Alma Mater Studiorum · Università di Bologna

Scuola di Scienze

Corso di Laurea Magistrale in Informatica

HUMAN ACTIVITY RECOGNITION IN SPORTS USING THE APPLE WATCH

Relatore:
Marco Di Felice

Presentata da:
Ramy Al Zuhouri

Sessione I

Anno Accademico
2017/2018

Dedicated to my family

Contents

List of Figures

List of Tables

Abstract

With the recent spreading of Internet of Things, the availability of a wide variety of cheap devices brought human activity recognition (HAR) to the broad audience, thus eliminating the need of using costly and obstructive hardware, and to constraint the users to remain in controlled environments. Human activity recognition finds application in the field of health monitoring, safety, context-aware behavior and fitness tracking; it can be applied in order to solve problems such as preventing and diagnosing medical conditions, automatic detection of dangerous events in order to trigger automatic emergency calls, automatic customization of the user's phone settings according to the activity that is being performed, and keeping track of physical activities in order to provide the user with useful metrics (e.g. calories consumption or steps taken). This thesis is focused on fitness tracking, with the aim of finding a way to perform activity recognition with a smartwatch in order to automatize activity tracking, exempting the user from manually interacting with the device in order to manage the workout sessions. For this purpose, the intertial sensors of an Apple Watch, along with the GPS and the heart rate monitor were used to train and test an activity recognition model. 4 subjects collected data for 8 different types of aerobic activities, populating a dataset of 4,083 instances, corresponding to about 20 minutes of physical activity for each subject. 9 different machine learning algorithms were evaluated using the holdout validation, trying different combinations of sensor data and features in order to find the optimal configuration. Due to its simplicity, it was chosen to use a decision tree for further validation

on unseen data. As expected, the accuracy of the decision tree was higher when validated on the test set, but dropped from 95.42% to 90.73% when tested on unseen data. The use of a history set increased the recognition accuracy up to 92.68%. More conclusions derived from validation: the models were able to recognize activities independently on the wirst location of the watch; moreover, the accelerometer and the gyroscope were enough to obtain a good recognition model, while the GPS and the heart rate monitor did not significantly increase the accuracy.

Chapter 1

Introduction

1.1 Overview

Human activity recognition (HAR) has became more and more popular in recent times, due to the widespread availability of smartphones, smartwatches and other types of wearable devices that allow to perform human activity recognition inexpensively, without wearing costly and obstructive devices, and without the need of constrainting the users to remain in controlled environments. Human activity recognition have found practical applications in the daily lives of people, mainly in the fields of health monitoring, safety, context-aware behavior and fitness tracking [1]. The goal of fitness tracking is to motivate the users to perform physical activites, counting the steps, the distance walked, the calories burned and other metrics in order to keep the user informed about their activities, and motivating them to reach some goals. Normal, a modern wearable device like a smartwatch or a wristband, needs input from the user in order to start tracking activities. Some devices automatically track physical activities, but in a trivial form, recognizing general activities such as "in movement" or "standing still". More specific physical activities such as "running" or "swimming" need manual input in order to be tracked. This study focuses on finding an application of human activity recognition in fitness tracking, more precisely it has the aim of finding a more

automatic and seamless way of tracking sport activities on smartwatches, in order to exempt the users from manually managing the workout sessions.
For this purpose, a HAR system based on machine learning was developed in order to train and test an activity recognition model, able to classify the following activities: push-ups, sit-ups, squats, lunges, jump rope, resting, walking and running. For such study, the sensors of an Apple Watch Series 2 were used, together with an iPhone 6S, which served as support for storing training data and for implementing more complex functionalities, as well as a MacBook Pro on which some learning algorithms were tested. Due to the restrictive CPU, memory and bandwidth limits of the Apple Watch, which made impossible to use more than 15% of the CPU for more than a minute, or to send large chunks of data to the iPhone without freezing the communications, it was chosen not to store the raw sensor data, which had an excessive size; rather, the feature extraction was carried out in the watch, and the features were sent to the iPhone on which they were stored in a local database. Due to these limitations, it was also chosen to extract a fixed set of sensor data and features, excluding the most expensive ones, and using a low sampling frequency of 16 HZ in order to avoid exceeding the CPU limits, which would imply the risk for the application to crash. Moreover, a fixed window size of 2.5 seconds with 50% overlaps was chosen. 4 subjects, all males, aged between 23 and 46 were included in the study, collecting more than 4,000 instances, corresponding to about 20 minutes of physical activity for every user. The subjects were free to wear the Apple watch in their favourite wrist. The data was therefore exported and analyzed on the MacBook machine using some machine learning scripts written in Python. Nine machine learning algorithms were evaluated using the holdout validation, with a test set including 30% of the data. The most accurate algorithm was random forest, which obtained an accuracy of 99.51%. Various configurations of features, sensors and options were tried, in order to find the optimal configuration. Due to its simplicity and understandability the decision tree model, which obtained an accuracy of 95.42%, was preferred over the random forest for

testing, and it was therefore imported in the watch application. The decision tree was validated within the application, obtaining a lower accuracy of 90.73%. The use of a history set improved the recognition accuracy, up to 92.68% with a history set of 3 elements. Moreover, an analysis of the CPU consumption of the application was performed, but it was able to capture mostly the CPU usage due to feature extraction, because sensors usage depends on system routines that cannot be analyzed in a proprietary operative system such as watchOS.
From the validation outside and within the application, the conclusion was that it is possible to recognize activities independently from the wirst location of the watch, and that the best configuration uses a history set of 3 predictions, and includes only the accelerometer and the gyroscope with automatic feature selection, which obtains a good accuracy with a reduced CPU consumption.

1.2 Structure of The Document

Chapter 2 is a review of the current literature on human activity recognition; current applications of human activity recognition are summarized in section 2.1, while different HAR techniques and algorithms are analyzed in section 2.2. In section 2.3, different ways and metrics to evaluate HAR systems are described. Finally, a review of the past works and studies on human activity recognition is presented in section 2.4.
The architecture of the proposed HAR system is presented in chapter 3, focusing on the goals and methodology in chapter 3.1, and explaining the high-level functioning of the architecture in section 3.2.
Chapter 4 focuses on the implementation details of the system, describing and illustrating the most important parts of the programs through class diagrams and reporting some code snippets. In section 4.1, a review of the frameworks used in the Apple Watch and iPhone application is presented. Sections 4.2 and 4.3 and focus respectively on the watch and phone side

of the application, while the machine learning scripts implemented on the external machines are described in section 4.4. Finally, section 4.5 presents a set of related issued found during the development of the system.
The data collection procedure is described in chapter 5. After a brief introduction, the sampling and feature extraction techniques are presented in section 5.1, while section 5.2 describes how the dataset was populated.
Chapter 6 discusses the evaluation of the developed HAR system, discussing the models validation in section, 6.1, and the in-app validation of the exported model in section 6.2.
Finally, 7 presents the conclusions, along with an overview of possible future work related to this study.

Chapter 2

State of The Art

2.1 Applications of Human Activity Recognition

With the recent spreading of Internet of Things, it has become increasingly common to adopt human activity recognition to recognize daily life activities with a broad variety of low-cost devices such as smartphones, smartwatches and other wearable devices, obtaining information from the device sensors such as the accelerometer, the gyscoscope and the heart rate monitor. Human activity recognition is mainly used in the following areas [1]:

2.1.1 Health Monitoring

Certain medical conditions cannot be prevented, diagnosed and treated in a traditional way, as they require the patients to be monitored during long periods of time, and often even during their daily routine activities. In the case of certain diseases, it is not said that the symptoms will appear during a medical examination, thus human activity recognition would provide doctors a tool to diagnose medical conditions, monitoring daily activities in order to detect abnormalities during the routine of a patient. Certain conditions such as strokes can be prevented and treated only if a diagnosis is made in useful

time. Hence, human activity recognition plays an important role in developing early-stroke diagnosis tools [2]. Human activity recognition is also useful in patients with Parkinson's disease to accurately identify the symptoms and monitor the course of the condition, as they need to be examinated during long periods of time [3].

Moreover, human activity recognition is used to monitor the rehabilitation of patients in order to give a more accurate status of their health condition and to assist them evaluating their improvements [4].

S. Patel *et al.* [5] illustrated a virtual reality approach to monitor the rehabilitation of patients through human activity recognition. Patients can be assisted in their rehabilitation program through *telerehabilitation*, delivering them activities in their homes, in a game-based context that enhances engagement and motivation.

Another application of human activity recognition in the field of health is to monitor eating habits in order to automatically mantain calories intake records that could replace manually-written food diaries [6]. An automatic method to detect drinking and eating activities could be used to provide suggestions that can help the users in the course of their diet. This is a way to tackle the hobesity problem, which is related to major health issues such as strokes and heart attacks. Moreover, human activity recognition could be used to detect smoking habits [7] in order to provide an individualized risk estimation which increases awareness and can motivate the users to quit smoking. Similary, human activity recognition can be used to automatically detect sleeping activities [8], which could be lead to the replacement of costly and invasive methods such as polysomnography.

2.1.2 Safety

Human activity recognition systems are able to automatically send an alert in case that the user falls; in this case, an automatic message can be sent to their relatives, or an automatic emergency call could be triggered to save the patient. The most common commercial systems are available

in form of wearable sensors, but they have a set of limitations such as high cost, limitation in terms of distance that can be traveled from the base, and a limited number of response methods. Smartphones could be used instead of specialised wearable systems, as they have greater flexibility because they do not need to be tied to a receiver, and they have a vast number of response methods such as sending a text message, an email or peforming an automatic call. Additionally, smartphones have a lower cost and they can be afforded by a greater audience of users. J. Dai and Z. Yang [9] proposed a pervasive fall detection system based on mobile phones, in order to address the issue of wearable systems, such as obstructivity and high cost. Y. He, Y. Li and S. D. Bao [10] implemented a fall detection system based on the data acquired from a waist-mounted smartphone in a real-time environment, using the phone built-in accelerometer to detect falls and send an automatic MMS to a pre-selected set of people, with information including the time, the GPS position and the Google Maps location of the suspected fall.

In addition, applications for the Apple Watch such as FallSafety [11] are able to perform fall detection and trigger an automatic emergency call in case that a fall is suspected. The Apple Watch Series 2 requires the Apple Watch to be paired to an iPhone to trigger the call, but since the Apple Watch Series 3, this is no longer required as the watch has an integrated SIM that can make calls autonomously.

Moreover, applications of human activity recognition can be extended to other areas where automatic emergency calls could provide greater safety. J. Dai *et al.* [12] proposed a system able to automatically detect dangerous vehicle maneuvers related to drunk driving in order to prevent car accidents, automatically alerting the driver or calling the police. This solution requires a mobile phone placed in the vehicle, whose accelerometer and gyroscope are used to collect data and compare it against typical drunk driving patterns obtained from real driving tests.

2.1.3 Context-aware Behavior

Human activity recognition can be used to customize the device's behavior according to the high-level activity that the user is performing, such as "at work" or "walking home"; for instance, the system may be able to disable incoming calls or setting the device mode to silent while the user is working, or if the user is exercising it may play music from a pre-selected playlist. When the user is moving, it is more difficult and also potentially dangerous to read the screen, hence Mashita *et al.* [13] suggest to address this problem recognizing the user context in order to automatically adapt the interface and the methods of display. Also the volume of the phone may be automatically adjusted and increased while the user is walking and it is thus submitted to extra noise caused by motion.
A subcase of the human activity recognition problem is the *transportation mode* recognition problem, which aims at detecting what kind of vehicle the user is using. Detecting if the user is moving by train, by car or by another kind of vehicle [14] is important in order to tune-up the context-based services that a mobile phone can provide to the users. For instance, the ringtone can be switched off while the user is walking, or the GPS navigator could be turned on when the user is traveling by car.
Additionally, human activity recognition can be useful for *self-managing systems*, which is an application similar to context-aware behavior, except for the fact that it does not focus on improving the user experience, but rather on managing efficiently the resources of a sytem. For instance, it would be efficient to turn off Bluetooth and WiFi when the user is walking or running, in order to save power, or to disable position sensors such as the GPS or GLONASS when the user is stationary. Another way is to reduce the frequency with which the sensors are read, which can save the battery life up to 20-50% [15].

Figure 2.1: Fitbit Alta^{TM}.

2.1.4 Fitness Tracking

The goal of fitness tracking is to monitor metrics such as steps taken, calories burned and heartbeat. Some advanced systems are able to monitor much more sophisticated metrics such as floors climbed and quality of sleep. Motivating the users to have physical activity is one more benefit of activity tracking: the activities can be shared among users on social networks, promoting challenge and competitiveness.
The Fitbit Alta^{TM} [16] is a wristband that includes a built-in accelerometer and it is able to automatically recognize activities such as walking, swimming and running, and to calculate some useful statistics such as total burned calories and steps taken. Another popular device is the Nike+ FuelBand [17], a wristband with an integrated accelerometer that is able to display a single value such as steps, distance and time, plus a motivational feature which encourages the users to move, earning Nike+ points everytime that they engage in a physical activity. A device with similar capabilities is the McRoberts MoveMonitor [18], which comes in the form of a wearable waistband, which can recognize physical activities and is able to measure energy expenditure and to perform sleep analysis.

Figure 2.2: Nike+ FuelBand.

Modern smartphones include a built-in accelerometer, a gyroscope and a positioning system such as GPS or GLONASS, making fitness tracking available to a broad audience. Indeed, while specialised devices such as Fitbit Alta™ and Nike+ FuelBand are special purpose, a smartphone is rather a general purpose device that is not specifically purchased by users who are exclusively interessed in activity tracking. Commonly, a smartphone application is able to perform a simplified version of human activity recognition, which consists in detecting the steps, the distance traveled and the pace, usually with a generic classification of activities such as "in movement" or "standing", and they need manual input in order to start collecting data about specific exercises like running or wimming. Another limitation of smartphones is that they can't be worn comfortably: users are ought to carry them in their pockets, or to wire them to a special band designed to carry a smartphone, which can be worn in the arm or in the waist. Between the most popular fitness tracking applications for smartphones, there is Nike+ Run Club, Runastic Running & Fitness and Human - Activity Tracker. They ara available for iOS and Android smartphones. More specific applications can calculate advanced metrics, like for example Pacer, an application designed to count steps, or Runastic Road Bike, an application exclusively designed to track

Figure 2.3: McRoberts MoveMonitor.

biking activities.

With the recent availability of more sophisticated smartwatches, equipped with accelerometers and position sensors, fitness tracking has made a further step ahead. The necessity of carrying a smartphone in the pocket or in a band tied to the body has been eliminated, and activity tracking can be performed directly on the smartwatch. The most popular smartwatches include the Apple Watch Series 3, the Garmin Watch and the Ticwatch. They can be paired to a smartphone to exchange data about fitness activities. The previously mentioned fitness applications available for Android and iOS come in a bundle that offer its smartwatch counterpart: the application can be used whether from the phone and from the watch, and the training data is seamlessly exchanged between the two devices. Given that a smartwatch is worn to the wrist, human activity recognition can be more accurate, and it may include more activities like swimming and sleeping, that would be difficult to track on a smartwatch. For instance, the Sleep+ application for watchOS - the Apple Watch operative system - is able to perform a very basilar sleep analysis: while the user sleeps, movements are tracked using the built-in accelerometer in order to classify the sleep into three categories: “awake”, “restless” and “restful”. The sleep data is saved and can be shared with the iPhone for better display, in order to give the user an idea on the sleep quality and duration.

Figure 2.4: Apple Watch Series 3.

2.2 Techniques

Human activity recognition can be approached in two ways [19], using:

- **External sensors**. In this case, the sensors are placed in predetermined points of interests, and the human activity recognition process depends on the voluntary interaction between users and sensors. It is the case of *intelligent homes* [20, 21, 22, 23, 24], where wireless sensors (e.g. RFID) are placed on key objects such as a washbawl or a TV, in order to recognize activities of daily living (ADLs), which include eating, watching TV, or washing dishes. Performing these activities is a good indicator of cognitive and physical capabilities, specially for the elderly and for people with dementia, whose health is required to be constantly monitored. Furthermore, cameras can be used as external sensors in human activity recognition, using features obtained from video sequences [25, 26, 27, 28]. This can be useful in the field of security, surveillance, entertainment and personal archiving.

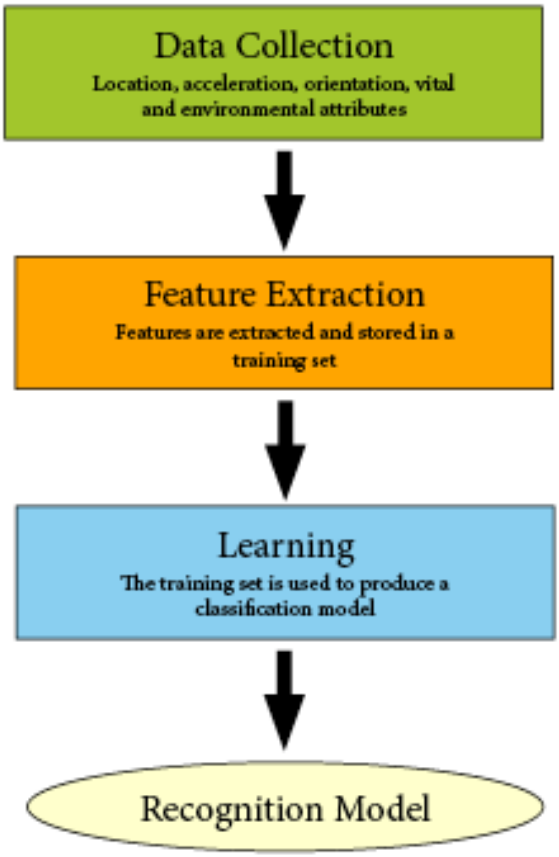


Figure 2.5: Phases of human activity recognition.

- **Wearable sensors**. Using sensors such as accelerometers, GPS, termometers or heart beat rate monitors is generally considered a better approach, because the techniques used in this approach are simpler than video processing techniques. Moreover, wearable sensors overcome more issues such as privacy and pervasiveness: indeed a permission is required from the users to be constantly recorded by cameras, and not everyone is willing to collaborate.

For these reasons, HAR with external sensors will not be examined, while the following remainder of this subsection will delve into the techniques used with wearable sensors.

Figure 2.5 shows the typical phases of human activity recognition. In the first phase, data are collected from sensors every fixed time interval. This time interval determines the *sampling frequency*: the number of times that the sensors are read every second. The data is then stored in a data structure like a queue, and every fixed or variable amount of time τ, the data is fetched and some features are extracted. τ is also called the *window size*, which can be measured in seconds or in the number of samples that are required to be collected before extracting the features. The features are tipically labeled with their class of activity and stored in a permanent memory. The feature extraction process is repeated until a large dataset is obtained, containing a statistically significant amount of data. This dataset is also called *training set*, which is used in the third phase, where the training data is used to produce a recognition model, using a machine learning algorithm.

2.2.1 Collectable Attributes

During the data collection phase of HAR, there are four types of attributes that can be collected:

- **Environmental attributes**. Attributes that depend on the surrounding environment such as humidity, temperature, noise and light levels can be measured. This provides contextual information that is useful to discriminate the activity performed by the users. For instance, if the user is operating in a noisy environment, it is more likely that they are walking or running rather than sleeping. These parameters might be useful to increase the accuracy of the recognition model, but alone they are not sufficient to produce an enough accurate recognition model. A study on human activity recognition that made use of environmental attributes was presented by Maurer *et al.* [29], who used a device containing a light sensor, a thermometer and a microphone, among with a dual axis accelerometer, to build an activity recognition system.
- **Motion and orientation**. An accelerometer is generally considered

a cheap device, which is usually integrated in most modern devices such as smartphones and smartwatches. Used in conjunction with a gyroscope, an accelerometer provides most of the data useful to build an accurate and reliable recognition model [30]. Typically, triaxial accelerometers are used to measure the total acceleration at every instant of time, which requires a gyroscope to isolate the gravity from the user-initiated acceleration. The gravity magnitude is fixed, but it can be decomposed along three axises to produce information about the orientation of the device. The user acceleration is instead given by the instantaneous acceleration caused by the user movements. Moreover, a gyroscope can also measure the rotation rate along the three axises of the device, which is useful since virtually all activities include swinging movements (e.g. swinging an arm during a run). For this reason, the position of the accelerometer is an important factor. For instance, placing an accelerometer inside the trousers pocket can be useful to recognize activities like walking and running, but it can hardly differentiate between activities that do not involve leg movements, like standing still or driving a car.

- **Location**. GPS and GLONASS are the most widely used positioning systems, which are integrated in most modern smartphones. There are two ways to use position sensors in human activity recognition: the first is to use the position to provide contextual information about the activity [31]. For instance, if the user is at home, it is unlikely that they could be swimming or riding a bus, but they might be eating or resting. The second way is to use the variations in longitude, latitude and altitude to calculate the speed and the course of the user during an activity. This can be useful to recognize activities where speed and course are key factors, like driving a car, running and walking. Using position sensors comes with a set of problems: first of all they are not suitable for indoor activities, or for activities performed in areas where the signal is weak. Moreover, the GPS/GLONASS systems are

expensive in terms of CPU usage and energy consumption.

- **Vital signs**. Heart rate, respiration rate, skin temperature and more signals can be used to obtain a more accurate recognition model [32]. However, Tapia *et al.* [33], in their proposed HAR system which used a triaxal accelerometer and a heart rate monitor, concluded that the heart rate is not useful in activity recognition, specially during transitions from intense to moderate activities, because the heart rate is slow to fall down to normal levels, therefore a low-level intensity activity performed after a high-intensity activity can be mistakenly classified as the latter.

2.2.2 Window Size Impact

Segmentation is the process of dividing data samples into smaller data segments, also called windows. There are three groups of segmentation techniques in activity recognition:

- **Activity-defined windowing**. The data is partitioned when an activity transition is dectected. For instance, Sekine *et al.* [34] proposed a model based on wavelet decomposition to detect frequency changes from three groups of walking activities. Another approach is to rely on user feedback to detect activity transitions. For instance, users may be required to stand still for a certain amount of seconds in order to dected activity transitions [35].
- **Event-defined windowing**. The event-defined approach is suitable for activities that can be considered an ordered sequence of actions or movements that occur sporadically, and are scattered through other gestures or movements. In this case, it can be enough to recognize specific movements or gestures that occur at the beginning of the activity. This means that the window size is not fixed, but it changes to fit the activity transitions. For instance, gait analysis can be used to detect

a walking activity, using a model that identifies the phases of a gait cycle, where a heel strike event is used to partition the data [36].

- **Sliding windows**. This is the most broadly used windowing method, because it is simple and it does not require any form of preprocessing, which makes it the most suitable method for real-time applications. It consists in simply dividing the data into a fixed set of samples, with no inter-window gaps. Sometimes it is convenient to reuse part of the data collected during the previous time window, allowing fixed size overlaps. For instance, if the features are extracted every time window of 2 seconds, it could be convenient to reuse half of the data. This means that the most recent data collected during the previous second is reused, while the older data is discarded. This halves the time required to extract the features, without requiring to half the window size.

Since the sliding window approach is the most used because of its simplicity and practicality, the rest of this chapter will focus on this method. With the sliding windows approach, the window size plays an important role in the efficacy and efficiency of a HAR system. Generally, a smaller window size allows for faster activity detection, and has a smaller energy impact. On the contrary, a bigger window size may come with greater energy consumption, but allows for recognition of complex activities, with the penalty of being slow to detect transitions. In previous studies, a window size ranging from 0.1 to 12.8 seconds or more has been used [37].

2.2.3 Feature Selection

Samples are collected at a certain sampling frequency and at the end of every time window, some features are extracted. The features will then serve as parameters useful to discriminate between one activity or another. Therefore, the recognition model obtained at the end of the whole process, will be able to recognize activities in a time window basis.

Among the most popular time domain features reported in literature [19],

there are **central tendency measures** such as the *mean* and the *root mean square*, as reported in the equations 2.1 and 2.2, where $1 \leq i \leq n$ represent the n samples collected during the time window:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.1}$$

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} {x_i}^2} \tag{2.2}$$

Another measure is the median, which is, given a set of samples ordered by value, the value that separates the lower half from the higher half.

Dispersion metrics include the *standard deviation*, the *variance* and the *mean absolute deviation*, as reported in the following equations:

$$\sigma_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{2.3}$$

$$\sigma_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2 \tag{2.4}$$

$$MAD = \sqrt{\frac{1}{n-1}\sum_{i=1} n\,|x_i - \overline{x}|} \tag{2.5}$$

Some features used in HAR are capable of describing the shape of the samples distribution, like the *skewness* [38], which measures the asymmetry of the samples distribution about its mean (equation 2.6). *Kurtosis* instead, gives an idea of the peakedness of the distrubution over the time window (equation 2.7).

$$Skewness = \frac{n}{(n-1)(n-2)}\sum_{i=1}^{n}\frac{(x_i - \overline{x})^3}{{\sigma_x}^3} \tag{2.6}$$

$$Kurtosis = \frac{n(n+1)\sum_{i=1}^{n}(x_i - \overline{x})^4 - 3\big(\sum_{i=1}^{n}(x_i - {\sigma_x}^2)\big)^2(n-1)}{(n-1)(n-2)(n-3){\sigma_x}^4} \tag{2.7}$$

The *interquartile range* is the differentce between the 75th and the 25th percentile (Q3, Q1 in the formula):

$$IR = Q_3 - Q_1 \tag{2.8}$$

The *Fourier transform* takes as input a time domain function such as the distribution of the acceleration samples, and produces a frequency domain function. This is useful to calculate the *energy*, which is defined as the sum of all the components of the Fourier trasnform $(F_1, \ldots, F_n)$, divided by the total number of samples:

$$Energy = \frac{\sum_{i=1}^{n} F_i^2}{n} \tag{2.9}$$

Another feature that is based on the Fourier transform is the *entropy* (H in 2.10) [39], which gives a measure of the amount of disorder in the data:

$$\begin{aligned} \hat{P}(F_i) &= \frac{|F_i|^2}{n} \\ P_i &= \frac{\hat{P}(F_i)}{\sum_{i=1}^{n} \hat{P}(F_i)} \\ H &= -\sum_{i=1}^{n} P_i \ln P_i \end{aligned} \tag{2.10}$$

2.2.4 Learning Algorithms

Machine learning, in particular *supervised learning*, is a widely used approach in human activity recognition. The training set contains the features collected during every time window, plus a categorical value that represents the activity, as shown in table 2.1. Every row of data is also called an *instance*. The goal of a supervised learning algorithm is to analize the training set in order to infer a model able to classify unlabeled instances. A good classification model is able to "guess" the acivity most of the times, but also the building time of the model and more metrics are taken in consideration. Many supervised learning algorithms have been described in literature [40]. A description of the most common algorithms will follow.

Instance	Speed $\left[\frac{m}{s}\right]$	Acceleration $\left[\frac{m}{s^2}\right]$	Activity
1	2.75	3.78	Running
2	3.39	4.41	Running
3	1.51	2.73	Walking
4	0.01	0.71	Resting
5	3.14	6.72	Cycling

Table 2.1: A dataset with labeled instances.

Decision trees

As shown in figure 2.6, in a decision tree, each nonterminal node represents a feature, and each branch represents a splitting point for the value that the feature can assume (e.g. in the root node if the speed is smaller than 2.75, then the leftmost branch is followed). To assign a class to an unlabeled instance, the classification algorithm starts from the root node, and moves to the child node following the branch according to the value of the feature that is being evaluated, until it reaches a leaf node, which contains the label that should be assigned to the instance.
To build a decision tree, the first step is to sort the features in order to choose which ones should appear first in the tree. A common criterion is to choose the feature that best divides the training data first, such as *information gain* [41] and *gini index* [42]. Another criterion is to choose the features not independently, but taking into account the context of other features, like described by Koronenko in his ReliefF algorithm [43]. However, many studies concluded that there is no single best method to choose the attributes, since the choice depends on the characteristics of the training set that is being used [44]. After choosing an attribute, the training data is split according to the chosen value, and then the procedure is repeated recursively on each partition of the data, until all the sub-trees belong to the same class.
A common problem of decision tree algorithms is *overfitting*, which occurs when the decision tree follows too closely the training set, but has a much

bigger error if tested on another dataset. In general, if a decision tree is larger than a decision tree with a similar performance in terms of accurary, then it is preferable to choose the smaller one, because it is less probable that it overfits the training data. To prevent overfitting in decision tree algorithms, there are usually two methods: the algorithm is stopped earlier, to prevent it from perfectly fitting all the training data, or the decision tree is pruned after being built.
The most common decision tree algorithm is the **C4.5 algorithm**, developed by Quinlan [45], which is based on the assumption that the training data fits in memory, and uses the concept of information gain to select the attributes. At each node, the C4.5 algorithm chooses the attribute that maximizes the information gain, then recurring on the subsets of the data obtained by splitting the dataset on the selected attribute. During classification, the decision tree can deal with unknown feature values by passing the instance down all the branches of the node where the unknown feature value was found, and using the averaged classification output of each branch as result. C4.5 is renowned for being a simple and easily comprehensible algorithm, being a good compromise between accuracy and efficiency. It tends to perform better with discrete and categorical features.

Neural networks

Neural networks are based on the concept of perceptrons [46]. As shown in figure 2.7, a **multilayer perceptron** [47] is a network composed of nodes connected among them, grouped by layers. Typically, the first layer contains a number of nodes equal to the number of features whose value should be evaluated in order to discrminate between activities. This layer is called *input layer*, because the functionality of its nodes is to send values to the nodes in the following layer. There is an arbitrarily large number of layers in the middle, called *hidden layers*, whose nodes receive an input from each node in the preceeding layer, and send an output to the nodes in the nodes in the following layer. Given that that m is the number of nodes in the layer

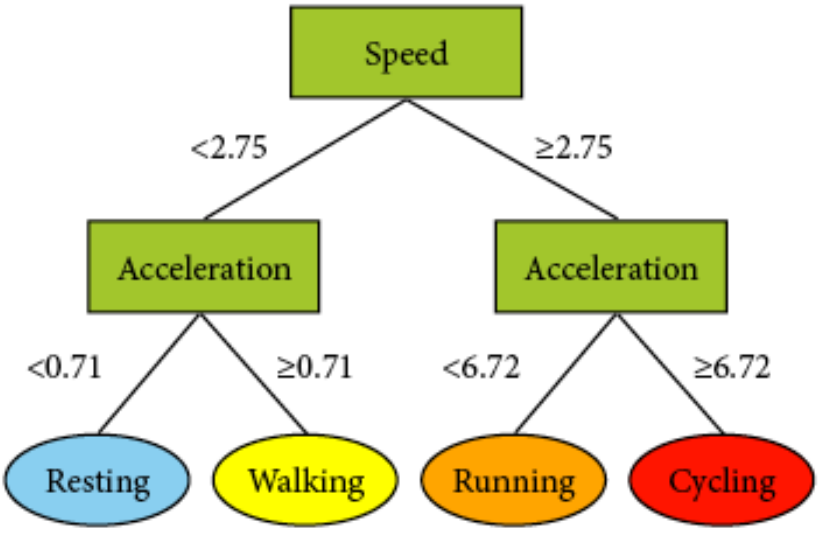


Figure 2.6: A decision tree that fits the dataset in table 2.1.

$k+1$, which is equal to the number of nodes in the preceeding layer (this condition is not necessarily true in all cases, because the number of nodes in any hidden layer can be arbitrarily large), then the input of the j-th node in the layer $k+1$ is equal to:

$$I_j^k = \sum_{i=1}^{m} x_i^k w_{ij}^k \tag{2.11}$$

Where x_i^k is the output of the i-th node in the layer k, and w_{ij}^k is the weight of the connection between the node i and j between the layer k and $k+1$. Each connection between any pair of nodes has its own weight that can be tailored in order to modify the behavior of the neural network. Each hidden node evaluates the input through an *activation function*, that usually adds a bias to the input and returns a value greater than zero only if the sum of the input and the bias is greater than a certain threshold. The behavior of a neural network is thus similar to the behavior of a human brain, where each neuron can fire and activate the synapses that link it to another neuron, until

the signal reaches a target neuron that in the neural network corresponds to the label that should be assigned to the instance that is being evaluated. The *output layer* is the layer than contains the terminal nodes, that are similar to the hidden nodes in the fact that they have an activation function, but they don't have any outbound link, and their output can be joined with the output of the other output nodes, to form a vector $[y_1, y_2, \ldots, y_n]$, where n is the number of activity labels, and only one value y_p among the members of the vector is equal to one, indicating that the predicated activity is the activity corresponding to the p-th output node of the perceptron, while all the other members of the vector are equal to zero.

Given a neural network with a certain number of nodes, three things can be manipulated in order to change its behavior: the network topology, the activation functions and the weights. Assuming that the first two are fixed, then the problem is to find the configuration of weights that gives the best accuracy. In order to build a multilayered perceptron that fits a certain dataset, initially, a network with random weights is built. Afterwards, for each instance in the dataset, the neural network is fed with its feature values, and the output is compared with the correspoding label in the training set. If the output is correct, then the neural network does not have to be corrected. Otherwise, the weights are updated with a certain bias that brings the output to be closer to the desired one. Usually all the instances are given in input to the perceptron, the weights are adjusted and then the process is repeated for a certain number of times (this is known as the propagation algorithm). The algorithm stops with many criteria, such as: i) after a certain, fixed number of steps, ii) when the error rate is smaller than a given threshold, iii) when no improvement on the error was experienced after a given number of steps. One of the advantages of multilayer perceptrons, is that when some instances are added to the training set, it is not necessary to rebuild the neural network from scratch. The perceptron can be tuned by adding instances, which cause the weights to be modified, without running the propagation algorithm again for older instances. Nevertheless, building

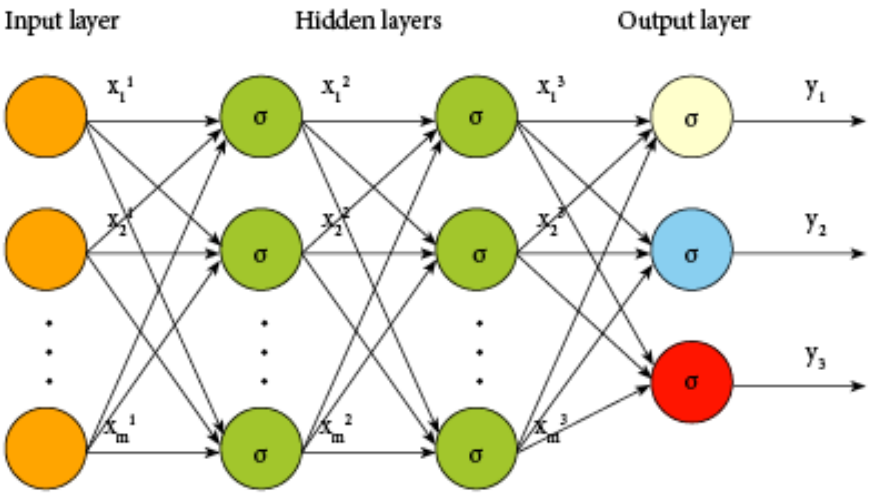


Figure 2.7: A multilayer perceptron with two hidden layers and three output nodes.

a neural network is usually a slow process, and the time complexity in the worst case can be exponential in the number of instances. The topology of the network also plays an important role, and can influence both the building time and the accuracy of the perceptron. A network with too many nodes can result in overfitting; on the contrary, a network with too few nodes can result in poor prediction accuracy.

Statistical classifiers

The training set can be used to build a probability model, which is able to estimate the probability that a certain instance belongs to a class. The most common statistical classifier is the **Naive Bayes** classifier [48]. It can be described as a network composed of directed, acyclic graphs with one parent and many nodes, where the parent is the unobserved node, and the children are the observed nodes, which are assumed to be independent. The probability that an instance A composed of n features $x_1, x_2, \ldots, x_n$ belongs to the class C_j can be estimated as:

$$P(A \in C_j) = P(C_j) \cdot \prod_{i=1}^{n} P(x_i|C_j) \tag{2.12}$$

Where $P(x_i|C_j)$ is the probability that an instance labeled with the class C_j has x_i as value of its i-th feature. Applying the Bayes theorem, the following formula can be obtained:

$$P(A \in C_j) = P(C_j) \cdot \prod_{i=1}^{n} \frac{P(C_j|x_i) \cdot P(x_i)}{P(C_j)} \tag{2.13}$$

$P(C_j|x_i)$, $P(x_i)$ and $P(C_j)$ stand respectively for: i) the probability that if an instance has x_i as value of its i-th feature, then it belongs to the class C_j, ii) the probability that the i-th feature of any instance has x_i as value, iii) the probability to find an instance labeled with the class C_j. All these probabilities can be extrapolated from the training data. Then, to label an instance it is enough to compute the probability $P(A \in C_j)$ for every class, and to pick the class whose probability is the highest.

The Naive Bayes classifier is simple and it requires a very short amount of time to be built, but it has an important limitation: it is based on the assumption that the child nodes are indepedent, which is rarely true. for this reason, this classifier has usually a low accuracy compared with other classifiers.

A more general class of statistical classifiers are **bayesian networks**, which are a superset of Naive Bayes classifiers. A bayesian network can be represented as a directed acyclic graph, where nodes represent the features, and arcs represent a relationship of dependence between two features. If there is no link between two nodes that correspond to the features x_1 and x_2, it means that the two features are independent. Statistically speaking, it means that given another feature x_3, then: $P(x_1|x_2, x_3) = P(x_1|x_3)$, for all the possible values for x_1, x_2, x_3. If instead there is an arc directed from x_2 to x_1, then it means that the value of x_1 is dependent on the value of x_2, and it is possible, given a third feature x_3 which has an outbound node adjacent to x_1, to calculate the conditional probability of the value x_1 for every configuration

of x_2, x_3. A bayesian network that fits a dataset of independent features is equivalent to a Naive Bayesian network, where the children of a node are not connected among them.
To build a bayesian network the first step is to determine its topology, in case that it is not already known. The number of possible network topologies is exponential in the number of features, therefore it is necessary to adopt an algorithm that tries to estimate the best topology, which is a good compromise between accuracy and time. Some approaches have been adopted: i) a scoring function is used to evaluate the "fitness" of a given network topology over the training data, then a local search can be used to find the best topology according to this score [49, 50], ii) using statistical tests such as *chi-squared* and *mutual information* test, the conditional independence between features can be inferred, thus obtaining a set of constraints that can be used to build the network [49]. Once a network has been built, the second step is to determine the conditional probability parameters $\big($e.g. $P(x_1|x_2, x_3)\big)$, which should be inferred from the dataset; they are usually stored in a table, called *conditional probability table* (CPT). The resulting bayesian network is able to return a label C_j, such that maximizes the probability $P(C_j|x_1, x_2, \ldots, x_n)$.
The advantage of bayesian networks is that it is possible to take into account domain-related information about a classification problem, in terms of dependencies between features (e.g. asserting that two features are correlated, or that a node is a leaf node, etc ...). But this comes with some limitations: first of all, a bayesian network is rarely fit for a training set with continuous variables, which are often needed to be discretized. Moreover, since the building time and the required space of a bayesian network is exponential, such model is unsuitable for training sets with a large number of features.

Instance-based learning

Lazy-learning algorithms delay the computation required to generalize the dataset until the classification is performed. They have the advantage

of requiring less time during the training phase, but they are slower during classification. The most popular instance-based learning algorithm is the **k-nearest neighbors algorithm (kNN)**, which is based on the assumption that instances that belong to the same class have similar underlying properties [51]. If this assumption is true, then to classify an unlabeled instance, a similarity measure that takes as input the features of two instances can be used to find the k instances that are closer to the instance that is being classified. The output is the most frequent class among these k instances. Many measures can be adopted, such as i) the Manhattan distance, ii) the euclidean distance, iii) the Minkowsky distance, iv) the Chebychev distance, v) the Canberra distance, vi) Kendall's rank correlation. Usually, a weighting scheme is adopted to alter the influence of each instance, which has to be tuned for the dataset that is being used [52]. Since the instances need to be mantained in memory in order to run the kNN algorithm, memory is clearly one of its most important limitations. This problem can be mitigated by using instance-filtering algorithms able to reduce the dataset while not significantly affecting the accuracy [53, 54]. Since the required time for classification is proportional to the number of instances, instance-based filtering has also a beneficial impact on the classification time. Another problem of the kNN algorithm is its dependence on the choice of k parameter, which can lead to different classification results of the same instance. The choice of the k parameter is even more crucial when the instance that is being classified is in a location affected by heavy noise (the location of the instance can be seen as its position in an n-dimensional space defined by the value of its features). In this case, the highest the k parameter, the lowest is the chance that noisy instances win the majority vote, leading to an incorrect classification. On the contrary, in the case that the instances belonging to a class are overnumbered by the instances of another class which are located in the surrounding region, the latter class can mistakenly win the majority vote if the value of k is too high. Wettschereck *et al.* [52] concluded that in case of noisy instances, the accuracy of the kNN algorithm is not sensitive to

the choice of k when k is large. Moreover, they found that the performance of kNN with $k = 1$ (1NN) is superior compared with higher choices of k for small datasets (with less than 100 instances). On the contrary, for medium and big datasets, 1NN is outperformed by kNN algorithms with higher values of k. Another factor that affects the performance of kNN is the choice of the distance measure, which usually affects the performance in different ways, according to the underlying properties of the training set. To summarize, the main problems of the kNN classifier are the large computational and space requirements for classification, and its sensitivity to the choice of the distance measure and the k parameter. Another disadvantage is the due to the possibility that a particular configuration of the kNN algorithm is "unstable", in the sense that a small change in the training set can lead to a very significant change in the resulting classifier. Nevertheless, delaying the efforts until the classification process makes instance-based learning a very flexible approach, since instances can be added without requiring to rebuild the model, which can be time costly in the case of other classifiers such as multilayer perceptrons and decision trees.

Support vector machines (SVMs)

This is the most recent machine learning approach [55, 56, 57], and it is based on the concept of *"margin"*, which is, given the $\mathbb{R}^n$ space where every axis represents a feature (n is the total number of features), the separation gap between two linearly separable classes C_1, C_2 in a dataset with m instances $x_1, x_2, \ldots, x_m$ and m labels $y_1, y_2, \ldots, y_m$. It has been demonstrated that the classification error can be minimized by maximizing the distance between the hyperplane and the instances on either side of the hyperplane. The problem consists in finding a pair $\big(w, b\big)$ such that:

$$\begin{aligned} w^T \cdot x_i + b \geq 1, \forall i \mid x_i \in C_1 \\ w^T \cdot x_i + b \leq 1, \forall i \mid x_i \in C_2 \end{aligned} \tag{2.14}$$

Where w is the weight vector and b is the bias. Then, the class is given

by the sign of $w^T + b$, which is positive if the instance belongs to C_1, and negative if the instance belongs to C_2.
When the two classes are linearly separable, it is possible to find an optimum hyperplane which maximizes the distance between the instances of the two classes and the hyperplane, by minimizing the squared norm of the separating hyperplane:

$$\begin{aligned} &\underset{w,b}{\text{minimize}}\ \Phi(w) = \frac{1}{2}\|w\|^2 \\ &\text{subject to } y_i(w^T \cdot x_i + b) \geq 1, i = 1, \ldots, m \end{aligned} \tag{2.15}$$

Given the optimum hyperplane, points that lie on the margin are called *support vectors*. The solution is represented by a linear combination of these vectors, while all the other vectors can be ignored. Since the number of support vector points are usually small, at classification time the SVM model complexity is unaffected by the number of features encountered in the training set. This makes SVMs suitable for problems where the number of features is large if compared with the number of instances in the training set.
In most of the problems, the training set is usually affected by a certain number of misclassified instances (e.g. due to noise); in such case, the SVM could not be able to find any separating hyperplane. Veropoulos *et al.* [58] addressed this problem through the use of a *soft margin*, which accepts some misclassifications of the training instances. A soft margin can be adopted by introducing slack ariables, in order to tolerate an error ξ, which would turn the inequalities in 2.14 to:

$$\begin{aligned} w^T \cdot x_i + b &\geq 1 - \xi, \forall i \mid x_i \in C_1 \\ w^T \cdot x_i + b &\leq 1 + \xi, \forall i \mid x_i \in C_2 \\ \xi &\geq 0 \end{aligned} \tag{2.16}$$

For a misclassification to occur, the error should be greater than 1. This means that $\sum_i \xi_i$ is the maximum number of training errors that can be tolerated.
Nevertheless, the SVM algorithm can only be applied to linearly separable

problems. In real-world problems such as human activity recognition, the training data is rarely linearly separable. This problem can be solved by mapping the data into a higher-dimensional space where the data is linearly separable. Then, the problem is solved by finding a separating hyperplane in this space (also called the *transformed feature space*), and any instance has to be mapped there before being classified. If the data can be mapped to a Hilbert space H throught a mapping function Φ, then to build an SVM it is necessary to calculate dot products on the mapped features of every instance in the training set. If there is a *kernel function* K, such that $K(x_i, x_j) = \Phi(x_i) \bullet \Phi(x_j)$, then the kernel function can be used in the training algorithm, thus eliminating the need of determining Φ and calculating inner products directly in feature space, as described in [56]. To determine the most appropriate kernel function, it is common practice to try the algorithm on different classes of kernel functions, and then using cross-validation to determine which function is the most suitable. For this reason, one of the most serious limitations of SVMs is the low speed during the training phase. The training phase can be reduced to a quadratic programming (QP) problem, whose complexity depends on the number of instances in the training set. Solving a QP problem requires to perform mathematical operations on large matrices, and it can be very slow for large datasets. **Sequential minimal optimization (SMO)** is a simple algorithm that requires less computation than traditional SVM algorithms, which decomposes the QP problem into a set of QP sub-problems, and requires a relatively low amount of time and space.

To summarize, SVMs have the advantage of being very fast during classification, and they are suitable for datasets with a large amount of features compared to the number of instances. Nevertheless, they are slow during the training phase. There is a workaround to adapt SVMs to non-linearly separable datasets, but they still can solve only binary classification problems. To adapt SVMs to multi-class problems, it is necessary to use a set of SVMs, one for every pair of classes. Unlike neural netoworks, SVM clasifiers are not

subject to the problem of local minima, as it is guaranteed that they will always find a global minimum.

Ensemble classifiers

An ensemble of classifiers, which is either composed by many versions of the same classifiers built in different ways (on different partitions of the training data, or varying the model parameters), or a collection of different classifiers, can be used to obtain more accurate prediction results. Every classifier makes its contribution by producing an output, and then a voting algorithm, either weighted or not, is run to choose the final output. The first category of ensemble methods will be described.

Bagging [59] is a an esemble method that relies on a single classification algorithm, called *inducer*, which is trained on many *boostrap samples* of the training set. A boostrap sample [60] is generated by uniformly sampling m instances from the training set with replacement. The sampling is repeated T times, and for each time an instance of the inducer classifier is generated. At the end of the process, a set of T classifiers is obtained, which is used to classify unlabeled instances through an unweighted voting algorithm. This technique relies on the instability of the inducer classifier, which produces different results if it is trained on a perturbated version of the original dataset. neural networks are a good example of unstable classifiers, and they generally produce good results if used as inducers of a bagging classifier. On the contrary, the performance of bagging classifiers is degraded if stable algorithms (e.g. kNN) are chosen as inducers.
Another similar ensemble method is **boosting** [59], which similary to bagging, relies on the concept of bootstrap sampling, with the difference that the classifiers are generated sequentially rather than in pararrel, and that the instances in the samples are weighted. Among the most popular boosting methods, there is the *AdaBoost* algorithm (adaptive boosting), which requires an inducer algorithm that supports weighted instances. For this purpose, the Naive Bayes and the multilayer perceptron algorithms may be

used, since they support weighted instances. In [59], a modified version of a decision tree algorithm is adopted, which supports weighting. Initially, the AdaBoost algorithm sets all the weights to 1, and a variable S' is used, which is initially equal to the whole training set. At each step, a classifier is generated using S' as input. The process of building the classifier might need to be repeated on a random boostrap sample of the training set in case that the classifier weighted error on the training set is is greater than 50%, up to a maximum number of 25 times. When a classifier with an error rate smaller than 50% is obtained, the correctly classified instances in S' are reweighted with a factor inversely proportional to the error rate on the training set, let is be β_i, which is always smaller than 1. Then the weights are normalized, and both the generated classifier and the value β_i are recorded. This means that correctly classified instances will have a smaller weight at the next iteration, while misclassified instances will have a greater weight, thus having more influence on the learning algorithm. When T steps are over, T classifiers $C_1, C_2, \ldots, C_T$ are obtained, the output of the resulting classifier $C^\star$ is obtained with a logarithmic weighted majority vote, dependent on the value β_i obtained at each step:

$$C^\star(x) = \arg\max_{y \in Y} \sum_{i:C_i(x)=y} \log \frac{1}{\beta_i} \tag{2.17}$$

This produces a more accurate classifier, whose performance is generally better than the performance of a single classifier. Despite this, due to the fact that misclassified instances are assigned a greater weight at every iteration, unlike bagging, boosting is very sensitive to noisy instances.

Another ensemble method is the **random forest** algorithm [61], which is based on decision trees. The underlying principle of the random forest algorithm is that an ensemble of smaller decision trees build on a subspace of the feature space, may generalize better than a single decision tree built on the whole feature space. Random forests are built using oblique decision trees, which are trees built choosing more than a feature at every nonterminal node. Since the decision trees must be smaller than a single decision

tree that fits all the feature space, usually a loose stopping rule is adopted. Criteria such as *central axis projection* and perceptron training (as described in [61]) are used to select the set of features that best divide the data at each nonterminal node. A certain fixed number of decision trees is built, using a randomly chosen subspace for each tree. Given a feature space of n features, there are 2^n possible combinations of features. If n is large enough, then it is not necessary to build a tree for every possible combination of the features, but a smaller number (e.g. 20) was demonstrated to be enough for most of the problems. A *discriminant function* combines the output of all the decision trees in the forest, and produces a final output. Random forests are usually more accurate than single decision tree, and their accuracy is proportional to the size of the forest. Another advantage of this algorithm is that it eliminates the problem of overfitting to which single decision trees are subject.

As for the question of finding the best machine learning algorithm, there is no general answer. It depends on what criterion is chosen to evaluate the learning algorithm such as accuracy, classification speed, learning speed, sensitivity to overfitting, etc... and many of these parameters depend on the nature of the data that is used to train the algorithm. For example, the Naive Bayes methods fits well will small datasets, and it is insensitive to noisy and missing data. On the contrary, given its nature, kNN is very sensitive to noisy and missing data. Neural networks, decision trees and SVMs are subject to the problem of overfitting, which in the case of decision trees is solved by using the random forest method. The Naive Bayes and the kNN algorithms are the fastest to train. Indeed, the required training time of a kNN algorithm is zero: it is enough to store the training set, and the work is deferred at classification time. This comes at the price of increasing the classification time and the required memory storage, which is exactly the storage required to keep the whole training set in memory. This problem does not affect the Naive Bayes algorithm, which requires low memory storage but it is usually less accurate than other algorithms due to the fact that

it is based on the assumption that the features are independent, which is usually false for real-world problems. Ensemble methods tend to be more accurate than their inducer algorithms, and they solve some of their issues, but they are generally more expensive. Moreover, in certain cases some ensemble methods may perform worse than their inducer algorithms. For instance, the AdaBoost algorithm, since it gives more weight to misclassified instances, tends to perform worse if trained with noisy datasets. For this reason, there is no single best training method, and the choice should be done case-by-case, considering the performance requirements and the nature of the training data.

2.3 Evaluation of a HAR system

Many aspects should be considered in order to evaluate a human activity recognition system [19]. Among the most important ones, there are:

- **Obtrusiveness**. Requiring the users to wear obtrusive sensors that limitate or condition their movements, or requiring too much user interaction, has a negative impact on the system. Discomfort is not the only consequence of an obtrusive recognition system; indeed, limiting user movements or requiring the user to interact too often with the system affects the quality of the collected data, which should be as close as possible to real data, collected in an environment where the users practice their activities naturally and spontaneously, without being influenced by the HAR system. Moreover, obtrusiveness is directly linked with *pervasiveness*: the capacity to persuade users to participate to the data collection process. The more users participate, the more data is collected, and the more the training data will be valuable.
- **Flexibility**. The recognition system should adapt to different users who perform activities in different ways (e.g. an elderly would not run as fast as a young person). A recognition model is said to be *subject-dependent* or *personal* if the training phase is executed for every

user; this is not always feasible because not every user is willing to cooperate in the trainign process, and because in certain cases repeating the training phase is not desirable (e.g. in a fall detection system). For this reason, an alternative is to build a unique model able to recognize the activities of every user, which is also called an *impersonal* model. This last approach comes with a price in terms of accuracy, because such system wouldn not keep account of the difference in which the activities are carried out by different users. An alternate approach that takes all the above mentioned problems into account, is to build per-group models.

- **Energy consumption**. Most of the hardware used to implement HAR systems is energy-constrainted. Extending the battery duration is desirable for devices such as mobile phones and wearable sensors; for this reason, it is preferable to turn on sensors such as the GPS only when it is strictly necessary (e.g. when the user is not resting). Moreover, communications have a great impact on energy consumption, and it is preferable to use low energy communication protocols. Also the architecture of the system, the number of sensors sampled, the number of features included - in case that the feature extraction is carried out directly in the device - and parameters such as sampling frequency and window length play an important role. The best scheme is usually a compromise between energy needs and other requirements.

- **Recognition performance**. The accuracy of the activity recognition model, together with other metrics that will be elaborated in detail afterwards, depend on many factors, such as the type of problem (e.g. a binary classification problem that should discriminate between resting and moving is much simpler that a complex problem that involves dozens of different activities), the quantity of data collected, the learning algorithm and the feature selection.

2.3.1 Validation

Validation is the procedure of verifying the performance of a human activity recognition model, according to some metrics that will be explored later in detail. Since the model is ought to make good predictions on unseen data, testing the performance on the same dataset used in the training phase is not an option. For this reason, many techniques may be used to validate a recognition model:

- **Holdout method**. The dataset is divided into a training set and a *test set*. The training set is used to build the model, while the test set is used for validation. The problem of this method is that it has a high variance, because the evaluation depends on which instances are included in the training set and which in the test set.
- **K-fold cross validation**. The data is partitioned into k subsets, and the validation is repeated k times, using the i-th subset of the data at the i-th iteration for validation, and the other subsets to train the model. The overall performance is averaged over the k iterations. This overcomes the problem of the holdout method of having a high variance, but since it requires k iterations it is slower than the holdout method. In multi-user HAR datasets, it is often convenient to use the *leave-one-subject-out* cross validation: a k-fold validation technique where k is equal to the number of users, and every subset includes the data of a single user.
- **Leave-one-out cross validation**. This is a k-fold validation method carried to the extreme where k is equal to the number of instances in the dataset. It is significantly slower than the holdout method and a cross validation method with smaller values of k.

2.3.2 Evaluation Metrics

Given a binary classifier, with reference to the values defined in table 2.2, the following metrics can be used to evaluate its performance:

True positives (TP)	The number of correctly classified positive instances
True negatives (TN)	The number of correctly classified negative instances
False positives (FP)	The number of misclassified negative instances
False negatives (FP)	The number of misclassified positive instances

Table 2.2: Values defined for a classification

- **True positive rate (recall)**. The true positive rate is the number of true positives divided by the total number of positive instances:

$$Recall = \frac{TP}{TP + FN} \tag{2.18}$$

- **True negative rate**. The true negative rate is the number of true negatives, divided by the total number of negative instances:

$$TNR = \frac{TN}{TN + FP} \tag{2.19}$$

- **Accuracy**. The most standard metric used for evaluation. It is defined as the ratio of correctly classified instances to the total number of instances:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.20}$$

- **Precision**. Precision is the ratio of true positives to the total number of instances classified as positive:

$$Precision = \frac{TP}{TP + FP} \tag{2.21}$$

- **F-measure (or F1 score)**. This metric combines precision and recall in a single value:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.22}$$

Although these metrics are defined for binary classification problems, they can be generalized for a problem with n classes, where an instance is positive

if it belongs to a certain class i, and negative if it does not belong to i . In this case, the classification results can be summarized in a **confusion matrix**, which is a matrix M of $n \times n$ elements, where n is the number of activities that can be recognized, and M_{ij} is the number of instances that belong to the class i that were classified as belonging to the class j.

2.4 HAR Systems and Studies in Literature

Similar studies have shown that it is possible to detect human activities using the previously mentioned techniques with good prediction results. For this study it is convenient to divide them into four categories, according to the hardware that was used to detect the activities, namely i) wearable sensors, ii) smartphones, iii) smartwatches, iv) miscellaneous.
Some studies are based on the **PAMAP2 dataset** [62, 63], which contains data collected from 9 subjects, using three wireless accelerometers with a sampling frequency of 100 HZ, placed on the dominant wirst and ankle, as well on the chest. The activities included in this dataset are: lying, sitting, standing, walking, running, cycling, Nordic walking, watching TV, computer work, car driving, ascending stairs, descending stairs, vacuum cleaning, ironing, folding laundry, house cleaning, playing soccer and rope jumping. The age of the subjects was 27.22 ± 3.31 years, with a BMI of 25.11 ± 2.62 kgm^{-2}. The dataset contains ten hours of data altogether, of which 8 hours were labeled.

2.4.1 Wearable Sensors

Lara and Labrador [19] presented a survey on human activity recognition using wearable sensors, illustrating its practical uses and reviewing the current techniques, architectures and the limitations of HAR systems. The article explores in detail general aspects and implementation choices of HAR systems, such as sensors and feature selection, common machine learning algorithms, performance metrics and evaluation aspects of HAR systems.

Bao and Intille [30] built a human activity recognition system, collecting data from 20 subjects for 20 different daily-life activities, in a semi-naturalistic environment. The activities included in the study are: walking, sitting and relaxing, standing still, watching TV, running, stretching, scrubbing, folding laundry, brushing teeth, riding elevator, walking carrying items, working on computer, eating or drinking, reading, bicycling, strenght-training, vacuuming, lying down and relaxing, climbing stairs and riding escalator. Five biaxial acceperometers were mounted on different body locations, and data was collected at a sampling frequency of 76.25 HZ, extracting the following features: mean, energy, frequency-domain entropy and correlation, using windows of 256 samples (6.7 seconds). Different meachine learning techniques were used: decision table, instance-based learning, decision tree and naive Bayes. The decision tree algorithm resulted the most accurate, obtaining an accuracy of 84% with the leave-one-subject-out cross validation. Using only two accelerometers caused the accuracy to drop slighly, decreasing of just 3.27% $\pm$ 1.062 with an accelerometer placed on the thigh and one on the wrist. This was the first study that proved the feasibility of using accelerometers to recognize household activities for context-aware computing.
A study based on the PAMAP2 dataset was conducted by Arif and Kattan [64], who extracted features from the dataset with 5 seconds windows with 1-second overlaps, obtaining a dataset of 18,664 instances. The dataset was partitioned into three feature sets, containing the features of the accelerometers placed in the wrist, in the chest and the ankle (FS1, FS2 and FS3 respectively). Three prediction models were build using the kNN algorithm, the rotation forest algorithm (a variant of the random forest algorithm), and neural networks. The models were then validated using the holdout method, with a training set containing 70% of the instances, and a test set containing the remaining 30% of the instances. Three metrics were used for the evaluation: precision, recall and F-measure. Then, the models were built again using all the feature sets, merging FS1, FS2 and FS3 in a unique feature set. The study showed that the models built on a single-accelerometer feature set,

had a great performance variance among different activities, due to the fact that some activities can be confused if only one accelerometer is used (e.g. standing, ironing and vacuum cleaning were confused if the accelerometer was placed on the ankle, while not if the accelerometer was placed on the wrist). Moreover, the kNN algorithm resulted in a great discrepancy between precision and recall, and between precision and F-measure. This is due to the fact that some activities had a high ratio of true positives, but also a large number of false negatives, resulting in low values of recall and F-measure, and higher values of precision. Tested on FS1, FS2 and FS3, kNN was the worst method, which obtained a precision of over 86.5%, but a recall ranging from 52.5% to 77.1%. The best method was the rotation forest, which obtained a precision and a recall in between 92.1 and 94.1%. The results of the models built on the merged fearture sets, show that the high variability between the performance of different activities, and the discrepancy between precision and recall of the kNN algorithm, were both eliminated, obtaining an average recall of 98.2% with the kNN and the rotation forest method.

Gyllensten and Bonomi [65] demonstrated the lack of accuracy of laboratory-trained recognition models if tested on free-living data. Data from 20 subjects was collected using a single waist-mounted accelerometer, during a laboratory trial. Three recognition models were trained on the laboratory data: an SVM, a neural network and a decision tree. The models were tested using the leave-one-subject-out cross validation, obtaining an accuracy of 95.1 $\pm$ 4.5%, 91.4 $\pm$ 6.7% and 92.2 $\pm$ 6.6% respectively. The same algorithms were tested on free-living data, showing a significantly decreased prediction accuracy. The algorithms obtained an accuracy of 75.6 $\pm$ 10.4%, 74.8 $\pm$ 9.7% and 72.2 $\pm$ 10.3%, validating the assumption that algorithms trained on laboratory data underperform if used on daily-activities data.

Another study conducted by Ellis *et al.* [66] further confirms this assumption, showing the poor performance of algorithms trained on data collected in a controlled environment, if tested against free-living data. Two datasets were used for this study: a controlled dataset, and a free-living dataset containing

data collected by 40 cyclists in their daily activities during 3-4 days, using an accelerometer, a GPS and a SenseCam device, with 60 seconds windows and 30-seconds overlaps. A random forest trained and tested on the controlled dataset with the leave-one-day-out validation obtained an accuracy of 94%, whereas the same algorithm tested on the free-living dataset scored an accuracy of just 70.9%. A random forest algorithm trained and tested on the free-living dataset obtained an accuracy of 89.2%, showing that classification of free-living activities is more complex due to high data variability, other than reinforcing the previously stated conclusion.

2.4.2 Smartphones

An overview of the state of the art of physical activity recognition through smartphones was presented by Morales and Akopian [67], surveying the relevant signals used in physical activity recognition, techniques of data acquisition and preprocessing, and methods to deal with the unknown location and orientation of the phone. Moreover, the article covers the topics of feature selection, activity models and classifiers, and reviews metrics for quantifying activity execution, along with ways to evaluate the usability of a HAR system.

Miao *et al.* [38] proposed a system to identify physical activities with a smartphone, regardless to the phone position and orientation. Seven subjects collected data for five physical activities: static, walking, running, walking upstairs and downstairs, carrying the phone in six possible positions, and varying the orienation. Orientation-independent signals like the acceleration and rotation rate magnitude were chosen, along with the magnetic orientation components on the three axises. The signals were sampled at 25 HZ, with 1.6 seconds windows and 50% overlaps. A proximity and a light sensor were used to determine whether the phone was on the pocket or not, in order to trigger the data collection automatically. The collected data was elaborated using the WEKA machine learning toolkit [68], testing three algorithms using the k-fold cross validation: decision tree, naive Bayes and SMO, scoring

an accuracy of 89.6%, 75.3% and 81.1% respectively. J48, the decision tree algorithm implementation of the WEKA toolkit, was the most accurate. The study demonstrated the possibility of building accurate, efficient and robust HAR systems, able to recognize physical activities regardless of the phone position and orientation.

Mitchell *et al.* [69] proposed a framework for automatic recognition of sport activities using smartphones. A phone was placed on the upper cervice of the users' back to collect data in order to recognize different activities in five-a-side soccer and field hockey. The included activities were: stationary, walking, jogging, spriting, hitting the ball, standing a tackle and dribbling the ball. The features were extracted using the discrete wavelet transform (DWT), and different configurations of DWT decomposition level, mother wavelet, window length and classifier were tested. The included classifiers were: SMO, kNN, bayesian networks, decision tree and multilayer perceptron. No classifier was shown to be more accurate than the others in a relevant way, except for the SMO that performed poorly for the soccer activities. Similar activities such as hitting the ball, standing a tackle and dribbing the ball were confused with each other, due to the fact that they involve similar movements. Finally, a fusion of classifiers was tested, using a different classifier for each activity. With this method, the highest accuracy was achieved: 86.3% for soccer and 88.8% for field soccer.

Bedogni, Di Felice and Bononi [14] developed a system to train and run a transportation mode recognition model, in order to enrich context-based services, switching on and off different sensors according to the motion type. The system relied on an Android application, WAID (What Am I Doing), which was able to collect accelerometer and gyroscope data, and to perform an online training in order to build a transportation mode classifier, with three different activities: walking, moving by train and moving by car. The classifier was then used in the WAID application to detect motion types, automatically switching the phone configuration, allowing to associate a profile with each type of motion, in order to turn off or on cellular data, Bluetooth,

WiFi and other sensors automatically, to provide an enhanced experience. 72,000 samples for each activity were collected, corresponding to 2 hours of data collection, without any restriction on the phone position and orientation. Three classifiers were trained: an SVM, a naive Bayes model and a random forest. The models were tested with different configurations, switching the window length and the sampling frequency. The window length resulted to be unrelated to the accuracy, while an important finding was reported about the sampling frequency: at the condition that the sampling frequency during the training phase and the predicting phase are the same, the sampling frequency is unrelated to the prediction accuracy of the trained classifier, opening thus the possibility of using lower sampling frequencies in order to minimize energy consumption. The random forest method was found to be the best, scoring a prediction accuracy of 97.71%. Also the use of a history set was investigated, which allowed to save the last n predictions in order to increase the overall accuracy, outputing the most frequent prediction inside the history set during each classification. The history set was found to increase the overall recognition accuracy in normal conditions, at the cost of reducing the accuracy during transitions.

A similar and more recent study was carried out by Testoni and Di Felice [70], who implemented a generic human activity recognition system for Android devices, taking into account energy constraints by limiting the amount of features and sensors included, and by delegating the most CPU-intensive tasks to an external machine. The proposed system automatized the full learning process, allowing the users to select the activity labels and the sensors to include in data collection. In training mode, the system was able to automatically extract features and send them to the external machine, ready to be stored in a MongoDB NOSQL database, and to be analyzed with the WEKA machine learning library. The system was able to train a random forest model on the stored data, and send it to the mobile client application. In detection mode, the installed model was run on the client application, and activity transitions were detected. It was possible to use a history set

to render activity transitions smoother. The system was afterwards used to collect data and train classifiers for two use cases: i) for transportation mode detection, discriminating between walking, traveling by train, by bus, by car or performing another activity. The achieved accuracy was around 92% with the random forest method, ii) for walking mode detection, discriminating between standing still, walking, and climbing/descending stairs. The random forest method achieved an accuracy of 96%. In both studies, the use of a history set allowed to score a higher accuracy, going up to 100% when the history set size was large anough.

2.4.3 Smartwatches

A comparative study between smartwatch and smartphone-based activity recognition was presented by Weiss *et al.* [71], which suggests that smartwatches are more accurate at recognizing some hand-based activities, and that similarly to smartphones, personal models outperform impersonal models. 18 activities were included in the study, which were grouped into three categories: 1) *Not hand-oriented general activities*: walking, jogging, climbing stairs, sitting, standing, kicking soccer ball, 2) *Hand-oriented general activities*: dribbling basketball, playing catch with tennis ball, typing, handwriting, clapping, brushing teeth, folding clothes, 3) *Eating activities*: eating pasta, eating soup, eating sandwich, eating chips, drinking from a cup. 17 subjects collected data for a duration of 2 minutes for each activity, using the accelerometer and the gyroscope of a LG G Watch and a Samsung Galaxy S4. Issues with the phone gyroscope prevented sufficient data from being collected, therefore the phone gyroscope data was not included in the study. The sensors were sampled at 20 HZ, extracting features every time window of 10 seconds, without overlaps. The data was then imported in the WEKA machine learning toolkit, studying each sensor for each device separatedly, testing several classification models: random forest, decision tree, IB3 (instance-based learning), naive Bayes and a multilayer perceptron. The

personal models were validated using the 10-fold cross validation, while the impersonal models were validated with the leave-one-subject-out validation. The radom forest method resulted the most accurate, scoring an accuracy above the average of the other classifiers for most of the activities. Considering only the models built using accelerometer data, the accuracy obtained with the personal random forest models was 75.5% for the phone and 93.3% for the watch, while the impersonal models obtained respectively 35.1% and 70.3%. This confirmed the hypothesis that personal models outperform impersonal models. Moreover, since the study included many hand-oriented activities, the watch models obtained a better accuracy, specially for activities in the second and third category. Indeed, the most confused activities with the phone model were eating activities, more precisely "eating a soup" was often confused with "drinking", because the acceleration caused to the phone in the pocket is similar. Another important finding was that personal models particularly increase the performance of eating activities, suggesting that there is a wide variance in how people eat.

Ahmad *et al.* [72] published an analysis of various feature and classification methods in order to find the optimum settings of physical activity recognition, with both personal and impersonal models. 6 healthy subjects collected data for more than a month, performing the following activities: walking, walking upstrairs, walking downstairs, running and jogging. The following classification models were tested: SVM, kNN (k = 10), bagging, decision tree, and naive Bayes. Two studies were carried out: in the first study, fixed-sized windows of 75 samples were used, and the performance of two different feature banks was tested, comparing the performance obtained with random permutation and normalization of the features, opposed to the case where no random permutation/normalization was applied. In the second study, the performance of the models built with different window sizes were compared, ranging from 25 to 300 samples. The study showed that feature random permutation and normalization increased the overall performance of the classifiers, and it was determined that the same features used in a previous study

on smartphones could be used in the case of physical activity recognition with smartwatches. The most performant models were SVM and decision tree, scoring an accuracy up to 98%. Moreover, personal models were found to be more accurate than impersonal models. The window size did not affect the performance of the SVM and decision tree classifiers, while it was found that increasing the window size decreased the performance of the kNN classifier. This was due to the fact that the kNN classifier has an accuracy that depends on the number of instances in the dataset, and increasing the window size using the same dataset causes the number of instances to decrease. A different approach was followed by Garcia *et al.*, who used a GENEActiv wristwatch to recognize long-term activities using hidden Markov models and conditional random fields. Two subjects collected data from the watch triaxial accelerometer for 11 and 10 days respectively. The activity labels of the first subject were: shopping, showering, dinner, working, commuting, brush teeth, and they had many ways of being performed: commuting could be performed by walking or using a vehicle, the "dinner" label included having lunch or breakfast, "showering" included dressing up, etc... the labels of the second subject were: commuting, lunch, working time, exercise, and another not tagged activity. Various features were extracted sampling the accelerometer at 20 HZ, using 3 seconds window with 33% overlaps. The activities were decomposed into strings of more primitives, which represented simple activities that composed a long-term activity. This study made use of *clustering* in order to produce a certain number of groups, such that instances that are similar belong to the same group. Various configurations were tested, using hidden Markov models and conditional random fields, adding prior knowledge regarding the duration of the activities and using clustering to further subclass fragmented activities in order to get more precise labels (e.g. commuting was a fragmented activitity because it was possible to commute by walking or using a vehcile), obtaining accuracy scores up to 77.1% using the leave-one-day-out cross validation. The study puts light on the possibility of using a different approach in order to classify long-term activities which

are composed by a sequence of simpler activities, obtaining an acceptable recognition performance.

2.4.4 Miscellaneous

A combined approach was followed by Ramos *et al.* [73], in order to demonstrate that using a smartwatch in combination with a smartphone increased the activity recognition performance. Accelerometer data was collected from 13 subjects using a Sony SmartWatch 2 SW2 and a Sony Smartphone Xperia Z1, for four types of activities: walking, sitting, standing and driving. Three features vectors were tried: standard deviation (STD), arithmetic mean (AM), and both (SA). For each activity, each subject collected one minute of data, which was afterwards analyzed and processed using R and the WEKA machine learning toolkit. Three machine learning algorithms were used: naives Bayes, SVM and decision tree, with the 10-fold cross validation computing the average accuracy over 30 runs. The accuracy of the recognition model that relied only on the smartphone was compared with the accuracy obtained using both the smartwatch and the smartphone; a t-test and a Wilcoxon test were performed in order to determine if there was an increase in the accuracy performance with 95% of confidence, in the case where both devices were used instead of using only the smartphone. For the decision tree model, it was determined that the smartwatch increased the recognition accuracy only for STD, but not for AM and SA. For the SVM classifier, it was determined that there was an increase in recognition accuracy for AM and STD, while for naive Bayes model the dependency was true for all the types of feature vectors. Therefore, for 6 configurations by 9 the recognition accuracy was increased using the smartwatch with 95% of confidence. The recognition accuracy of the combined approach for SA reached the value of 88.47% with the SVM algorithm, and 87.33% with decision tree. Al-Naffakh *et al.* [74] proposed an architecture for performing human activity recognition in the context of transparent authentication systems (TAS), in which the accelerometer, gyroscope and GPS sensors are used to distinguish

the true device user from imposters, relying on the fact that different subjects tend to perform activities in different ways. 10 subjects collected data wearing a Microsoft Band 2 (a wristband with smartwatch features), walking in two five-minutes sessions on a flat floor, on two different days. Accelerometer and gyroscope data was sent to a smartphone via Bluetooth, and 88 features were extracted for every 10-seconds window, with no overlaps. Afterwards, the average euclidean distance from other subjects and from self was measured, separatedly for the accelerometer and the gyroscope. Two scenarios were considered: same-day and cross-day. The first finding was that, as expected, for every subject the average euclidean distance from other subjects was greater than the average distance from self. The gyroscope resulted less reliable than the accelerometer because it resulted in a higher intra-subject variance of the euclidean distance. Moreover, the study showed that the average euclidean distance from the same subject and from others is greater in the cross-day scenario. The study suggests an approach to perform activity recognition, proposing an architecture based on a multi classifier approach, in which activities are recognized with the aid of other smartwatch sensors (e.g. GPS), for the use within a TAS.

Chapter 3

System Architecture

3.1 Goals and Methodology

The goal of this study is to demonstrate that it is possible to build a system capable of recognizing sport activities automatically, in order to facilitate physical activity tracking, exempting the user from manually managing sport sessions, in a non-invasive and user-friendly fashion. Since the goal of fitness tracking is to motivate the users to perform physical activity, and to promote competitiveness, a totally automatic tracking system that does not require user input would drive closer to such goals.
For a similar study, it is necessary to build an activity recognition system, able to collect data from sensors such as the accelerometer, gyroscope, etc... in order to build a dataset and train a sport activity recognition model through the use of supervised machine learning. Such model can to be integrated in a fitness tracking application. The steps involved in the process are the following:

- Choosing the data collection parameters.
- Collecting and storing data in a dataset.
- Selecting the window length and the features.
- Training different classifiers on the dataset.

- Evaluating and choosing a classifier.
- Testing the classifier in a demo fitness application.

The process can be repeated many times in order to select the best configuration of features, window lenght, sensors included and many machine learning parameters, which would give the best compromise between recognition accuracy, power saving and simplicity. A human activity recognition system should allow to perform such steps, providing an interface to drive its users during the realization of the study.

To carry out the study, a distributed application was developed, capable of collecting and storing data it in a local dataset, along with an activity recognition module which was fed with the model obtained after the training phase. Due to a certain number of issues that were encountered during the development phase, which will be explained later in section 4.5, it was chosen to store the extracted features in the dataset, instead of storing the raw sensor data.
The devices used for this study were:

- An Apple Watch Series 2
- An Apple iPhone 6s

The two devices are able to communicate through Wi-Fi or Bluetooth, automatically switching between the two technologies as needed. In order to save power, if the phone is nearby the watch, the Bluetooth is used. If the two devices are out of range and there is an available Wi-Fi network, then the wireless network is used.

3.2 Architecture Review

Figure 3.1 shows the architecture of the developed HAR system, which relies on a distributed application and on external scripts hosted on a third machine.

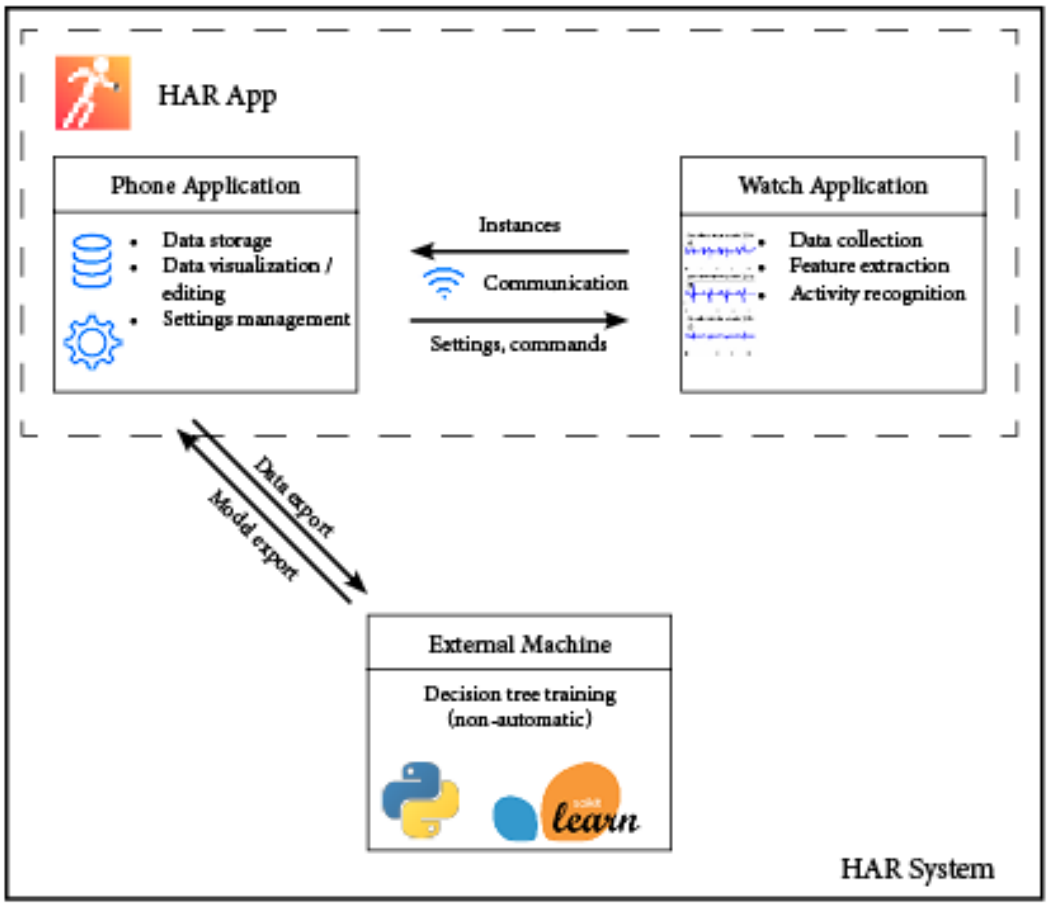


Figure 3.1: Architecture of the developed HAR system.

HAR App

The distributed application, Human Activity Recognition (or its abbreviated name HAR App), was developed both for the Apple Watch and for the iPhone side. The application on the watch is able to communicate with the application on the phone and viceversa when the two devices are paired. The watch application is responsible for the data collection and feature extraction phases, and for running the recognition algorithm obtained after the training phase. The phone application receives instances from the watch application, and stores them is a local dataset, which can be visualized and edited by the user; the dataset can also be exported to a CSV format. Another func-

tionality of the phone application is settings management, which allows the user to modify settings related to the data collection and feature extraction phases, such as which sampling frequency to use, etc . . . the settings are regularly synchronized with the watch application. The phone application is also able to send commands to control the activity recording, such as pausing, resuming or stopping the activity.

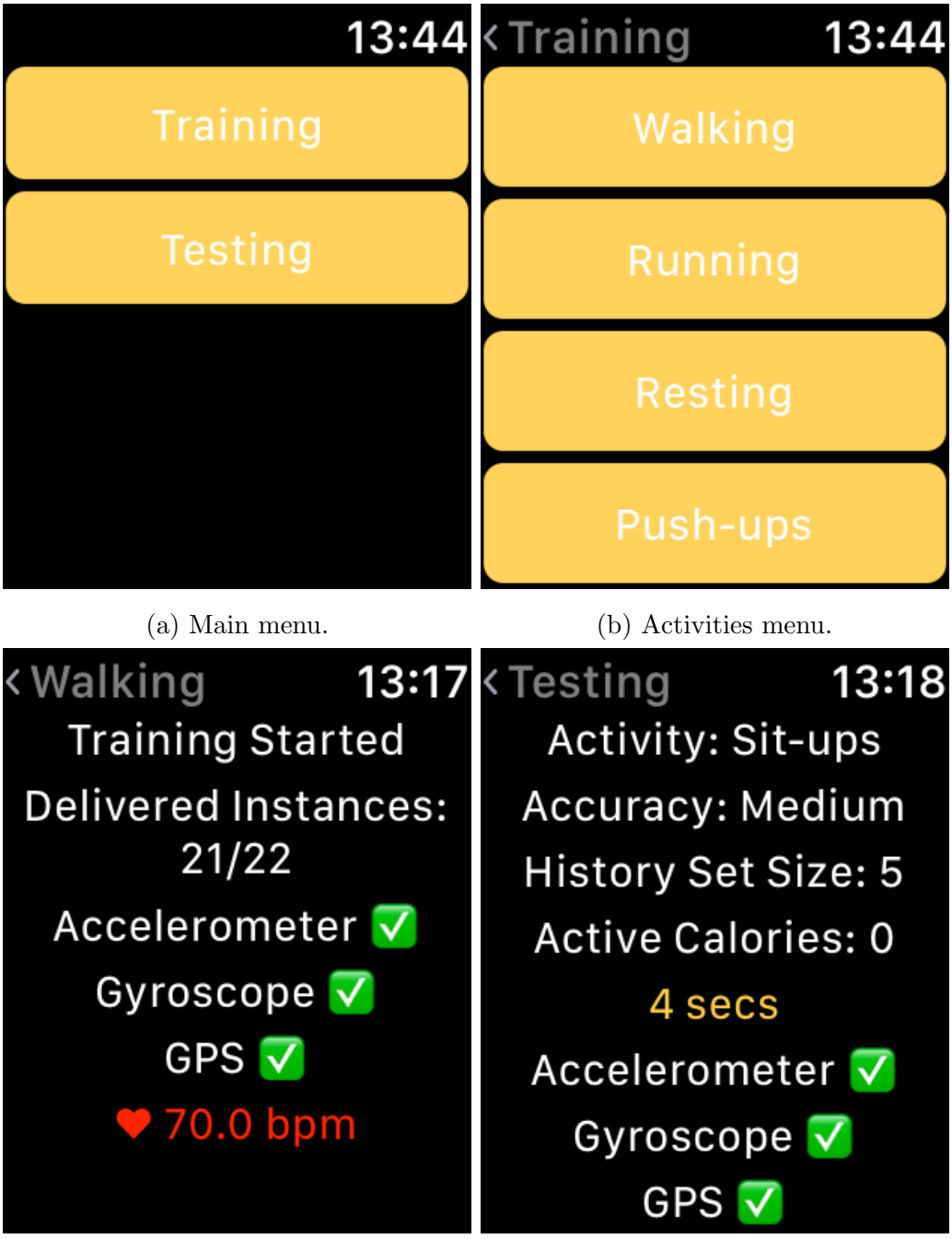
Scikit-learn

Scikit-learn [75] is a machine learning library for the Python programming language. It includes many of the classification algorithms needed for human activity recognition, and it is able to interoperate with the Python numerical library NumPy [76].
The learning phase was carried out in Python, with the use of Scikit-learn and NumPy to elaborate the dataset exported by HAR App, and produce a classification model. The learning phase is not automatic, and requires manual input and actions to export the dataset, build the model and import it to the application.

Figure 3.2 shows the HAR App interface on the Apple Watch. In the main menu the user can choice between training and testing. In the training interface, the user can choose between a list of activities, and then the data recording starts, as shown in subfigure 3.2c. Subfigure 3.2d shows the testing interface.

Workouts

As stated, the goal beyond this study is to automatically recognize sport activities in order to exempt the user from manually starting, pausing or stopping workouts, to facilitate activity tracking with capabilities such as



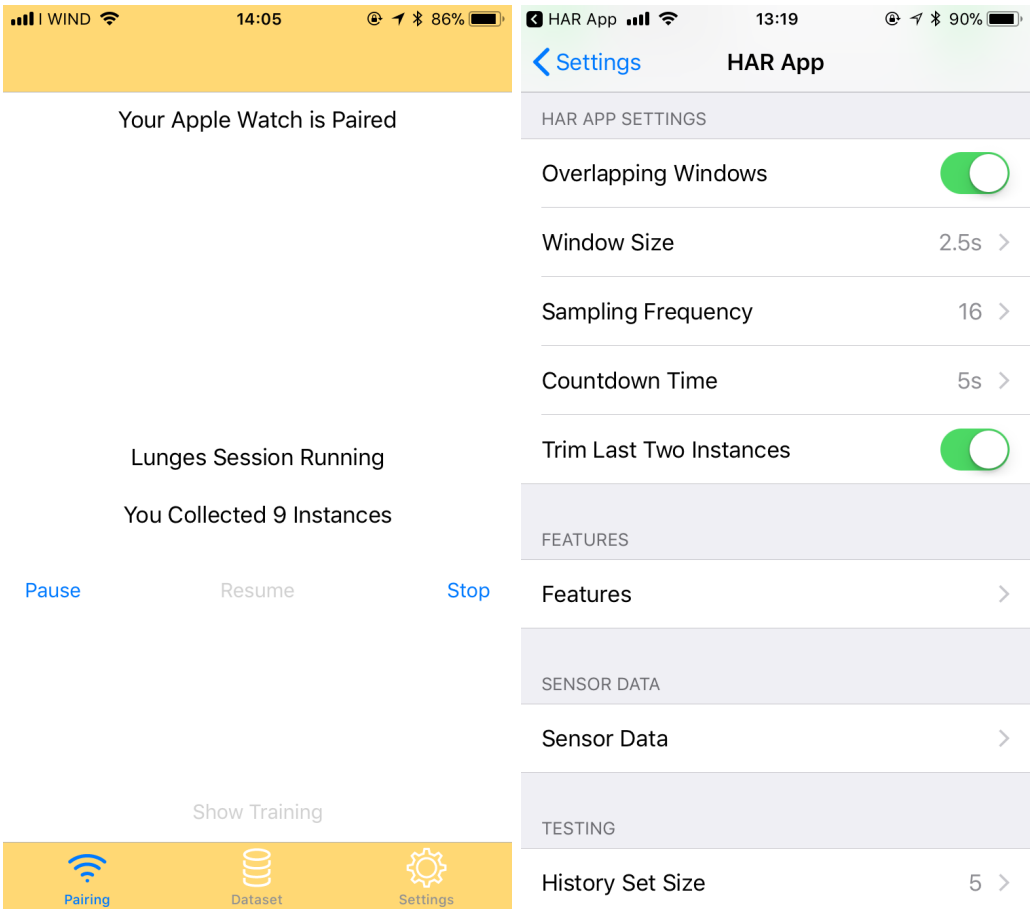
(a) Main menu.

(b) Activities menu.

(c) Activity recording interface.

(d) Activity recognition interface.

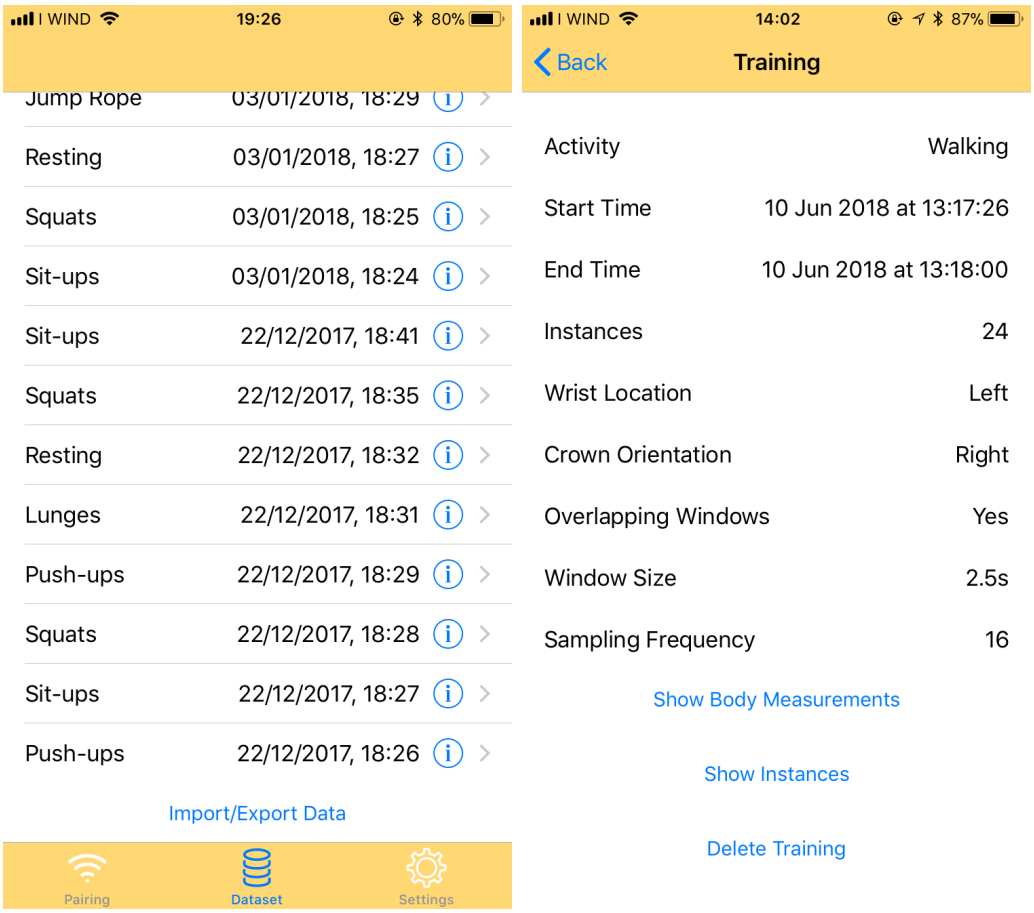
Figure 3.2: HAR App interface on the watch.



(a) Pairing interface. (b) Settings interface.

Figure 3.3: HAR App interface on the phone.

calories counting and other advanced features. HAR App automatically saves the workouts during the testing phase. When an activity beyond resting is dectected for more than 30 seconds, the application saves the workout at the end of the session, including the type, duration of the workout and the active calories burned. The workout is saved in a centralized repository made available by the iOS and watchOS operative systems, which can store health and fitness data. Applications can write or read data from this repository, such as workouts, heart rate, energy burned, weight, height and other user data. When an application writes in this database, all the information become

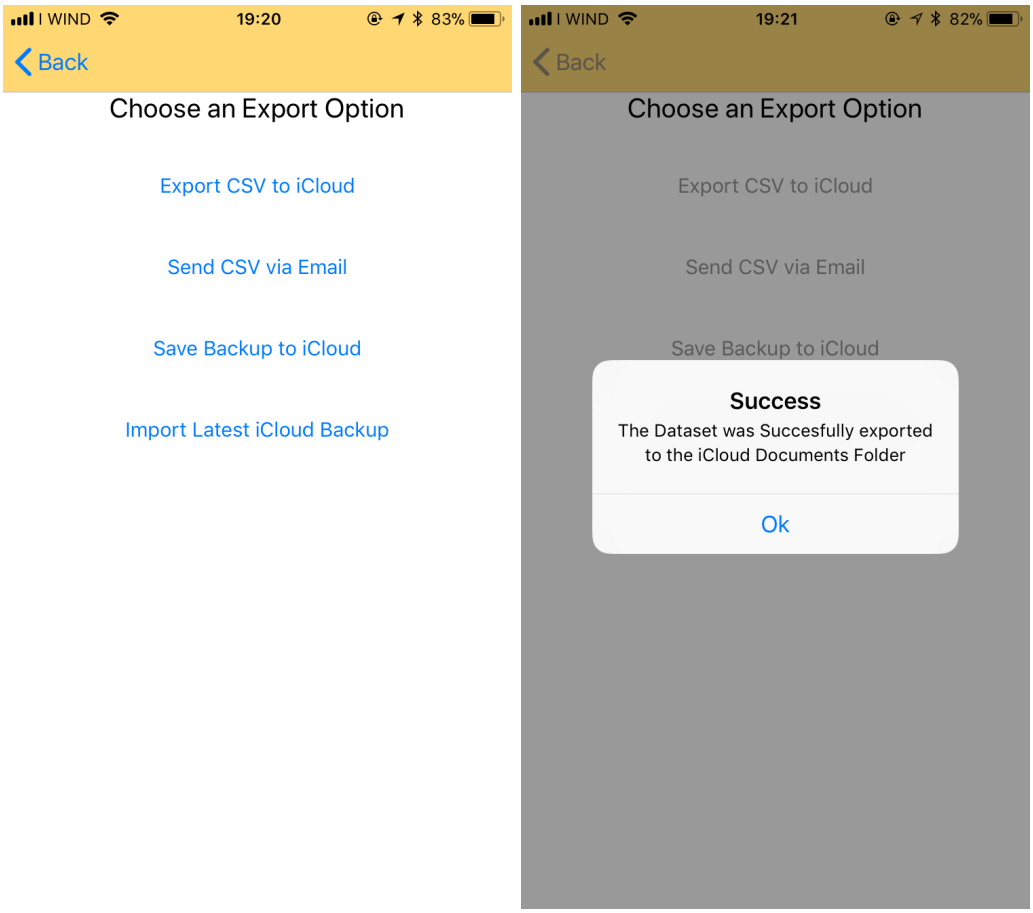


(a) Dataset visualization. (b) Training information.

Figure 3.4: HAR App interface on the phone.

available for other applications to be read. All the data can be visualized using Health app, an application natively included in iOS.

The phone interface is shown in figures 3.3, 3.4 and 3.5: in the pairing interface, the user receives information about the data received from the watch, and they can control the session sending a pause, resume and stop command; in the settings interface, the user can choose i) activity recording options such as window size, sampling frequency, etc . . . , ii) which features



(a) Import/export interface. (b) Export success message.

Figure 3.5: HAR App interface on the phone.

to extract, iii) which sensors to include, iv) the history set size for activity recognition. Indeed, to render activity transitions smoother, a history set can be used. The classification algorithm is a decision tree obtained from the external machine and imported in the application. The history set contains the last n decision tree classifications, and the final prediction is the most frequent activity in the history set. The full list of all the included options is shown in table 3.1.

The data can be visualized and edited, deleting specific instances. Moreover, it is possible to save and restore a dataset backup on iCloud, and to export

Option	Values
Window size	1s, 2s, 2.5s, 4s, 5s, 8s, 10s
Sampling frequency	16, 24, 32, 40, 50
Countdown time[1]	None, 2s, 5s, 10s, 15s, 20s
Trim last two instances[2]	True, false
Features	Maximum, minimum, median, mean, standard deviation, variance, skewness, kurtosis IQR, energy, cross-domain entropy
Sensor data	1D: heart rate, latitude, longitude, altitude, course, speed. 4D: user acceleration, gravity, orientation, rotation rate.
History set size	1, 3, 5, 7

Table 3.1: Available options for data collection, feature extraction and classifcation in HAR App.

the dataset to a CSV format and send it via email, or save it to the iCloud folder, as shown in figure 3.5.

The classification model was manually imported from the external scripts that rely on scikit-learn. The learning phase of the system is not automatic and requires manual actions, like:

- Importing the training set to the external machine.
- Possibily elaborating the dataset, removing features and/or outliers.
- Running the scripts and exporting a decision tree in a textual format.
- Importing the model back to the application.

[1]The countdown time before starting the activity, has the purpose of giving time to the user to prepare before that the activity recording starts.

[2]The last two instances might be deleted, taking into account the time needed to interact with interface in order to stop the activity recording.

Chapter 4

System Implementation

As stated in chapter 3, the system relies on three main components: a watch application, a phone application and some scripts located in an external machine. The watch and its phone counterpart application were developed using the Swift programming language [77], through the Xcode IDE [78]. The scripts were written in Python, using the scikit-learn machine learning library [75] and the NumPy numerical library [76].
Applications that target both the iOS and watchOS operative systems, can be developed in a unique project, developing an iPhone application, a WatchKit app that is responsible for storing static resources such as images and graphics elements of the watch interface, and a WatchKit extension that is responsible for implementing the dynamic behavior of the watch application, such as responding to notifications, events and for performing continuous tasks. After presenting a list of the frameworks included in the Swift project in section 4.1, in order to follow a modular approach the watch and the phone applications will be presented separatedly, respectively in sections 4.2 and 4.3. Section 4.4 will describe the machine learning scripts.

4.1 Swift Frameworks

The following is a list of the frameworks that were included in the HAR App project, either in the phone, the watch application or both.

HealthKit

HealthKit is a framework provided by Apple, available in the Swift programming language, which allows applications to access and share health and fitness data such as physical activities, calories intake, sleep, etc... providing a centralized database accessible by applications. HealthKit manages all the privacy aspects of the users, requiring an explicit authorization by the user in order to allow applications to read or write specific health and fitness data. In the HAR App, the HealthKit framework was used in the watch application in order to:

- Collect user information such as age, height, weight and gender.
- Record the heart rate.
- Keep track of the resting heart rate, which is a metric useful to calculate the calories consumption of physical activities.
- Save workouts and the active energy burned.

The application needs first to be authorized by the user in order to perform the above operations; the authorization is asked on the phone application, where the user is presented with the interface shown in figure 4.1. Once given the authorizations, the watch application can start functioning.

CoreMotion

CoreMotion is a framework that allows to access accelerometer, gyroscope and magnetometer data. Since the magnetometer is unavailable on the Apple Watch, it has not be included in the study. When the application wants to read motion data, it can requests updates at a fixed rate, in order to read:

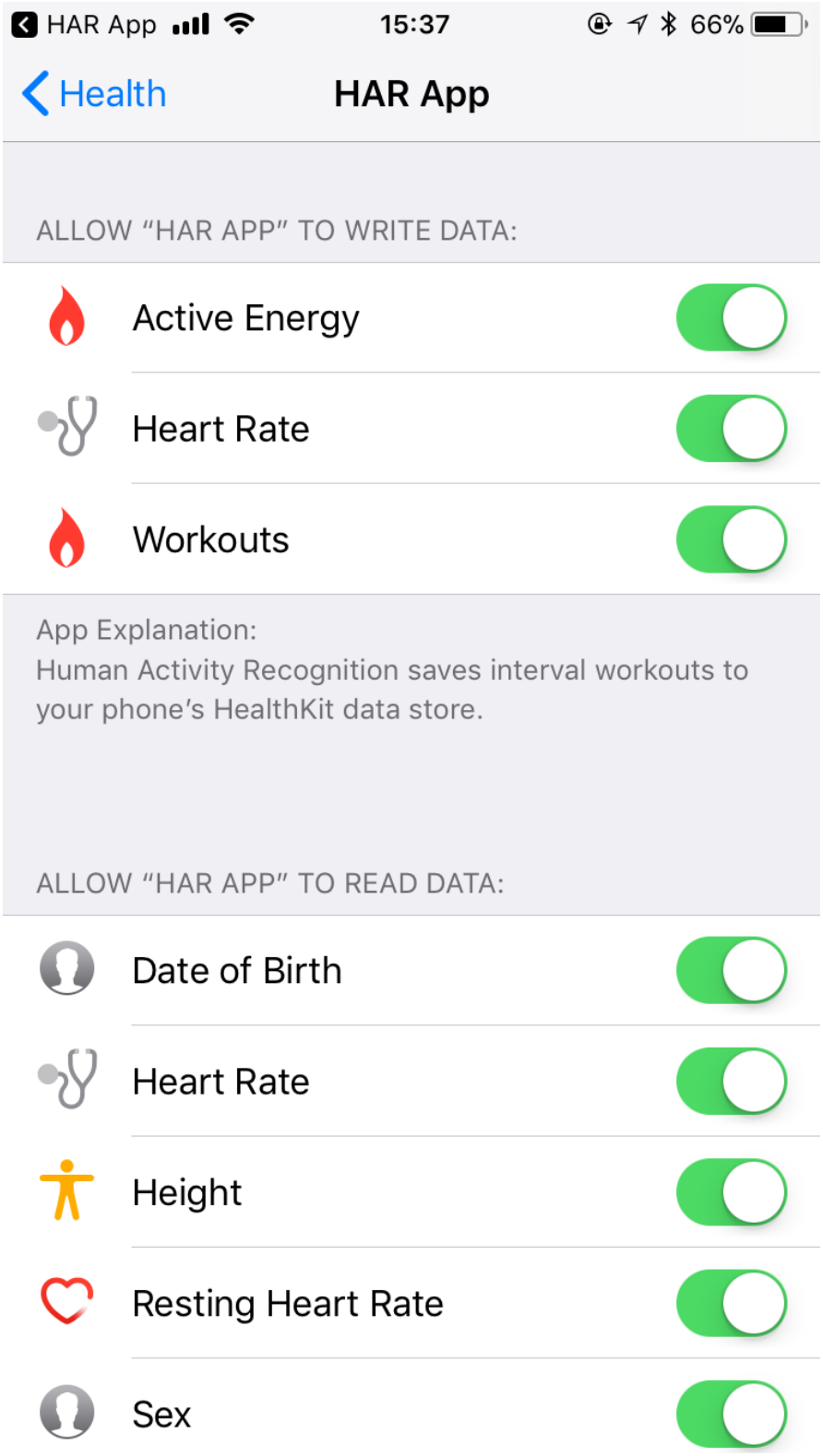


Figure 4.1: HealthKit authorizations interface.

- Attitude
- Rotation rate
- Gravity
- User-initiated acceleration

CoreLocation

CoreLocation provides services for determining the device geographic location, using all the available onboard hardware, including WiFi, GPS, Bluetooth, magnetometer, barometer, and cellular hardware. If the location is requested in a watch application, it is possible for CoreLocation to return the location obtained from its paired iPhone, in order to save the watch battery power. CoreLocation seamlessly switches between available technologies in order to calculate the position, using beacons, WiFi fingerprinting or satellite signals according to the situation and the environment, in order to get the most accurate information. Since determining the user position involves privacy concerns, it is necessary to authorize location services for applications that request them. The authorization is requested on the phone and once obtained, the watch application is free to use CoreLocation services.

Watch Connectivity

The WatchConnectivity framework implements two-way communications between a watch application and its paired phone application counterpart. There are many ways to communicate between paired applications, but since in this application WatchConnectivity was required to perform real-time synchronous operations, it was chosen to use the *interactive messaging mode*, which allows to send messages between the iOS application and the watchOS applications immediately, awakening the application that received the message in case that it is not active, and executing a completion callback in case that the message is delivered successfully. The communications were required in order to perform the following operations:

- Transfering instances from the watch application to the paired phone application.
- Transfering data collection, feature extraction and prediction options (e.g. sampling frequency) from the phone application to the watch application.
- Sending interactive commands from the phone application to the watch application, in order to start, pause, resume and stop the session.
- Sending feedback from the watch application to the phone application.

Surge

Surge [79] is a Swift library that relies on the Accelerate framework, which provides high-performance functions matrix math, digital signal processing, and image manipulation. Accelerate is able to improve the performance of certain math calculations, using the SIMD instructions implemented in the most modern CPUs. Surge is built on top of the Accelerate framework in order to provide a simplified, user-friendly API.
This framework was used in HAR App in order to execute efficiently operations on vectors, such as the sum, the fast Fourier transform and many more operations useful in the feature extraction phase, which was carried out in the Apple Watch application.

Core Data

The CoreData framework allows to manage model layer objects and provides automated solutions to persistence and other tasks associated with object life cycle and object graph management. CoreData simplifies and speeds up the development effort required to mantain a relational database. A CoreData database was used to locally save the training data received from the watch application.

Other Frameworks

Other standard frameworks were used in the development phase: UIKit and WatchKit provide the user interface functionalities on iOS and watchOS respectively, and they both rely on Foundation, a framework that provides a base layer of functionalities for applications, which implements the most commonly used data types. These three frameworks are generally used in any iOS and watchOS application developed with Xcode.
Moreover, ClockKit was used on the Apple Watch to handle *watch complications*, which are elements of the watch face that display small, immediately relevant bits of information. A complication of the application was added in order to have a prompt way to launch the application from the watch face, and to check for relevant information (e.g. the number of instances sent). Another framework used in this project is the MessageUI framework, which was used in the iPhone application project in order to send the dataset via email; exporting the dataset was necessary to run the machine learning scripts on an external machine.

4.2 Watch Application

The watch application is divided in two modules: a training module, responsible for the data collection and the feature extraction phases, and a training module responsible for running activity recognition.

4.2.1 Training Module

Figure 4.2 shows the class diagram for the `TrainingController` class, which manages the data collection and feature extraction phases, and updates the relevant UI in order to give feedback and interact with the user, allowing to control the session. `ActivityController` implements the functionalities of a `WKInterfaceController`, a WatchKit MVC controller. `ActivityController` relies on the `SensorsRecorder` class, which is a service

that was developed in order to automatically collect data, querying the user preferences to detect data collection parameters, and which sensors and features to include; at each window frame, features are extracted and the delegate object of the `SensorsRecorder` is notified. In order to be notified, an object should register itself as delegate and implement the delegate methods, which will be invoked by the `SensorsRecorder` object. The classes `HealthKitService`, `CoreMotionService` and `CoreLocationService` manage the HealthKit, CoreMotion and CoreLocation operations respectively, querying the sensors and providing data, which is made available to the `SensorsRecorder` object. Moreover, the `SensorsRecorder` class manages the authorizations, querying the authorization status and prompting a visual alert in case that the user did not grant the necessary authorizations (e.g. location services are not enabled). The following code snippet shows the necessary steps to instantiate and use a `SensorsRecorder` object:

```
let recorder = SensorsRecorder(activity:activity)!
recorder.delegate = self
recorder.startSession()
recorder.start()
```

The `startSession()` method is required to activate a workout, which is handled by the `HealthKitService` class. Once a workout is active, the application is allowed to stay in foreground and it is not killed until the workout is finished. Otherwise, during data collection the application would be deactivated and killed in a short amount of time. The `start()` method causes the data collection and feature extraction process to start, and the client object is notified by the `SensorsRecorder` through this method:

```
func sensorsRecorder(_ recorder: SensorsRecorder,
   didExtractFeatures features: [String : Any])
```

Which receives a dictionary as argument, containing the extracted features for each sensor. The recording can then be paused, resumed and stopped calling the corresponding methods.

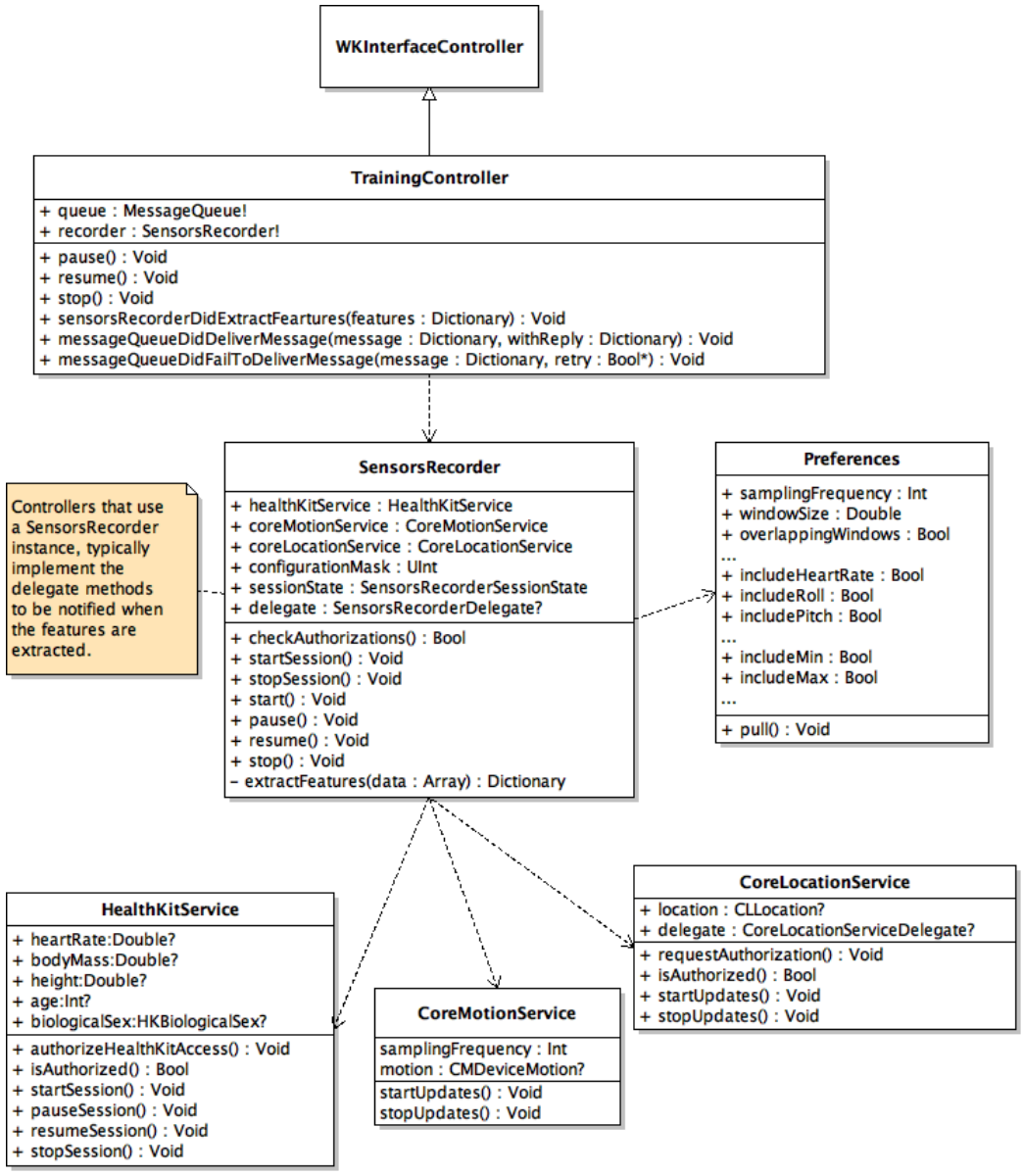


Figure 4.2: Class diagram for the `ActivityController` class and its related classes.

The `Preferences` class automatically manages the user preferences, which are synchronized with the iOS application preferences using the `pull()` method, which asynchronously sends a message to the iPhone application, requesting to update the preferences, to which the iOS application responds sending a dictionary containing the preferences. The watchOS application can receive this dictionary not exclusively after requesting a pull, but also at the initiative of the iOS application, which can decide to send the preferences dictionary after noticing that the user has recently changed some settings.
Most of the communications are handled using the `MessageQueue` class, which manages a queue of messages, allowing to send FIFO messages with a simplified mechanism of congestion and flow control, receiving notifications when the messages are successfully delivered, or when there is an error. Nominally, the WatchConnectivity framework guarantees that when a message is sent, either a success or failure callback is called. But in practice, it may happen than no callback is actually called, and that sometimes the communications freeze, without giving any notification. The `MessageQueue` class was built on top of the WatchConnectivity framework to overcome these flaws. The following code snippet shows how to instantiate a `MessageQueue` object and send a dictionary containing the extracted features:

```
self.queue = MessageQueue(timeout: 5.0)
self.queue.delegate = self
let message = [
        "type":"info",
        "state":state,
        "activity":self.recorder.activity,
        "time":Date(),
        "instance":features
]
self.queue.enqueue(message)
```

After enqueueing the message, the delegate is asynchronously notified through any of these two methods:

```
func messageQueue(_ queue: MessageQueue,
   didDeliverMessage message: [String : Any],
   withReply: [String : Any])

func messageQueue(_ queue: MessageQueue,
   didFailToDeliverMessage: [String : Any], retry:
   UnsafeMutablePointer<Bool>)
```

If the message fails to deliver, either because there is a timeout or because there is a network error, the queue will retry to send the message in case that the `retry` parameter is set to true.
`ActivityController` sets itself as delegate of its `SensorsRecorder` and `MessageQueue` objects. When a new set of features is extracted, they are enqueued, and the controller is notified when a message is delivered with success or when there is a delivery error. Moreover, the controller sends feedback to the phone application and receives commands (e.g. to pause the session remotely).

4.2.2 Testing Module

Like the training module, the testing module relies on the `SensorsRecorder` class in order to carry out the features extraction from the sensor data, with the difference that it does not need to send any data beyond control information to notify the phone application. Figure 4.3 shows the class diagram for `TestingController` and its related classes. The `DecisionTree` class implements a classifier using the output of the machine learning scripts, in a textual format that is parsed and converted into a decision tree. The `HistorySet` class stores the last n predictions in a queue, which can be inserted calling the `insert()` method, and outputs the most frequently predicted activity and the accuracy, which are stored respectively in the `activity` and `accuracy` properties. The accuracy is given by the number of predictions in the queue that are equal to the most frequent prediction, di-

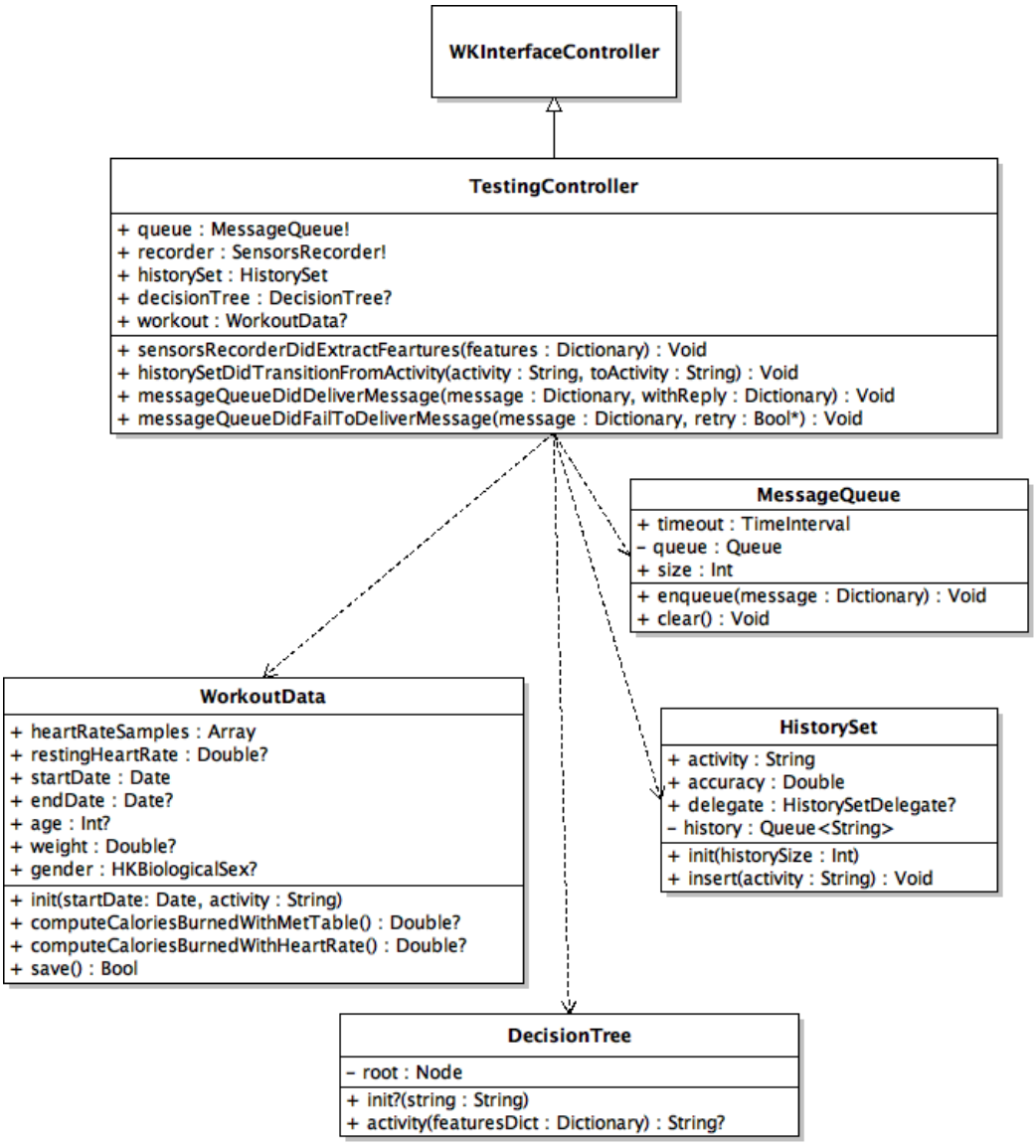


Figure 4.3: Class diagram for `TestingController` and its related classes.

vided by the length of the queue. When `TestingController` is notified by the `SensorsRecorder`, because a new set of features was extracted, it executes the decision tree classification algorithm and inserts the classification result inside the history set. The final output is the activity predicted by the history set.
Another functionality implemented by `TestingController` is calories counting: when an activity different than resting is predicted for more than 30 seconds, the workout is saved. The workout is managed by the `WorkoutData` class, which implements calories estimating using two methods:

- In case that the resting heart rate value of the user is not available in the HealthKit database, a metabolic equivalent table [80] is used, which contains a calories consumption estimate for every type of physical activity.
- In case that the resting heart rate of the user is available in the HealthKit database, the calories burned are estimated using the average heart rate and the resting heart rate, estimating the VO2 and the VO2 max in order to calculate the total calories burned over the activity duration.

The second method is more precise, and it takes account of the heart rate and the age, other than the weight. Either methods take account only of the *active calories burned*: the energy consumption due *only* to physical exercise, obtained by subtracting the calories burned in resting conditions from the total calories burned.
Another functionality implemented by `TestingController` is logging: the predicted activities can be logged in the documents folder of the application, and the logs can be retrieved in order to have a window per window report of the predicted activities.
Similarly to `TrainingController`, also `TestingController` inherits from `WKInterfaceController` and it is thus responsible for managing the user interface. The key difference is that instead of enqueueing the extracted features, `TestingController` use them for classification and the message queue is used for control information only.

4.3 Phone Application

The phone application is mainly responsible for receiving the instances from the watch application, and storing them into a database, allowing the user to visualize, edit and export the dataset.

CoreData was used to define a relational data model, with the following entities, as shown in figure 4.4:

- `Training`. This entity contains the overall information of the activity: the parameters used for data collection, the body measurements, the age and the gender of the user, the location and orientation of the watch, the start and end time of the activity, and the name of the physical activity. It has a one-to-many relationship with the `FeatureSet` entity.
- `FeatureSet`. This entity represents an instance, which is directly associated with a `Training` entity, and it has a one-to-many relationship with the `FeatureGroup` entity. For each sensor data (see table 3.1) it has a relationship with a `FeatureGroup`, and four relationships in the case of four-dimensional data such as acceleration, gravity, orientation and rotation rate.
- `FeatureGroup`. This entity is directly associated with a `FeatureSet` entity, and it has one value for each feature, and a `data` property that describes the sensor data whose features were extracted (e.g. "attitude-roll").

There might be not available values in case that some sensors or features were not included in the settings, or in case that the computed feature returned NaN, e.g. in the case of the skewness, which can't be calculated if all the samples have the same value.

A `TrainingController` class controls the insertion, deletion and editing of the training data, allowing to insert records simply by providing a

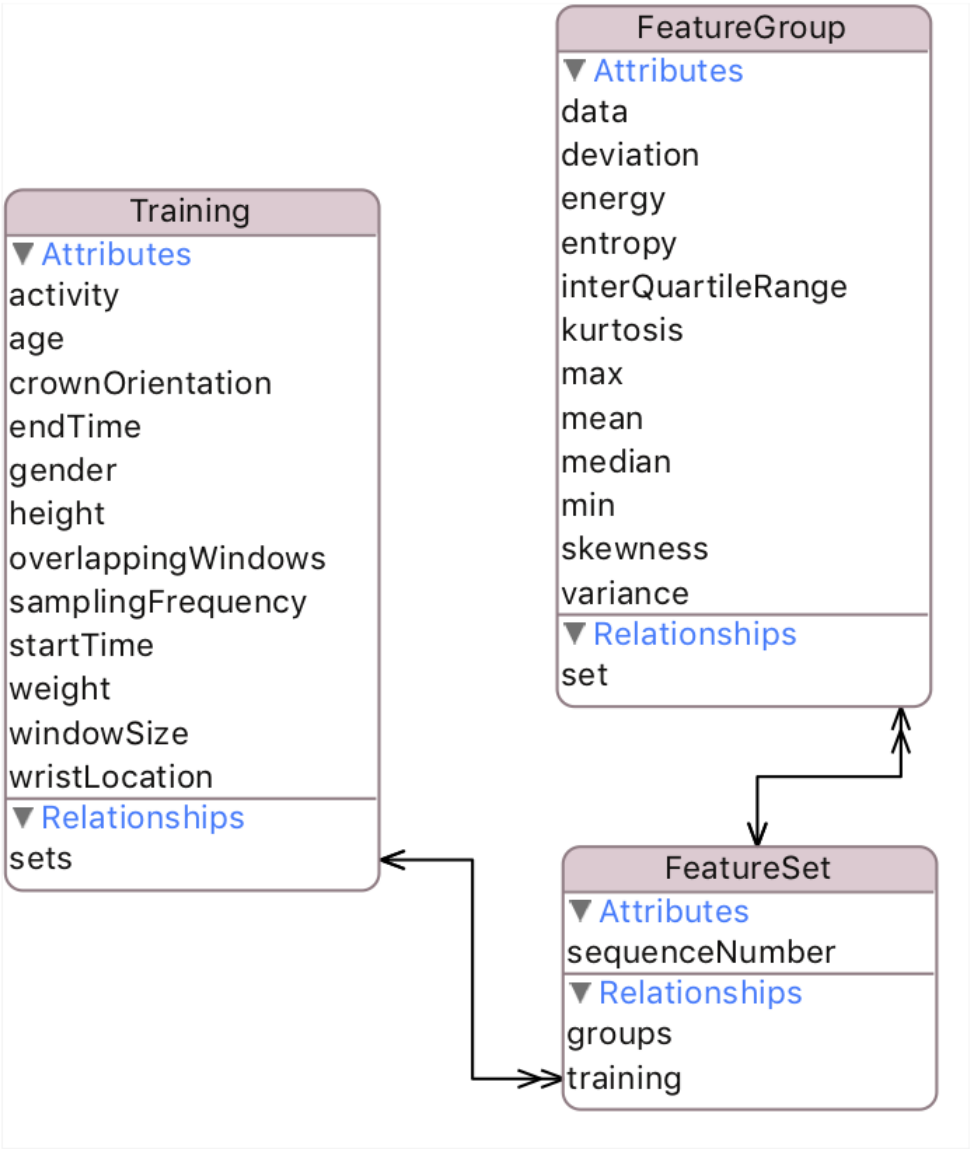


Figure 4.4: CoreData model of the training set.

dictionary of features and some other user information. When the phone application receives a message which notifies that the recording session has started, it is enough to call the `insertTraining()` method, and then provide other information such as the end time (which is constantly updated as the recording session goes on) and other user data, and the `Training` entity is created and inserted in the database automatically, as shown in the following code snippet:

```
/* This dictionary contains training information
   like start time, wrist location, etc... */
guard let trainingInfo = message["userInfo"] as? [
   String:Any] else { break }
self.training = self.controller.insertTraining(
   dictionary: trainingInfo)
self.training?.endTime = time
self.controller.saveContext()
```

To add an instance, it is enough to call the `insertFeatureSet()` method, possibly update the end time, and then save the context again:

```
let _ = self.controller.insertFeatureSet(instance,
   inTraining: training)
self.training?.endTime = time
/* Saving the CoreData context causes the data to be
    actually inserted into the database */
self.controller.saveContext()
```

The instances are received using the WatchConnectivity framework, without the need to use a queue of messages because the phone application does not need to send a considerable amount of data, but only feedback and control information (e.g. the user wants to remotely pause the activity recording). One or more `Training` entities can be converted to a CSV file, containing a row for every instance. The data can be exported from a dedicated controller, which can send the dataset via email, or save it to the iCloud folder.

The following code shows how to convert an array of `Training` entities into a CSV string:

```
self.controller.context.perform { [weak self] in
        /* The code is executed in a background
   thread, safe from race conditions */
        let trainings = self?.controller.trainings
        let csv = trainings?.csvString()
}
```

The phone application is also responsible for presenting the data to the user, but this section will not dig into the details of how the UI was implemented.
The last aspect that needs to be mentioned is the management of the user preferences. Like stated in the previous section, the watch application mantains its own copy of the settings, which is periodically synchronized with the settings of the phone application, due to the impossibility, or at least the difficulty for the user to enter the settings from the apple watch. For this purpose, the phone application might receive a message requesting for an updated version of the settings, to which is simply responds calling a reply handler that contains a dictionary representation of the settings, as allowed by the WatchConnectivity framework. Moreover, the phone application constantly monitors the user settings, and it receives a notification when the user has made some changes. In this case, the preferences are pushed: a message is proactively sent to the watch application, which proceeds to update the settings.

4.4 Machine Learning Scripts

After transfering the dataset in a CSV format, it needs to be elaborated in order to produce a classification model, which can be manually imported in HAR App. The elaboration can be divided in two steps:

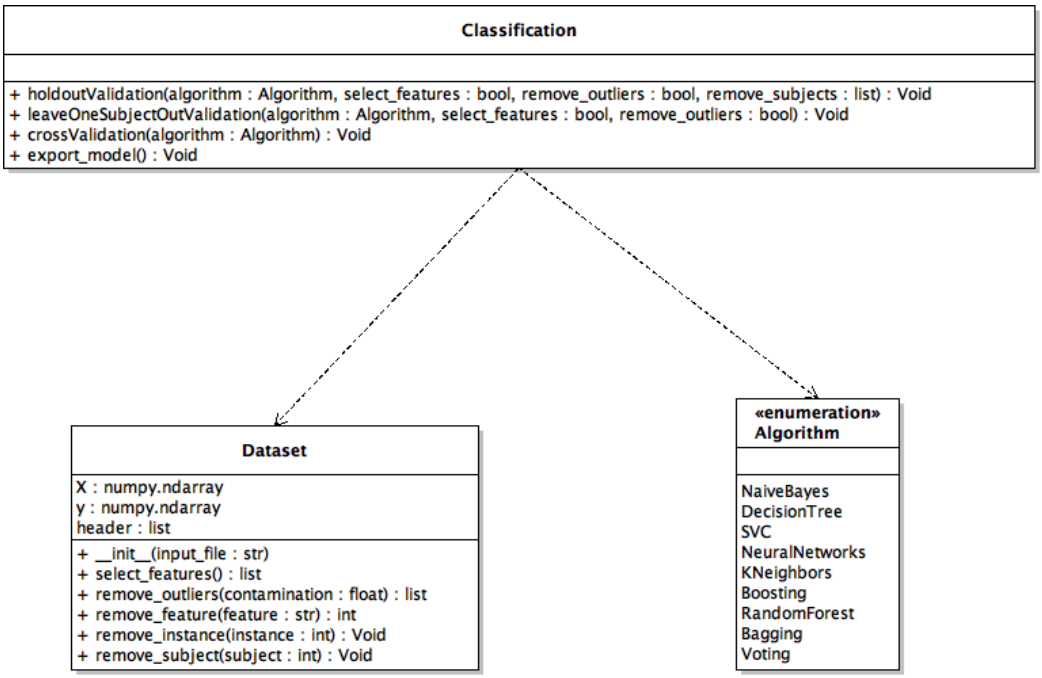


Figure 4.5: Class diagram of the machine learning scripts.

- The dataset needs to be imported and converted into a bidimensional array. Operations like feature selection, outlier detection and removal of specific instances or users from the dataset need to be supported.
- Many classification algorithms are evaluated, comparing their performance metrics. Finally, a classification model in a textual format is exported.

Two Python scripts were developed: i) `Dataset.py` for the dataset elaboration, which relies on NumPy, ii) `Classification.py` for the machine learning step. Figure 4.5 shows the class diagram of the classification scripts. The `Classification` class automatically uses the methods defined in `Dataset.py`, loading the exported CSV dataset and calling the appropriate methods in case that it is necessary to modify the dataset (e.g. removing outliers). The below script shows how to execute the holdout validation (which is by default implemented using 70% of the training data and the rest for validation) using a decision tree algorithm, and then exporting the

classification model:

```
import Classification
Classification.holdoutValidation(algorithm = Classification.
                                 Algorithm.NaiveBayes,
                                 select_features = True,
                                 remove_outliers = False,
                                 remove_subjects = [0])
Classification.export_model()
```

An algorithm for automatic feature selection is executed, and the subjects corresponding to the indexes contained in the `remove_subjects` list, are removed; in this case, the first user is removed. Afterwards, the decision tree is exported in a textual format, which will be parsed by the watch application. The following is the list of all the supported algorithms:

1. Naive Bayes
2. Decision tree
3. Linear SVC (support vector clustering)
4. Multilayer perceptron
5. kNN (k = 5)
6. Gradient boosting
7. Random forest
8. Bagging
9. Voting (majority voting of classifiers 1 to 5)

The holdout validation by default is run 10 times, and an average of the following metrics is computed:

- Accuracy
- Precision
- Recall

- F1 score
- Confusion matrix
- Model building time

Figure 4.6 shows a possible output of the classification scripts.

```
>>> import Classification
>>> Classification.holdoutValidation(algorithm = Classification.Algorithm.NaiveBayes, select_features =
True, remove_outliers = False, remove_subjects = [0])
User 0 removed
Selected 14 features
Run      Accuracy        Precision       Recall          F1              Time
Run # 1  0.884406        0.889193        0.884406        0.883999        0.003110
Run # 2  0.881134        0.887578        0.881134        0.880154        0.003848
Run # 3  0.869138        0.881222        0.869138        0.869307        0.002685
Run # 4  0.872410        0.878730        0.872410        0.870265        0.002257
Run # 5  0.874591        0.885307        0.874591        0.875909        0.002207
Run # 6  0.875682        0.884757        0.875682        0.874581        0.002172
Run # 7  0.893130        0.900793        0.893130        0.892635        0.002330
Run # 8  0.866957        0.875974        0.866957        0.866087        0.002196
Run # 9  0.878953        0.883628        0.878953        0.877678        0.003747
Run #10  0.875682        0.883140        0.875682        0.874595        0.003618
Average  0.877208        0.885032        0.877208        0.876521        0.002817
Confusion Matrix:
[[106   0   1   0   0   1   0   0]
 [  0  64  16   1   2  24   0   0]
 [  0   6  87   2   0  15   2   0]
 [  0   2   2 109   0   0   0   0]
 [  0   0   0   0 114   0   0   0]
 [  1   4   1   0   0 107   0   0]
 [  0   5  18   0   0   0  99   0]
 [  0   0   0   0   0   0   0 114]]
```

Figure 4.6: Output of the classification scripts.

4.5 Related Issues

A certain number of issues were encountered during the development phase, which have posed constraints on some choices, particularly on those described in section 5.1. Most of the development efforts were employed to solve the following issues:

- CPU limits. There is an implicit, non-official CPU limit for Apple Watch applications, which was discovered during the development phase. This limit was discovered to be 15% during a workout. If an application uses more than 15% of the CPU for 60 seconds, it might be killed by the watchOS watchdog. For this reason, the number of included features

was reduced, excluding the most expensive features. Moreover, a low sampling frequency was chosen to avoid having the application killed.

- Memory limits. RAM and disk memory are also important factors. An option was to store the raw sensor data directly on the Apple Watch, and export it later to the iPhone, but this was impossible due to RAM and disk usage limitations. For this reason, it was chosen to directly extract the features on the watch application, and send them to its phone application counterpart in real-time.
- Communication problems. Communications from the Apple Watch to the iPhone are unstable, and they can often be frozen, making impossible to send instances to the phone application. Sometimes it is necessary to restart the Apple Watch in order to restore the communications. Some complicated mechanisms were implemented to ensure that the messages were enqueued and resent later in case that the delivery failed, along with some congestion control techniques. For this reason, sending large chunks of data to the phone application was impossible, and this contributed to the decision of sending the extracted features to the phone, instead of sending the raw data, which has a larger size than the features.

For these reasons, parts of the steps mentioned in section 3.1 were skipped, and a configuration of sampling frequency, window length, sensors and features was chosen beforehand, instead of varying the configurations, due to the impossibility of including those configurations that involved a high CPU, memory and bandwidth usage (e.g. choosing a higher sampling frequency, or adding more features). Moreover, these issues were not encountered during the debug phase, because the CPU and memory limits are absent during debug, and the communication issues are not reproducible during debug, which requires the Apple Watch to be connected to the iPhone. Crash logs were often unavailable, and this made the discover of the issues more difficult, slowing down significantly the development phase.

Chapter 5

Data Collection

This chapter focuses on the methodoloy adopted for data collection, which includes technical details about sampling and feature extraction in section 5.1, and a review of the subjects and the activity recording methodology in section 5.2, with details on the statistical features of the constructed dataset. The activities included in this study are: push-ups, sit-ups, squats, lunges, jump rope, resting, walking and running. The activities could be performed in any way by the subjects, and they were free to choose their own variant of the exercise in case that it could be executed in multiple ways (e.g. curtsy lunges over regular lunges, or for the sit-ups, the bycicle instead of crunches). For the resting activity, the subjects were instructed to rest in any position that they found comfortable and that they would choose naturally to rest between activity sessions.
For this study, the subjects were required to wear the watch on their favourite wrist, with the watch crown oriented as they wished, and they were monitored with an iPhone during the activities, which was placed at a reasonably close distance in order to communicate with the watch and record the activity data.

5.1 Sampling and Feature Extraction

For the data collection process, it was chosen to store the features in the dataset instead of storing the raw data [1]. During the data collection process, the following sensors were sampled:

- **Accelerometer**. The Apple Watch Series 2 includes a triaxial accelerometer able to distinguish between user-initiated movements and gravity. Since gravity is dependent on the orientation of the watch, which is measured by the gyroscope, the gravity components were considered redundant and they were not included.
- **Gyrscope**. The components of the watch orientation, and the rotation rate were sampled.
- **GPS**. Built-in GPS capabilities are included in the Apple Watch. However, often the watch uses its paired iPhone GPS capabilities in order to save battery and to gain accuracy. Since the low-level functioning of the GPS is not transparent in WatchOS, it is not possible to know if the watch is using its own GPS or if it is receiving GPS information from the paired phone. However, since the phone was always placed at close distance from the watch, it is plausible that the phone GPS was used during most of the time.
- **Heart rate monitor**. A *photoplethysmography*-based heart rate monitor is included in the watch, which exploits the fact that blood adsorbs green light to record the heart rate using two light sensors and two green LEDs, in the range of 30-120 beats per minute. The heart rate was asynchronously sampled and included in the sensor data.

[1] As stated in section 4.5, some choices were dictated by the strict CPU, memory and bandwidth limits on the Apple Watch, such as: i) storing the features in the dataset instead of the raw data, ii) choosing a low sampling frequency, iii) choosing a low, fixed window length, iv) choosing a fixed subset of features, v) choosing not to include all the sensor data.

Sensor	Components	Sensor	Components
Accelerometer - user acceleration	X	Gyroscope - rotation rate	X
	Y		Y
	Z		Z
	Magnitude		Magnitude
Gyroscope - attitude	Roll	GPS	Course
	Pitch		Altitude
	Yaw		Speed
	Magnitude	Heart rate monitor	Beats per minute

Table 5.1: Sampled data.

As shown in table 5.1, in total, 16 components were extracted at every sampling.

A sampling frequency of 16 HZ was chosen, with a window length of 2.5 seconds and 1.25 overlaps, for a total of 40 samples for every window. The following features were extracted from each component:

- Maximum
- Minimum
- Mean
- Median
- Standard deviation
- Skewness
- Kurtosis
- Inter-quartile range

Every instance was labeled, and the following user information were added:

- Weight

- Height
- Age
- Gender
- Wrist location
- Watch crown orientation

For a total of 134 features for each instance.

5.2 Dataset Population

Four subjects participated to the study. Their information is reported in the below table:

User	Weight	Height	Age	Gender	Wrist location	Crown orientation
#1	69	1.75	23	Male	Left	Right
#2	76	1.79	28	Male	Left	Right
#3	75	1.78	29	Male	Right	Right
#4	58	1.66	46	Male	Right	Right

Table 5.2: Information on the subjects that participated to the study. The weight is reported in kilograms and the height in meters.

The environments in which the data was collected were a boxing gym, a service area and a park. The subjects were asked to perform the activities in any order, possibly taking breaks between one activity and another, and breaking the activity recording along multiple days if necessary. No limitations or instructions about how to perform the exercises were given. Each user collected data for about 20 minutes for all the activities altogether, for a total of 4083 instances collected.
Figure 5.1 and figure 5.2 show how the instances are distributed among the activities and the subjects, respectively.

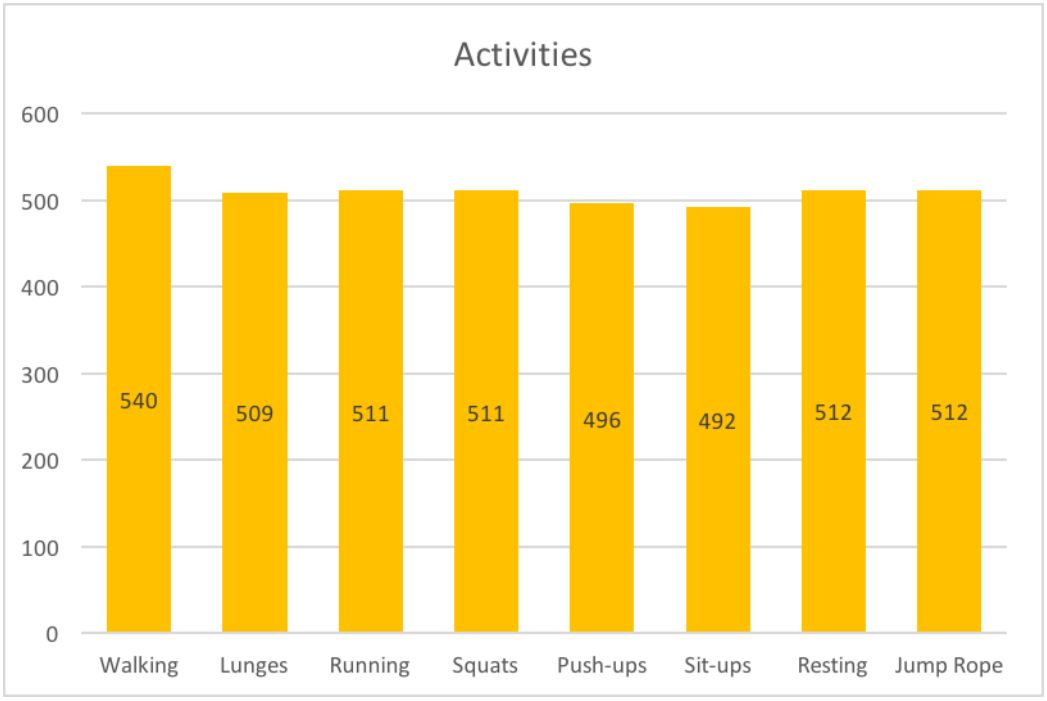


Figure 5.1: Instances distribution per activity.

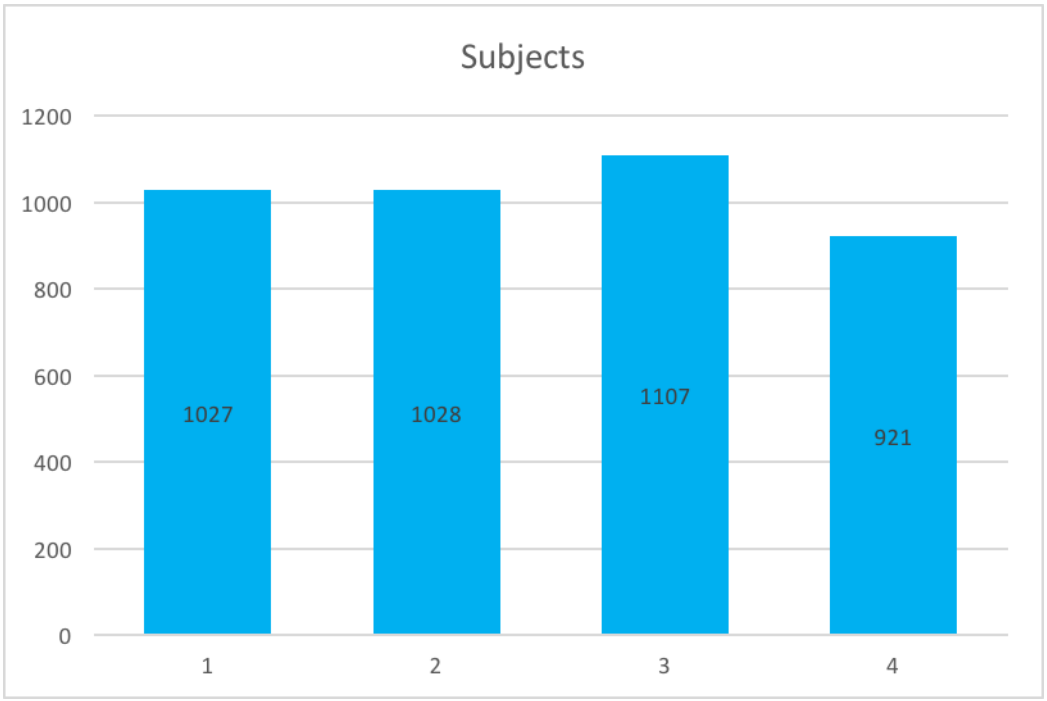


Figure 5.2: Instances distribution per subject.

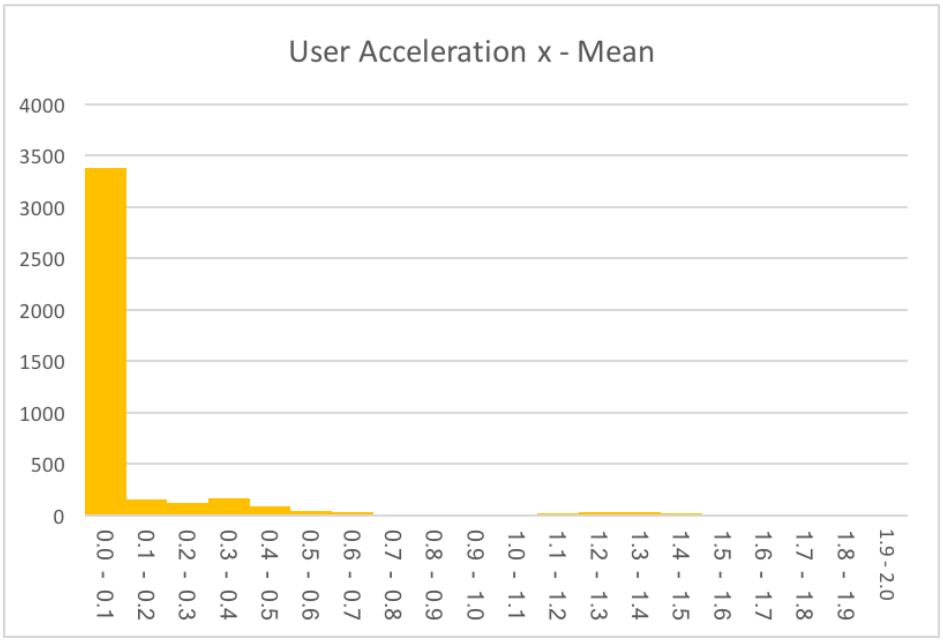


Figure 5.3: Distribution of instances for the user acceleration x mean feature.

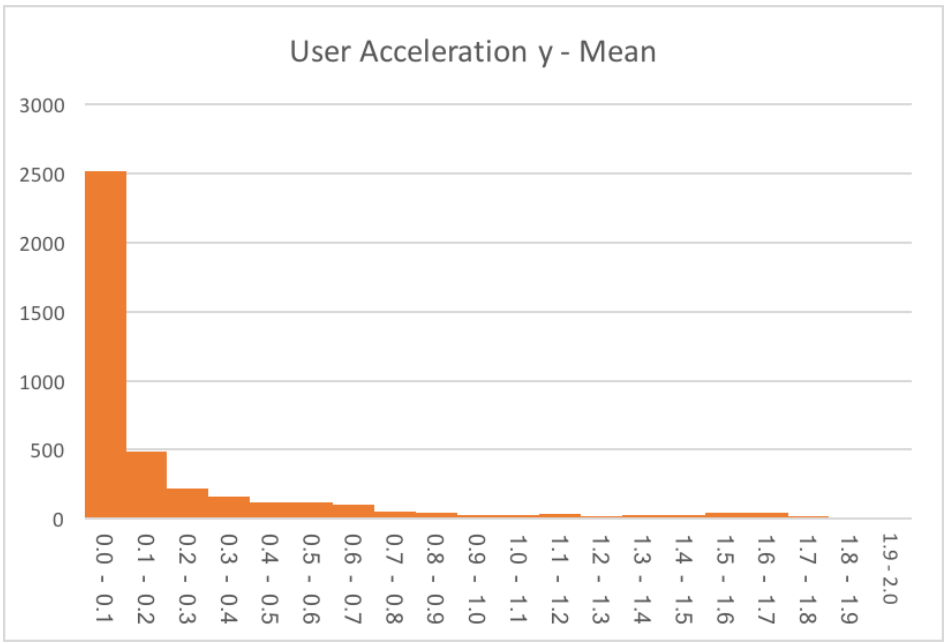


Figure 5.4: Distribution of instances for the user acceleration y mean feature.

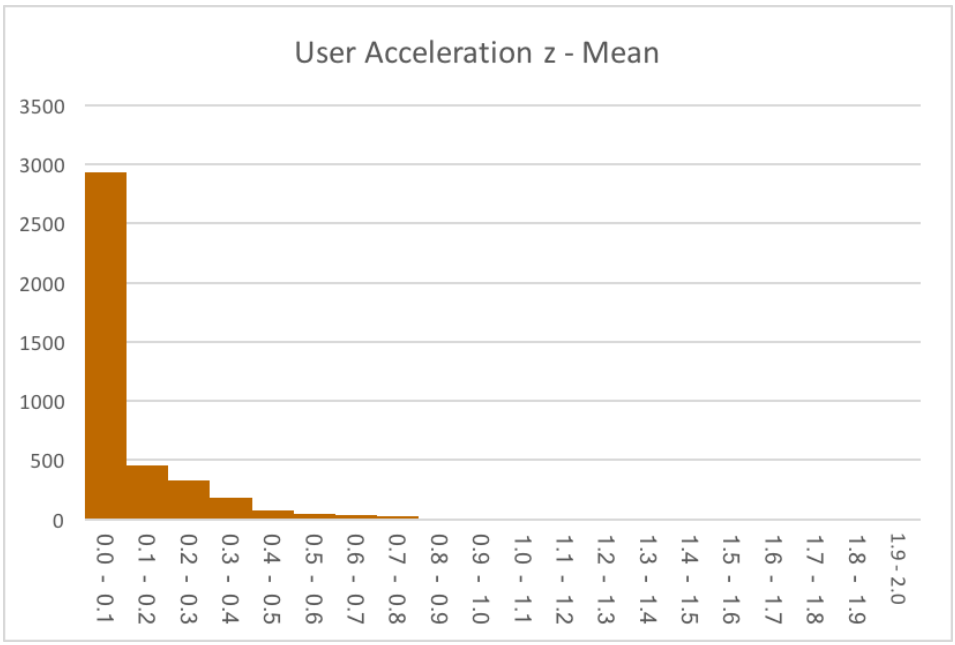


Figure 5.5: Distribution of instances for the user acceleration z mean feature.

Figures 5.3, 5.4 and 5.5 show the distribution of instances for every interval of the user acceleration mean value on the x, y and z axis respectively. As the figures show, the acceleration mean on the three axises show the characteristics of a long tail distribution. The higher the acceleration interval, the lesser the probability of finding an instance whose acceleration mean value falls within such range. As for the magnitude of the user acceleration mean, whose distribution is shown in figure 5.6, it still resembles a long tail distribution, but with a much less regular shape. Every distribution of the user acceleration mean along the three exises is not completely regular, and shows some points where the frequency increases instead of decreasing (e.g. between 0.2 and 0.3 in figure 5.3). The less regular shape of the distribution of magnitude values of the acceleration could explained with the fact that the acceleration magnitude is itself given by the squared sum of three acceleration components, thus inheriting the irregular shape of the three distributions, obtaining an irregularity that is the sum of more irregular

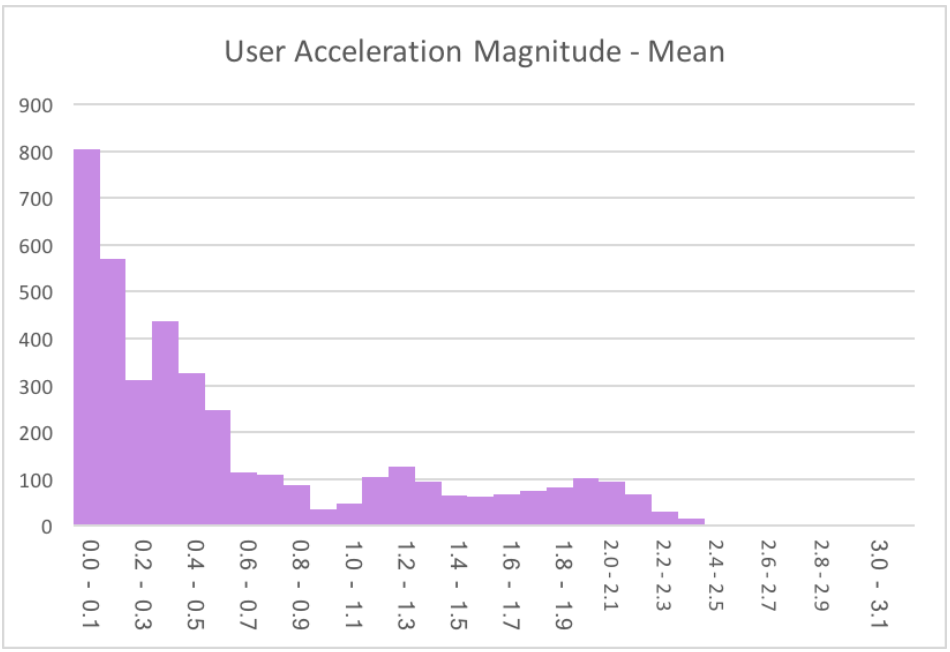


Figure 5.6: Distribution of instances for the user acceleration magnitude mean feature.

distributions.

Chapter 6

System Evaluation

As explained in chapter 5, the dataset was populated with 4083 instances, and as shown in figures 5.1 and 5.2, the data is well balanced among activities and subjects. For validation, the holdout method was used, employing 70% of the data for training the model, and the remaining 30% for validation. Each test was repeated 10 times, and the metrics were averaged over all the runs. The HAR system was evaluated considering many metrics, which summarise the performance of the system, in terms of time and recognition performance. The system was first evaluated using the machine learning scripts described in section 4.4, varying different configurations of algorithms [1], subjects, sensors, features and other options such as outlier removal and varying the dimension of the test set size. In the second place, the system was tested using the testing module of HAR App, the developed application, exploiting the logging capabilities described in subsection 4.2.2 in order to obtain a report of the predicted activities and calculate various metrics, which were compared with the metrics obtained from the machine learning scripts. Moreover, the CPU consumption of the testing module of the application was evaluated, varying different configurations of sensors.

[1] Details about the employed machine learning algorithms can be found in section 4.4

6.1 Machine Learning Results

6.1.1 Algorithms and Subjects

The first test compared different machine learning algorihtms, considering the model building time, which is shown in figure 6.1, and the recognition performance, shown in figure 6.2. As expected, the naive Bayes and kNN algorithms were the fastest, the former because it is based on simple probability computations, and the latter because it is based on lazy learning, which means that the learning process simply consists in copying the training set in memory, and all the computation is deferred at classification time. The slowest algorithm was multilayer perceptron, because it requires complex computations in order to taylor the neural network, finding the optimal configuration of connection weights; as expected, since the voting algorithm is a majority vote between five models, including multilayer perceptron, it inherits its slowness from the multilayer perceptron method, and obtained a building time of 3.89 seconds, which is close to the sum of the building time of the first five algorithms shown in figure 6.1. The most accurate recognition model was random forest, which scored an accuracy of 99.51%, followed by voting, which obtained an accuracy of 98.66%, and the SVC method, with a score of 97.79%. The decision tree method obtained a good combination of building time and accuracy, respectively 95.42% and 0.259 seconds; also because it is a simple and understandable model, it can be considered a good compromise, and for this reason the following metrics in this chapter will be reported using the random forest and the decision tree methods.

The so far reported results considered impersonal models, obtained using the whole dataset. A personal model includes only one subject, and it is generally more accurate than an impersonal model. Figure 6.3 shows the accuracy scores obtained with impersonal and personal models; the latter included one subject at time, whose information can be found in table 5.2. As expected, personal models outperformed the impersonal model; moreover, the performance discrepancy between the random forest and decision tree

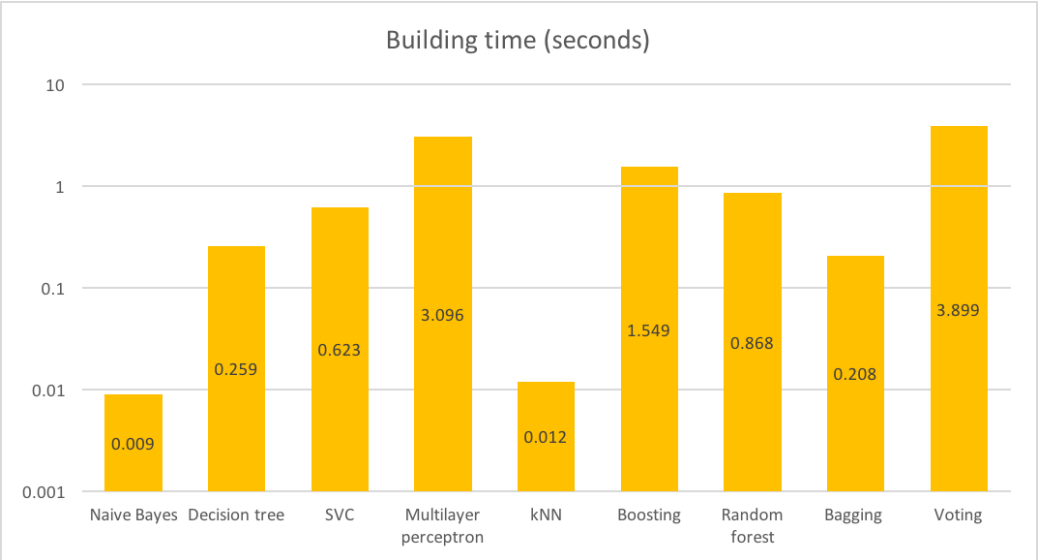


Figure 6.1: Model building time for different machine learning algorithms. The building time is reported in logarithmic scale.

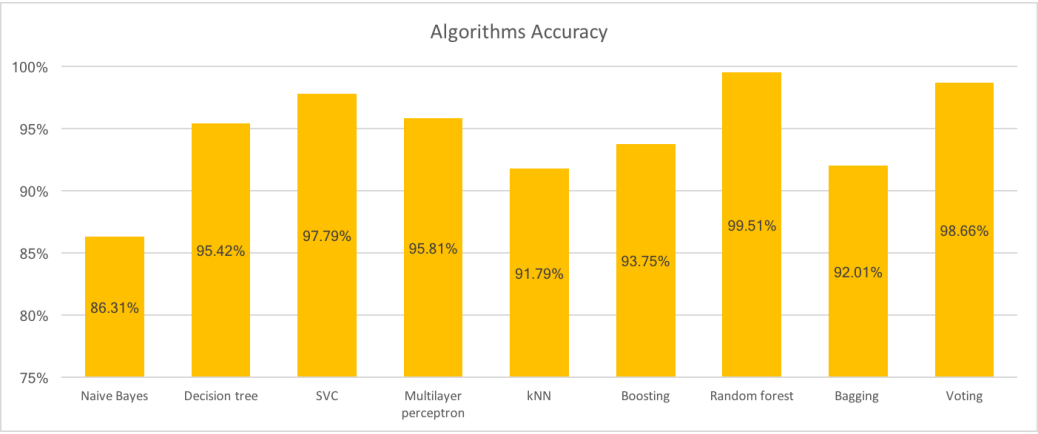


Figure 6.2: Accuracy obtained for different machine learning algorithms.

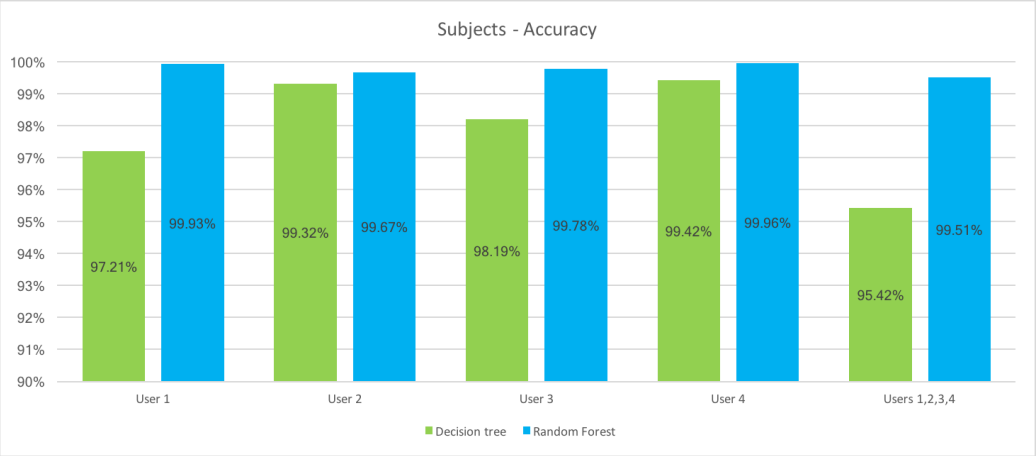


Figure 6.3: Accuracy of impersonal and personal models.

methods is higher for the impersonal model, with an accuracy difference of 4.09%, while the average gap of the personal models is 1.3%. This can be explained with the fact that impersonal models are more prone to overfitting, whose problem is eliminated by the random forest method.

The decision tree confusion matrix is shown in 6.4. As can be seen, resting was sometimes confused with sit-ups and viceversa, and this was due to the fact that the subjects were free to rest in the position that they found more comfortable, and sometimes they were laid in the same position used for performing sit-ups. Moreover, some instances of squats were confused with sit-ups and viceversa despite the fact that the two exercises are performed in completely different positions, but this can be explained with the fact that both squats and sit-ups are often performed crossing the arms. Sit-ups and resting were the least accurately predicted activities, because of their high variability. Indeed, the subjects rested in various positions, standing up, sitting down or lying down; as for the sit-ups activity, it has many variants and it has thus a higher inter-subject and intra-subject variability compared with other exercises. The random forest confusion matrix is shown in 6.5; as can be seen, the number of misclassified instances is highly reduced.

Activity	Push-ups	Sit-ups	Squats	Lunges	Jump Rope	Resting	Walking	Running
Push-ups	144	2	2	0	0	2	0	0
Sit-ups	1	133	4	2	1	5	2	0
Squats	1	4	144	2	0	2	0	0
Lunges	0	1	3	147	0	1	1	0
Jump Rope	0	0	0	0	152	0	0	0
Resting	2	6	4	1	0	140	1	0
Walking	0	1	0	1	1	1	157	0
Running	0	0	0	0	1	0	0	152

Figure 6.4: Decision tree confusion matrix.

6.1.2 Sensors and Features

Various sensors and features configurations were tested with the same methodology, using the decision tree and random forest methods. The following data was considered [2]:

- Heart rate
- Acceleration
- Rotation rate
- GPS data
- Subject information

Subject information included weight, height, age, gender, wrist location and crown orientation. A set of configurations containing only some of the above data was tested, and the experiment was repeated for both the decision tree and random forest methods, as shown in figures 6.6 and 6.7, where

[2]More details can be found in table 5.1

Activity	Push-ups	Sit-ups	Squats	Lunges	Jump Rope	Resting	Walking	Running
Push-ups	149	0	0	0	0	0	0	0
Sit-ups	0	144	0	0	0	2	0	0
Squats	0	0	153	0	0	0	0	0
Lunges	0	0	0	153	0	0	0	0
Jump Rope	0	0	0	0	154	0	0	0
Resting	0	1	0	0	0	152	1	0
Walking	0	0	0	0	0	0	162	0
Running	0	0	0	0	0	0	0	153

Figure 6.5: Random forest confusion matrix.

a checkmark under a sensor data label indicates that the data was included, while a ballot mark indicates that the data was discarded. At a first glance, it can be noticed that the heart rate monitor alone is not sufficient to obtain accurate results. Indeed, the accuracy was 71.5% and 71.23% for the decision tree and random forest algorithms. Also GPS data was not sufficient to obtain an accurate model, which obtained accuracy scores of 63.12% and 62.99% with decision tree and random forest. The sensor that obtained the greatest accuracy score alone was the gyroscope, more specifically the orientation alone scored 91.28% and 97.14%, while a combination of orientation and rotation scored scored 92.6% and 98.66%. The accelerometer alone scored an accuracy of 89.92% and 95.82%, which is enough to recognize activities with the sole sensor. Each sensor data positively contribued to the accuracy scores: the heart rate, even if it not enough if used alone, increased the accuracy from 63.12% to 91.81% if added to GPS data with the decision tree method, and from 62.99% to 92.37% with the random forest method. Similarly, removing GPS data from this configuration caused the accuracy to

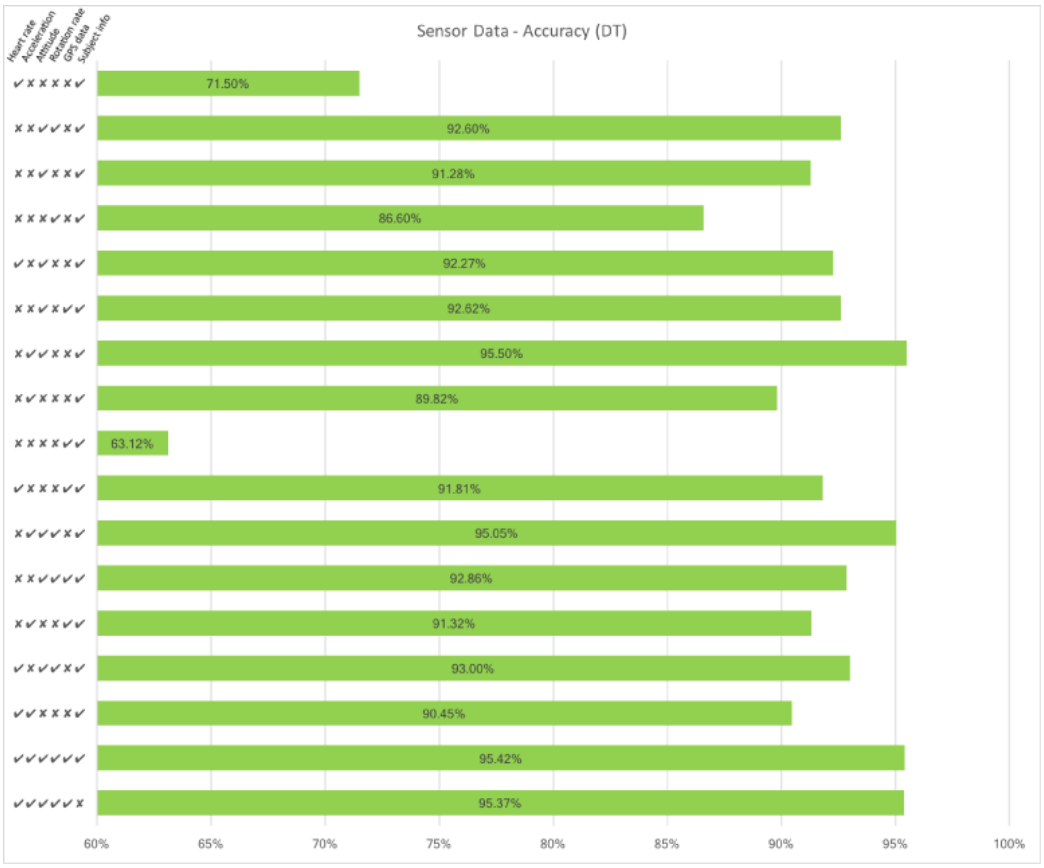


Figure 6.6: Accuracy scores of various sensor data configurations for decision tree. A checkmark indicates that the sensor data was included.

drop to 71.23% and 71.5%, as stated previously. Another important finding was that subject information did not significantly improve the recognition accuracy, which dropped only by 0.05% and 0.13% with decision tree and random forest. Information such as age, weight and height did not significantly contributed to the accuracy, but more importantly, the wirst location did not have a significant contribute, in spite of the fact the feature set included wrist location-dependent data. This suggests that the data used in this study is fit to build a wrist location-independent model. Indeed, the chosen features and data were similar to those included in [38], with the exception that the gyroscope and accelerometer data included all the four components, instead of using the magnitude only.

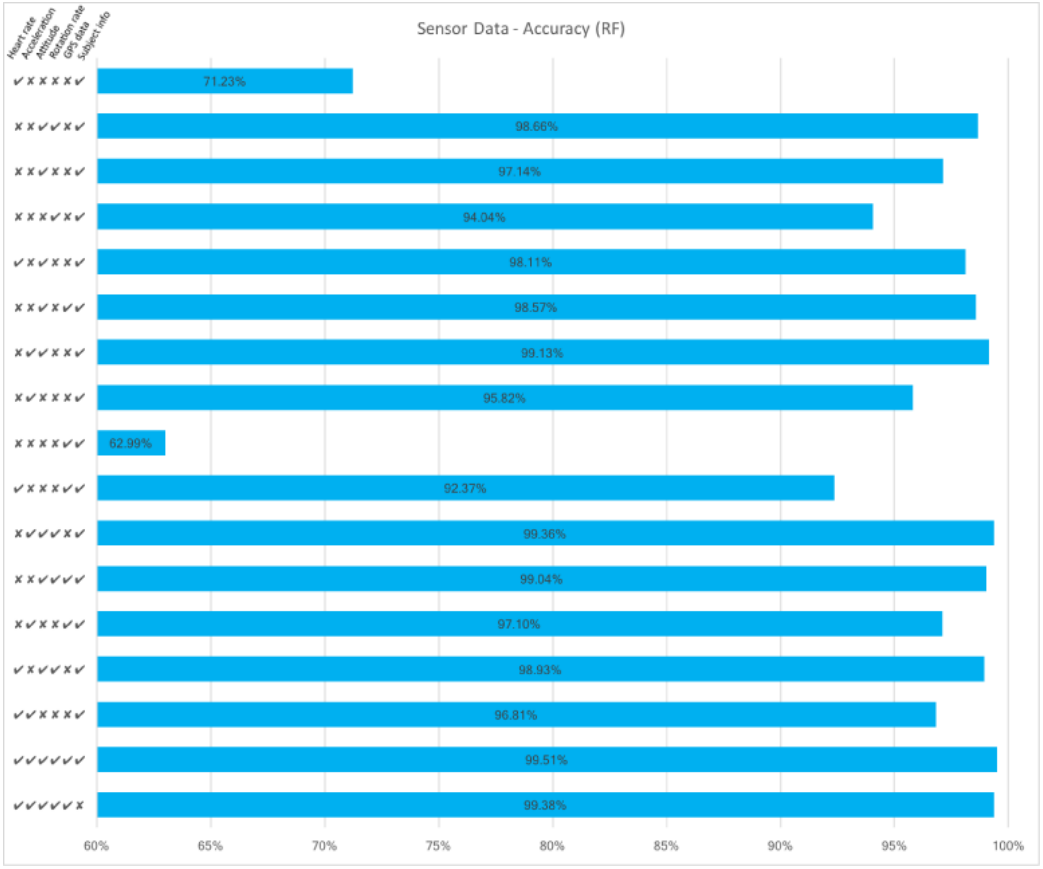


Figure 6.7: Accuracy scores of various sensor data configurations for random forest. A checkmark indicates that the sensor data was included.

Another experiment was carried out measuring the accuracy score of models trained using only one feature in isolation, comparing them with the accuracy obtained using all the features. For the decision tree method using some features in isolation did not significantly degrade recognition accuracy. Like shown in figure 6.8, all the features except skewness and kurtosis were enough, if used alone, to obtain more than 92% of accuracy; the mean and the median obtained even a greater accuracy than the model built including all the features. This is probably due to the tendency of decision trees to go in overfitting when a too large number of features is used. To prove this hypothesis, the results of the same experiment carried out with the random

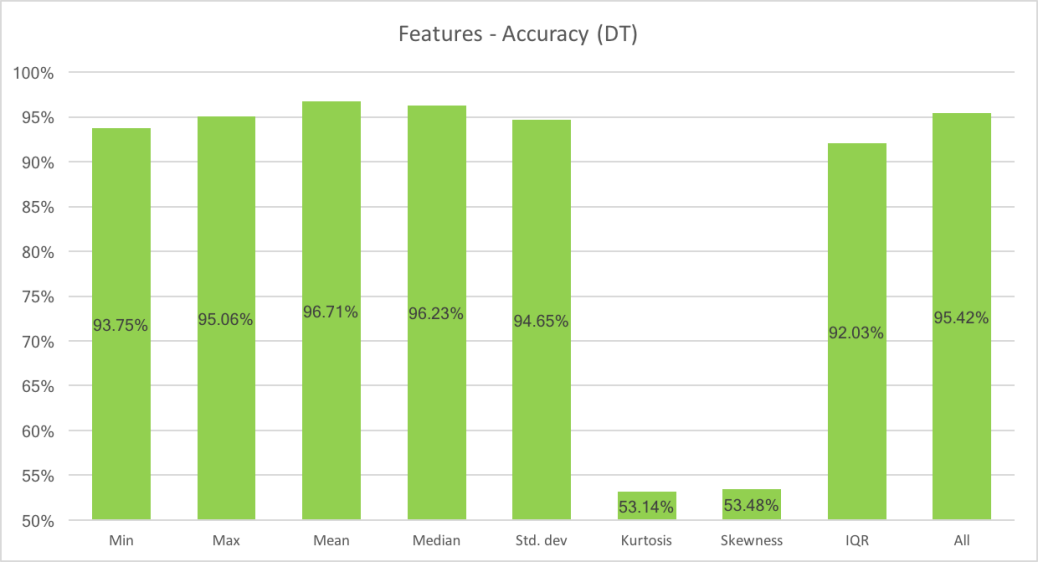


Figure 6.8: Accuracy of each feature used in isolation, compared with the accuracy obtained using all the features (decision tree).

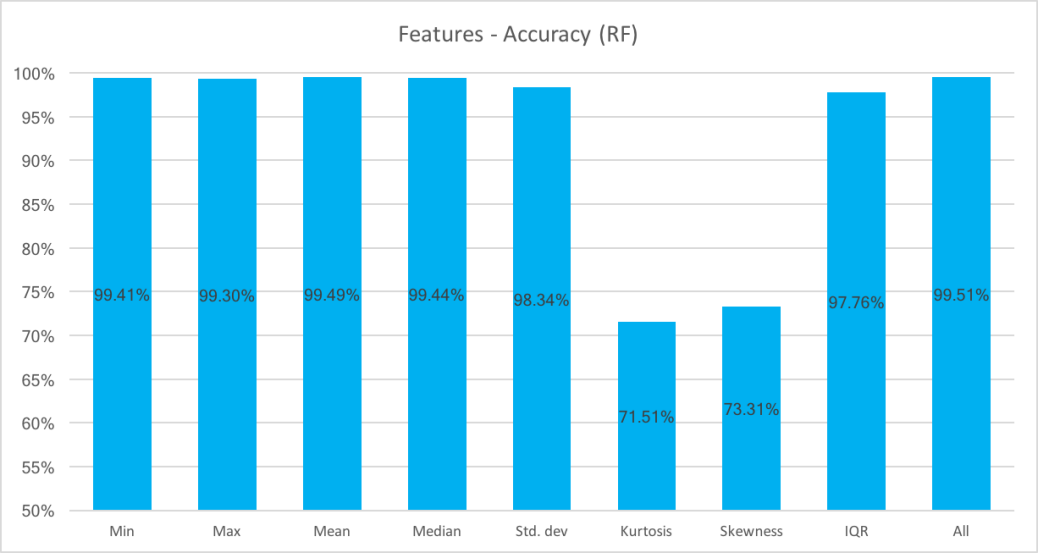


Figure 6.9: Accuracy of each feature used in isolation, compared with the accuracy obtained using all the features (random forest).

forest algorithm are shown in figure 6.9; while it is still true that some features used alone have a performance that is comparable with the accuracy score of all the features, no fearture alone obtained a greater accuracy than the all-features configuration. Also for the random forest method skewness and kurtosis resulted the least accurate features if used alone. This suggests that the shape and the irregularity of the samples distribution is not much relevant to recognize this kind of physical activities with the watch.
Finally, an experiment that tested various configurations of sensor data and features was performed. The configurations are shown in the below table:

Configuration	**Sensor data**	**Features**
1	Gyroscope - attitude	Mean
2	Gyroscope - attitude and rotation rate	Mean
3	Gyroscope - attitude and rotation rate	Median
4	Automatic sensor data and feature selection[3]	
5	Gyroscope - attitude and rotation rate	Automatic feature selection[4]
6	All	All

Table 6.1: Configurations of sensor data and features.

[3]Selected Features: attitude-roll-max, attitude-yaw-deviation, attitude-pitch-median, attitude-pitch-mean, attitude-pitch-deviation, attitude-magnitude-max, rotationRate-x-IQR, rotationRate-y-min, rotationRate-z-IQR, rotationRate-magnitude-median, userAcceleration-x-IQR, userAcceleration-y-max, userAcceleration-y-deviation, userAcceleration-z-deviation, userAcceleration-magnitude-median, userAcceleration-magnitude-mean, altitude-max.

[4]Selected features: attitude-roll-median, attitude-pitch-max, attitude-pitch-min, attitude-pitch-median, attitude-pitch-mean, attitude-pitch-deviation, attitude-magnitude-max, attitude-magnitude-median, attitude-magnitude-mean, rotationRate-x-deviation, rotationRate-x-IQR, rotationRate-z-deviation, rotationRate-z-kurtosis, rotationRate-magnitude-median, rotationRate-magnitude-mean, age.

An algorithm of automatic feature selection based on decision trees was used to automatically select features in configuration 4. In configuration 5, the same algorithm was applied after discarding all the sensors except the gyroscope. In all the configurations, subject information were kept and in the case of automatic feature selection, they were among the other candidate features.
Figure 6.10 show the accuracy results of activity recognition performed with the configurations of table 6.1. Again, the decision tree algorithm improved its accuracy with automatic feature selection (configuration 4), increasing from 95.42% to 96.08%, due to its tendency to overfit the training data if the number of features is too high. On the contrary, for the random forest method no configuration resulted in higher accuracy than the configuration with all sensor data and features. Both for decision tree and random forest, the best configuration was the one with automatic feature selection on all the features. Considering that only 17 features were included in configuration 4, and that 134 were included in configuration 6, automatic feature selection with random forest can still be considered a good compromise between accuracy and efficiency.

6.1.3 Other Experiments

The first test was about outlier removal, which relied on an outlier detection algorithm which received a contamination parameter as input, the greater its value, the greater the number of detected outliers. The outlier detection algorithm was run on the training set only, and it was used to train a recognition model to be tested on the test set, which contained all the data, without applying the outlier removal. Figure 6.11 shows the results for both the decision tree and random forest methods. The contamination parameter is on the x axis, and between 0 and 1% there are some apparently random fluctuations, with the accuracy that goes up and down but without showing any significative improvement or deterioration. After 2%, the accuracy starts decreasing slowly, losing about the 1%. The conclusion of this experiment is

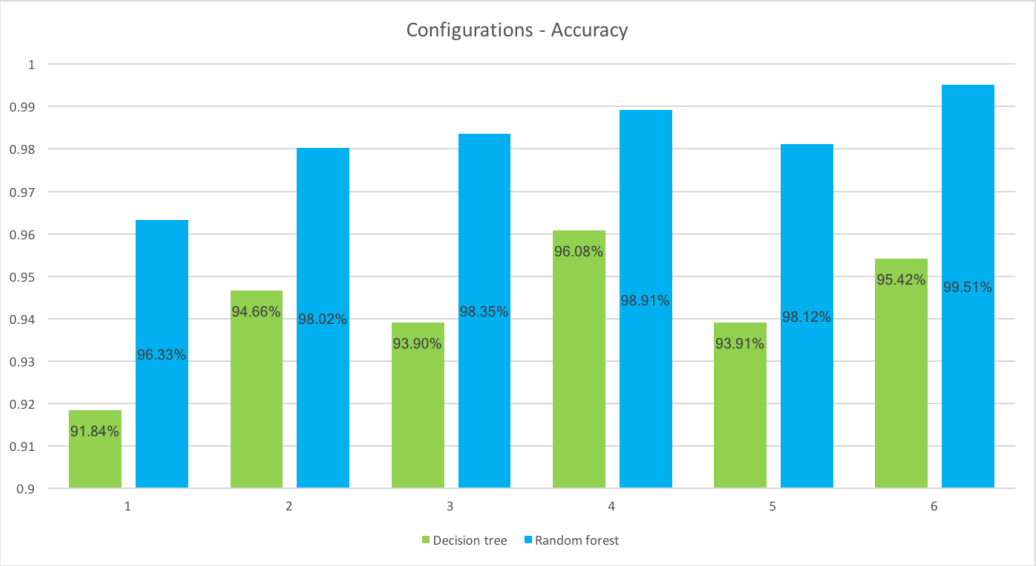


Figure 6.10: Accuracy of various configurations of sensor data and features with decision tree and random forest. The configurations are shown in table 6.1

that outlier removal did not bring any benefit to activity recognition.

The second experiment reported in this subsection is about the accuracy variation with different values of the test set size. As shown in figure 6.12, as the test set size is increased, the accuracy went down, because a smaller test set size means that there is more data available to train the recognition model, which is therefore more accurate. For the random forest algorithm, the accuracy starts from 99.75% with a size of 1%, and goes slightly down until it reaches the value of 99.51% with the nominal size of 30%, and it starts abruptly decreasing around 80%. The decision tree algorithm shows a similar behavior, except for the fact that the accuracy starts with a value of 95.36% with a test set size of 1%, and it increases up to 96.29% with a size of 5%, and then starts decreasing.

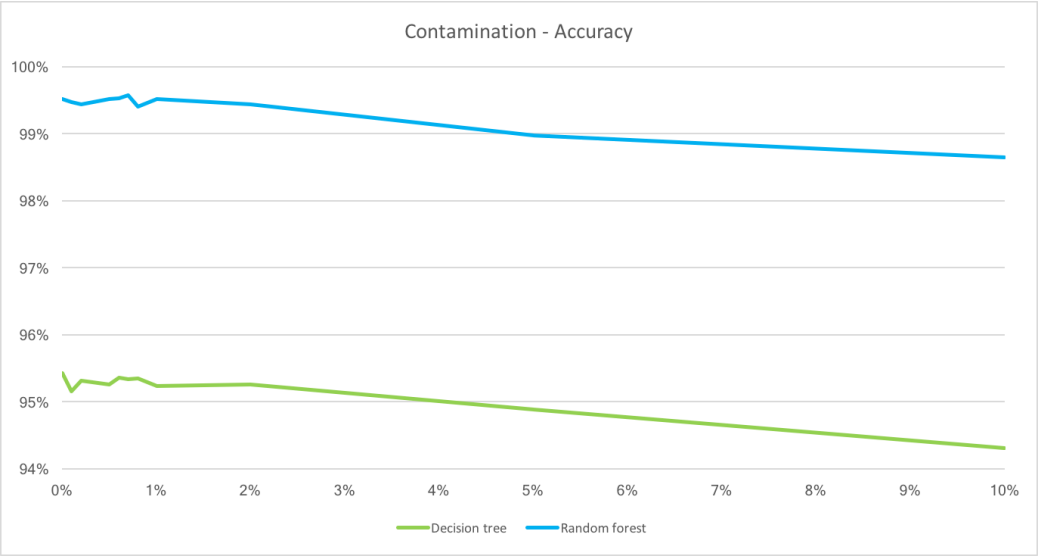


Figure 6.11: Accuracy of outlier removal varying the contamination parameter.

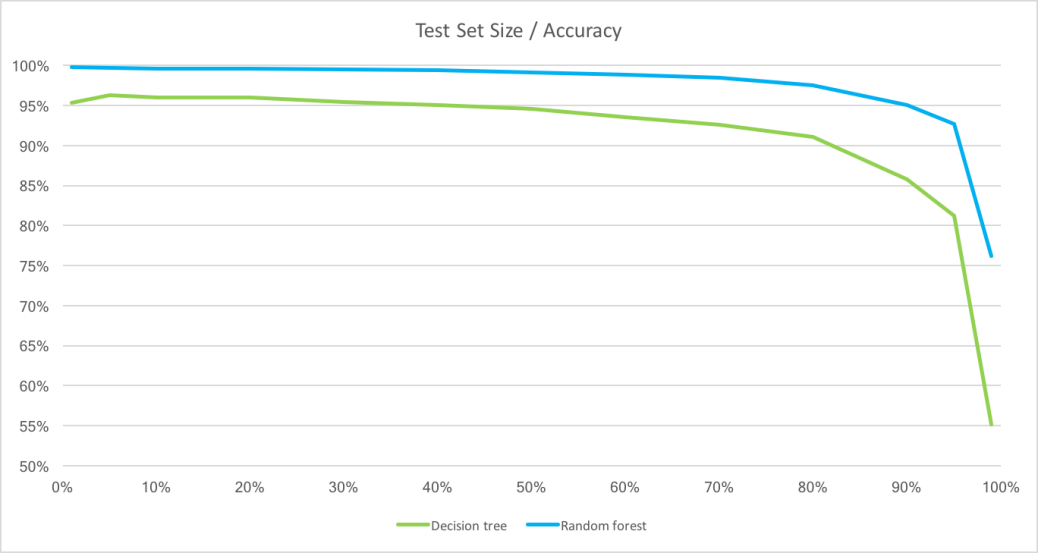


Figure 6.12: Accuracy varying the test set size.

6.2 In-App Tests

To further validate the HAR system, two more tests were performed from within the application, in order to measure: i) recognition performance, ii) CPU consumption. The recognition performance test was expected to score different results from the test carried out on the test set, because there is always some dependency between training set and test set data; indeed, since it was chosen to use windows with 50% overlaps, instances calculated in contiguous window were partly calculated on the same data, and if the criterion is simply to randomly select 30% of the data for testing, then some couples of instances with similar features were split between the training set and the test set. Moreover, even instances that were not calculated in contiguous time windows may have some dependency if they were calculated during the same activity in a close lapse of time.
As there is no way to measure the battery consumption of the watch application, the CPU test should give an approximate idea of the battery consumption. Nevertheless, there are some problems that should be taken into account:

- The CPU usage does not take in consideration other hardware that consume battery power, such as the watch green LEDs, the light sensors and the accelerometer-gyroscope.
- Services that allow to read the accelerometer, the gryscope, the GPS position rely on external libraries and routines that are not run within the application. It is not possible to know if a certain algorithm (e.g. the GPS triangulation algorithm) is run within the application or in another program.
- When available, the apple watch uses the GPS position obtained from its paired iPhone instead of locally querying its GPS. In this case, the CPU usage of the watch application does not take into account the CPU time spent on the iPhone to calculate the position.

- During a workout, the heart rate monitor is always active, even if the heart rate is not being queried. This means that if the heart rate monitor is excluded from the sensors used for HAR, it is still active and it consumes its amount of CPU (either within the application or outside).

For these reasons, the CPU usage of the watch application is not directly linked with the total power consumption needed in order to recognize activities. The CPU usage should only give an idea of what the application consumes in order to run classify activities, considering that some variables may be taken outside of the equation.

6.2.1 Recognition Performance

As mentioned in subsection 4.2.2, the testing module of HAR App relies on a history set, which stored the last n classifications inside a queue, and returns the most frequent activity inside the queue as the final prediciton. The size of the history set is configurable with a value of 1, 3, 5 or 7. A decision tree [5] was validated within the application by the user number 2 in table 5.2, classifying a total of 1442 instances, which corresponds to 35% of the dataset, slightly more than the test set size. Every activity was included in the test, collecting 173 ± 27 instances for every activity. Figure 6.13 shows the confusion matrix with no history set applied. 27 instances of squats were misclassified as resting, and 10 as walking. Moreover, 50 instances of jump rope were misclassified as walking, and the sit-ups activity was misclassified 21 times as squats and 13 times as resting. The interesting fact about this confusion matrix is that the misclassifications are not symmetrical: despite a large number of instances of jump rope were misclassified as walking, no instances of walking were misclassified as jump rope; similarly, no instances of resting and walking were misclassified as squats, and no instances of squats

[5] The decision tree was trained on the whole dataset, using all the features and the sensor data available.

Activity	Push-ups	Sit-ups	Squats	Lunges	Jump Rope	Resting	Walking	Running
Push-ups	143	0	3	0	0	0	0	0
Sit-ups	3	163	21	13	0	0	0	0
Squats	0	0	132	0	0	27	10	0
Lunges	0	0	0	192	0	0	0	0
Jump Rope	0	0	1	0	129	0	50	0
Resting	1	3	0	0	0	151	0	0
Walking	0	6	0	0	0	2	192	0
Running	0	0	0	0	2	0	0	198

Figure 6.13: Confusion matrix without history set.

or lunges were misclassified as sit-ups. While there is some similarity between some of these activities (e.g. between sit-ups and squats if they are performed crossing the hands, or between jump rope and walking because of the similar positions), the misclassifications seem to be mostly due to overfitting more than because there are confusing patterns in activities than can be performed similarly.

Figures 6.14, 6.15 and 6.16 show the confusion matrices for history set sizes of 3, 5 and 7. Interestingly, as the history set size is increased, as the number of misclassified instances of some pairs of activities either increase of decrease. For example, increasing the history set size from 1 to 7, decreases the number of squats instances misclassified as resting from 27 to 10. On the contrary, increasing the history set size from 1 to 7, increases the number of sit-ups instances misclassified as squats from 21 to 31. There is an explanation for this behavior; for instance, let's consider this sequence of classifications of the jump rope activity: "walking, jump rope, jump rope, walking, walking, walking, jump rope, jump rope, walking", which was taken

Activity	Push-ups	Sit-ups	Squats	Lunges	Jump Rope	Resting	Walking	Running
Push-ups	143	0	3	0	0	0	0	0
Sit-ups	4	164	21	11	0	0	0	0
Squats	0	0	154	0	0	15	0	0
Lunges	0	0	0	192	0	0	0	0
Jump Rope	0	0	0	0	130	0	50	0
Resting	0	2	0	0	0	153	0	0
Walking	0	4	0	0	0	1	195	0
Running	0	0	0	0	0	0	0	200

Figure 6.14: Confusion matrix with history set size = 3.

Activity	Push-ups	Sit-ups	Squats	Lunges	Jump Rope	Resting	Walking	Running
Push-ups	143	0	3	0	0	0	0	0
Sit-ups	1	162	27	10	0	0	0	0
Squats	0	0	158	0	0	11	0	0
Lunges	0	0	0	192	0	0	0	0
Jump Rope	0	0	0	0	123	0	57	0
Resting	0	0	0	0	0	155	0	0
Walking	0	4	0	0	0	0	196	0
Running	0	0	0	0	0	0	0	200

Figure 6.15: Confusion matrix with history set size = 5.

Activity	Push-ups	Sit-ups	Squats	Lunges	Jump Rope	Resting	Walking	Running
Push-ups	146	0	0	0	0	0	0	1
Sit-ups	0	158	31	10	0	0	0	0
Squats	0	0	159	0	0	10	0	0
Lunges	0	0	0	192	0	0	0	0
Jump Rope	0	0	0	0	121	0	59	0
Resting	0	0	0	0	0	155	0	0
Walking	0	4	0	0	0	0	196	0
Running	0	0	0	0	0	0	0	200

Figure 6.16: Confusion matrix with history set size = 7.

from real classification data. If the classifications start from the fifth entry in order to allow a history set sized 5 to classify as many activities as in the case where no history set is used, then the recognition accuracy with no history set is 50% (2 of last 4 activities are correctly classified), which is the same accuracy obtained with a history set large 3. But if the history set size is increased to 5, the accuracy drops to 0%. The conclusion is that a history set allows for a greater recognition accuracy only when the recognition is enough accurate, while in case that the misclassified instances outnumber the correctly classified instances, a larger history set size only decreases the accuracy. Analyzing the classification data, it was found that this kind of incorrect behavior occurred only during certain classifications, while in other situations the prediction accuracy was high (e.g. during certain tests, no jump rope instances were misclassified as walking).

Figures 6.17 to 6.24 show the accuracy, recall, precision and F1 scores for all the activities. The most well recognized activities were: push-ups, lunges, resting and running. Jump rope was the most problematic, due to the fact

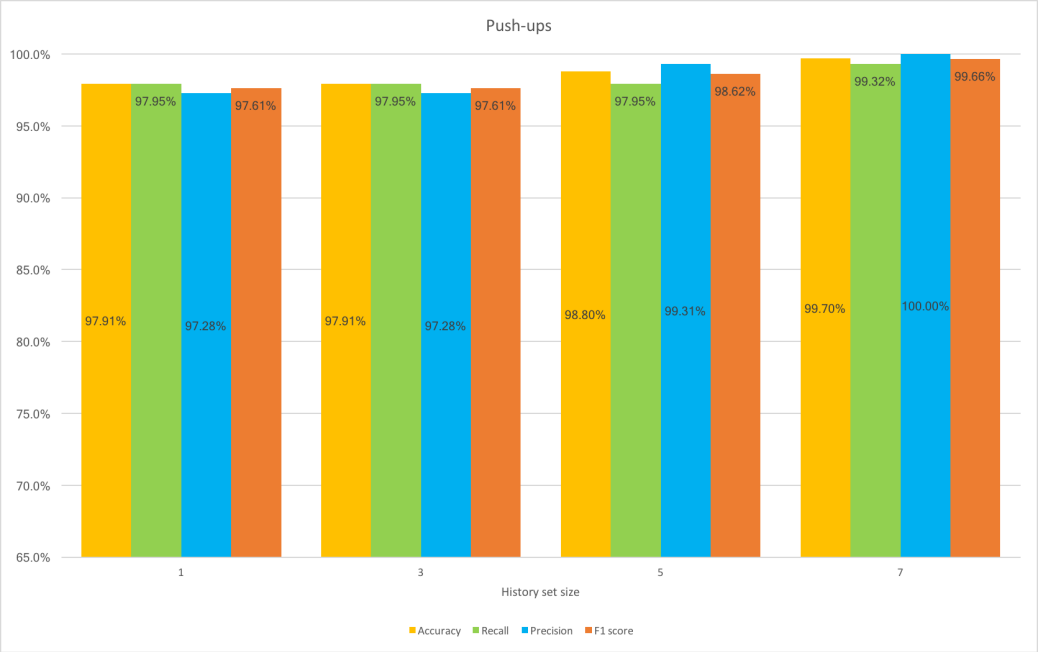


Figure 6.17: Push-ups metrics.

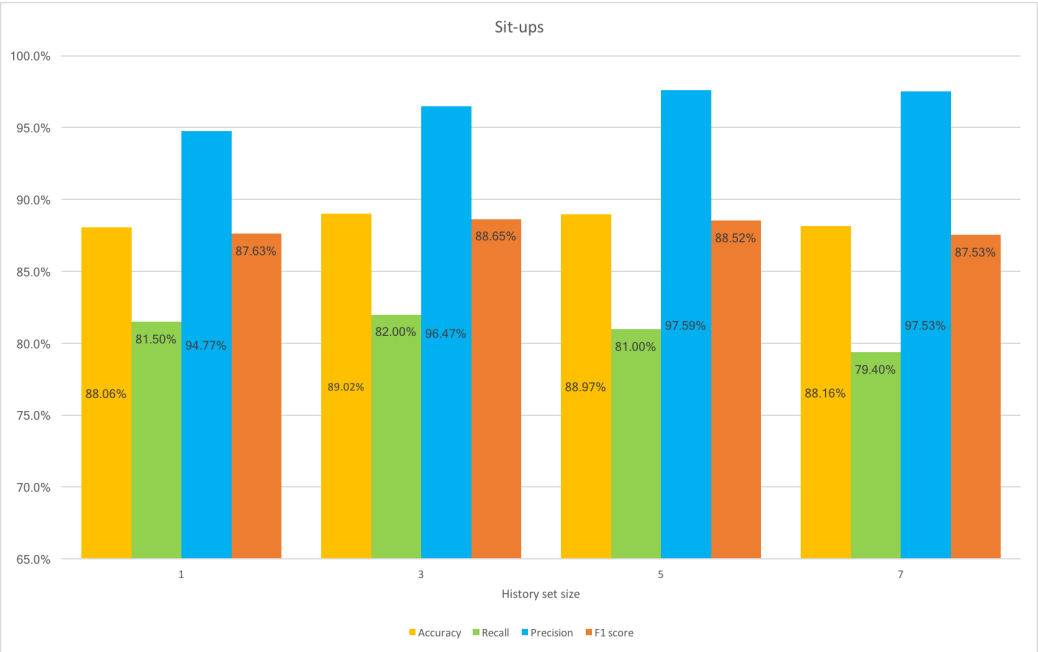


Figure 6.18: Sit-ups metrics.

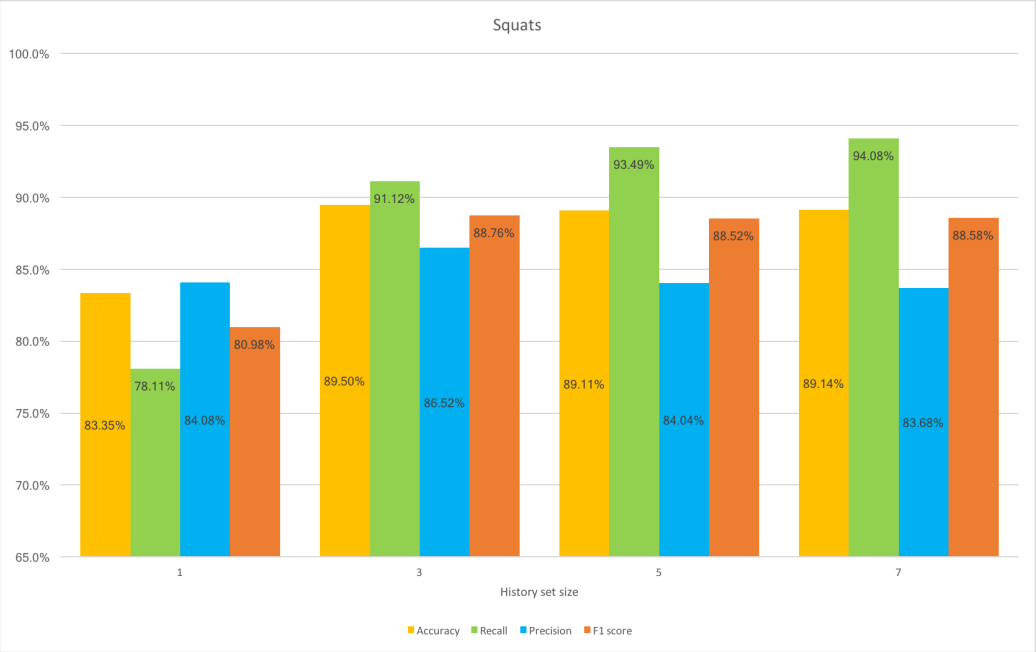


Figure 6.19: Squats metrics.

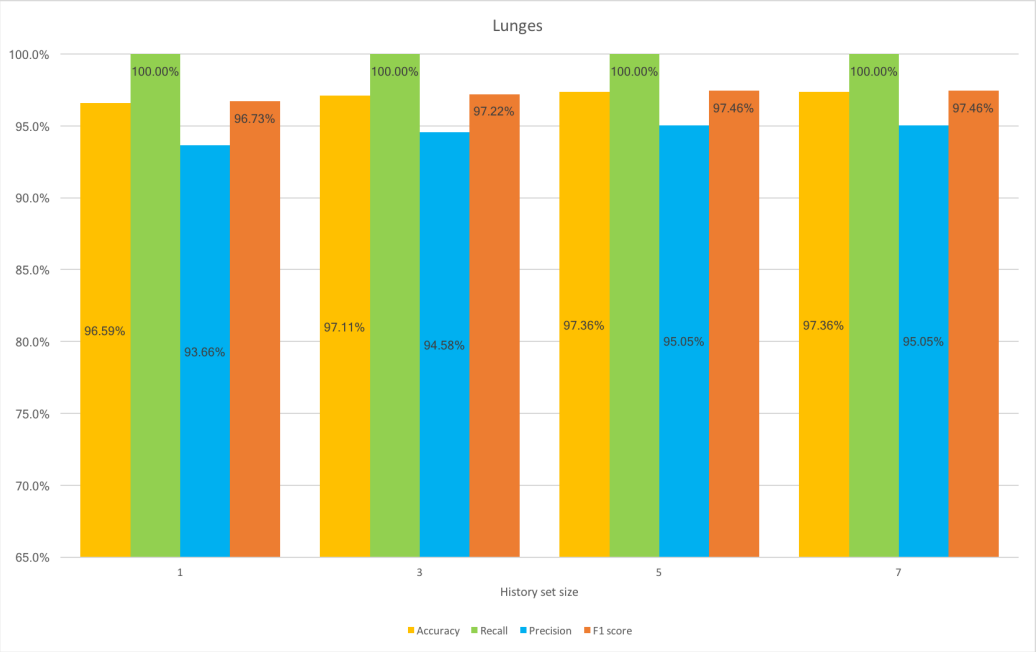


Figure 6.20: Lunges metrics.

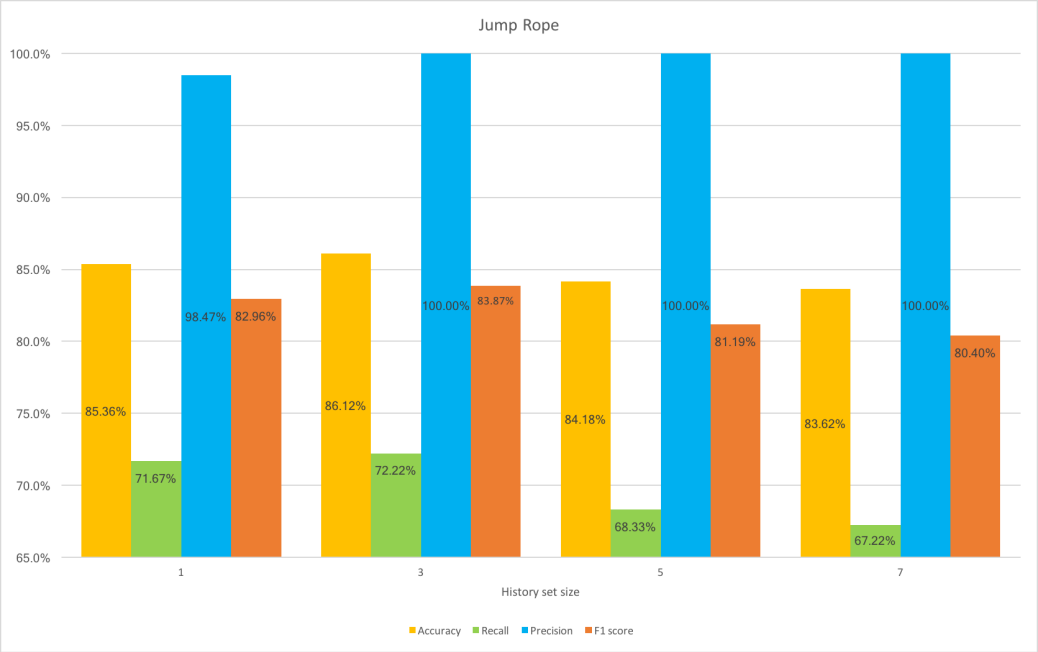


Figure 6.21: Jump Rope metrics.

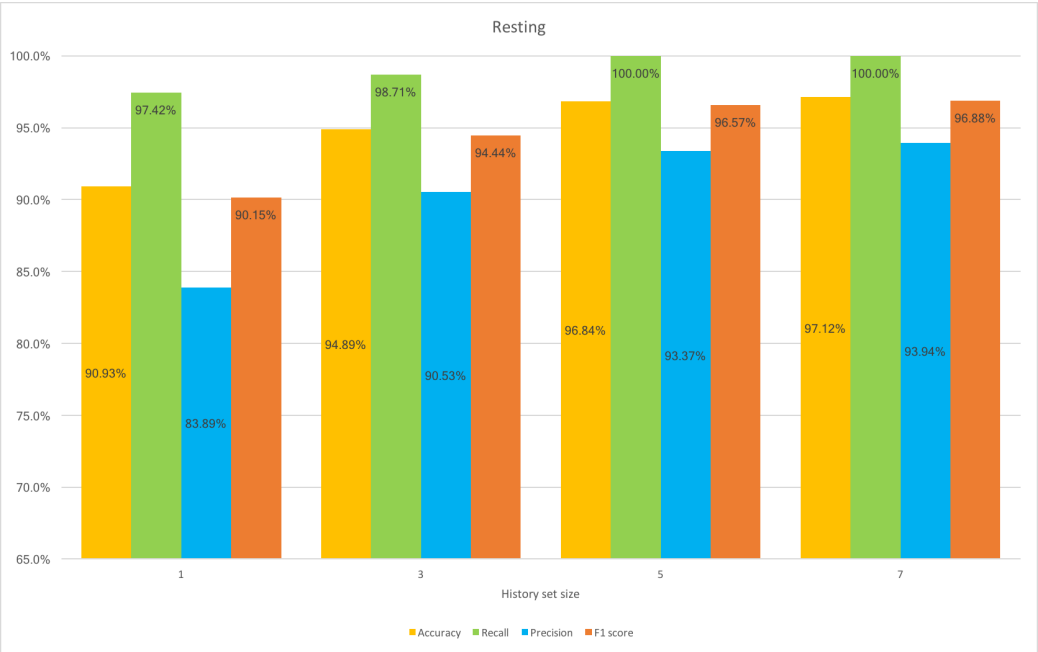


Figure 6.22: Resting metrics.

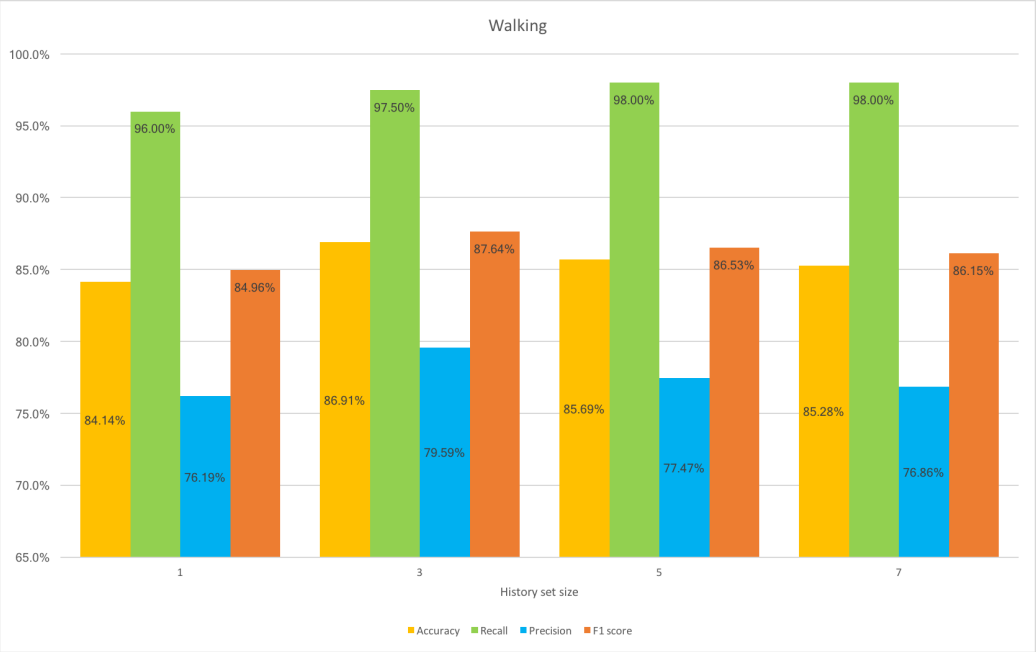


Figure 6.23: Walking metrics.

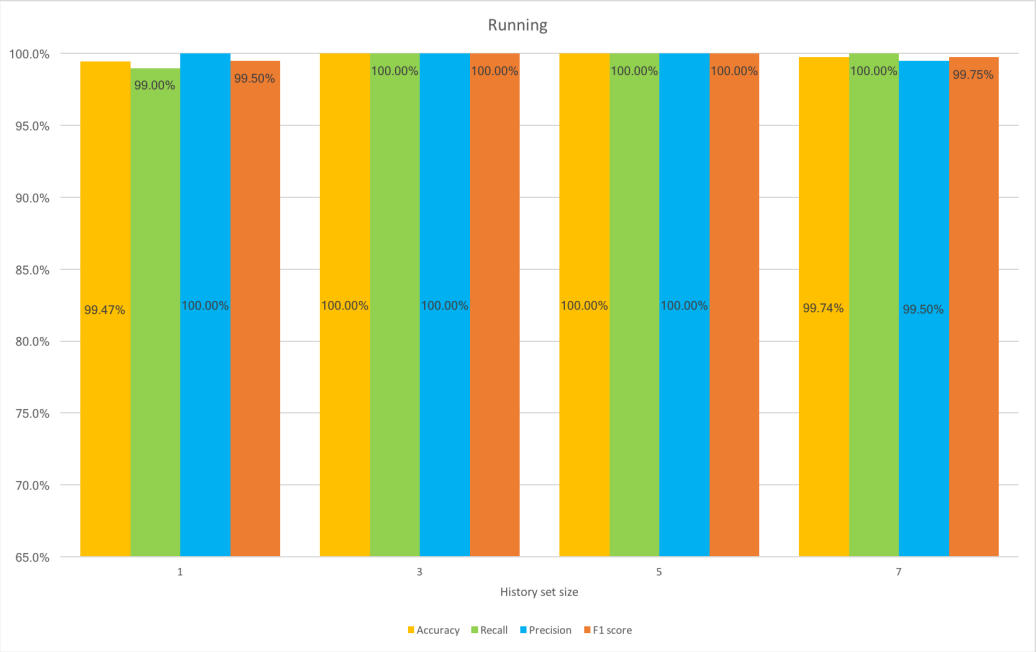


Figure 6.24: Running metrics.

that it was often confused with walking. There is an interesting thing to notice about these metrics: some activities have balanced values of recall and precision, while some show a great discrepancy, having a recall value significantly greather than precision or viceversa. Like explained in subsection 2.3.1, the recall inversely depends on the number of false negatives: if the activity is often misclassified as another activity, then the recall is low. Precision instead inversely depends on the false positive rate: given an activity C_j, if other activities are often misclassified as C_j, then the precision score of C_j will be low. The F1 score takes both the recall and precision into account. For this reason, an activity like jump rope has a great precision score (up to 100% for a history set size greater of equal than 3), but a very low recall. Moreover, because of the previously mentioned behavior of increasing misclassifications between some activities as the history set size grows, also the recall of jump rope decreases as the history set size grows. The F1 score is a balanced metric which takes into account of both precision and recall, and it shows the highest value for the jump rope activity with a history set size of 3. For the opposite reason, walking has a high recall score but a low precision score, because most of the instances of the walking activity were correctly classified, but many instances of jump rope were misclassified as walking. For the walking activity, the recall increases as the history set size grows, while the precision first increases up to 79.59% with a history set size of 3, and then decreases. From 3 to 5 there is an inverse tendency between recall and precision: the recall increases, while the precision decreases. Another activity that show an inverse tendency between precision and recall in some points of the bar chart are resting and sit-ups.

Figure 6.25 show the average metrics for all the activities. If the F1 score is chosen to evaluate the recognition performance due to the fact that it takes into account both recall and precision, then the best history set size is 3, with an F1 score of 92.27%. If the history set size is further increased, the F1 score slighly goes down and the same is true for the accuracy, which reaches the value of 92.68% and then slighly decreases. A history set of 7 activities is

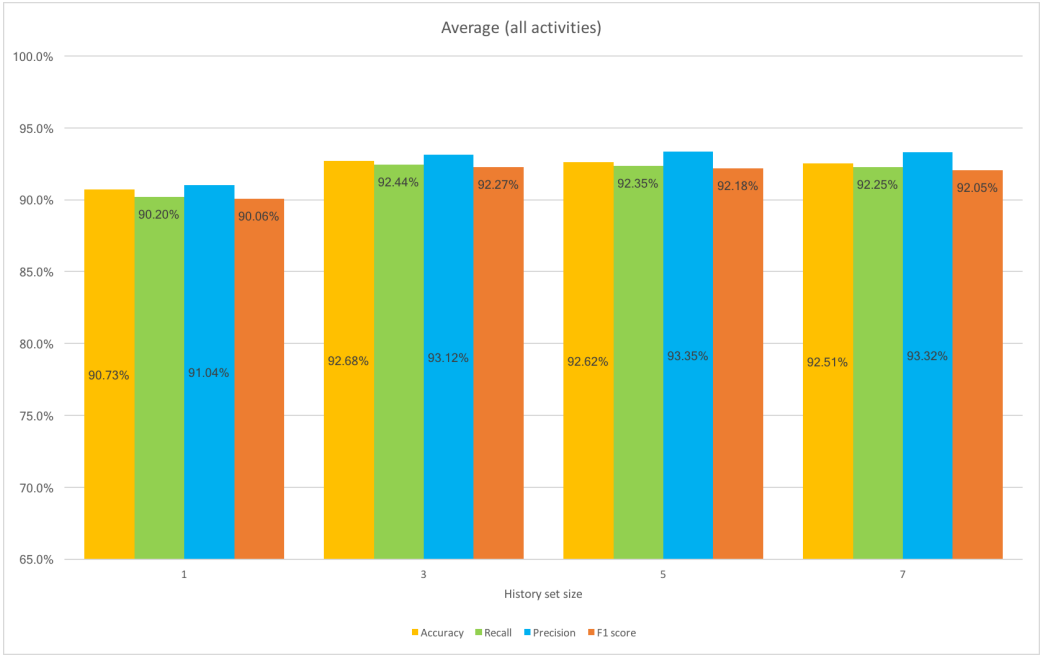


Figure 6.25: Average metrics (all ativities).

almost as accurate as a history set of 3 activities, but since a greater history set decreases the recognition accuracy during transitions, it is preferable to use a history set of 3 elements, which can be considered the best size for this kind of activities in this study.

6.2.2 CPU Time

The last in-app test regarded the CPU consumption for each configuration of sensors, which is shown in figure 6.26. Because of the previously mentioned problem, the CPU time is not directly linked with battery consumption, and it is unbalanced, taking into account mostly the overhead due to feature extraction rathern than the overhead due to the use of sensors, such as the CPU needed to run the GPS triangulation algorithm, which most likely runs in a watchOS system routine or in the iPhone. The CPU time was measured over 60 seconds of use of the application, which gives an idea of how much CPU the application needs to allocate in order to perform activity

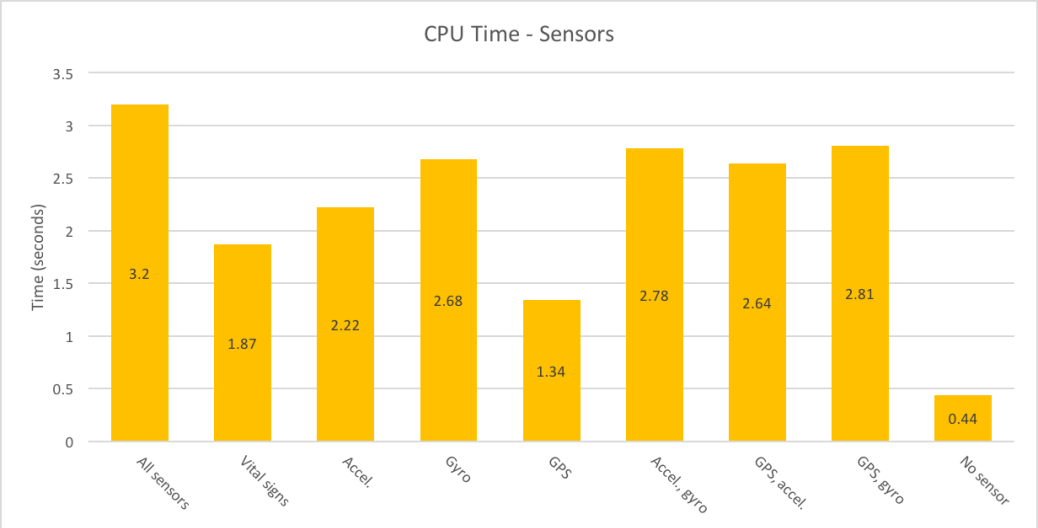


Figure 6.26: CPU time for different configurations of sensors, measured over 60 seconds.

recognition. As expected, the results are highly dependent on the number of features included. The number of features associated with each sensor are: i) gyroscope: 64, ii) accelerometer: 32, iii) GPS: 24, iv) heart rate monitor: 8. With the exception of the GPS which consumed 1.34 seconds, less than the heart rate monitor which consumed 1.87 seconds, the results seem to be ordered by the number of features included. Considering that the configuration that included all the sensors consumed 3.2 seconds versus 2.78 seconds consumed with only the acceleromter and the gyroscope, and that the accuracy obtained using only the acceleration and the attitude was greater than the accuracy of all the sensor data (see figure 6.6), then the conclusion is that the GPS and the heart rate monitor can be considered redundant in this study, and they did not significantly increase the accuracy. This reinforces the findings in [33], where it was concluded that the heart rate monitor was not useful in HAR, because the heart rate is slow to change and it is therefore sensible to the activities performed in the past minutes. The best configuration would use only the accelerometer and the gyroscope, and to further save energy it would make use of feature selection, which also

reduces overfitting with decision trees.

Chapter 7

Conclusions

A human activity recognition system based on the Apple Watch was developed, using an iPhone and an external machine, in order to recognize the following activities: push-ups, sit-ups, squats, lunges, jump rope, resting, walking and running. The system is able to collect data in order to train and test an activity classifier. The data is stored in the phone, while the machine learning scripts are located in a third machine, and they need manual actions in order to build and export a recognition model, which can be manually imported in the watch. The data collection phase involved four subjects, all males, aged between 23 and 46. Due to the strict CPU, memory and bandwidth limits on the Apple Watch, it was chosen to directly extract the features on the watch and send them in real-time to the iPhone, instead of sending the raw sensor data. The choice of the sampling frequency and which features and sensor data to use was also dictated by these problems, therefore a low sampling frequency of 16 HZ was chosen in order to reduce the CPU consumption of the application, and a fixed subset of computationally efficient features and sensor data was chosen, with a window length of 2.5 seconds with 50% overlaps. 4,083 instances were collected, corresponding to about 20 minutes of physical activity for every subject. 9 different recognition algorithms were trained and evaluated on the external machine, using the holdout validation with a test set of 30%, obtaining an accuracy

of 95.42% with decision tree, and 99.51% with the random forest method, which resulted the most accurate. Various configurations of sensor data and features were tried, coming to the conclusion that the heart rate monitor and the GPS did not significantly increase the recognition accuracy when the gyroscope and accelerometer are used, but they are useful when there is not enough sensor data from the accelerometer and the gyroscope (e.g. adding the heart rate monitor to the GPS increases the accuracy from 62.99% to 92.37% with the random forest method). The accelerometer and the gyroscope alone were able to achieve an accuracy of 99.36% with random forest and 95.05% with decision tree. Another important finding derived from validation: despite each subject wore the watch in a different location during the study (two on the left wrist, two on the right), the wrist location information did not significantly increase the recognition accuracy, demonstrating that the recognition model is independent from the wrist location. A similar conclusion was made in a study on smartwatches that included similar features [38]. Moreover, activities that involve similar gestures or that are performed in similar positions were confused more often with each other: sit-ups were confused with squats due to the fact that they are both often performed crossing the hands, while resting was often confused with sit-ups due to the fact that both activities can be perfomed while lying down. Automatic feature selection showed an increase in recognition accuracy for the decision tree learning method, because it is more prone to overfitting when too many features are included, while it did not increase the accuracy of the random forest model. Another finding from the validation was that outlier detection and removal did not significantly increase the recognition accuracy.

The recognition model was afterwards validated within the application with a decision tree, obtaining different results: the average recognition accuracy dropped to 90.73%, and there was a significative discrepancy between the accuracy of different activities, as well as a discrepancy between the recall and precision of certain activities, for the reason that some activities were confused with other activities but not viceversa. While there is some similar-

ity between some of the activities that scored a low precision or recall, this seems to be more related to decision tree overfitting.
Using a history set capable of storing the last n classification, outputing the most frequent predicted activity in the sequence, showed an improvement in the recognition accuracy, which went up to the value of 92.68% with a history set of 3 predictions, and then slightly decreased as the history set size was increased. A history set of 3 elements gives the best accuracy for this kind of problem, and also allows for recognizing transitions fast, in a time window of maximum 3.75 seconds.
An analysis of the CPU consumption of the application with different configurations of sensors was not able to truly capture the total CPU consumption caused by the application, because the services that allow to read the GPS, the heart rate monitor, the accelerometer and the gyroscope are centralized, and there is no way to know if a given computation (e.g. GPS triangulation) is taking place within the application or outside (e.g. in a system routine). The results were mostly dependent on the feature extraction, which was implemented in the application and it can be considered a bottleneck. The only conclusion from this test is that discarding the heart rate monitor and the GPS slightly drops the CPU consumption of the application. Therefore using only the accelerometer and the gyroscope with automatic feature selection would achieve a good accuracy and a lower CPU consumption.

To summarize, considering the accuracy result of 92.68% with a history set of 3 elements, the conclusion is that automatic recognition of sports activities is feasible on the Apple Watch, independently on the watch location. The best configuration includes only the accelerometer and the gyroscope, with automatic feature selection using a history set of 3 elements. The most accurate algorithm is random forest.

7.0.1 Future Work

The first problem of this study was the impossibility of storing raw sensor data instead of the features, due to CPU and memory limits, as well as bandwidth limits that would cause the communications between the Apple Watch and the iPhone to freeze if large chunks of data are transmitted. Given the recent technology developments, it would be possible in future to use a watch with less limitations, that would allow to store all the raw data locally or by communicating with the phone. Storing the raw data instead of storing the features would give some advantages in the following choices:

- Choice of the features.
- Choice of the window length.
- Choice of the sampling frequency.
- Possibility of filtering the data.

Moreover, more possible future improvements of the system would include:

- Automatic energy monitoring of the application, included the energy consumed by the sensors.
- Duty cycles in order to save batter power and use the sensors only when movement is detected.
- Automatic recognition model installation, using more methods (e.g. random forest).

The last option requires an external server, which should allow to send sensor data. The server would elaborate the sensor data with the parameters chosen by the user, and it would reply by sending an encoded recognition model, and, if requested, some statistics (e.g. accuracy score on the test set). The system should be more automatic, it should not require any manual step during the training and testing phase, and it should be more customizable

(e.g. allowing to choose the activity labels).
Finally, the training/testing phases should be integrated into a fintess application, which would provide an impersonal recognition model with the possibility of manually training the activity classifier, in order to track physical activities automatically, without requiring any input.

Bibliography

[1] J. W. Lockhart, T. Pulickal, and G. M. Weiss, "Applications of mobile activity recognition," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, (New York, NY, USA), pp. 1054–1058, ACM, 2012.

[2] J. Villar, S. González, J. Sedano, C. Chira, and J. Trejo, "Improving human activity recognition and its application in early stroke diagnosis," vol. 25, pp. 1–20, 11 2014.

[3] V. Loseu, H. Ghasemzadeh, S. Ostadabbas, N. Raveendranathan, J. Malan, and R. Jafari, "Applications of sensing platforms with wearable computers," in *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '10, (New York, NY, USA), pp. 53:1–53:5, ACM, 2010.

[4] S. L. Lau, I. König, K. David, B. Parandian, C. Carius-Düssel, and M. Schultz, "Supporting patient monitoring using activity recognition with a smartphone," in *2010 7th International Symposium on Wireless Communication Systems*, pp. 810–814, Sept 2010.

[5] S. Patel, H. Park, P. Bonato, L. Chan, and M. Rodgers, "A review of wearable sensors and systems with application in rehabilitation," *Journal of NeuroEngineering and Rehabilitation*, vol. 9, p. 21, Apr 2012.

[6] R. I. Ramos-Garcia and A. W. Hoover, "A study of temporal action sequencing during consumption of a meal," in *Proceedings of the In-*

ternational Conference on Bioinformatics, Computational Biology and Biomedical Informatics, BCB'13, (New York, NY, USA), pp. 68:68–68:75, ACM, 2013.

[7] P. M. Scholl and K. van Laerhoven, "A feasibility study of wrist-worn accelerometer based detection of smoking habits," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 886–891, July 2012.

[8] G. Jean-Louis, D. F. Kripke, R. J. Cole, J. D. Assmus, and R. D. Langer, "Sleep detection with an accelerometer actigraph: comparisons with polysomnography," *Physiology % Behavior*, vol. 72, no. 1, pp. 21 – 28, 2001.

[9] J. Dai, X. Bai, Z. Yang, Z. Shen, and D. Xuan, "Perfalld: A pervasive fall detection system using mobile phones," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 292–297, March 2010.

[10] Y. He, Y. Li, and S. D. Bao, "Fall detection by built-in tri-accelerometer of smartphone," in *Proceedings of 2012 IEEE-EMBS International Conference on Biomedical and Health Informatics*, pp. 184–187, Jan 2012.

[11] `https://www.fallsafetyapp.com/blog/apple-watch-fall-detection`.

[12] J. Dai, J. Teng, X. Bai, Z. Shen, and D. Xuan, "Mobile phone based drunk driving detection," in *2010 4th International Conference on Pervasive Computing Technologies for Healthcare*, pp. 1–8, March 2010.

[13] T. Mashita, K. Shimatani, M. Iwata, H. Miyamoto, D. Komaki, T. Hara, K. Kiyokawa, H. Takemura, and S. Nishio, "Human activity recognition for a content search system considering situations of smartphone users," in *2012 IEEE Virtual Reality Workshops (VRW)*, pp. 1–2, March 2012.

[14] L. Bedogni, M. D. Felice, and L. Bononi, "By train or by car? detecting the user's motion type through smartphone sensors data," in *2012 IFIP Wireless Days*, pp. 1–6, Nov 2012.

[15] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer, "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach," in *2012 16th International Symposium on Wearable Computers*, pp. 17–24, June 2012.

[16] `https://www.fitbit.com/it/alta`.

[17] `https://www.nike.com/fuelband`.

[18] `http://www.mcroberts.nl/products/movemonitor/`.

[19] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," *IEEE Communications Surveys Tutorials*, vol. 15, pp. 1192–1209, Third 2013.

[20] T. L. M. van Kasteren, G. Englebienne, and B. J. A. Kröse, "An activity monitoring system for elderly care using generative and discriminative models," *Personal and Ubiquitous Computing*, vol. 14, pp. 489–498, Sep 2010.

[21] A. Tolstikov, X. Hong, J. Biswas, C. Nugent, L. Chen, and G. Parente, "Comparison of fusion methods based on dst and dbn in human activity recognition," *Journal of Control Theory and Applications*, vol. 9, pp. 18–27, Feb 2011.

[22] J. Yang, J. Lee, and J. Choi, "Activity recognition based on rfid object usage for smart mobile devices," *Journal of Computer Science and Technology*, vol. 26, pp. 239–246, Mar 2011.

[23] J. Sarkar, L. T. Vinh, Y.-K. Lee, and S. Lee, "Gpars: a general-purpose activity recognition system," *Applied Intelligence*, vol. 35, pp. 242–259, Oct 2011.

[24] J. Hong and T. Ohtsuki, "A state classification method based on space-time signal processing using svm for wireless monitoring systems," in *2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 2229–2233, Sept 2011.

[25] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea, "Machine recognition of human activities: A survey," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, pp. 1473–1488, Nov 2008.

[26] J. Candamo, M. Shreve, D. B. Goldgof, D. B. Sapper, and R. Kasturi, "Understanding transit scenes: A survey on human behavior-recognition algorithms," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, pp. 206–224, March 2010.

[27] C. N. Joseph, S. Kokulakumaran, K. Srijeyanthan, A. Thusyanthan, C. Gunasekara, and C. D. Gamage, "A framework for whole-body gesture recognition from video feeds," in *2010 5th International Conference on Industrial and Information Systems*, pp. 430–435, July 2010.

[28] M. A. R. Ahad, J. K. Tan, H. S. Kim, and S. Ishikawa, "Human activity recognition: Various paradigms," in *2008 International Conference on Control, Automation and Systems*, pp. 1896–1901, Oct 2008.

[29] U. Maurer, A. Smailagic, D. P. Siewiorek, and M. Deisher, "Activity recognition and monitoring using multiple sensors on different body positions," in *International Workshop on Wearable and Implantable Body Sensor Networks (BSN'06)*, pp. 4 pp.–116, April 2006.

[30] L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," in *Pervasive Computing* (A. Ferscha and F. Mattern, eds.), (Berlin, Heidelberg), pp. 1–17, Springer Berlin Heidelberg, 2004.

[31] D. Riboni and C. Bettini, "Cosar: hybrid reasoning for context-aware activity recognition," *Personal and Ubiquitous Computing*, vol. 15, pp. 271–289, Mar 2011.

[32] ı. D. Lara, A. J. PéRez, M. A. Labrador, and J. D. Posada, "Centinela: A human activity recognition system based on acceleration and vital sign data," *Pervasive Mob. Comput.*, vol. 8, pp. 717–729, Oct. 2012.

[33] E. M. Tapia, S. S. Intille, W. Haskell, K. Larson, J. Wright, A. King, and R. Friedman, "Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor," in *2007 11th IEEE International Symposium on Wearable Computers*, pp. 37–40, Oct 2007.

[34] M. Sekine, T. Tamura, M. Ogawa, T. Togawa, and Y. Fukui, "Classification of acceleration waveform in a continuous walking record," in *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Vol.20 Biomedical Engineering Towards the Year 2000 and Beyond (Cat. No.98CH36286)*, vol. 3, pp. 1523–1526 vol.3, Oct 1998.

[35] Z. He and L. Jin, "Activity recognition from acceleration data based on discrete consine transform and svm," in *2009 IEEE International Conference on Systems, Man and Cybernetics*, pp. 5041–5044, Oct 2009.

[36] M. Benocci, M. Bächlin, E. Farella, D. Roggen, L. Benini, and G. Tröster, "Wearable assistant for load monitoring: recognition of on-body load placement from gait alterations," in *2010 4th International Conference on Pervasive Computing Technologies for Healthcare*, pp. 1–8, March 2010.

[37] O. Banos, J.-M. Galvez, M. Damas, H. Pomares, and I. Rojas, "Window size impact in human activity recognition," *Sensors*, vol. 14, no. 4, pp. 6474–6499, 2014.

[38] F. Miao, Y. He, J. Liu, Y. Li, and I. Ayoola, "Identifying typical physical activity on smartphone with varying positions and orientations," *Biomedical engineering online*, vol. 14, no. 1, p. 32, 2015.

[39] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "Physical human activity recognition using wearable sensors," *Sensors*, vol. 15, no. 12, pp. 31314–31338, 2015.

[40] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, pp. 3–24, 2007.

[41] E. Hunt and J. S. Martin, "P.(1966), experiments in induction."

[42] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees. monterey, calif., usa: Wadsworth," 1984.

[43] I. Kononenko, "Estimating attributes: analysis and extensions of relief," in *European conference on machine learning*, pp. 171–182, Springer, 1994.

[44] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data mining and knowledge discovery*, vol. 2, no. 4, pp. 345–389, 1998.

[45] J. Quinlan, "C4. 5: Programs for machine learning. morgan kaufmann, san francisco.," *C4. 5: Programs for machine learning. Morgan Kaufmann, San Francisco.*, 1993.

[46] F. Rosenblatt, *Principles of Neurodynamics.* Spartan Books, 1962.

[47] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462, 2000.

[48] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[49] D. Heckerman, C. Meek, and G. Cooper, "A bayesian approach to causal discovery, chapter 4, camputation, causation, and discovery," 1999.

[50] S. Acid and L. M. de Campos, "Searching for bayesian network structures in the space of restricted acyclic partially directed graphs," *Journal of Artificial Intelligence Research*, vol. 18, pp. 445–490, 2003.

[51] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[52] D. Wettschereck, D. W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 273–314, 1997.

[53] M. Kubat and M. Cooperson Jr, "A reduction technique for nearest-neighbor classification: Small groups of examples," *Intelligent Data Analysis*, vol. 5, no. 6, pp. 463–476, 2001.

[54] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data mining and knowledge discovery*, vol. 6, no. 2, pp. 153–172, 2002.

[55] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[56] B. Schölkopf, C. J. Burges, and A. J. Smola, *Advances in kernel methods: support vector learning*. MIT press, 1999.

[57] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. New York, NY, USA: Cambridge University Press, 2000.

[58] K. Veropoulos, C. Campbell, N. Cristianini, *et al.*, "Controlling the sensitivity of support vector machines," in *Proceedings of the international joint conference on AI*, vol. 55, p. 60, 1999.

[59] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine learning*, vol. 36, no. 1-2, pp. 105–139, 1999.

[60] B. Effron and R. J. Tibshirani, "An introduction to the bootstrap," *Monographs on statistics and applied probability*, vol. 57, p. 436, 1993.

[61] T. K. Ho, "Random decision forests," in *Document analysis and recognition, 1995., proceedings of the third international conference on*, vol. 1, pp. 278–282, IEEE, 1995.

[62] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *Wearable Computers (ISWC), 2012 16th International Symposium on*, pp. 108–109, IEEE, 2012.

[63] A. Reiss and D. Stricker, "Creating and benchmarking a new dataset for physical activity monitoring," in *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*, p. 40, ACM, 2012.

[64] M. Arif and A. Kattan, "Physical activities monitoring using wearable acceleration sensors attached to the body," *PloS one*, vol. 10, no. 7, p. e0130851, 2015.

[65] I. C. Gyllensten and A. G. Bonomi, "Identifying types of physical activity with a single accelerometer: evaluating laboratory-trained algorithms in daily life," *IEEE transactions on biomedical engineering*, vol. 58, no. 9, pp. 2656–2663, 2011.

[66] K. Ellis, S. Godbole, J. Chen, S. Marshall, G. Lanckriet, and J. Kerr, "Physical activity recognition in free-living from body-worn sensors," in *Proceedings of the 4th International SenseCam & Pervasive Imaging Conference*, pp. 88–89, ACM, 2013.

[67] J. Morales and D. Akopian, "Physical activity recognition by smartphones, a survey," *Biocybernetics and Biomedical Engineering*, vol. 37, no. 3, pp. 388–400, 2017.

[68] `https://www.cs.waikato.ac.nz/~ml/weka/`.

[69] E. Mitchell, D. Monaghan, and N. E. O'Connor, "Classification of sporting activities using smartphone accelerometers," *Sensors*, vol. 13.

[70] A. Testoni and M. Di Felice, "A software architecture for generic human activity recognition from smartphone sensor data," in *Measurement and Networking (M&N), 2017 IEEE International Workshop on*, pp. 1–6, IEEE, 2017.

[71] G. M. Weiss, J. L. Timko, C. M. Gallagher, K. Yoneda, and A. J. Schreiber, "Smartwatch-based activity recognition: A machine learning approach," in *Biomedical and Health Informatics (BHI), 2016 IEEE-EMBS International Conference on*, pp. 426–429, IEEE, 2016.

[72] M. Ahmad and A. M. Khan, "Seeking optimum system settings for physical activity recognition on smartwatches," *arXiv preprint arXiv:1706.01720*, 2017.

[73] F. B. A. Ramos, A. Lorayne, A. A. M. Costa, R. R. de Sousa, H. O. Almeida, and A. Perkusich, "Combining smartphone and smartwatch sensor data in activity recognition approaches: an experimental evaluation.," in *SEKE*, pp. 267–272, 2016.

[74] N. Al-Naffakh, N. Clarke, P. Dowland, and F. Li, "Activity recognition using wearable computing," in *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference for*, pp. 189–195, IEEE, 2016.

[75] `http://scikit-learn.org/stable/`.

[76] `http://www.numpy.org`.

[77] https://developer.apple.com/swift/.

[78] https://developer.apple.com/xcode/ide/.

[79] https://github.com/mattt/Surge.

[80] B. E. Ainsworth, W. L. Haskell, A. S. Leon, D. R. Jacobs, H. J. Montoye, J. F. Sallis, and R. S. Paffenbarger, "Compendium of physical activities," *Medicine & Science in Sports & Exercise*, vol. 25, no. 1, pp. 71–80, 1993.