
    double[] results = new double[8];
    List<Double> qf = new ArrayList<>(); //Query Feature vector
    for (int k = 0; k < attr.length; k++) {
      qf.add(q.getFeature(attr[k])); //Only considered features are added
    }
    double weight;
    for (int i = 0; i < types.length; i++) {
      result = 0;
      for (int j = 0; j < meanVectors.get(types[i]).size(); j++){
        if (weighted)
          weight = weights[j];
        else {
          weight = 1;
        }
        result += Math.log(weight
          * p(qf.get(j), meanVectors.get(types[i]).get(j),
              varVectors.get(types[i]).get(j)));
    }
    results[i] = result;
   }
```

Probability of sample being of particular type n is stored at index n of the `results` array.

4.5.4 Multivariate Gaussian Bayes Classifier

Beside the Naïve Gaussian Bayes Classifier implementation 5.1, additionally a Multivariate Gaussian Bayes Classifier was implemented, similarly in a format allowing sharing of the code in unchanged form between mobile and workstation modules of the solution.

The structure of the class allows easy incorporation of outcomes of Weka-based analysis. The classifier will only consider features included in the `attr` array. 4.12

Generating classification model - μ and σ vectors & Covariance Matrix

If the class is instantiated with a set of μ vectors and σ vectors, as is the case when the classifier is used on a mobile platform, this section is skipped.

Otherwise μ and σ vectors filling process is conducted, similarly as in Naïve Gaussian Bayes Classification implementation 4.5.3

In addition to the μ and σ vectors, n covariance matrices are generated for n types to classify.

Custom 'CMatrix' class was written, that is instantiated with type that it refers to and fills itself with appropriate covariance values.

```
class CMatrix {
  double[][] mat;
  int d;
  CMatrix(int type) {
    this.d = attr.length;
    mat = new double[d][d];
    for (int i = 0; i < attr.length; i++) {
      for (int j = 0; j < attr.length; j++) {
        double covariance = getCovariance(attr[i], attr[j], type);
        this.set(i, j, covariance);
      }
    }
  }

  double get(int i, int j) {
    return mat[i][j];
  }

  int getD() {
    return d;
  }

  void set(int i, int j, double x) {
    mat[i][j] = x;
  }

  double[][] getMatrix() {
    return mat;
  }

}
```

The 'getCovariance' method makes use of μ and σ vectors in order to calculate

covariance between attribute values. The format of vectors follows from Naïve Gaussian Bayes Classification implementation 4.5.3

```
double getCovariance(int featureX, int featureY, int type) {
  double meanX = meanVectors.get(type).get(featureX);
  double meanY = varVectors.get(type).get(featureY);
  double sum = 0;
  int count = 0;
  for (AccFeat a : lib) {
    if (a.getType() == type) {
      sum += (a.getFeature(featureX) - meanX)
      * (a.getFeature(featureY) - meanY);
      count++;
    }
  }
  double result = sum / (count - 1);
  return result;

}
```

As soon as covariance matrices are generated, classification can take place.

Classification

Similarly to the the Naïve Bayes Classifier implementation, classification method receives the AccFeat object as an argument and then builds a vector `qf` of features to be considered in the classification.

$$p(\mathbf{x}|\omega_i) = (2\pi)^{-d/2} |\mathbf{C}_i|^{-1/2} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T \mathbf{C}_i^{-1}(\mathbf{x}-\mathbf{m}_i)}$$

Figure 4.14: Class-conditional probability for class i function

Implementation of the probability function 4.14:

```
private double p(List<Double> qf, int type) {
    Matrix coeff = new Matrix(cMatrices.get(type).getMatrix());
    Matrix coeff_inv = coeff.inverse();

    double result = Math.pow((2 * Math.PI), -coeff.getRowDimension() / 2);
    result *= Math.pow(coeff.det(), -0.5);
```

```
    double[][] xminusmt = new double[1][attr.length];
    double[][] xminusm = new double[attr.length][1];
    double difference;
    for (int i = 0; i < (attr.length); i++) {
      difference = qf.get(i) - meanVectors.get(type).get(i);
      xminusmt[0][i] = difference;
      xminusm[i][0] = difference;
    }
    Matrix x_minus_m = new Matrix(xminusm);
    Matrix x_minus_m_T = new Matrix(xminusmt);

    Matrix half_times_x_minus_m_T = x_minus_m_T.times(-0.5);

    Matrix x_minus_m_T_times_C_inv = half_times_x_minus_m_T
        .times(coeff_inv);
    double norml1 = x_minus_m_T_times_C_inv.times(x_minus_m).norm1();
    double n = Math.exp(norml1);
    result *= n;

    return result;
}
```

The calculation of probabilities for each feature value follows from the description of Multivariate Bayes Classifier in the **Technical Research** 2.3.2 section of this chapter.

```
public void classify(AccFeat q) {
  double[] results = new double[9];
  List<Double> qf = new ArrayList<Double>();

  for (int j = 0; j < attr.length; j++) {
    qf.add(q.getFeature(attr[j]));
  }

  for (int i = 0; i < 9; i++) {
      results[i] = p(qf, i);
  }
```

Probability of sample being of particular type n is stored at index n of the `results` array.

4.5.5 Uploading Classifier Models

Using methods 'sendMeanVarVectors' and 'sendMvcMatrices' the Workstation module extracts μ and σ vectors as well as covariance matrices in case of the Multivariate implementation from the classifiers, converts them to JSON format and uploads to the cloud database.

Chapter Summary:

This chapter concerns talks in detail about Java and JavaScript implementation of the solution. It opens with a very important diagram (4.1) on how data is exchanged between different modules of the solution. The data flow is additionally described in first section of the chapter.

Web Module implementation is introduced in section 4.2. Details of retrieving and aggregating activity classification data through REST API and JSON data representation across the module are presented, together with appropriate UI screenshots.

The section that follows () focuses on how the Mobile Module implementation of the solution. First part of the section describes how accelerometer data is recorded and its parameters are being set. The user interface is introduced, describing different recording modes. It is shown how the data is stored and finally how it is being sent to the cloud database. The second part describes implementation of Activity Recognition engine on the mobile phone. When identification is attempted, application will check if classifier model is downloaded, and if it is not, it will download appropriate data from the classifier model - μ and σ vectors for the Naïve Gaussian Bayes Classifier and additional covariance matrices for Multivariate. Each of classification results can be sent to the cloud database, with appropriate userID and timestamp.

Section 4.4 begins with a description of decoupled approach of storing acceleration signal data and signal features separately. Custom FeatureExtractors class implementing feature extraction methods described in previous chapters is introduced together with the outline 4.9

Final section describes the Workstation Module and how design goals outlined in the previous chapter (3.5) have been achieved.

Implementation of Naïve and Multivariate Gaussian Bayes Classifiers is described in detail, following from technical research in chapter . Generation and uploading of classifier models is explained.

Chapter 5

Evaluation

The aim of the evaluation stage is to judge solution performance through a series of experiments as well as influence final implementation decisions.

5.1 Experiment 1: Algorithm Evaluation

With both Naïve Gaussian Bayes Classifier 2.3.2 and Multivariate Gaussian Bayes Classifier 2.3.2 implemented (4.5.4 &) it has to be decided which algorithm will be employed in the solution.

One of the major concerns is the smallest viable training library size for each of the classification models. Preferably the training library size requirement should be kept to a minimum, to minimise the setup time required for to use the solution by the end user - and this is the **criteria** that will be used in deciding on which algorithm will be used in the solution.

5.1.1 Background

Algorithms

Through use of the **Workstation Module** 3.5 and its ability to work with different classification algorithms the Naïve Gaussian Bayes Classifier will be compared to the Multivariate variety. The Naïve Gaussian Bayes Classifier will be tested in two forms - first one being the "vanilla" classifier, which assumes equal weights to every feature that is extracted, while the other one will be of **Weighted** variety, with weight values obtained through analysis

with **Weka**,3.5.1 made possible via **Weka Interface** 4.5.1 which is part of the Workstation Module.

Sample Recording

50 samples of each activity will be recorded. Where it is possible to perform activity continously for the duration of 50 samples i.e. 5 minutes and 20 seconds (50 samples of 6.4 seconds) the **Constant Recording** option will be used 4.3.4. Activities in following 8 categories will be considered:

- **Sitting**
- **Standing**
- **Cycling**
- **Jogging**
- **Walking**
- **Fast Walking**
- **Walking up the stairs**
- **Walking down the stairs**

Phone will be kept in the left pocket of the trousers when recording each of the test samples.

Classification

Test samples will be uploaded to the cloud database and classified using the **Workstation Module** with appropriate Classification Algorithm implementations plugged in.

Each of classifier algorithms which will build their classification model using a training library of 150 samples per activity class, loading the training samples from the cloud database.

Naïve Bayes Classifier (2.3.2) This variety of the Naïve Bayes Classifier will use full set of features and consider each feature to be independent of each other and of equal importance.

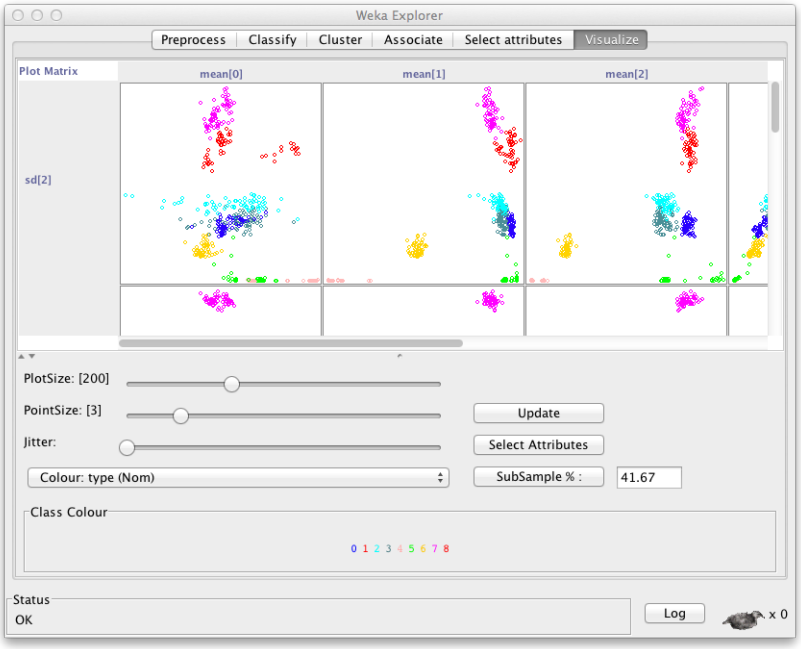

Figure 5.1: Clustering of activity types visualised by feature in Weka.

Weighted Naïve Bayes Classifier (2.3.2) Through use of **(3) Export to .arff file** function in the workstation module, all of training data is exported to Weka-compatible arff format.

The "weighted" variety of Naïve Bayes Classifier will only consider features which have positive information gain ratio. The ratio is obtained through analysis with **InfoGainAttributeEval** method in **Weka** 3.5.1. Additionally, the information gain ratio of each attribute will be used as its weight.

Multivariate Gaussian Classifier (2.3.2) The Multivariate Gaussian Classifier will consider only a small subset of features. These features will be as follows:

- **Mean [3]**
- **Standard Deviation [3]**

Only a small subset of features can be used as Multivariate Gaussian Classifier usually requires large training libraries for accurate results.

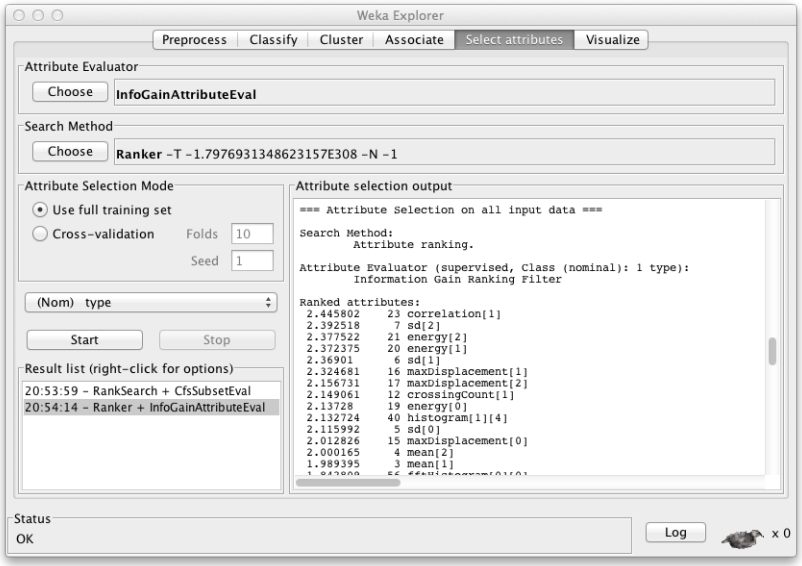

Figure 5.2: Output of information gain evaluation per feature in Weka.

5.1.2 Methodology

The **Workstation Module** of the solution will be used to quantify data from the experiment. All of the functionality needed to perform the experiment is done through a custom implementation.

Experiment will be performed as follows:

1. Full training sample library will be loaded into the workstation module.
2. **AccFeat** (4.4.1) library - library of objects representing extracted features will be created, however only 10 random samples from the training set (per activity) will be used for this purpose.
3. Three classifiers will be built (Naïve/Weighted/Multivariate) using the AccFeat library.
4. 50 test samples per activity will classified. Classification results will be put into a confusion matrix.
5. Each stage will be repeated 50 times to mitigate impact of 10 randomly chosen samples. Results for each stage will be summed up to produce a confusion matrix. Three confusion matrices, one per algorithm will be produced per stage.

6. Random sample count will be increased by 10 and next stage will begin, by repeating the procedure from **step 2**.

5.1.3 Methodology Code

The code of the implementation used for the analysis can be seen below.

```
public static void confusionMatrix(Classifier c, int repeats) {
  int[][] matrix = new int[8][8];
  int count = 0;
  int right = 0;
  for (int i = 0; i < repeats; i++) {
    buildAccFeatLibrary(true, libLimit);
    c.reloadLibrary(accFeatLibrary);
    for (AccFeat a : accTestFeatLibrary) {
      Pair<ArrayList<Double>, String> p = c.classify(a);
      ClassificationResult r = new ClassificationResult(p.getFirst(),
          null);
      if (a.getType() == r.getResult()) right++;
      count++;
      matrix[a.getType()][r.getResult()]++;
    }
  }
  System.out.println("Correct classifications: " + right + "/" + count);
  print(matrix);
}
```

5.1.4 Results

The correlation between library size and number of correct classifications can be seen in figures 5.3 and 5.4.

Full set of confusion matrices that were produced in the experiment can be found in the Appendix.

5.1.5 Conclusions from the experiment

It is apparent that Multivariate classifier only becomes effective after large amount of training samples is provided for the classifier model generation

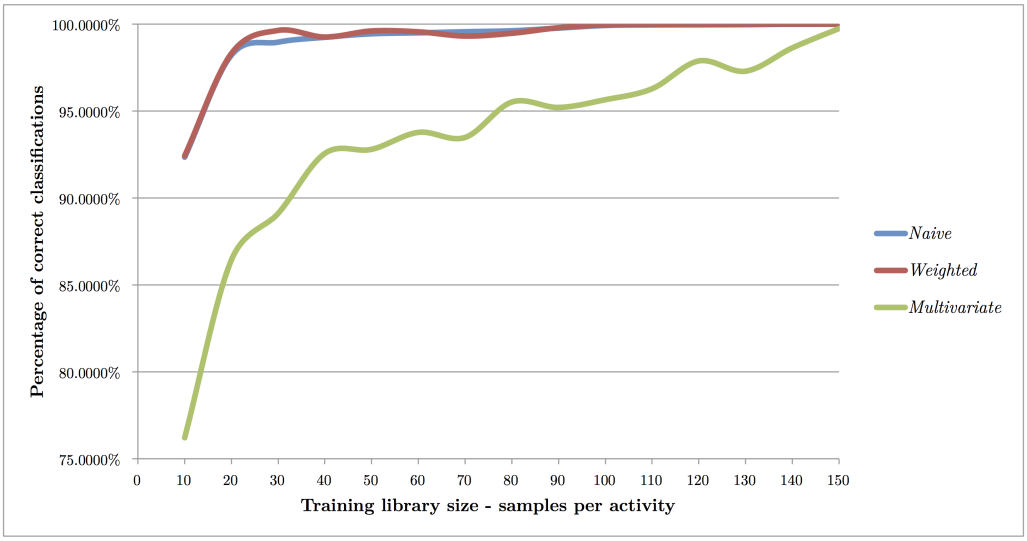

Figure 5.3: Impact of training library size on classification accuracy.

Library size	Naive	Weighted	Multivariate	Naive	Weighted	Multivariate
10	18469	18484	15242	92.3450%	92.4200%	76.2100%
20	19643	19662	17286	98.2150%	98.3100%	86.4300%
30	19793	19928	17822	98.9650%	99.6400%	89.1100%
40	19849	19852	18513	99.2450%	99.2600%	92.5650%
50	19889	19922	18560	99.4450%	99.6100%	92.8000%
60	19900	19912	18755	99.5000%	99.5600%	93.7750%
70	19913	19863	18697	99.5650%	99.3150%	93.4850%
80	19924	19895	19104	99.6200%	99.4750%	95.5200%
90	19955	19959	19042	99.7750%	99.7950%	95.2100%
100	19985	19991	19131	99.9250%	99.9550%	95.6550%
110	19991	19996	19257	99.9550%	99.9800%	96.2850%
120	19992	19998	19577	99.9600%	99.9900%	97.8850%
130	19994	20000	19459	99.9700%	100.0000%	97.2950%
140	20000	20000	19727	100.0000%	100.0000%	98.6350%
150	20000	20000	19950	100.0000%	100.0000%	99.7500%

Figure 5.4: Correct classifications for training library sizes 10-150.

process. As we can see, it takes 150 training samples per activity for it to reach same precision as Naïve Gaussian Bayes Classifier does using only 90 samples. Overall, the Weighted Naïve Gaussian Bayes Classifier performs slightly better than the Naïve classifier in most of the cases and additionally it reaches the 100% level using 10 less training samples. Additional advantage of the Weighted variant of the classifier is that less data has to be stored in the cloud database, as smaller set of features is considered. I have therefore decided that the Weighted Naïve Gaussian Bayes Classifier will be the one recommended to use in the solution, however Multivariate Gaussian Bayes Classifier is fully implemented together with full cloud database compatibility.

In practical terms, we can see that 30 samples per activity are enough to achieve at least 99% accuracy in activity recognition. This means that user is required to spend only a total of 192 seconds to record all needed training samples for a particular activity (one sample is 6.4 seconds long).

5.2 Experiment 2: Solution Performance Evaluation

The performance of the solution will be evaluated in real life scenario.

5.2.1 Methodology

A path through the Huxley Building will be set out, which will consist of majority of activities included in the **Activity Set** (3.3).

The Mobile module will be deployed, using following recording settings:

- **Constant Recording** set to on
- **Constant Identifying** set to on
- **Speak out results** set to on

The classifier in the mobile module will be built using same training data as in 5.1.

5.2.2 Quantification

While performing the activities, the subject will be wearing headphones. As each result of the classification is spoken out, the subject will use an appropriate **Tally counter** to count correct (green counter) and incorrect/unknown classifications (red counter). The data will be recorded after each completed path, and aggregated to find out the precision of classification.

Figure 5.5: Tally counters used to count correct and incorrect classifications through the duration of the experiment

5.2.3 Results

The classification takes place every 3.2 seconds, with 1.6 second overlap. Through the experiment I have found out that solution has close to 100% precision when detecting **currently** performed activity. However, when changing from one activity to the other, for example standing up after sitting, there are usually 2-3 unknown activity classifications before new activity is classified. The amount of unknown activity classifications when going from one activity to the other, depends on how fast the transition is done. We can see the reason why unknown classifications will appear in the diagram 5.6.

When user switches from sitting position to standing position, sudden acceleration changes are registered and a classification is attempted. Because a

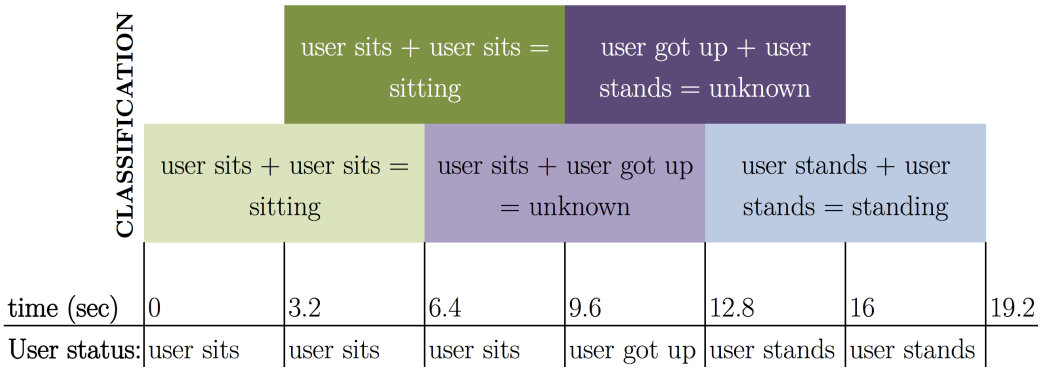

Figure 5.6: Why switching current activity causes unknown classification.

sliding window is used, these sudden acceleration changes "contaminate" two classification attempts. Of course example presented in the diagram assumes that user does his switch in less than 1.2 seconds, and he does it exactly after last successful "sitting" classification. If user does the activity switch a little bit later, or he performs it for longer than 1.2 seconds, three or more classification attempts may be contaminated.

In the experiment, it was found out that on average between 20% and 25% of classifications are of "unknown" variety. This result however is heavily influenced by how the path was laid out, with user often changing the performed activity.

In even more "real life" scenario, example being walking from student halls to the campus ratio of correct to unknown results exceeded 9:1, as "walking" activity was performed for extended period of time, with brief pauses caused by traffic lights.

5.2.4 Conclusion from the experiment

The solution has proven to be extremely reliable, correctly identifying every of the performed activities. The main drawback of the solution is that it is sensitive to frequent change of performed activity. Ratio of correctly classified activities to unknown activities can be increased by further decreasing the classification window size. Despite this sensitivity, even when changing activities very often, we can expect around 80% of the classifications to be reflecting activities performed in that period of time, as proven in the experiment.

5.3 Treatment of "unknown" classification result

The solution sends each classification result to the cloud database, tagged with the timestamp of classification and userid of the user sending the result. Results can be browsed through the Web Module.

As has been discovered via **Experiment 2** (5.2), changing activity from one to the other causes between 2-4 misclassifications - marked as "unknown" classification. The solution therefore will therefore allow up to 4 consecutive misclassifications, sending last identified result instead. For example in the situation shown in the diagram 5.6, instead of sending two "unknown" classifications at times t=12.8 and t=16, two "sitting" classifications will be sent. However, if more than 4 consecutive "unknown" classifications are received, application will not send any more results to the database, until known classification result is received.

5.4 Compatibility with other Android Phones

The mobile module has been deployed and tested on the following phones:

- LG Nexus 4
- HTC One
- HTC One X

5.4.1 Methodology

1. Application was installed on the phone
2. Classifier models were downloaded from the cloud database
3. **Experiment 2** 5.2 has been performed using the new phone

5.4.2 Results

There have been no differences in classifier performance when compared to running on Samsung Galaxy S3. The interface of the application was rendered in the exactly same way on every phone except for HTC One. HTC One,

equipped with 1080p screen, as opposed to 720p used in other phones allows more data to be displayed at once - most noticeable difference has been noted in **Activity Recording** tab, where more lines of feature values were displayed.

5.5 Use of Gyroscope

At early stages of development, I have been experimenting with use of gyroscope recordings together with accelerometer recordings for classification purposes, essentially creating a Sensor Fusion solution.

5.5.1 Methodology

1. Two sets of test samples were loaded into the Workstation Module: one set of samples consisting solely of accelerometer data, while the other consisting of both accelerometer and gyroscope data.
2. Following features were extracted from gyroscope data:

- **Mean** [3]
- **SD** [3]

2. Naïve Gaussian Classifier was used to classify all of the samples

5.5.2 Results

The samples with accelerometer data were often misclassified, which has shown that (as predicted in 2.1.1) gyroscope readings are not relevant in classification of daily activities.

5.6 256 vs. 512 sample size

Originally the solution was intended to be using 512 sample size. This is reflected for example in the **Activity Recording** tab of the Mobile Module, where option **Twice-the-size (512) samples** can be seen. The switch to 256 samples has been made after noticing that smaller sample size, and therefore shorter timeslice that is classified (6.4 vs 12.8 seconds) reduces

impact of going from one activity to the other. This is because a smaller window, and therefore smaller overlap means that fewer classifications would be contaminated by the movement executed to go from one activity to the other.

All of the training samples recorded in the 512 mode are still used in the training process - when the Workstation Module detects a 512 samples it splits it into two samples and creates two `AccFeat` objects from them which are later used for Machine Learning Process.

5.7 Battery Life Impact

An investigation was made into how the solution impacts battery life of the phone.

5.7.1 Methodology

1. Phone was charged to 100% level.
2. Mobile module was deployed on the phone with Monitor Tab enabled, therefore forcing constant sampling of accelerometer data.
3. Phone was left with application running overnight, for 9 hours. Any kind of data connection like WiFi or 3G has been disabled to prevent automatic updates from affecting the battery drain.
4. Level of battery was noted in the morning.
5. To have a comparison, on the next day phone was charged to 100% level and left for 9 hours again, but without the Mobile module.
6. Level of battery was noted and compared.

5.7.2 Results and Conclusions

The battery drain by the enabled accelerometer has been proven to be surprisingly small, to the point of being negligible. Battery level after 9 hours of idle usage has been only 2% higher than that of 9 hours of sampling. Although no figures are available on the accelerometer power usage in the Samsung Galaxy S3, similar high-end handset - iPhone 4 is equipped with accelerometer that consumes 0.75mW of power at 100Hz sampling rate. This figure explains the low battery impact of the solution, which is sampling at much lower rate of 40Hz.

Chapter Summary:

This chapter presents experiments used to evaluate the solution. First Experiment 5.1 evaluates impact of training library size on the precision of three classifier algorithm implementations. It has been shown that for small library size, Naïve Gaussian Bayes Classifier performs significantly better than Multivariate Gaussian Bayes Classifier. This influenced the decision of choosing the Naïve Gaussian Bayes Classifier as the default classification algorithm in the solution, however it was also shown that the Multivariate Gaussian Bayes Classifier eventually reaches the comparable precision level, therefore both implementations are available to chose from. As number of training samples increases, a switch can be made to the Multivariate Classifier. The experiment is performed through self-implemented analysis methods using the Workstation Module.

Experiment 2 5.2 looks at the performance of the solution in real life example. Subject is asked to repeatedly perform activities while wearing headphones, which enables him to hear the results of classification. As result is heard, he uses one of two counters depending on whether the result is corret or incorrect/unknown. It was found that solution is very precise, however sensetive to frequent changing of currently performed activity.

The other sections of the chapter describe less complex issues such as treatment of "unknown" result in terms of quantifying daily activity, compatibility with other handsets, battery drain, sample size impact and use of gyroscope.

Chapter 6

Conclusion

This project was an attempt at creating a long-term activity recognition solution through use of smartphone accelerometer. The goal of the project was to successfully identify and quantify activities performed by the user during the day, motivating the user to do more of physical exercise every day - one example being choosing stairs over the lift.

6.1 New Features

Following the trend of improving classification precision noticed when researching past works in the area, an attempt was made at discovering new signal features. Past works have not made an attempt at using histograms of FFT data as a feature of acceleration sample. This solution has employed this technique, and as was shown by analysis of features with Weka, **FFT histograms** (described in 3.4.1) have proven to be a feature of high information gain, with some bins of the histogram scoring higher than tried-and-tested features like correlation between axes or even mean value of acceleration values.

Another unique feature which was used in the solution was the **Relative Crossing Count** (described in 3.4.1). This feature has proven to be very effective - analysis via Weka has shown that relative crossing count on y axis is one of TOP8 features in terms of information gain, with relative crossing count of the z axis being TOP16. Full output from Weka can be found in the appendix.

6.2 Scalability

Never before the possibilities that came with cloud technology have been exploited like in this implementation.

6.2.1 Data Gathering

While other works in the field have used Bluetooth (Nishkam Ravi 2005) or offline data gathering (Jennifer R. Kwapisz 2010), this solution does not even require the researcher to see the subjects. Through use of cloud database for storage of training and test samples, subjects can be based anywhere around the world, and no direct access to them is needed like in the past works in the field. The number of subjects can be scaled to any amount, and as number of subjects increases the cloud database will scale automatically.

6.2.2 Classifier Model Generation

By generating the classifier model on a workstation, and distributing it to users via cloud database, solution is not limited by the processing power of the phone or the amount of by data-transfer limits. In work of (Jennifer R. Kwapisz 2010), where multiple subject approach was chosen, 4526 training samples were recorded by multiple subjects - assuming same sample size as in this solution, this equates to over 76MB of data. With such large training library size building classifier model on a mobile phone would require downloading 76MB of samples every time a new classifier model was to be built. If we compare it to this solution, which builds the classifier model on a workstation, only 0.0033MB of data has to be downloaded by the phone. The size of classifier model data is completely independent of training library size. Additionally, all of data processing required to build the model is mitigated to a workstation.

6.3 Single Subject vs. Multiple Subjects

This solution was focusing on single-subject activity recognition, instead of the approach undertaken by previous works in the field, where samples were recorded using many human subjects.

The choice of focusing on single subject has one major drawback which made itself apparent through testing - efficiency of the solution decreases when somebody else than person who trained it wants to use it.

However, when we compare results of this solution to the results of past works in the area the difference this decision has made becomes more apparent.

Below we can see confusion matrices from work of (Jennifer R. Kwapisz 2010)

		Predicted Class					
		Walk	Jog	Up	Down	Sit	Stand
Actual Class	Walk	**1543**	5	73	60	1	1
	Jog	3	**1299**	16	3	0	0
	Up	84	24	**335**	98	2	2
	Down	108	10	136	**206**	2	3
	Sit	0	2	4	1	**268**	7
	Stand	1	0	5	4	8	**205**

Figure 6.1: Kwapisz et al. - Confusion matrix

The training library size in (Jennifer R. Kwapisz 2010) research was of between 223 to 1683 samples per activity, with a mean of 754 samples per activity, collected from multiple human subjects. Each of samples was 10 seconds long, which means that over 2 hours of samples were recorded per activity.

Compare this with confusion matrix from this solution, where only 30 samples per activity are used:

Weighted Naive Bayes Classifier (Library size: 30)

		Classified as							
		Walk	**F. Walk**	**StairUp**	**StairDwn**	**Sit**	**Stand**	**Cycle**	**Jog**
Actual class	**Walk**	56	0	0	0	0	0	0	0
	F. Walk	0	53	1	0	3	0	0	0
	StairUp	0	0	52	0	0	0	0	0
	StairDwn	0	0	1	63	3	0	0	0
	Sit	0	0	0	0	72	11	0	0
	Stand	0	0	0	0	0	52	0	0
	Cycle	0	0	0	0	0	0	52	0
	Jog	0	0	0	0	1	0	0	49

Figure 6.2: Confusion matrix - library of 30 samples per activity

If we remember that samples in this solution are only 6.4 seconds long, it is easy to calculate that only 192 seconds of samples per activity have to be recorded to achieve over 99% recognition precision.

It is therefore worth asking the question - is it better to ask users to train their applications themselves as opposed to having a global training database?

In my opinion, answer lies in the middle. There are some drawbacks of following the single-subject approach:

- Solution is not effective when used by somebody else
- Solution is sensitive to external factors - example: if user has done all of the training in same pair of trousers, it might work worse with a different pair
- If solution was to be marketed, users would have to be trusted to record proper data for each activity, and not for example walking data for climbing stairs

This is why despite deciding to focus on single-subject activity recognition, through use of cloud database for storage of classifier models and training samples, it is possible to employ hundreds of human subjects to record and submit training data without even a slight change in the code of the solution.

6.4 Future Work

The major limiting factor on the solution described in the report was time. If one was to improve on the solution, suggested improvements are as follows:

- **Multiple subject training data** - employ multiple human subjects for gathering of training data. Semi-supervised approach to gathering of training data should be explored, where new training data is compared to existing data to prevent contaminating training set with erroneous data. Because of cloud based architecture, thousands of subjects can be used, allowing us to build a more general human activity recognition model.
- **Use of Hidden Markov Models** - Hidden Markov Models would be useful in improving classification precision. For example, it is unlikely that next activity after sitting is walking up the stairs. Looking at performed activities as a sequence, rather than a single activity can be a major improvement in terms of precision, and might be needed if a more general, and therefore harder to classify model was to be built.

- **Improvements to the Web Module** - Current proof-of-concept web module is lacking authentication mechanisms, which should be implemented if solution was to be marketed. Adding social features and potential Facebook integration can be explored as well, increasing the motivational factor of the solution.

Chapter 7

Bibliography

A. M. Khan, Y.-K. Lee, S. Y. Lee. 2010. "Human Activity Recognition via An Accelerometer-Enabled-Smartphone Using Kernel Discriminant Analysis." http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05482729.

D. Minnen, T. Starner, J. A. Ward, P. Lukowicz, G. Troster. 2005. "Recognizing and discovering human actions from on-body sensor data." http://cecs.uci.edu/~papers/icme05/defevent/papers/cr1851.pdf.

Digithoughts. 2012. "Android and Apple smartphone sales, 2007-2012 Android and Apple sales 2012" (April). http://digithoughts.com/post/21428797724/android-and-apple-smartphone-sales-2007-2012-by.

Dominic Maguire, Richard Frisby. 2009. "Comparison of Feature Classification Algorithm for Activity Recognition Based on Accelerometer and Heart Rate Data." http://arrow.dit.ie/cgi/viewcontent.cgi?article=1002&context=ittpapnin.

Harris, Jamie. 2013. "Apple iPhone 5 remains Britain's most popular smartphone." http://www.digitalspy.co.uk/5K43ztEXvFfzQjiqSbpeye4c6DDFkmpP8knp3bZgrpCxcD5cs
tech/news/a486807/apple-iphone-5-remains-britains-most-popular-smartphone.
html.

Jennifer R. Kwapisz, Gary M. Weiss, Samuel A. Moore. 2010. "Activity Recognition using Cell Phone Accelerometers." http://www.cis.fordham.edu/wisdm/public_files/sensorKDD-2010.pdf.

Johnson, Luke. "Samsung Galaxy S3 remains UK's most popular handset as iPhone 5 slumps." http://www.trustedreviews.com/news/samsung-galaxy-s3-remains-uk-s-most-popular-handset-as-iphone-5-slumps.

Kozmann G, Lux RL, Scott M. 1991. "Sample size and dimensionality in

multivariate classification: implications for body surface potential mapping." http://www.ncbi.nlm.nih.gov/pubmed/2036782.

L. Sun, D. Zhang, N. Li. 2011. "Physical Activity Monitoring with Mobile Phones:" 104–111. http://link.springer.com/chapter/10.1007%2F978-3-642-21535-3_14.

Lau, S. L., and K. David. 2010. "Movement Recognition using the Accelerometer in Smartphones." *Communication*: 1–9. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5722356.

M. Ermes, J. Parkka, J. Mantyjarvi,, and I. Korhonen. 2008. "Detection of daily activities and sports with wearable sensors in controlled and uncontrolled conditions." http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4358887&userType=inst.

Mohd Fikri Azli bin Abdullah, Ali Fahmi Perwira Negara, Md. Shohel Sayeed, Deok-Jai Choi, Kalaiarasi Sonai Muthu. 2012. "Classification Algorithms in Human Activity Recognition using Smartphones." *International Journal of Computer and Information Engineering 6 2012* 6/1012: 77–84. http://www.waset.org/journals/ijcie/v6/v6-15.pdf.

MongoLab.com. "MongoLab.com." www.mongolab.com.

Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, Michael L. Littman. 2005. "Activity Recognition from Accelerometer Data." http://www.aaai.org/Papers/IAAI/2005/IAAI05-013.pdf.

Poovandran, R. 2008. "Human activity recognition for video surveillance." *IEEE International Symposium on Circuits and Systems*: 2737–2740. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4542023.

S.-I. Yang, S.-B. Cho. 2008. "Recognizing human activities from accelerometer and physiological sensors." *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*: 100–105. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4648116.

T. S. Saponas, J. Lester, J. Froehlich, J. Fogarty,, and J. Landay. "iLearn on the iPhone: Real-Time Human Activity Classification on Commodity Mobile Phones.." http://research.microsoft.com/en-us/um/people/ssaponas/publications/UW-CSE-08-04-02.pdf.

Tasse, Dan. 2012. "Android accelerometer sampling rates." http://ilessendata.blogspot.co.uk/2012/11/android-accelerometer-sampling-rates.html.

Thomas Holleczek, Jona Schoch, Bert Arnrich, Gerhard Troster. 2010. "Recognizing Turns and Other Snowboarding Activities with a Gyroscope." http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5665871.

U. Maurer, A. Smailagic, D. Siewiorek,, and M. Deisher. 2006. "Activity recognition and monitoring using multiple sensors on different body positions." http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1612909&tag=1.

Wikipedia. "Wikipedia." https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

Wilcox, Joe. "Android wins the smartphone wars." http://betanews.com/2012/09/13/android-wins-the-smartphone-wars/.

Yang, J. 2009. "Toward Physical Activity Diary: Motion Recognition Using Simple Acceleration Features with Mobile Phones." *Data Processing*: 1–9. http://cvrr.ucsd.edu/ece285/papers/Yang_towardsPhysicalActivityDiary.pdf.

irisCouch.com. "irisCouch.com." http://www.iriscouch.com/.

Appendix - confusion matrices

Naive Bayes Classifier (Library size: 10)

		Classified as							
		Walk	**F. Walk**	**StairUp**	**StairDwn**	**Sit**	**Stand**	**Cycle**	**Jog**
Actual class	**Walk**	2465	0	2	0	0	33	0	0
	F. Walk	0	2475	5	10	3	7	0	0
	StairUp	0	0	2356	82	21	41	0	0
	StairDwn	0	2	59	2263	28	148	0	0
	Sit	17	0	0	0	2321	127	35	0
	Stand	0	0	0	0	0	2500	0	0
	Cycle	0	0	2	1	671	22	1804	0
	Jog	0	0	0	0	112	103	0	2285

Weighted Naive Bayes Classifier (Library size: 10)

		Classified as							
		Walk	**F. Walk**	**StairUp**	**StairDwn**	**Sit**	**Stand**	**Cycle**	**Jog**
Actual class	**Walk**	2401	0	1	1	0	97	0	0
	F. Walk	0	2280	5	2	39	174	0	0
	StairUp	0	0	2382	38	2	78	0	0
	StairDwn	0	1	13	2332	3	151	0	0
	Sit	3	0	0	0	2346	114	37	0
	Stand	0	0	0	0	37	2463	0	0
	Cycle	0	0	0	0	399	17	2084	0
	Jog	0	0	0	0	35	269	0	2196

Multivariate Bayes Classifier (Library size: 10)

		Classified as							
		Walk	**F. Walk**	**StairUp**	**StairDwn**	**Sit**	**Stand**	**Cycle**	**Jog**
Actual class	**Walk**	1999	20	286	195	0	0	0	0
	F. Walk	3	2469	13	15	0	0	0	0
	StairUp	0	1	2254	245	0	0	0	0
	StairDwn	0	3	600	1897	0	0	0	0
	Sit	647	0	9	0	1277	0	567	0
	Stand	500	100	624	919	0	357	0	0
	Cycle	2	0	5	0	0	0	2493	0
	Jog	0	3	1	0	0	0	0	2496

Naive Bayes Classifier (Library size: 20)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2486	1	10	0	3	0	0
StairUp	0	0	2476	24	0	0	0	0
StairDwn	0	0	51	2443	3	3	0	0
Sit	2	0	0	0	2420	61	17	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	159	0	2341	0
Jog	0	0	0	0	0	23	0	2477

Weighted Naive Bayes Classifier (Library size: 20)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2498	0	0	0	0	2	0	0
F. Walk	0	2450	1	14	0	35	0	0
StairUp	0	0	2494	6	0	0	0	0
StairDwn	0	0	11	2484	0	5	0	0
Sit	2	0	0	0	2431	59	8	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	182	0	2318	0
Jog	0	0	0	0	0	13	0	2487

Multivariate Bayes Classifier (Library size: 20)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2437	0	45	18	0	0	0	0
F. Walk	0	2484	7	9	0	0	0	0
StairUp	0	0	2446	54	0	0	0	0
StairDwn	0	0	281	2219	0	0	0	0
Sit	76	0	0	0	2053	0	371	0
Stand	100	100	251	1402	0	647	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 30)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2492	2	4	1	1	0	0
StairUp	0	0	2494	5	1	0	0	0
StairDwn	0	0	2	2496	1	1	0	0
Sit	0	0	0	0	2440	58	2	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	1	128	0	2371	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 30)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2485	6	9	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	2	0	0	0	2465	30	3	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	22	0	2478	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 30)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2488	6	6	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	170	2330	0	0	0	0
Sit	0	0	0	0	2355	0	145	0
Stand	100	0	252	1499	0	649	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 40)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2493	1	6	0	0	0	0
StairUp	0	0	2499	1	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2432	67	1	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	75	0	2425	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 40)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2484	1	15	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	1	2499	0	0	0	0
Sit	0	0	0	0	2447	52	1	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	78	0	2422	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 40)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2497	0	0	3	0	0	0	0
F. Walk	0	2496	1	3	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	123	2377	0	0	0	0
Sit	0	0	0	0	2445	0	55	0
Stand	0	0	54	1248	0	1198	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 50)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2492	1	7	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	1	2499	0	0	0	0
Sit	0	0	0	0	2412	87	1	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	14	0	2486	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 50)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2496	0	4	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2453	46	1	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	27	0	2473	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 50)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2497	0	3	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	107	2393	0	0	0	0
Sit	0	0	0	0	2432	0	68	0
Stand	0	0	0	1262	0	1238	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 60)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2494	0	6	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2407	93	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	1	0	2499	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 60)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2487	0	13	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2429	71	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	4	0	2496	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 60)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2496	3	1	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	84	2416	0	0	0	0
Sit	0	0	0	0	2489	0	11	0
Stand	0	0	2	1144	0	1354	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 70)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2494	0	6	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2419	81	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 70)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2485	0	15	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2378	122	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 70)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2498	1	1	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	81	2419	0	0	0	0
Sit	0	0	0	0	2490	0	10	0
Stand	0	0	0	1210	0	1290	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 80)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2497	0	3	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2427	72	1	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 80)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2485	0	15	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2410	90	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 80)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2499	0	1	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	87	2413	0	0	0	0
Sit	0	0	0	0	2495	0	5	0
Stand	0	0	0	803	0	1697	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 90)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2497	0	3	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2460	40	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	2	0	2498	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 90)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2490	0	10	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2469	31	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 90)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	77	2423	0	0	0	0
Sit	0	0	0	0	2496	0	4	0
Stand	0	0	0	877	0	1623	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 100)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2499	0	1	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2486	14	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 100)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2493	0	7	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2498	2	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 100)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	74	2426	0	0	0	0
Sit	0	0	0	0	2499	0	1	0
Stand	0	0	0	794	0	1706	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 110)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2499	0	1	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2492	8	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 110)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2496	0	4	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 110)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	67	2433	0	0	0	0
Sit	0	0	0	0	2499	0	1	0
Stand	0	0	0	675	0	1825	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 120)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2492	8	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 120)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2498	0	2	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 120)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	55	2445	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	368	0	2132	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 130)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2494	6	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 130)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 130)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	60	2440	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	481	0	2019	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 140)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 140)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 140)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	50	2450	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	223	0	2277	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Naive Bayes Classifier (Library size: 150)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Weighted Naive Bayes Classifier (Library size: 150)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	0	2500	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Multivariate Bayes Classifier (Library size: 150)

Actual class \ Classified as	Walk	F. Walk	StairUp	StairDwn	Sit	Stand	Cycle	Jog
Walk	2500	0	0	0	0	0	0	0
F. Walk	0	2500	0	0	0	0	0	0
StairUp	0	0	2500	0	0	0	0	0
StairDwn	0	0	50	2450	0	0	0	0
Sit	0	0	0	0	2500	0	0	0
Stand	0	0	0	0	0	2500	0	0
Cycle	0	0	0	0	0	0	2500	0
Jog	0	0	0	0	0	0	0	2500

Appendix - Weka Attribute Analysis

```
=== Run information ===

Evaluator:    weka.attributeSelection.InfoGainAttributeEval
Search:weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation:     Object
Instances:    1200
Attributes:   97
              type
              mean[0]
              mean[1]
              mean[2]
              sd[0]
              sd[1]
              sd[2]
              avPeakDistance[0]
              avPeakDistance[1]
              avPeakDistance[2]
              crossingCount[0]
              crossingCount[1]
              crossingCount[2]
              resultantAcc
              maxDisplacement[0]
              maxDisplacement[1]
              maxDisplacement[2]
              0
              energy[0]
              energy[1]
              energy[2]
              correlation[0]
              correlation[1]
              correlation[2]
              SMA
              histogram[0][0]
              histogram[0][1]
              histogram[0][2]
              histogram[0][3]
              histogram[0][4]
              histogram[0][5]
```

histogram[0][6]
histogram[0][7]
histogram[0][8]
histogram[0][9]
histogram[1][0]
histogram[1][1]
histogram[1][2]
histogram[1][3]
histogram[1][4]
histogram[1][5]
histogram[1][6]
histogram[1][7]
histogram[1][8]
histogram[1][9]
histogram[2][0]
histogram[2][1]
histogram[2][2]
histogram[2][3]
histogram[2][4]
histogram[2][5]
histogram[2][6]
histogram[2][7]
histogram[2][8]
histogram[2][9]
fftHistogram[0][0]
fftHistogram[0][1]
fftHistogram[0][2]
fftHistogram[0][3]
fftHistogram[0][4]
fftHistogram[0][5]
fftHistogram[0][6]
fftHistogram[0][7]
fftHistogram[0][8]
fftHistogram[0][9]
fftHistogram[1][0]
fftHistogram[1][1]
fftHistogram[1][2]
fftHistogram[1][3]
fftHistogram[1][4]
fftHistogram[1][5]
fftHistogram[1][6]

```
                fftHistogram[1][7]
                fftHistogram[1][8]
                fftHistogram[1][9]
                fftHistogram[2][0]
                fftHistogram[2][1]
                fftHistogram[2][2]
                fftHistogram[2][3]
                fftHistogram[2][4]
                fftHistogram[2][5]
                fftHistogram[2][6]
                fftHistogram[2][7]
                fftHistogram[2][8]
                fftHistogram[2][9]
                AR[0][0]
                AR[0][1]
                AR[0][2]
                AR[0][3]
                AR[1][0]
                AR[1][1]
                AR[1][2]
                AR[1][3]
                AR[2][0]
                AR[2][1]
                AR[2][2]
                AR[2][3]
Evaluation mode:evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
  Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 1 type):
  Information Gain Ranking Filter

Ranked attributes:
 2.445802    23 correlation[1]
 2.392518     7 sd[2]
 2.377522    21 energy[2]
```

```
2.372375    20 energy[1]
2.36901      6 sd[1]
2.324681    16 maxDisplacement[1]
2.156731    17 maxDisplacement[2]
2.149061    12 crossingCount[1]
2.13728     19 energy[0]
2.132724    40 histogram[1][4]
2.115992     5 sd[0]
2.012826    15 maxDisplacement[0]
2.000165     4 mean[2]
1.989395     3 mean[1]
1.842809    56 fftHistogram[0][0]
1.804546    13 crossingCount[2]
1.777991    51 histogram[2][5]
1.722827    66 fftHistogram[1][0]
1.720558    46 histogram[2][0]
1.720542    53 histogram[2][7]
1.709533    22 correlation[0]
1.702252    57 fftHistogram[0][1]
1.701058    77 fftHistogram[2][1]
1.687486    76 fftHistogram[2][0]
1.678168    39 histogram[1][3]
1.676683    24 correlation[2]
1.670715    67 fftHistogram[1][1]
1.667289    14 resultantAcc
1.646021    41 histogram[1][5]
1.638985    25 SMA
1.628569    50 histogram[2][4]
1.594474    78 fftHistogram[2][2]
1.572379    47 histogram[2][1]
1.558043    36 histogram[1][0]
1.539034    52 histogram[2][6]
1.524737    68 fftHistogram[1][2]
1.476224    38 histogram[1][2]
1.465283    31 histogram[0][5]
1.464312    49 histogram[2][3]
1.455779     9 avPeakDistance[1]
1.447798    54 histogram[2][8]
1.444024    11 crossingCount[0]
1.413786    30 histogram[0][4]
1.408629    58 fftHistogram[0][2]
```

```
1.404607    44 histogram[1][8]
1.380724    32 histogram[0][6]
1.364258     2 mean[0]
1.354444    37 histogram[1][1]
1.335999    45 histogram[1][9]
1.283928    43 histogram[1][7]
1.278898    79 fftHistogram[2][3]
1.251914    90 AR[1][0]
1.250086    55 histogram[2][9]
1.211083    29 histogram[0][3]
1.185737    60 fftHistogram[0][4]
1.175164    59 fftHistogram[0][3]
1.153681    81 fftHistogram[2][5]
1.153382    87 AR[0][1]
1.147942    69 fftHistogram[1][3]
1.143712    61 fftHistogram[0][5]
1.141305    94 AR[2][0]
1.117684    80 fftHistogram[2][4]
1.11019     91 AR[1][1]
1.088294    62 fftHistogram[0][6]
1.086949    82 fftHistogram[2][6]
1.073276    42 histogram[1][6]
1.067558    73 fftHistogram[1][7]
1.063439    72 fftHistogram[1][6]
1.037149    74 fftHistogram[1][8]
1.024419    34 histogram[0][8]
1.021513    86 AR[0][0]
1.014614    63 fftHistogram[0][7]
1.010464    83 fftHistogram[2][7]
1.008854    85 fftHistogram[2][9]
1.007201    75 fftHistogram[1][9]
1.006015    84 fftHistogram[2][8]
0.994498    71 fftHistogram[1][5]
0.988826    48 histogram[2][2]
0.987986    70 fftHistogram[1][4]
0.965907    33 histogram[0][7]
0.963876    64 fftHistogram[0][8]
0.959181    95 AR[2][1]
0.932519    65 fftHistogram[0][9]
0.902623    93 AR[1][3]
0.874647    35 histogram[0][9]
```

```
 0.825033    28 histogram[0][2]
 0.778244     8 avPeakDistance[0]
 0.751999    92 AR[1][2]
 0.663552    27 histogram[0][1]
 0.62577     26 histogram[0][0]
 0.443255    96 AR[2][2]
 0.387968    10 avPeakDistance[2]
 0.350318    89 AR[0][3]
 0.339412    88 AR[0][2]
 0           18 0
 0           97 AR[2][3]

Selected attributes: 23,7,21,20,6,16,17,12,19,40,5,15,4,3,56,13,51,66,46,53,
```