Design and Development of a Smartwatch Application for Accelerometer Data Labelling

By

Hamza Ahmed

School of Computer Science

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Masters of Science in the Faculty of Engineering.

December 2024

Abstract

Non-communicable diseases (NCDs) cause 41 million deaths annually, and key risk factors include physical inactivity, unhealthy diet, alcohol abuse and smoking. This project explores the application of Human Activity Recognition (HAR) using machine learning (ML) models and smartwatch accelerometer data to monitor behaviours contributing to NCD-related deaths. A key contribution of this project is the development of a custom-built data collection application that allows researchers such as the Tobacco and Alcohol Research Group (TARG) to collect labelled data on human activity, store the data online in a secure database and configure the activities easily using an online system without the need of any programming knowledge. Personal models were trained on user-specific data using the publically available WISDM sensor dataset, achieving high accuracy across Decision Trees, Random Forests and Convolutional Neural Networks (CNNs). Decision tree models achieved up to 96.5% accuracy on balanced data, while Random Forest models surpassed 99% accuracy, demonstrating their robustness in addressing class imbalance. CNN showed comparative performance, with strengths in handling overrepresented classes with the cost of higher computational demand. Data collected from the custom built app highlighted the challenges of real-world activity recognition. Due to the discontinuous nature of activity execution during collection, models performed slightly lower than on WISDM data. Nevertheless, Random Forest models achieved 78.95% accuracy, while CNNs did well in recognising specific activities, such as walking and vaping. In this work I demonstrate the feasibility of using machine learning to monitor human activities, compare the effect of class imbalance in model performance, and highlight the use of a data collection application to collect labelled data in real-time.

Acknowledgements

I would like to express my gratitude to everyone for this support in this project. First and foremost, I am deeply thankful to my project supervisor, **Dr Jon Bird** for his invaluable guidance and encouragement during every stage of this work. I would also like to thank **Sydney Charitos** for their help and feedback with the app development process, which significantly improved my confidence. I would like to extend my appreciation for **Professor Angela Atwood** for their help in establishing the requirements for this project and feedback on the finalised application.

[Hamza Ahmed
[December, 2024]

Author's declaration

I declare that the work in this report was carried out in accordance with the requirements of the University of Bristol's regulations and Code of Practice and Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific references in the text, the is the candidate's own work. Work done in collaboration with, or with the assistance of others is indicated as such. Any views expressed in the dissertation are those of the author.

[Hamza Ahmed
[December, 2024]

Contents

1 Introduction **7**
- 1.1 What Is Human Activity Recognition? . . . 7
- 1.2 Motivation . . . 8
- 1.3 Objective . . . 9

2 Background **10**
- 2.1 Overview of Machine Learning . . . 10
- 2.2 The Four Stages of Human Activity Recognition . . . 11
  - 2.2.1 Activity Capture . . . 11
  - 2.2.2 Data Pre-Processing . . . 12
  - 2.2.3 Model Training and Prediction . . . 14
- 2.3 Common Metrics For Model Performance . . . 16
- 2.4 The Need For A Data Collection App for Human Activity Recognition . . . 17

3 Related Work **19**
- 3.1 Background Of Human Activity Recognition Using Sensors . . . 19
- 3.2 Detection Of Harmful Habits Related to Non Communicable Diseases . . . 20
  - 3.2.1 Detecting Smoking . . . 20
  - 3.2.2 Detecting Alcohol Drinking Behaviour . . . 21

4 Technical Background **22**
- 4.1 Android Studio . . . 22
- 4.2 Python . . . 23
- 4.3 Power Automate . . . 23

5 Project Execution : Data Collection Application **25**
- 5.1 Introduction . . . 25
- 5.2 App Development . . . 25
  - 5.2.1 Phase 1: Getting Familiar With Android Studio And Collecting Accelerometer Data . . . 26
  - 5.2.2 Phase 2: Labelling Data And Expanding App Functionality 28
  - 5.2.3 Phase 3: Sending Data To An Online Database . . . 31

5.2.4 Phase 4: Easy Configurable List Of Activities 35

6 Critical Evaluation **40**
6.1 Strength and Achievements 40
6.2 Limitations and Area for Improvements 40
6.2.1 Configurable Data Collection Parameters 40
6.2.2 Data Upload Delays 41
6.3 Conclusion 41

7 Project Execution : Using Machine Learning for Human Activity Detection **43**
7.1 Using Machine Learning To Predict Human Activity In the Wireless Sensor Data Mining (WISDM) dataset 43
7.1.1 Overview (WISDM) dataset 44
7.1.2 Data Cleaning and Preprocessing 44
7.1.3 Exploratory Data Analysis 45
7.1.4 Feature Extraction 47
7.1.5 Extracting Data From A Single User To Build A Personal Model 48
7.2 Data Collected from Data Collection Application 49
7.3 Model Training 50
7.3.1 Training Decision Tree and Random Forest Classifiers . . 50
7.4 Summary 51

8 Results and Evaluation **53**
8.1 Personal Models : User 8 Data From WISDM 53
8.1.1 Data Collected from App 56

9 Conclusion **60**

10 Future Work **62**
10.1 Expanding App Configurability 62
10.2 Improving Data Upload Delays 62
10.3 Storing Data In Various Formats 62
10.4 Improving Model Performance 63
10.5 Incorporating Machine Learning In Power Automate Flow 63
10.6 Machine Learning Libraries for Android Studio 64
10.7 Using the App As A Fully Realised Personal Informatics Tool . . 64

Appendix **65**
A Feature Stat Calculation Class 65
B Power Automate Flow to Upload Accelerometer Data 70
C Power Automate Flow to retrieve activities list from excel when a Microsoft Form is submitted 71
D Power Automate Flow to send selected Activities List from One Drive as a HTTP response to smart watch 72

Chapter 1

Introduction

In this chapter, I will introduce the concept of Human Activity Recognition (HAR), its stages, and its potential applications in addressing health-related challenges. I will then outline the motivation for this project, focusing on how HAR can help mitigate Non-Communicable Disease (NCD)-related deaths by tracking unhealthy behaviours. Finally, I will present the specific objectives of this thesis and its contributions to the field.

1.1 What Is Human Activity Recognition?

Human activity recognition, a versatile concept in artificial intelligence, involves identifying activities based on collected raw data from various sources such as wearable sensors, smartphone sensors, and camera devices [1]. The applications of this technology are vast, ranging from healthcare [2] to surveillance, sports and exercise monitoring, and personal habit tracking [3]. HAR, specifically in the context of monitoring activities such as smoking, drinking, eating, and sedentary behaviour, can be deployed as a personal informatics device to help reduce the impact of these factors on mortality. Typically, HAR consists of four stages, as illustrated in Figure 1.1 : 1) capturing signal activity, 2) data pre-processing, 3) machine learning model training, and 4) Activity Prediction [3]. These are expanded upon in chapter 2.2

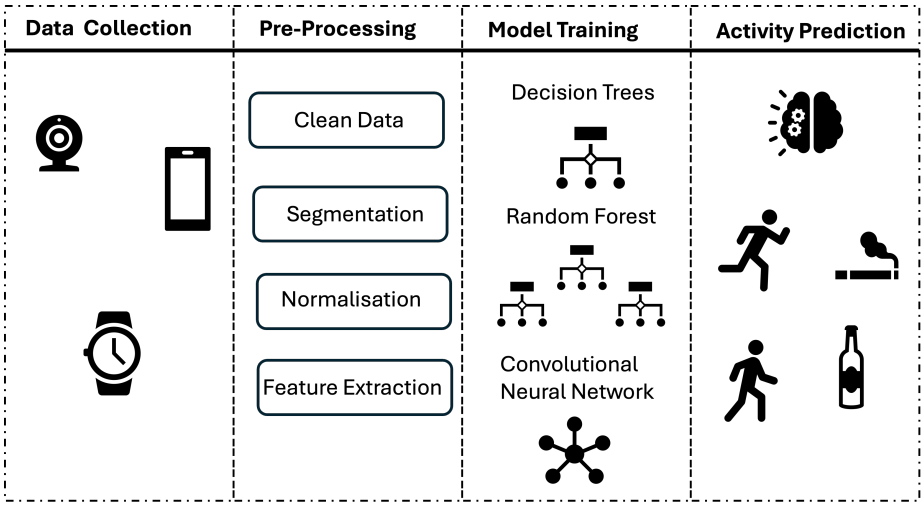

Figure 1.1: Four Stages of HAR: 1: Data Collection, 2: Pre-Processing of collected data and feature generation 3: Training machine learning model(s) 4: Model predicts the activity being preformed from data

1.2 Motivation

The number of deaths caused by Non-Communicable Diseases (NCDs) (diseases that do not spread through infection or other people but are typically caused by unhealthy behaviours) is estimated to be around 41 million people per year, according to the World Health Organisation (WHO) [4]. The most significant causes of death caused by NCDs are cardiovascular, cancer, respiratory diseases and diabetes, which in total account for 80% of all premature NCD-related deaths [4]. Physical inactivity, unhealthy diet, harmful abuse of alcohol and tobacco use are all factors that increase the risk of developing these diseases. Traditional methods can make it challenging to track habits that increase the risk of NCD-related deaths. These methods rely on user input for data such as the number of cigarettes a person has smoked or the number of drinks they have had daily. Here, machine learning models in HAR can be a valuable tool to allow autonomous tracking of these behaviours and provide additional information, such as social behaviours [5] that lead to these habits.

Machine learning models perform best with large volumes of accurate data; the more extensive and precise the dataset used for training, the greater the model's predictive accuracy. Therefore, data collection is arguably the most crucial step in HAR. Most modern smartphones and smartwatches have accelerometers that provide three-dimensional axial data, one of the most reliable methods for tracking physical activity [6]. However, the availability of high-quality datasets, particularly those collected from smartwatches, remains limited. Furthermore,

labelling the collected data is often a time-consuming and labour-intensive process [7]. One of the contributions of this thesis will be to develop a data collection tool for researchers to collect labelled accelerometer data via a smartwatch device. A research tool which enables users to collect labelled data, represents a vital advancement in using HAR to address NCD-related deaths.

1.3 Objective

The Tobacco and Alcohol Research Group (TARG) investigates the psychological and biological factors underlying health-related behaviours. To facilitate their research, this thesis focuses on the development of a data collection tool designed to meet the following client requirements:

- Collect labelled accelerometer data via a smartwatch device.
- Store the smartwatch data securely in an online database.
- Provide a configurable activity list to allow multiple activities to be tracked and labelled.

I will then use the data collected from this application as well as existing datasets on human activity to train machine learning models such as Decision Trees, Random Forest and Convolutional Neural Networks (CNNs) to demonstrate how machine learning can be used in HAR.

Chapter 2

Background

This chapter provides a foundational knowledge required to understand the key concepts underpinning this project. Here I cover the conceptual background of machine learning, its role in Human Activity Recognition (HAR), the four stages of activity recognition and the significance of accelerometer data and feature extraction. The chapter also introduces the metrics used to evaluate machine learning models, such as accuracy, precision, recall and F1-score. Finally, I highlight the limitations of existing HAR datasets and justify the need for a a data collection tool for HAR tailored for researchers.

2.1 Overview of Machine Learning

Machine Learning (ML) as defined by IBM [8] is a branch of computer science that focuses on using data and algorithms to enable artificial intelligence to mimic how humans learn, gradually improving their accuracy. ML models fall into two main categories:

Supervised Machine Learning

Supervised learning is defined by its use of **labelled** datasets to train algorithms (models) to classify data or predict outcomes accurately; types of ML models used for this project are expanded upon in section 2.2.3. As input data containing the features (such as accelerometer data) and the label (such as activity being performed) are fed into the model, the model adjusts its weight until data has been fit appropriately. Types of supervised ML algorithms include neural networks, linear regression, logistic regression, random forest, decision trees and support vector machine (SVM).

Regression vs Classification Supervised ML algorithms can be split into regression or classification. Regression finds correlation between dependant and independent variables, therefore, help predict contentious variables such as market trends, weather patterns and oil and gas prices. A classic example of regression based ML are house price predictor. Here, data such as size of house

and house prices is used to train the model. A regression ML model will then predict the price of a house given its size.
Classification models on the other hand, define functions that help divide the data into distinct classes. An example of a classification model is a cancer production tool, where a model may be provided with various features such as tumore size, mean texture, mean area, smoothness and more. Based on these features a classification model could predict if the tumore is benign or malignant.

Unsupervised Machine Learning

Unsupervised learning uses ML to analyse and cluster unlabelled datasets. These algorithms group data according to hidden patterns without the need for human intervention. Types of unsupervised ML algorithms include Principal component analysis (PCA), Singular Value Decomposition (SVD), neural networks, k-means clustering and probabilistic clustering methods.

The aim of this project is to develop a data collection tool to collect labelled data for supervised ML algorithms for HAR.

2.2 The Four Stages of Human Activity Recognition

2.2.1 Activity Capture

The diagram in Figure 1.1 shows that human activity recognition can be split into four stages. In the first stage, the activity signal is captured, which can be done using various methods:

- **Video:** Video-based HAR involves identifying actions performed by humans from video sequences. Various modalities can be extracted from video, such as Red-Green-Blue (RBG) channels, which provide colour and intensity information, and optical flow, which captures motion patterns between frames [9]. CNNs are commonly used in video-based HAR, as they are effective in learning spatial features from RGB images and can be extended with temporal networks [10], such as Long Short-Term Memory (LSTM) layers, to analyse sequential data over time [11].

- **Sensors:** Sensor-based HAR involves capturing activity signals through wearable or ambient sensors, such as accelerometers [6], gyroscopes [12], and magnetometers [13]. These sensors measure physical quantities like acceleration, angular velocity, or magnetic field strength in three dimensions, providing precise data about body movements. Accelerometers, for instance (Figure 2.1), record changes in velocity across the three axis : x, y and z. making them particularly suitable for detecting patterns of physical activity like walking, running, or sitting. The raw sensor data is often time-series, segmented into smaller windows to extract meaningful

features [14]. Machine learning models, including traditional classifiers like Decision Tree, Random Forest and Support Vector Machines (SVMs) and neural networks such as CNNs, are used to process this data.

Figure 2.1: Three axial movement detected by accelerometers in smart watches [15]

2.2.2 Data Pre-Processing

The second stage of Human Activity Recognition (HAR) involves pre-processing the raw data collected during the activity capture stage. Pre-processing is a critical step as it ensures that the data is clean, structured, and ready for feature extraction and model training. This stage includes several essential steps:

1. **Data Cleaning** Data cleaning involves removing or handling missing, inconsistent, or erroneous data points, such as null values or outliers. Null values can arise due to sensor errors or transmission failures, and they are often removed.

2. **Segmentation** Segmentation refers to dividing continuous time-series data, such as one collected by accelerometers, into smaller, fixed-length intervals called "windows." Each window represents a segment of the activity and captures enough data to identify patterns. For example, in this project, data is segmented into 10-second windows at a sampling frequency of 20 Hz, resulting in (20 x 10) 200 data points per window. These windows are essential for feature extraction as they standardise the data format, allowing models to process consistent input sizes.

3. **Normalisation / Standardisation** Normalisation and standardisation are both techniques to scale the data values so they can be analysed more easily. The main difference between the two is how the data is scaled.

In normalisation, data is scaled to specific ranges. This can be useful when scale of features varies greatly. In standardisation, data values are rescaled to have a mean of 0 and a standard deviation of 1. Standardisation of accelerometer data is an effective method for improving the validity and comparability of results as it minimises biases caused by wear-time variations such as collecting data between users [16].

4. **Window Size Creation** Standard classification algorithms (such as decision trees and random forests) cannot be applied directly to raw time series data. Instead, the data is first transformed into discrete 'windows', which divide the data into small segments such as 5 seconds, 10 seconds, 1 hour etc, (a 5 second window on a dataset collected at 20 Hz would contain 100 data points). The choice of window size is a key decision in HAR. A window that is too short may not capture sufficient activity information, while a window that is too long may dilute the activity's distinguishing features. To ensure that no data is missed for the model, windows are also created with overlap which can also vary. An illustration of windowing with 4 data points and 50% overlap is shown in Figure 2.2

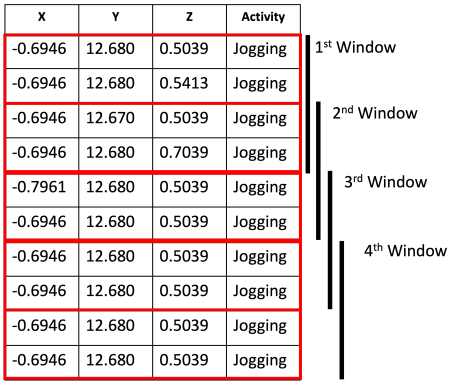

Figure 2.2: Demonstration of creating windows from tri axial acclereometer data. Each window in this example contains 4 data points and a 50% overlap

5. **Feature Extraction** Feature extraction transforms raw time-series data into meaningful statistical data that machine learning models can use. For this project, 18 statistical features are extracted from the accelerometer data in each window. These features include measures like mean, standard deviation, variance, skewness, kurtosis, and energy for each axis (x, y, z). By summarising the raw data, feature extraction reduces dimensionality while preserving essential information about the activity patterns. For example, calculating the mean and variance provides insights into the overall magnitude and spread of movements, while skewness and kurtosis describe the shape of the activity distribution. Energy features indicate the intensity of activity within the window.

2.2.3 Model Training and Prediction

Types of Machine Learning Models

Decision Trees

A decision tree is a type of supervised learning technique that used for classification and regression tasks. A decision tree is made up of root node, branches, and inner nodes. As illustrated by Figure 2.3 decision trees recursively divide the dataset based on a given feature. For example, in the classification of cats and dogs based on feature sets such as ear shape, face shape, and presence of whiskers, a decision tree will first find the best feature to split on based on the purity of the resulting split. The purity of a split is the frequency of its most common constitution, for example if a set (after a split) leads to 60% Cats, and 40% dogs then the purity of that split is 60% if we were classifying cats from dogs. This process is repeated until a node is either 100% of one class, if the splitting exceeds the maximum defined depth or if the improvement in purity is below the defined threshold.

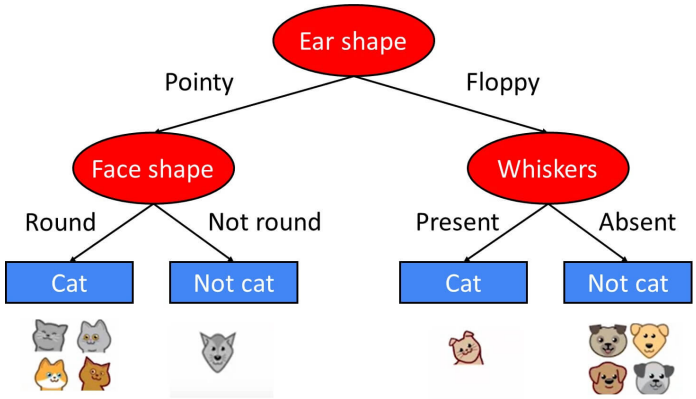

Figure 2.3: An illustration of a classification decision tree structure used to classify cats from dogs based on Ear shape, Face shape, and Whiskers [17]

Similarly, we can use time series data such as accelerometer data (change in velocity in the x, y, z-axis), extract features from the data and use the features to train a Decision Tree model to predict the activity being preformed.

Random Forest

Random forests are ensemble learning methods that build multiple decision trees and combine their prediction to improve classification performance. Each tree in the forest is a trained on a subset of the data, and during the training process, a random subset of feature is considered for each split. The final prediction of a random forest is typical obtained through majority voting, i.e. the classification reached by the majority of the trees in the forest. Random forests are robust

to noise, handle high-dimensional data well and are less sensitive to outliers compared with a single decision tree. However, they use more processing power and can take longer for prediction.

Convolutional Neural Networks

A neural network is a ML model that makes decisions in a manner similar to the human brain, where nodes (or neurons) work together to identify, weight options and arrive at conclusions [8]. There are three layers to a neural network as shown in Figure 2.4; the input layer, a hidden layer and the output layer. Think of each node as its own regression model (like a regression decision tree) composed of input data, weights, threshold and an output. If the output of any individual node is above the threshold then the data is passed to the next layer, otherwise no data is passed.

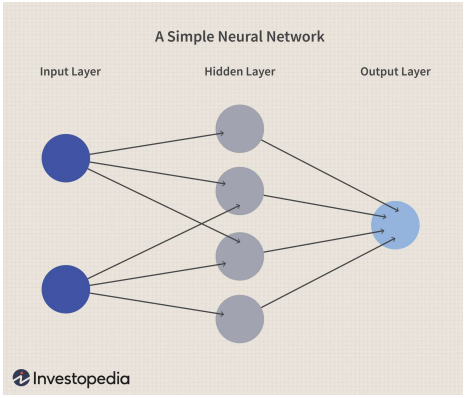

Figure 2.4: A simple neural network, consisting of an input layer, one hidden layer and one output node [18]

Convolutional Neural Networks (CNNs) are a type of neural network, they show superior performance with image, speech or audio signal inputs. Like neural networks there are three layers to a CNN:

- Convolutional Layer
- Pooling Layer
- Fully-connected (FC) layer

The convolutional layer is the core building block of a CNN and is where majority of the computation occurs. In the context of image processing, the input will be a colour image, which is made up of a matrix of pixels in 3D. This means the input has three dimensions - a height, width and depth - which corresponds to RGB of an image. There is also a feature detector, also known as a kernel or filter, which moves across the receptive fields of the image checking

if the feature is present.
Although mostly applied to image and video based methods, CNNs also show great promise for signal data, such as accelerometers to detect HAR [19]. The filters in CNN layers extract meaningful features like periodicity in jogging or sharp changed in acceleration when climbing stairs. These features are then passed to dense layers for classification, where the network predicts the specific activity.

2.3 Common Metrics For Model Performance

- **True Positive (TP):** The number of instances where the model correctly predicts a sample as belonging to a positive class, where positive refers to a specific outcome or label of interest in the given context. For example, in the context of activity recognition, a positive might represent a specific activity such as Drinking, here a true positive would be number of times the model correctly identifies jogging.
- **False Positive (FP):** A False Positive occurs when the model incorrectly predicts a sample as belonging to the positive class when, in reality it does not. For example, if Jogging is considered as the positive, a false positive is when the model predicts jogging, but the user was walking or standing.
- **True Negative (TN):** The number of samples that are actually negative and predicted negative by the model
- **False Negative (FN):** A False Negative is the number of instances where the model incorrectly predicts a sample belonging to a positive class when in reality it does not.
-

Accuracy

Accuracy is a performance metric that measures the proportion of correctly classified instances out of the total instances. It is a calculated by the following formula 2.1

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

In the context of activity recognition, accuracy represents the percentage of times the model correctly identifies an activity. For example, if the dataset includes activities such as walking, jogging and sitting, accuracy measures how often the model predicted the correct activity. Although a useful and widely adopted metric for model performance, in cases, of imbalance datasets where one class significantly outweighs others, accuracy can be misleading. In such cases, metrics like precision, recall and F1-scores are more informative.

Precision

Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive. It is calculated by the following formula 2.2

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

In the context of activity recognition, if running is considered the true positive class, precision indicates how many of the instances predicted as running by the model are actually running.

Recall

Recall or the rate of true positives, measures the proportion of actual positive instances that the model correctly predicts as positive. It is calculated by the following formula 2.3

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

In the context of activity recognition, recall measures how many actual activity instances were correctly identified by the model. For example, if there are 100 "jogging" instances in the dataset and the model correctly identifies 90 of them, the recall would be 90%.

F1-Score

The F1 score is particularly useful when there is a class imbalance in the dataset. The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both. It is calculated by the equation 2.4

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.4}$$

In activity recognition, the F1-score provides an overall measure ofhte models ability to correctly identify the positive class while minimising both false positives and false negatives. For example, if hte precision for "jogging" is 80% and the recall is 90%, the F1-score would provide a balance view of model's performance, calculating to approximately 84.2%.

2.4 The Need For A Data Collection App for Human Activity Recognition

Unlike image or video data, which humans can annotate easily, labelling sensor data is a time-consuming and complex process [20]. Human activity recognition (HAR) methods require large volumes of labelled data to train classifiers effectively. While many sensor-based datasets for HAR exist [21, 14, 22, 23, 24, 20, 25], they often focus on specific subsets of activities. These include whole-body movements like "walking," "jogging," and "sitting" [21], hand movements

such as "drinking," "eating chips," "brushing teeth," and "clapping" [14], or niche activities like "passing," "shooting," and "laying-up" in basketball [22]. Most datasets typically include features such as "user," "timestamp," "x-axis," "y-axis," "z-axis," and "activity", which can be used for feature generation to extract features for ML model training. However, while these datasets are widely used and valuable, they often fail to cater to the specific needs of researchers. For instance, training machine learning models to differentiate between activities like drinking wine, beer, or water may require highly specialised unavailable datasets. Additionally, inconsistencies in data collection—such as sensor placement, the number of sensors used, and collection durations—pose significant challenges for generalising machine learning models. A customisable data collection app addresses these limitations by enabling researchers to collect and label data tailored to their objectives. Features like easy labelling, intuitive user interfaces, and secure online data storage make such an app a powerful tool for advancing HAR research. Simplifying the data collection process accelerates the study of activities related to non-communicable diseases (NCDs) and serves as a foundation for personal informatics tools, empowering individuals to track and reflect on their habits.

Chapter 3

Related Work

In this chapter, I explore the foundations and advancements in HAR using sensor based technology. I discuss the methodologies used in early activity detection, to the evolution of using smartphone and smart watch based detection. This chapter also covers the previous research conducted in detecting some of the behaviours related to NCD related deaths, I explore the methodologies used by the researchers and explain their findings, noting any limitations if present.

3.1 Background Of Human Activity Recognition Using Sensors

The use of technology to track activity for self-improvement is not a new one. As early as the 1970s, people have been using wearable sensors to gain insight into behaviour and activity [26]. Human Activity Recognition (HAR) has gained a significant interest in recent years, thanks to the advancements in vision and sensor-based technology. The predecessor of the sensor-based method of activity recognition was using cameras to track human behaviour and then processing via data prepossessing, object segmentation, feature extraction and model implementation [27, 28]. However, vision based techniques still present their challenges when it comes to HAR, for example: the body's size, effects of clothing, background colour, angle of observer and light intensity [29].

Thanks to the rapid development of Microelectromechanical Systems (MEMS), internal sensors have become much smaller, lighter, less expensive and more accurate. This makes internal sensors more robust in various environments to be used for HAR. Many previous studies have been conducted using machine learning classification algorithms and sensor data to identify activity:

Gyllensten et al. [30] proposed a classification model and single sensor data to identify activities such as running, walking, sitting down, lying down and cycling. The researchers noted that whilst in laboratory conditions the out-

comes were generally acceptable, the performance of the model on untrained data would be lacking.

Esfahani et al. [31] found that placing multiple sensors (accelerometers and gyroscopes) on the body provided a higher level of precision for classification for similar activities, reaching a 85% model accuracy compared with the WISDM dataset [14]. However, they noted that in a real world environment, complex activities would not be classified as well under laboratory conditions.

Activity recognition using **smartphone sensors** were shown to be a successful route as shown by Ahmed et al. [32] where the use of feature called Average peak-through-distance (APTD) and use of Grid Search Cross Validation to tune the hyper parameters in their Radial Basis Function Support Vector Machine (RBG SVM) to achieve 95% F-score of activities: sitting, jogging, skipping, upstairs, downstairs.

Rubén San-Segundoa et al. [33] collected data from smartwatches and smartphones using three-axis accelerometer signals and applied two methods: Convolutional Neural Networks (CNNs) and Hidden Markov Models (HMMs). They demonstrated that CNNs were more effective in predicting human activity, particularly for smartwatches then smartphones which presented greater data variability then smart watches.

3.2 Detection Of Harmful Habits Related to Non Communicable Diseases

3.2.1 Detecting Smoking

There are few studies on activity recognition using wrist-worn devices to detect hand to mouth movements. One example is of a research by Phillipp M.Scholl and Kristof van Laerhoven [34] where the authors use smart watches to show the feasibility of detecting smoking using built in accelerometers. The authors used a wearable device, known as a Hedgehog platform with a built in accelerometer, that the participants in the study (n=4) wore for a week and tapped every time they had a cigarette. From this study, the researchers found the average smoking time was about 4-8 minutes for one cigarette. The Gaussian model detected smoking gestures with a precision of 51.2% and a user specific recall of over 70%. However, the researchers noted that some manual labelling of data was required. In another study by Casey A Cole. et al. [35] 10 participants were given smartwatch devices with a custom built in data collection app that was paired via Bluetooth to a smartphone. This app allowed for data recording and transmission to a cloud storage database. Smoking behaviour was recorded over three days, where accelerometer data was continuously recorded for 12 hours. Participants logged smoking details such as start and stop time and puff counts on an online form after each smoking session. Data was segmented into 1, 2,

3, and 4 hours, however an exact average smoking time was not provided in the study. The researchers used neural networks to classify smoking and non smoking behaviour patterns with an accuracy score of 90%.
From these studies, it can be concluded that detecting smoking using machine learning algorithms is a feasible task, with neural networks demonstrating superior model performance compared to Gaussian models. However, these studies also underscore the critical need for a dedicated data collection app. For instance, Cole et al. [35] highlighted the importance of such a tool to simplify data collection processes, while the study by Philipp M. Scholl and Kristof van Laerhoven [34] faced challenges due to the lack of an automated solution, relying on manual labelling when the Hedgehog device failed to register taps. A limitation with both studies is also the small number of participants.

3.2.2 Detecting Alcohol Drinking Behaviour

A notable paper by Sngwon Bae et al.[5] use **smartphone** sensors to track physical and social behaviours that are associated with drinking episodes in young adults. The researchers used 30 participants with a past of hazardous drinking, they used mobile sensors to capture accelerometer data contentiously for 28 days. Data was collected via a data collection app, and labelled by sending prompts to the users every day at 10 am asking the user if they had any alcohol the previous day, if yes, what time did the drinking start and stop and how many drinks were did the user consume in this period. The researchers defined a heavy drinking episode of $\geq$4 (for women) and $\geq$5 (for men). The Random Forest model used in the study correctly classified an individual drinking, heavy drinking episode or not drinking with an accuracy of 96.6%. There are important conclusions that can be made from this study for this project and its future work. **Validity of self-reported data:** Self-reported onset and ending drinking episodes combined with number of drinks consumed, correlate strongly with objective breathalyser readings, which indicates that self reported data can be a reliable source for collecting real world data for drinking. **Time lag of wearable sensors :** The researchers found wearable trans-dermal alcohol monitoring (measuring ethanol excreted through the skin) gave a time lag (up to several hours) in detecting alcohol use. This makes them unsuitable for applications requiring real time data. **Type of data collection for drinking detection:** To accurately predict drinking behaviour, a data collection app should record start and end time of drinking, the number of drinks consumed in real time. It is also important to consider activities that lead up the a drinking session (especially heavy drinking episodes).

Chapter 4

Technical Background

In this chapter I will go over the programming languages, packages and frameworks used for developing the accelerometer data collection application. This chapter will provide an overview of the main tools used and the rationale behind the selection of each tool.

4.1 Android Studio

Android Studio is the official integrated development environment (IDE) for Android app development. Android studio provides a comprehensive set of tools and features specifically designed for app development such as, code editors, debugging tools, built in support for a popular app development programming language called Kotlin and a real time visual layout screen as shown in Figure 4.1

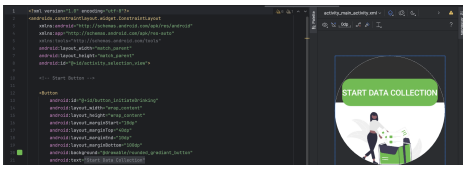

Figure 4.1: Visual Layout and Code Editor within Android Studio

View Based UI development : XML provides a structured way to define the appearance of the User Interface (UI) elements such as views, widgets and layouts. Each new UI in an android application is referred as an "**activity**". XML layouts support a wide range of UI widgets that enable full freedom to customise and enhance user experience. Examples of that have been utilised in this project include:

- Text Views

- Buttons
- Image Views
- Recycler Views (allowing for a scrollable list of items)

Although these features are available in other platforms such as Flutter and React, they are especially easy to integrate within Android Studio.

Use of Emulators : Android Studio also provides emulators to simulate how the app preforms. The emulators come with predefined configurations for various Android phones, tablets, Android TV devices and Wear OS devices. The emulators provide all the functionality of a real device, most useful of which for this project were the sensor simulations allowing me to simulate movement to collect accelerometer data. Being able to emulate a real OS environment with APIs functionality included, significantly improved the iterative design process as features can be tested with ease as they are added.

Shared Preferences : In this project, accelerometer data was saved using the SharedPreferences API. A SharedPreferences object point to a file containing a key : value pairs which can be easily accessed to read and write data to the file within the device. Ease of access of this data from anywhere in the code made debugging and posting data via a request/response API a simpler task.

4.2 Python

According to IEEE spectrum's 11th annual ranking [36], the most popular programming language is Python, which continues to dominate as one of the most widely used programming languages, due to its popular libraries particularly in the rising interest of machine learning. Libraries such as Stickit-learn which offers robust variety of AI and ML models. The pre written code allows developers to easily train models and test models with a few lines of code. This project uses Python to load the dataset using data handling libraries such as Pandas to pre-process the data and extract features, then using Scikit-Learn library to train ML models.

4.3 Power Automate

Power Automate is a cloud based service from Microsoft which enables workflow automation across applications and services. It allows users to create automated workflows without the need for extensive programming language experience by allowing users to simply drag and drop various interfaces.
In this project, Power Automate is utilised to automate the process of configuring the smart watch to allow users to select the various activities. It is also used to transfer the collected data at the end of each session to an online database, where accelerometer, user and device information are stored in an Excel sheet.

One of the key advantages of Power Automate is its seamless integration with other Microsoft services such as **OneDrive**, **Forms** and **Excel**. This integration made it an ideal choice for configuration and handling of data transfer pipeline. By leveraging HTTP actions in Power Automate, accelerometer data collected on the smart watch can be sent via a POST request.
By automating the data flow, Power Automate significantly streamlined the workflow, allowing the focus to remain on data collection and model training rather than data management. Its flexibility and ease of integration made it a critical tool in this project's development pipeline. Below is an example of how Power Automate was configured to handle incoming data:

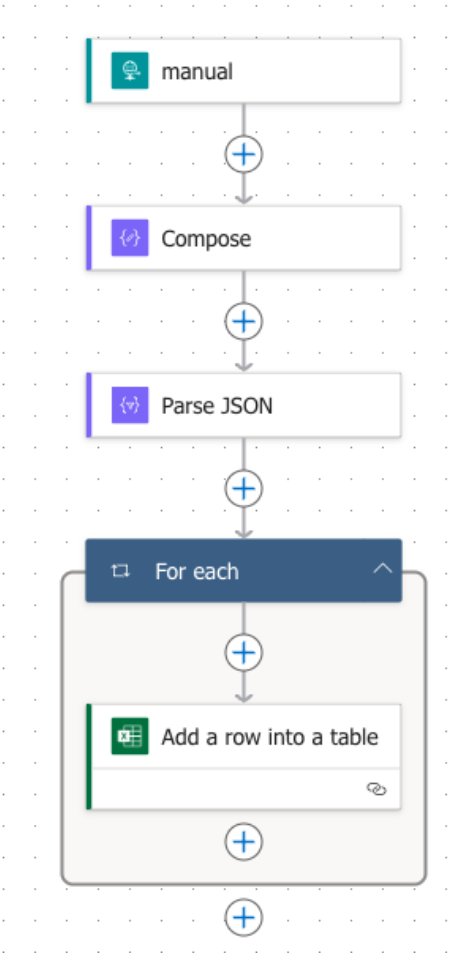

Figure 4.2: Power Automate flow showing a Trigger Action (HTTP manual request), Data Processing (Compose and Parse JSON) and updating each row of an Excel with the processed accelerometer data (Add a row into a table)

Chapter 5

Project Execution : Data Collection Application

5.1 Introduction

This chapter outlines the design and development process of the smartwatch data collection application created to meet the TARG requirements. The chapter is describes the iterative development process used to add features to the application, following the Agile development methodology. Each phase progressively adds functionality, starting from basic accelerometer data collection to a robust configurable application that stores data in an online database.

5.2 App Development

The client requirements I received for the data collection app are re-iterated here and are as follows:

- Collect labelled accelerometer data via a smartwatch device.
- Store the smartwatch data securely in an online database.
- Provide a configurable activities list to allow multiple activities to be tracked and labelled.

I used the Agile methodology to implement features iteratively to achieve the criteria above. The Agile methodology is a project management framework that breaks down projects into several dynamic phases. Each phase involves the following iterative design process:

1. **Planning and Requirement:** In this phase I identify the feature(s) I want to implement.

2. **Analysis and Design:** Here, I develop the User Interface (UI) on Android Studio and devise the best strategies for feature implementation. I use diagrams to help me visualise how I want each activity screen to look.
3. **Implementation:** Here, I implement the feature(s)
4. **Evaluation and Review:** The features are tested and evaluated at each phase by running it on a physical device or on an Android Studio emulator.

5.2.1 Phase 1: Getting Familiar With Android Studio And Collecting Accelerometer Data

Planning and Requirements

The primary objective of this phase was to lay the foundation for data collection application by understanding Android Studio's capabilities and implementing basic functionality to collect accelerometer data from the sensors; I wanted to ensure that raw accelerometer data (x, y and z-axis) could be captured.

Analysis and Design

The design of this stage was intentionally minimalistic to reduce complexity while I became familiar with APIs such as **SensorManager** which makes the device sensors such as accelerometers available in the Android application. The application consisted of :

- **Start Activity:** A simple button to initiate data collection
- **Stop Activity:** A button to stop data collection and store recorded values locally

The UI was designed using XML layouts, allowing me to define the appearance of each activity screen. At this stage, no additional configuration options or user prompts were implemented. Figure 5.1 shows the app's primary interface during this phase; pressing the Start button loaded the second "Stop" activity screen.

Figure 5.1: Diagram illustrating UI interface of app during Phase 1. User presses START to start accelerometer data collection and presses STOP to stop data collection

Implementation

To implement the data collection, I used the **SensorManager** class in Android Studio to register and manage the device's accelerometer sensor (Listing 5.1. The **onSensorChange()** method is overridden to continuously capture three-axial accelerometer data when the **Start** button is pressed. Data collection ceases when the **Stop** button is pressed. At this stage, the data collection is printed on screen using the **Log** function in Android Studio, which is also useful for debugging purposes.

```
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
//Initialise the SensorManager and accelerometer
sensorManager = getSystemService(Context.SENSOR_SERVICE) as
    SensorManager
accelerometer =
    sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)

// Log the initialisation
if (accelerometer != null) {
    Log.d(TAG, "Accelerometer initialized successfully")
} else {
    Log.d(TAG, "No accelerometer available on this device")
    }
}

override fun onSensorChanged(event: SensorEvent?) {
// Capture and log accelerometer data
event?.let {
    val xAxis = it.values[0]
    val yAxis = it.values[1]
    val zAxis = it.values[2]
    Log.d(TAG, "Accelerometer Data - x-axis: $xAxis, y-axis:
        $yAxis, z-axis: $zAxis")
    }
}
```

Listing 5.1: Initialising SensorManager and retrieving accelerometer data on Sensor Change. Data is printed to Log screen.

Evaluation and Review

To evaluate the functionality, I connected a physical Google Pixel watch device to Android Studio via wireless debugging setting on the watch. When I moved the watch I could see the accelerometer data on my log screen. I confirmed successful data retrieval by measuring changes in the x, y and z-axis values through the watch's movement compared with when the watch was left stationary. The next stage would expand the functionality of the app by:

- Collecting labelled accelerometer data

- Data to be collected within a specific time window
- Data is to be saved on the device in a CSV format.

5.2.2 Phase 2: Labelling Data And Expanding App Functionality

Planning and Requirement

The second phase of development aimed to address the limitations of Phase 1 by enabling the collection of labelled data. To accomplish this, I wanted to introduce a user-friendly interface for activity selection, allowing users to select from a list of activities that would act as the label for the data collected. I also wanted to save the data on the watch device for ease of access. Furthermore, I wanted each activity collected in a specific time window to give a balanced dataset.

Analysis and Design

To facilitate the activity selection screen, I evaluated various methods such as ListView and RecyclerView; both offer vertical lists of scroll-able views, and each view is placed one below the other.

Activities stored in an ArrayList are bound with the UI widgets displaying each activity's names on the screen. I chose RecyclerView to display my activities, as it offers significant advantages such as :

- Improved performance by reusing views for scrolling items
- Greater flexibility to customise how data is displayed
- Support for complex layouts (such as text and images).

This design allowed me to easily configure the list of activities, enabling future customisation without requiring substantial changes to the underlying code.
The app is further enhanced by a TextEdit field, which allows researchers to input user IDs before data collection. This addition ensures that data can be personalised.

The Choice of a Time Window for Data Collection

In HAR, the duration of the data collection window is a crucial design choice that impacts both the usability of the application and the downstream processing of the collected data. In this project, a **10-second time window** for data collection is implemented, and this decision is supported by both established datasets and practical considerations.

Data Collection and Example Generation in the WISDM Dataset

The WISDM datasets provide foundational insights into data collection and example generation. The original WISDM dataset [37] collects accelerometer data from smartphones at a frequency of **20Hz**. While the exact duration of data collection per activity is variable, the researchers segment the collected raw data into **10-second examples**, referred to as **example durations (EDs)**. Each 10-second segment contains **200 data points** (20Hz $\times$ 10 seconds), which are used to extract features for training machine learning models. The 10-second ED was chosen as it balances the need to capture sufficient activity repetitions (e.g., walking strides or jogging steps) and ensures high classification performance, as shorter or longer durations were found to be less effective.

In the later WISDM dataset [14], which includes accelerometer and gyroscope data from smartphones and smartwatches, a **3-minute time window** is used to collect data for each activity. However, even in this dataset, the raw data is segmented into **10-second windows** during preprocessing to generate examples for model training. This demonstrates that the 10-second duration is a widely accepted standard for HAR and is effective for feature generation, regardless of the total collection window.

Justification for a 10-Second Data Collection Window

This project adopts a **10-second time window** for data collection directly in the app, rather than a longer collection window (e.g., 3 minutes) with subsequent segmentation. This choice is supported by the following arguments:

1. **Alignment With Existing HAR Datasets**:
Both WISDM datasets process their raw data into 10-second segments for example generation. By collecting data in 10-second windows directly, this project aligns its raw data with the structure typically used in HAR studies, ensuring compatibility with existing methodologies.

2. **Improved Usability For Users**:
A shorter data collection window improves the usability of the application. Asking users to perform specific activities for only 10 seconds is more practical and less burdensome than requiring longer durations (e.g., 3 minutes). This ensures greater compliance and consistency, particularly for activities that require significant effort, such as jogging or running.

Implementation

The following key features were implemented in this phase:

1. **Activity Selection Using RecyclerView:**
   The RecyclerView was populated with activity options such as Walking, Jogging, Sitting, and Standing, which were dynamically displayed in the

UI. When a user selects an activity, the app uses this label to tag the accelerometer data collected during the session.

2. **User ID Input:**
   A simple TextEdit field was added to allow users to input a unique identifier. This ID was appended to the collected data, enabling future analysis on a per-user basis.

3. **Timed Data Collection:**
   The app was updated to record data for a **10-second window** once the user selected an activity and pressed "Start." The accelerometer data was collected using the SensorManager API, implemented in Phase 1, but now labelled with the chosen activity.

4. **Saving Data in CSV Format:**
   At the end of each session, the collected data was saved locally in a CSV file with the following headers: user, x-axis, y-axis, z-axis, activity. This format ensures compatibility with machine learning workflows and facilitates easy data export for further analysis.

Evaluation and Review

The updated app was tested on an Android Wear emulator and a physical smartwatch. The following aspects were evaluated:

- **RecyclerView Functionality:** The activity list displayed smoothly, with no lag or rendering issues, and users could easily select the desired activity.
- **User ID Input:** The TextEdit field successfully captured and saved user IDs, ensuring each session's data was appropriately labelled.
- **Timed Data Collection:** The 10-second window operated as expected, collecting 200 data points per session at a sampling rate of 20Hz.
- **CSV Data Saving:** The CSV files generated at the end of each session were verified to contain the correct data format and labels, ensuring they could be used directly for machine learning model training.

This phase marked a significant improvement in the app's functionality, transitioning it from a simple data logger to a fully functional research tool capable of collecting labelled accelerometer data. The features laid the groundwork for the personalised data collection approach central to the project's success. An updated visual representation of the UI flows in the app is shown in Figure 5.2.

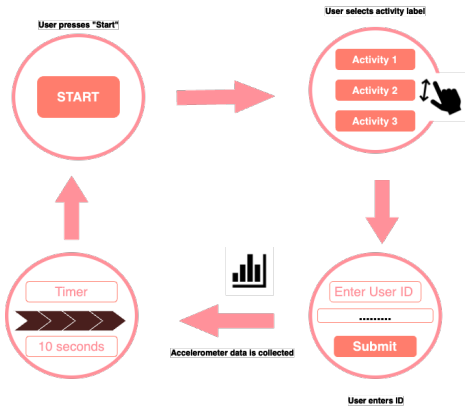

Figure 5.2: Diagram illustrating UI interface of app during Phase 2. Here, the user presses START to go to the second activity screen, where the user is prompted to select from a list of activities (e.g. Running, Walking, Drinking etc.). Following this, the user is the prompted to enter in a user ID (such as a name or numeric identifier). Once the user has entered in the user ID, accelerometer data is collected within a 10 second time window after whcich the activity screen switches back to the initial START screen

5.2.3 Phase 3: Sending Data To An Online Database

Planning and Requirement

The objective of this phase was to enable the seamless transfer of the collected accelerometer data from the smartwatch to an online database, making it accessible for further analysis. The essential tasks I wanted to achieve in this phase were:

- Implement a method in the app to send data to Power Automate
- Design a Power Automate flow to process and store accelerometer data
- Add a new activity screen to allow users to "Send Data"

Analysis and Design

Sending Data To Power Automate Server Using HTTP POST

Data from the app can be **sent** to Power Automate using a Hypertext Transfer Protocol (HTTP) POST request. HTTP is designed to enable communication between clients and servers. Communication between clients and servers works as a **request - response** protocol. A classic example of this is the browser (which acts as a client) sends a HTTP request to a web server. The response contains the requested content (such as a web page) and additionally may also contain status information. There are many methods in a HTTP protocol, the

most common of which are POST and GET. The POST method is used to send data to the server, the **request-body** of a POST method contains the data to send.

As illustrated by Figure 5.3 accelerometer data from the watch is sent via a HTTP POST where the request body contains the accelerometer data, here Power Automate acts as the server.

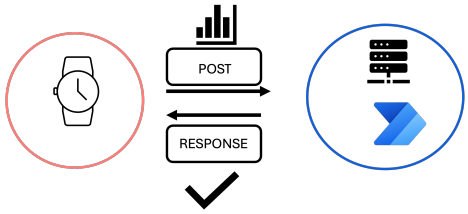

Figure 5.3: HTTP POST method to send accelerometer data from the watch to a Power Automate flow, where the data can be processed and stored.

Receiving And Storing Accelerometer Data From The Watch in Excel

The easiest method of storing data online was in an Excel file on a One Drive account. Using a Power Automate flow, **incoming data** can be easily processed as a JavaScript Object Notation (JSON) payload, a text-based format for storing and exchanging human-readable and machine-parsable data. Each JSON object contains the following information collected from the data collection app: user ID, x-axis value, y-axis value, z-axis value, and selected activity. This can be parsed in Power Automate and used to update a row of the Excel worksheet. There are several technical and practical advantages to storing accelerometer data in an Excel file:

- **Seamless integration with Power Automate:** Using the Excel connector in Power Autoamte, data can be easily appended directly into an Excel table during a flow execution. Power Automate also allows Excel Scripts to run in a flow, which would be helpful in data processing in later steps.
- **Alignment with existing datasets:** The WISDM dataset stores its raw accelerometer data in CSV format, a commonly used structure for machine learning workflows. Data stored in an Excel file can be easily converted to a CSV file.
- **Enhanced data accessibility and filtering:** Excel provides user-friendly tools for sorting, filtering, and visualising data directly within spreadsheets, which makes it easy for researchers to review and validate collected data. These capabilities also enhance the dataset's accessibility and usability for non-technical stakeholders.

Implementation

Figure 4.2 shows the Power Automate flow to handle incoming accelerometer data as a JSON payload. The flow encompasses the following steps:

1. A **manual trigger** to initiate the process: This is set to be when an HTTP POST request is received to the specific URL.
2. A **compose step** to handle incoming data
3. A **Parse JSON** step to ensure data structure is validated
4. A **For Each loop t**o iterate through the list of accelerometer readings.
5. An **Excel connector** to append the data as rows into a pre-configured table within an Excel file.

The data transfer functionality was implemented using the **HTTPRequest-Management** class (code for which is shown in Listing 5.2). Accelerometer data is extracted from SharedPrefrences and transformed into a JSON payload. This payload is then sent to the URL corresponding to the Power Automate HTTP Request trigger. The HTTP POST response is logged for debugging purposes.

```

class HTTPRequestManagement(context: Context) {

// Initialize SharedPreferences
sharedPrefs = context.getSharedPreferences("accelerometerData",
    Context.MODE_PRIVATE)
// Data Class to hold the data
data class PostData(
    val userID : String,
    val xAxis: String,
    val yAxis: String,
    val zAxis: String,
    val activity: String
)
// Function to send a single POSTData Object
private fun sendDataToDataBase(postData: PostData) {
    val postDataJson = gson.toJson(listOf(postData))

    Log.d(Tag, "Sending JSON Body: $postDataJson")

    val requestBody =
        postDataJson.toRequestBody("application/json".toMediaType())
    val request = Request.Builder()
        .url(endpointURL)
        .header("Content-Type", "application/json")
        .post(requestBody)
        .build()
    try {
```

```
27          client.newCall(request).execute().use { response ->
28              if (!response.isSuccessful) {
29                  Log.d(Tag, "Failed to send data : $response")
30                  throw IOException("Unexpected response")
31              } else {
32                  Log.d(Tag, "Data sent successfully")
33              }
34          }
35      } catch (e:IOException) {
36          Log.d(Tag, "Error while sending data : ${e.message}")
37          e.printStackTrace()
38      }
39
40  }
```

Listing 5.2: HTTPRequestManagement class to handle data transformation and sending HTTP POST request to Power Automate flow.

Evaluation and Review

The integration was tested to ensure data transfer and user experience. I tested the Power Automate flow and the successful transfer of data by collecting data from an emulator. The data arrives in the excel file as shown by Figure 5.4.

UserID	X	Y	Z	Activity
HAMZA	0.8054936	10.294489	0.8554068	Sitting
HAMZA	0.8054936	10.294489	0.8554068	Sitting
HAMZA	0.8054936	10.294489	0.8554068	Sitting
HAMZA	0.8054936	10.294489	0.8554068	Sitting
HAMZA	0.8054936	10.294489	0.8554068	Sitting
HAMZA	0.8054936	10.294489	0.8554068	Sitting
HAMZA	0.8054936	10.294489	0.8554068	Sitting

Figure 5.4: Screenshot of example accelerometer data in an Excel file on One drive collected from watch device and uploaded via HTTP POST request and custom Power Automate flow

Overall, in this phase I implemented the following features:

- A Power Automate flow to accept accelerometer data as a JSON payload and add the data to an Excel File.
- An HTTPRequestManagement class to transform the accelerometer data into a JSON object and send an HTTP POST request.
- An activity screen containing a button to ”Send Data” after the 10-second collection session.

5.2.4 Phase 4: Easy Configurable List Of Activities

Planning and Requirements

The primary goal of this phase was to allow users to configure the list of activities dynamically without requiring programming knowledge. This feature is handy for researchers who want to adjust the tracked activities based on their study requirements. The planned solution involved:

- Enabling users to select predefined activity lists via a Microsoft Form.
- Store the activities list in an Excel file on One Drive.
- Dynamically retrieve and update the activity list on the smartwatch app.

Analysis and Design

System Overview
The system uses **Microsoft Forms**, **Power Automate flows**, and **Excel integration** to achieve dynamic activity configuration. The flow operates in three key steps:

1. **User Input via Microsoft Form**: A form (as shown in Figure 5.5) allows users to select a preconfigured activity list from a drop down menu. Each option corresponds to a sheet name in an Excel file ("ActivitiesList") stored on OneDrive (Figure 5.6)

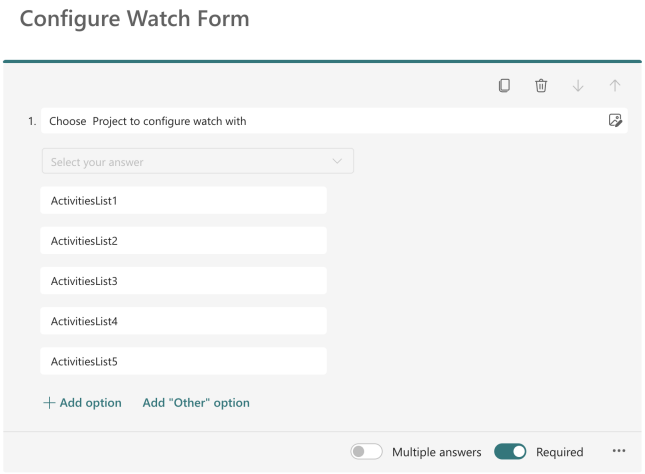

Figure 5.5: A Microsoft Form containing a drop-down list of activities, each corresponding to an Excel Sheet in an Excel File that can be parsed into a JSON array in Power Automate

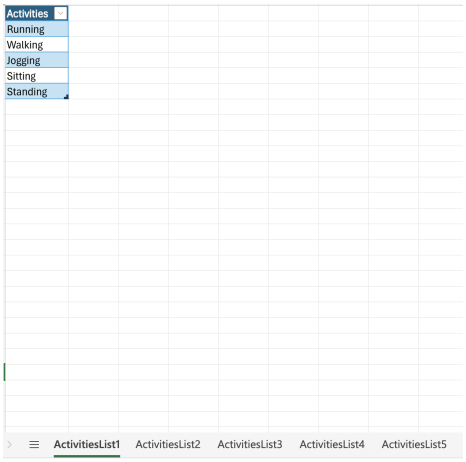

Figure 5.6: List of activities corresponding to the Microsoft Form

2. **Power Automate Flows**:

   - **Flow 1** : (Appendix: Figure 3) This flow retrieves the selected sheet's activity list from the "Activities List.xls" Excel file using a custom script and appends it to a "Processed Answers.xls" file. This file serves as a static reference for the smartwatch app.
   - **Flow 2**: (Appendix: Figure 4) Processes the activities from the "Processed Answers.xls" file and sends them to the app via an HTTP POST request.

3. **App Integration**: The app retrieves the activity list from "Processed Answers.xls" by sending a POST request and reciving the selected acitvities in the response body. These activities are then used to populate the RecyclerView on the activity selection screen.

The diagram in Figure 5.7 illustrates the implementation approach how of how the the data collection app is linked with power automate to configure activities list in the app and then to transmit the collected data to an Excel online database

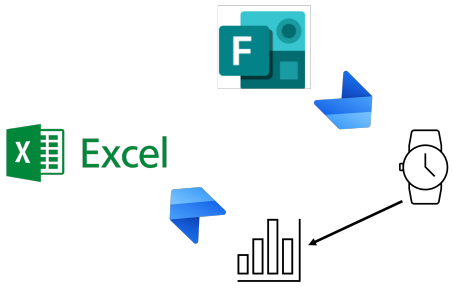

Figure 5.7: Illustration of the integration between the smartwatch app, Microsoft Forms, and Power Automate (blue arrows). The app dynamically configures activities using Microsoft Forms, retrieves activity lists stored in Excel via Power Automate, Collects the accelerometer data and uploads collected data to an online Excel spreadsheet using Power Automate.

Implementation

Flow 1 : Handling Form Responses

When a user submits a response in the Microsoft Form, a custom Excel script retrieves the selected activity list. The script 5.3 fetches the activities from the corresponding sheet in the "Activities Lists.xls" Excel file and appends them to the "Processed Answers.xls" file. This script is triggered in Power Automate Flow 1 (Appendix : Figure 3)

```
function main(workbook: ExcelScript.Workbook, sheetName: string):
    string[] {
    const sheet = workbook.getWorksheet(sheetName);
    if (!sheet) throw new Error(`Worksheet with name ${sheetName}
        not found.`);
    const table = sheet.getTables()[0];
    const values = table.getRange().getValues();
    const activities = [];
    for (let i = 1; i < values.length; i++)
        activities.push(values[i][0]);
    return activities;
}
```

Listing 5.3: Excel script to return a list of activities from "Activities Lists.xls" file

Flow 2: Sending Activities To The App

The app sends a POST request to Power Automate Flow 2 (Appendix : Figure 4) using the HTTPRequestManagement class's retrieveActivitiesData() method (Listing 5.4). The retrieved activity list is saved locally in SharedPreferences.

```
    fun retrieveActivitiesData() {
// remove previous activities from sharedprefrences
    sharedPrefActivities.edit().remove("activitiesList").apply()
// request body for POST request
    val requestBody = "Requesting activities
        list".toRequestBody("text/plain".toMediaType())
    val request = Request.Builder()
        .url(configureURL)
        .post(requestBody)
        .build()
    client.newCall(request).execute().use { response ->
        if (response.isSuccessful) {
            val jsonData = response.body?.string()
// If the request is sent successfully, then the response body
    contains the list of activities  from "ProcessedAnswers" file
    and is added to the sharedprefrence instance
            sharedPrefActivities.edit().putStringactivities
                listst", jsonData).apply() }
    }
}
```

Listing 5.4: Method to retrieve activities list as a response from a POST request

Activity list is accessed from SharedPreferences and loaded into the RecyclerView on the **Activity Selection** screen. Each activity is represented as a widget, allowing users to select an activity to track. The above function (Listing 5.4) is called in a Configure Activities List class, which prompts the user to "Configure Watch" and sends the HTTP POST request. The **Activity Selection** screen dynamically updates to reflect the configured activity list.

Evaluation and Review

Flow Functionality: Both Power Automate flows were tested to ensure the correct retrieval and processing of activity lists. Errors, such as missing sheets or tables, were handled gracefully with appropriate error messages.

App Integration: The app successfully retrieved activity lists and populated the RecyclerView dynamically. Changes in the Excel file were reflected immediately upon reconfiguration.

User Experience: Researchers could easily configure the watch without technical expertise, fulfilling the requirement for a dynamic and user-friendly setup.

Overview of Data Collection App

The sequence diagram shown in Figure 5.8 illustrates the app's complete workflow.

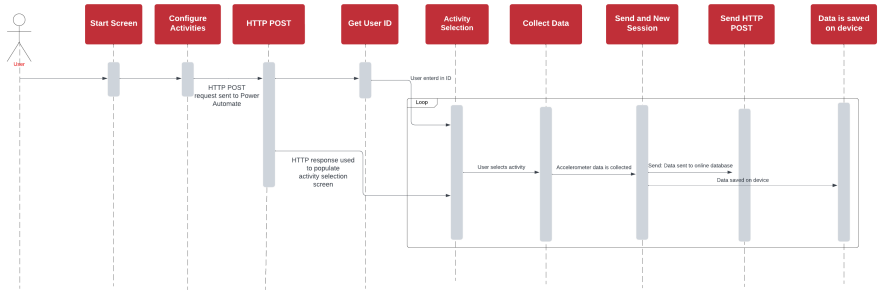

Figure 5.8: Sequence Diagram, illustrating the flow of Data Collection App

The user begins at the Start Screen and configures activities via a Power Automate flow. Activities are loaded from an Excel sheet stored on OneDrive, which can be edited through a Microsoft Form containing a dropdown menu of various activity lists. On the app, the user is prompted to enter their User ID (such as a name or a unique numerical identifier), select an activity from the configured list, and initiate data collection. Upon completing the data collection, the user can upload the collected data to an online database, while a copy of the accelerometer data is also saved on the smartwatch for local access.

Chapter 6

Critical Evaluation

In this chapter I cover the critical evaluation of the finished product, noting the strength and achievements as well as the limitations of the current application and space for future work.

The completed data collection application successfully meets the core requirements outlined by the Tobacco and Alcohol Research Group. The application enables users to collect labelled accelerometer data via a smartwatch device, store the resulting data securely in an online database, and dynamically configure the activity list through a user-friendly interface. Additional functionality was incorporated to enhance the app's robustness as a research tool, including capturing timestamp, data, and unique device id from the smartwatch used to collect the data.

6.1 Strength and Achievements

The app fulfils all the primary requirements provided by the client. By integrating features such as user-configurable activity lists and data collection within a 10-second time window, the app offers significant flexibility for researchers. Furthermore, using a Form-based method to configure the app allows it to be used efficiently, and due to the modular design of the app, further configurable components can be added and integrated into the Form.

6.2 Limitations and Area for Improvements

6.2.1 Configurable Data Collection Parameters

While the app allows researchers to collect accelerometer data in a standard 10-second time window, there is no option to configure the sampling frequency or time window duration. Related work notes that different activities may require varying data collection durations. For instance, a study on smoking

[35] found the average smoking session to last 4-5 minutes, while another study on drinking [5] had self-reported drinking episodes averaging 15 minutes. While the 10-second window remains justified due to its prevalence in existing datasets like WISDM, offering researchers the ability to customise the time window or frequency would make the app even more versatile.

6.2.2 Data Upload Delays

A notable limitation of the app is the delay in data upload when transferring data to Power Automate via the HTTP POST requests. The processing time can lead to the mixing of activity data when two sessions occur in quick succession. For example, accelerometer data from a drinking session could overlap with data from a subsequent smoking session. This issue arises because the transmission process does not pause data collection between activity changes. To mitigate this, a queuing system was implemented (Listing 6.1), ensuring data is processed sequentially.

```
1 private fun sendDataFromQueue() { executor.execute { while
      (postDataQueue.isNotEmpty()) { val
2 // Retrieve and remove the head of the queue
3 currentPostData = postDataQueue.poll() currentPostData?.let {
      sendDataToDataBase(it) } } } }
```

Listing 6.1: Queueing mechanism for managing data uploads to Power Automate

While this method reduced delays, it did not eliminate the issue of overlapping data, leading to the hypothesis that the issue may be due to the 'for each' loop in the Power Automate flow used to populate the Excel rows, causing the bottleneck in data processing and upload. Nonetheless, the accelerometer data remained uncorrupted; the mixing occurred only in data organisation, with rows from different sessions being interleaved. A unique session ID was introduced for each data collection session to address this. Researchers can now filter data by session ID, ensuring clean segmentation of activities despite transmission delays.

6.3 Conclusion

The development of the data collection app demonstrates significant progress in making HAR research more accessible and efficient. I have also demonstrated the efficiency in using Agile methodology to develop such an app, adding to the modularity of the application to allow future features to be implemented more efficiently. While the app successfully meets client requirements, further improvements such as configurable collection parameters and optimised data upload mechanisms could enhance its utility. Nevertheless, the app provides a robust foundation for collecting sensor data in a structured and scalable manner,

supporting current research needs and providing a stepping stone in the development towards a personal informatics tool to be used for activity monitoring to mitigate behaviours that lead to premature deaths.

Chapter 7

Project Execution : Using Machine Learning for Human Activity Detection

In this chapter, I describe the process of using machine learning models to predict human activity based on accelerometer. I use the WISDM dataset, building on existing work done by Fang Wang [38] and Patrick Nabryia [39] by exploring the use of a balanced dataset and CNN models. I also use the data collected from the data collection app and the same machine learning models to distinguish between activities : "Drinking Water", "Typing", "Eating popcorn", "Vaping", and "Walking". This chapter focuses on the methodologies for data preprocessing, feature extraction and training.

7.1 Using Machine Learning To Predict Human Activity In the Wireless Sensor Data Mining (WISDM) dataset

Fang Wang in their thesis [38] explored multiple ML performance on WISDM dataset. The most optimal models suited for smart watch applications were found to be Decision Trees and Random Forest due to the good balance between model performance and processing power (measured as time taken to train model). In this thesis I will build upon this and the work of Patrick Nabriya [39] by comparing the use of a **unbalanced** vs **balanced** data set on Decision Trees and Random Forests as well as exploring the effect of a CNN model. I use a target individuals dataset to build a personalised model rather then an impersonal model (trained on data from all users) for activity prediction. The evidence for a personal model preforming significantly better at activity recognition then impersonal models is demonstrated by the accuracy

scores of various ML models tested in Fang Wang's thesis (Tables 7.1 and 7.2) on accelerometer and gyroscope WISDM dataset [14]

Model	**Overall Accuracy (%)**
Logistic Regression	62.32
Decision Tree	55.82
Support Vector Machines	63.27
Multilayer Perceptron Classifier	61.73
Random Forest	65.69
Xgboost	66.16
Bayesian	46.25
Average (AVE)	**60.17**

Table 7.1: Overall accuracy for impersonal models. [38]

Model	**Overall Accuracy (%)**
Logistic Regression	88.60
Decision Tree	84.73
Support Vector Machines	88.80
Multilayer Perceptron Classifier	89.64
Random Forest	90.92
Xgboost	90.22
Bayesian	82.60
Average (AVE)	**87.93**

Table 7.2: Overall accuracy for personal models. [38]

7.1.1 Overview (WISDM) dataset

WISDM is a widely used dataset for Human Activity Recognition research. It consists of triaxial accelerometer data collected from 36 different users preforming six different activities: Walking, Jogging, Going Upstairs (Upstairs), Going Downstairs (Downstairs), Sitting, Standing. The dataset is structured with six different columns: users, activity, timestamp, x, y, z axis, where each is axis corresponds to the three dimensional data collected by an accelerometer. Data is collected at a frequency of 20 Hz, meaning that one data point (x, y, z change in velocity) is collected every 50 milliseconds.

7.1.2 Data Cleaning and Preprocessing

Data Download and Cleaning

The WISDM dataset was imported and preprocessed using Python and data handling libraries such as Pandas and NumPy as shown by code in Listing 7.1 Columns were assigned names, and missing or malformed values were handled

by dropping null entries. The z-axis data which contains trailing semicolons , was cleaned and converted to numeric format for consistency.

```
column_names = ['user', 'activity', 'timestamp', 'x-axis',
    'y-axis', 'z-axis']
data = pd.read_csv('../data/WISDM_ar_v1.1_raw.txt', sep=',',
    names=column_names, on_bad_lines='skip')
data.dropna()
data['z-axis'] = data['z-axis'].str.strip(';').astype(float)
```

Listing 7.1: Python code showing, data cleaning of WISDM raw dataset: 1. Reading the text data into a pandas dataframe, 2. dropping null values, 3. converting the datatype of z axis of type float

A preview of the data after pre-processing is shown in Table 7.3

user	activity	timestamp	x-axis	y-axis	z-axis
33	Jogging	49105962326000	-0.694638	12.680544	0.503953
33	Jogging	49106062271000	5.012288	11.264028	0.953424
33	Jogging	49106112167000	4.903325	10.882658	-0.081722
33	Jogging	49106222305000	-0.612916	18.496431	3.023717
33	Jogging	49106332290000	-1.184970	12.108489	7.205164

Table 7.3: Preview of WISDM data set after cleaning, showing user, acitvity, x-axis value, y-axis values, and z-axis values

7.1.3 Exploratory Data Analysis

Total Activity Distribution

Figure 7.1 illustrates the class distribution of activities within the raw WISDM dataset. Walking and Jogging are over represented, whereas Sitting and Standing are under represented activities, leading to an imbalanced dataset. This imbalance can cause machine learning model bias as their predictions toward the dominant classes, as we will explore here.

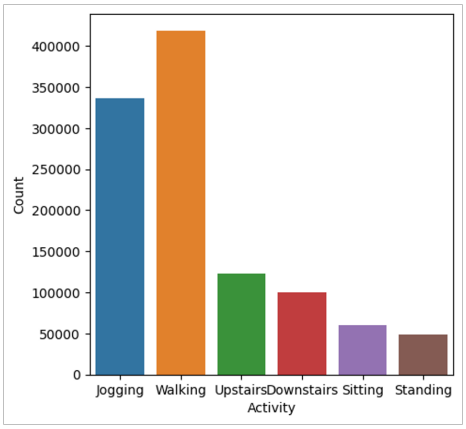

Figure 7.1: Activity distribution in WISDM dataset

Distribution Of Activities Per User

From this data, we can also see there is large imbalances between the activities preformed by each user as shown in Figure 7.2

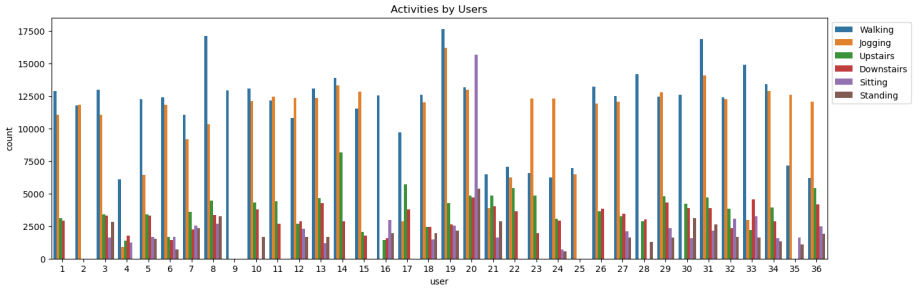

Figure 7.2: Distribution of each activity per user in the WISDM dataset

Changes in Accelerometer Data across Various Activities

Figure 7.3 shows the changes in acceleration as shown by the variation in x, y, and z-axis in a 5 second window for a user 33 across the six different activities.

As we can see, activities that require more movement, such as Jogging and walking, have much more acceleration changes than sedentary movements, such as Sitting and Standing. This can lead to challenges for traditional ML models, as they rely on clear feature distinctions to classify activities. Therefore, dynamic movements like Joggings or Walking that generate more pronounced changes in acceleration across the axes provided a richer dataset with more easy-to-identify patterns than sedentary movements, which produce fewer unique features to leverage for predictions. However, advanced models, like Neural Networks, can capture subtle differences and interactions.

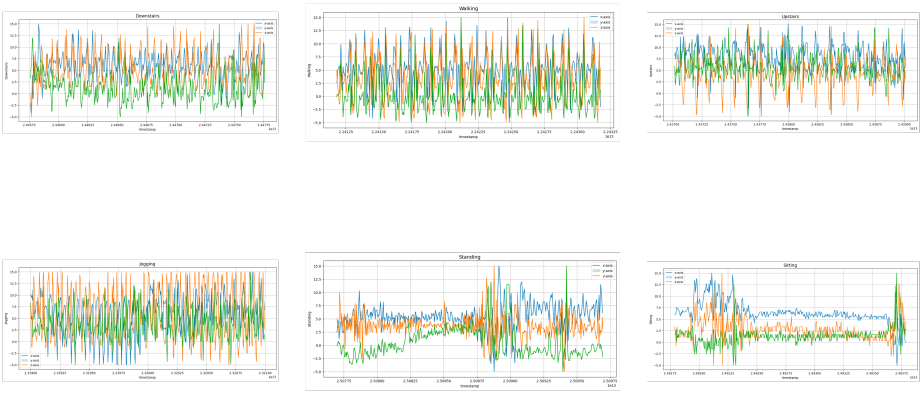

Figure 7.3: Accelerometer reading within a 5 second window for user 33 across Jogging, Walking, Upstairs, Downstairs, Sitting and Standing

7.1.4 Feature Extraction

Windowing To convert the raw time-series accelerometer data into features suitable for machine learning, overlapping windows of data were created, an detailed explanation of windows is provided in section 2.2.2. Here, I use a window size of **100 data points with a 50% overlap** as in the original article by [39]. For each window containing a set of data , 18 simple statical features were calculated for x, y and z axis data:

- **Mean:** The average value of each dataset value.
- **Standard Deviation:** a measure of the variation of the value from the mean.
- **Mean Absolute Deviation:** The average of the absolute values of the deviations of each data from the mean.
- **Minimum value:** The lowest value in the dataset
- **Maximum value:** The maximum value in the dataset
- **Difference between minimum and maximum value**
- **Median**
- **Median absolute deviation:** The median of the absolute value of the new data, obtained by subtracting the median from the original data
- **Interquartile range**
- **Negative value count:** Values below 0
- **Positive value count:** Values above 0

- **Number of values above the mean**
- **Number of peaks:** Count of the distinct high points in the signal, identifying rapid changes or repetitive patterns within the data.
- **Skewness:** Measures of the asymmetry of a probability distribution. Positive skew indicated a longer tail on the right, while negative skew indicates a longer tail on the left
- **Kurtosis:** Assesses the 'tailedness' of the distribution, indicating whether data points are concentrated around the mean (low kurtosis) or in the tails (high kurtosis)
- **Energy:** Represents the signal strength, calculated as the sum of the square of all values.
- **Average composite acceleration:** combined magnitude of acceleration in all axes.
- **Signal amplitude area:** area under the curve of the absolute signals values.

This step is conducted for both the training and testing set. For this project I created a custom Python class to calculate the 18 statistical features mentioned above. The full code for which is shown in Appendix : A.

7.1.5 Extracting Data From A Single User To Build A Personal Model

To build a personal ML model, I extracted the data for a single user from 36 users in the WISDM dataset and preform the data preparation steps described above. I chose User 8's data because this user had the most comprehensive data across all activity classes (Figure 7.2. The class distribution of User 8's dataset is shown in Figure 7.4 . I then made two configurations of the dataset:

1. **Unbalanced Dataset**: Retaining the original class distributions.
2. **Balanced Dataset**: Downsampling the majority classes (e.g., Walking and Jogging) to match the smallest class, "Sitting" with 2699 samples: Figure 2.

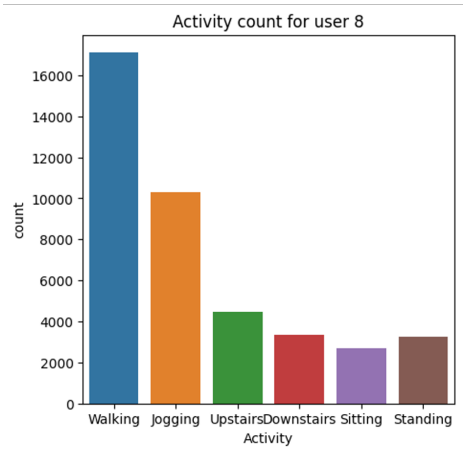

Figure 7.4: Activity Class distribution for User 8 from the WISDM dataset

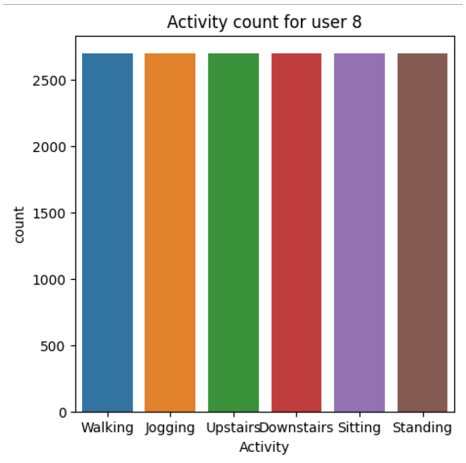

Figure 7.5: Activity Class distribution for User 8 **after** downsampling

7.2 Data Collected from Data Collection Application

For the final phase of this project, data was collected using the developed smartwatch application to evaluate its performance in real-world scenarios. The activities selected for data collection included "Drinking Water" "Vaping" "Eating Popcorn," "Walking," and "Sitting" (Figure 7.6) as these represent a diverse range of movements and behaviours (and relate back to activities that cause NCD-related deaths). Data was collected at a frequency of 20 Hz, meaning 20

data points per second were recorded for the x, y, and z-axis accelerometer readings. Each activity session lasted for 10 seconds, and six sessions were recorded for each activity, resulting in a total of 60 seconds of data per activity.

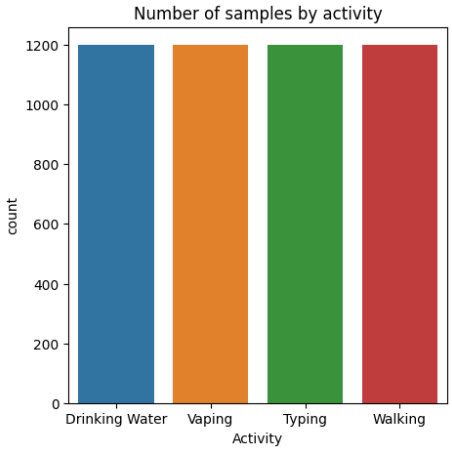

Figure 7.6: Data Collected from data collection app for Drinking Water, Vaping, Typing, Walking. Data was collected at a frequency of 20 Hz

7.3 Model Training

For this study, I utilised three machine learning approaches: Decision Trees, Random Forest, and 2D Convolutional Neural Networks (CNNs). Each model was trained on the feature engineered user 8's data. Below, I describe the training process for each approach.

7.3.1 Training Decision Tree and Random Forest Classifiers

After creating windows of size 100 with 50% overlap and extracting features from the accelerometer data, the data is standardised. For training and testing sets I use (k=5) K-fold validation to split the data. The model is trained on k-1 folds and the remaining folds are used to test the model. This process is repeated five times, where each fold serves as a test once. K-fold validation ensures that accuracy is consistent and that model is generalising well and data over fitting, data leakage of inflated accuracy from repeated runs are limited.

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
# Decision Tree with Cross-Validation
decisionTree = DecisionTreeClassifier(random_state=529)
cv_scores_dt = cross_val_score(decisionTree, X_train, y_train,
    cv=skf, scoring='accuracy')
```

```
decisionTree.fit(X_train, y_train)
dT_yPred = decisionTree.predict(X_test)
dT_accuracy = accuracy_score(y_test, dT_yPred)

print(f"Decision Tree Accuracy on test set: {dT_accuracy *
    100:.2f}%")
print("Decision Tree Cross-Validation Scores:", cv_scores_dt)
print(f"Mean Cross-Validation Accuracy: {np.mean(cv_scores_dt) *
    100:.2f}%")
print("\nDecision Tree Classification Report:")
print(classification_report(y_test, dT_yPred))
```

Listing 7.2: Training Decision Tree on user 8 data

Training a 2D Convolutional Neural Network (CNN)

Preprocessing for CNN

For the CNN model, data cleaning and balancing steps were repeated. Window sizes and over lap were kept consistent for comparison with Decision Trees and Random Forest models. However, given the rich time-series nature of the accelerometer data, a 2D CNN was employed to learn spatial and temporal patterns from the raw data and unlike the Decision Tree and Random Forest models, the CNN was trained directly on raw accelerometer windows without explicit feature extraction.

After segmentation the data set is not a 3D list with shape [N, 3, 100] (number of data points, number of features, number of features, and window size). To prepare the data for CNN, the data must first be re-shaped into a 4D array format, this prevents loss of meaningful data.

CNN Architecture The CNN architecture was designed to balance performance with computational efficiency, inspired by similar work in human activity recognition using accelerometer data [40].

Training and Evaluation The model was trained for 10 epochs with a batch size (the number of training examples, before the model adjusts its weight) of 32, using an 80-20 split for training and validation data. Training metrics (accuracy and loss) were monitored across epochs.

7.4 Summary

This chapter demonstrates the application of machine learning techniques to classify human activities using accelerometer data. I use the WISDM data and data collected from the data collection app to train three ML models: Decision Tree, Random Forest and CNN. Decision Tree and Random Forest models were chosen due to the balance between high accuracy and F1-scores on the WISDM dataset but low processing power, which is essential for deployment of models on

smartwatches for real time processing and activity detection. I train the models on a target user from the WISDM dataset, training models on a balanced dataset (uneven activity distribution) and balanced dataset (downsampled to create a even activity distribution). In the next Chapter I will be comparing the results from these models.

Chapter 8

Results and Evaluation

In this chapter, I present the results of machine learning model training for human activity recognition. I use the WISDM dataset and data collected from the custom built app (covered in Chapter 5), comparing the effect of balanced and unbalanced datasets. For each dataset I evaluate three ML models: Decision Trees, Random Forest and CNN. Models are evaluated using their accuracy score, recall (positive calls) and visualised using Confusion matrices. Overall I demonstrate the strength and limitations of each model on and provide evidence for feasibility of real time human activity recognition.

8.1 Personal Models : User 8 Data From WISDM

The personal model is trained on the data of a target user. This user was selected from the 36 users present in the WISDM dataset [21] due to the comprehensive presence of all target activity classes. After data cleaning, the target user's data is extracted, and used to extract features as described in Chapter 7. I use K-fold validation with five folds to train and test the data. All models preform exceptionally well, with the accuracy of the Decision Tree model being 94.5% on unbalanced data (Figure 8.1 set and 96.5% on balanced dataset (Figure 8.2). In the unbalanced model, there is high classification for over represented activities such as Walking (recall = 1.00) and Jogging (recall = 0.92), with lower performance on under represented activities such as Upstairs (recall = 0.67).

```
Decision Tree Accuracy on test set: 94.92%
Decision Tree Cross-Validation Scores: [1.         0.95744681 0.95744681 0.95744681 0.93617021]
Mean Cross-Validation Accuracy: 96.17%

Decision Tree Classification Report:
              precision    recall  f1-score   support

  Downstairs       0.94      0.94      0.94        18
     Jogging       1.00      0.92      0.96        12
     Sitting       0.93      1.00      0.96        13
    Standing       1.00      0.83      0.91         6
    Upstairs       0.75      0.67      0.71         9
     Walking       0.97      1.00      0.98        60

    accuracy                           0.95       118
   macro avg       0.93      0.89      0.91       118
weighted avg       0.95      0.95      0.95       118
```

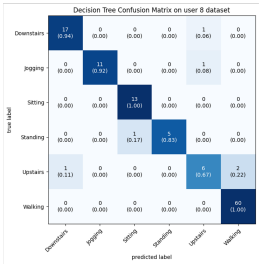

Figure 8.1: Decision Tree model performance and Confusion matrix for unbalanced data from User 8 in the WISDM dataset. Activities: Downstairs, Jogging, Sitting, Standing, Upstairs, Walking

The balanced dataset performs slightly better in accuracy, however there is consistent high performance across all activity classes (e.g. Downstairs, and Walking both achieve a recall of 1). Under represented activities such as Upstairs and Standing show better recognition in the balanced model, with the recall of 0.91 for both.

```
Decision Tree Accuracy on test set: 96.92%
Decision Tree Cross-Validation Scores: [0.96153846 1.         0.98039216 0.98039216 0.98039216]
Mean Cross-Validation Accuracy: 98.05%

Decision Tree Classification Report:
              precision    recall  f1-score   support

  Downstairs       1.00      1.00      1.00        11
     Jogging       0.92      1.00      0.96        11
     Sitting       0.92      1.00      0.96        11
    Standing       1.00      0.91      0.95        11
    Upstairs       1.00      0.91      0.95        11
     Walking       1.00      1.00      1.00        10

    accuracy                           0.97        65
   macro avg       0.97      0.97      0.97        65
weighted avg       0.97      0.97      0.97        65
```

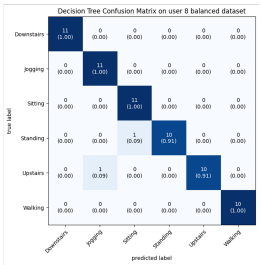

Figure 8.2: Decision Tree model performance and Confusion matrix for balanced data from User 8 in the WISDM dataset. Activities: Downstairs, Jogging, Sitting, Standing, Upstairs, Walking

The Random Forest models preform even better, classifying with near perfect activity prediction. Accuracy for the unbalanced data was 99.15% with high represented activities (Jogging and Walking) having a recall of 1, meaning that all instances of Jogging and Walking were identified 100% of the time. There was also improved performance for under represented activities over the unbalanced Decision Tree model. This means that Random Forest approach helps address class imbalances similar to Decision Trees with balanced data.

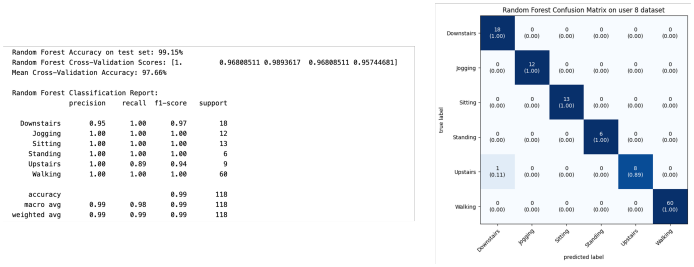

Figure 8.3: Random Forest model performance and Confusion matrix for unbalanced data from User 8 in the WISDM dataset. Activities: Downstairs, Jogging, Sitting, Standing, Upstairs, Walking

The recall for all activities (except for Upstairs and Standing) was 1, however there is a slight miscalculation for Standing, which had a recall of 1 in the unbalanced dataset. It is not clear as why this is the case (re running the model and checking for errors in code did not reveal bugs). I was however, able to increase the model accuracy and recall for Standing to 1 using 10 fold cross validation. Interestingly, recall for Upstairs could not be increased to 1 for these models, despite higher k splits. Due to the similarity in movement in Upstairs and Downstairs it is highly likely that this phenomena is due to disparity in User 8's data specifically. I was able to confirm this by testing the models on another user and attaining a recall of 1 across both Random Forest models and balanced Decision Tree model.

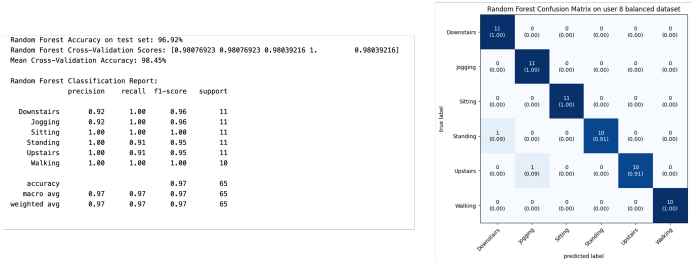

Figure 8.4: Random Forest model performance and Confusion matrix for balanced data from User 8 in the WISDM dataset. Activities: Downstairs, Jogging, Sitting, Standing, Upstairs, Walking

The CNN models also preform very well, with high accuracy scores across all activities. The balanced and unbalanced CNN models show slightly different trends in miscalculation. The unbalanced model shows highest miscalculation with Upstairs and Downstairs (recall of 0.54), which is most likely due to the similarities in accelerometer patterns between these two activities. Activities like Jogging perform better in the unbalanced model then the balanced one, which indicates the suitability of CNN to over represented classes similar to

traditional models.

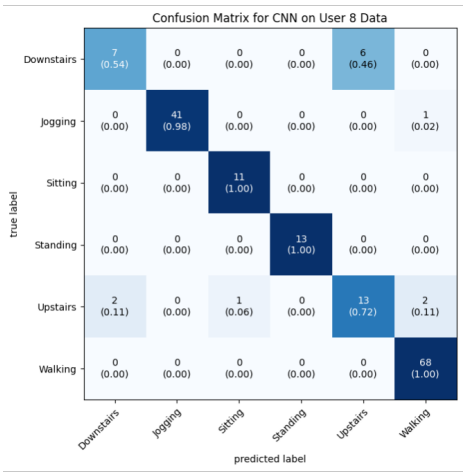

Figure 8.5: CNN on unbalanced User 8 Data. Activities: Downstairs, Jogging, Sitting, Standing, Upstairs, Walking

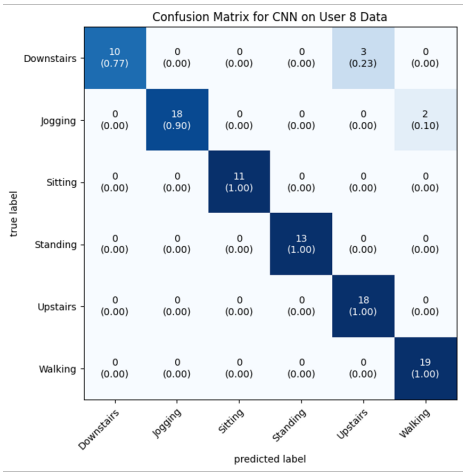

Figure 8.6: CNN confusion matrix on balanced User 8 data. Activities: Downstairs, Jogging, Sitting, Standing, Upstairs, Walking

8.1.1 Data Collected from App

Using the data collection app developed as a research tool for TARG group, I collected data from four activities: “Drinking Water”, “Typing”, “Vaping”, and “Walking”. Each activity was performed for 10 seconds, with 200 data points being collected within that 10 second time frame. Each activity session

was repeated 6 times for a total of 1 minute collection for each activity. Data processing was the same as for preparation for user 8's data with dropping null values and balancing performed.

The Decision Tree model achieved a test accuracy of 63.15%, showcasing a moderate performance with recall scores ranging from 0.4 for"Drinking Water" and "Vaping" to 1 for "Walking". This suggests that the decision tree struggled with classifying activities that had less distinct features in the data, were "Drinking Water" was misclassified across all activities.

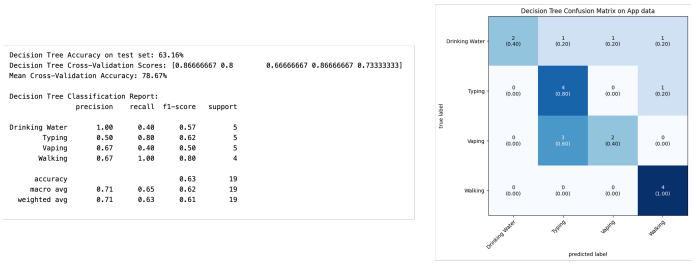

Figure 8.7: Decision Tree model on accelerometer data collected from data collection app. Activities: Typing, Vaping, Walking, Drinking Water

The Random forest model makes a large improvement in accuracy with 78.95%. Activities. This model generalises well across all activities, the recall for drinking water, typing and Walking being quite high.

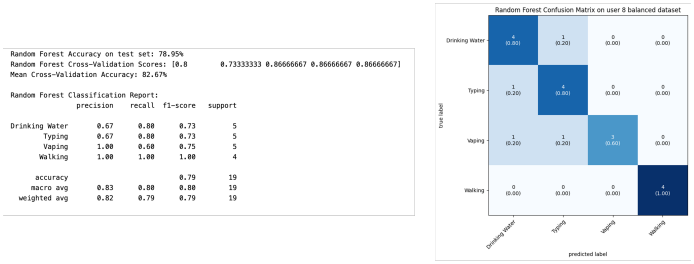

Figure 8.8: Random Forest model on accelerometer data collected from data collection app. Activities: Typing, Vaping, Walking, Drinking Water

In contrast, the CNN model achieved more nuanced predictions, with perfect recall for Walking (recall = 1) , and near perfect for Vaping (recall =0.98), but there was distinctive lower performance of CNN for "Drinking Water" compared to both Random Forest and Decision Trees.

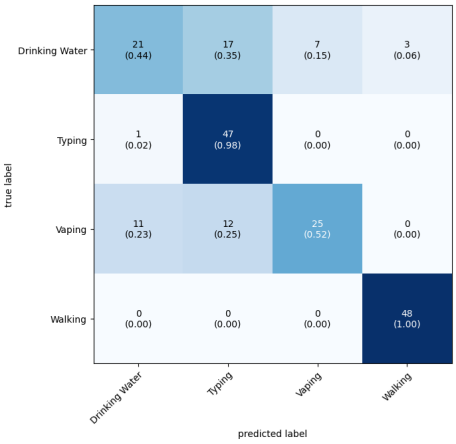

Figure 8.9: CNN model on accelerometer data collected from data collection app. Activities: Typing, Vaping, Walking, Drinking Water

These results demonstrate the strong performance of Decision Trees, Random Forest and CNNS in recognising human activity behaviour within a personalised model approach. For user 8's dataset, Decision Trees and Random Forest achieved exceptional results, with certain activities showing higher recall compared to the CNN model. Analysis of data collected from the app showed a noticeably lower model performance despite consistency in data processing and model training techniques. This reduction in accuracy is likely due to the method of data collection. To simulate "realistic" movements, activities were not performed continuously within the 10-second window, leading to large time gaps in the data, as illustrated in Figure 8.10.

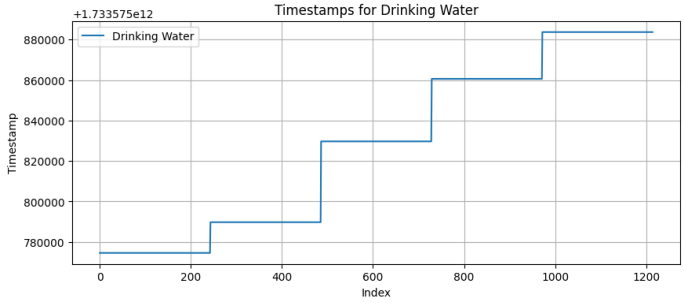

Figure 8.10: TimeStamp data for activity "Drinking Water"

In summary, this work successfully demonstrates human activity recognition using machine learning models. Notably, traditional models like Decision Trees provide comparable performance to computationally demanding neural networks. This finding underscores the feasibility of deploying simpler models,

such as Decision Trees, on smartwatches for real-time data collection, processing, feature extraction, and activity prediction.

Chapter 9

Conclusion

This project successfully demonstrates the application of machine learning models for human activity recognition (HAR), offering insights into both controlled datasets and real-world scenarios. This project aimed to facilitate research to combat non-communicable diseases (NCD) and related deaths. These deaths result in claiming 41 million lives annually, according to the latest WHO report. Activities such as Drinking alcohol, smoking, eating and sedentary behaviour can all lead to premature death, and thus, a system to measure and control these behaviours is an imperative goal for HAR research. Machine learning models for HAR provide a promising avenue to track these habits autonomously through smartwatches, which are ubiquitously used in most third-world countries [41]. Smartwatches contain sensors like accelerometers, which provide velocity changes in three axes (x, y, z) to detect changes in movement. Thus, by embedding HAR techniques in smartwatches to detect real-time activity detection, we can provide feedback to users to track their behaviour and reduce the effect of NCD-related deaths. The first step in HAR is to have a reliable data collection method. Having an application to collect labelled data is an invaluable tool in the researcher's toolbox. The main objective of this project was to build a data collection app for smartwatches that meets the following requirements :

- Collect labelled data
- Save data on an online database.
- Configurable list of activities.

In this project, I describe the Agile methodology and iterative development process to design an application using Android Studio for a smartwatch that meets all the client requirements. Evaluation of the final application from the Tobacco and Alcohol Research Group was immensely positive, with plans to use the application in their research to collect data for the detection of binge drinking and vaping.

Additionally, I use the Wireless Sensor Data Mining (WISDM) dataset on human activity and data collected from the custom-built data collection app in this project to explore machine learning models: Decision Trees, Random Forests, and Convolutional Neural Networks (CNN). The WISDM data contains tri-axial accelerometer data from smartphones of various activities, including: "Sitting", "Standing", "Walking", "Jogging", "Upstairs" and "Downstairs". Similarly, using the same sampling frequency as most datasets, including the WISDM, I collected data in 10-second intervals for the following activities: "Walking", "Typing", "Vaping" and "Drinking Water". I collected each activity for a total of 1 minute.

Personalised models trained on User 8's data specifically achieved outstanding performance with near-perfect recall for some activities in all models. Using a balanced dataset by downsampling data showed a slight improvement in the overall accuracy of all models. However, the most noticeable change was the improved detection of less represented activities, such as "Upstairs" and "Standing" which had fewer data points than other activities. Data collected through the custom smartwatch app added a practical layer to the project, as the activities simulated realistic scenarios with non-continuous movements. While this approach introduced challenges such as data gaps, Random Forests demonstrated strong generalisation capabilities, outperforming Decision Trees and CNNs across most activities. Despite the reduced performance compared to controlled datasets, the app's success in collecting labelled accelerometer data is crucial for researchers studying behavioural patterns linked to NCD risks.

A significant outcome of this project is the demonstration that traditional models, such as Decision Trees, can perform comparably to computationally demanding CNNs. This finding has practical implications for deploying lightweight models on wearable devices for real-time HAR. By enabling continuous monitoring of physical activity and behaviours, such systems could play a critical role in NCD prevention and management. In conclusion, this work combines machine learning, personalised modelling, and smartwatch-based data collection to address a pressing global health challenge. By developing a scalable tool for researchers to collect labelled data and showcasing the feasibility of deploying efficient, simpler models on wearable devices, this project lays the groundwork for future advancements in HAR and personalised healthcare. These contributions bring us closer to leveraging technology to reduce NCD-related deaths.

Chapter 10

Future Work

10.1 Expanding App Configurability

The app currently allows a configurable list of activities to be added, which can be selected via a Microsoft Form to send the selected list of activities from an excel sheet to the app. The app functionality can be expanded by adding in more information to be stored and configured, such as the time window for data collection which is currently 10 seconds, however as shown by previous research various activities take various amounts of time to complete and thus to get the most accurate representation of an activity in a real world scenario, a configurable time window should be added. Additionally, information pertaining to user information such as User ID is also typed in to the app directly by the user, this information can also be added in via the Microsoft Form.

10.2 Improving Data Upload Delays

Uploading data to the online Excel sheet is not entirely a seamless process, due to the bottleneck created in the Power Automate flow used to process the incoming data, there are instances where data mixing can occur, this has been partially fixed by recording unique session IDs with each collection session, however a future work could to use advanced scripts to process the incoming accelerometer data such that the upload is near instantaneous.

10.3 Storing Data In Various Formats

Currently the data is only stored in an Excel format and while this allows the data to be easily readable and searched through the use of excel tools, future work should add in the option to configure which type of data format should be used to save the accelerometer data.

10.4 Improving Model Performance

Currently 18 features are engineered from raw accelerometer data to train basic models such as Decision Trees and Random Forest and although the results of these models are excellent, future work could incorporate more features to increase model performance. CNN based models could also be used by using more layers and fine tuning hyper parameters such as number of filters, batch size and number of epocs could increase model performance.

10.5 Incorporating Machine Learning In Power Automate Flow

Currently, the machine learning was down by manually downloading the accelerometer data from the online data base into a local python script. If this process in a future work can be added to a power automate flow to run the script autonomously whenever data is uploaded it will greatly enhance the research tool by providing report feedbacks to researchers on the trained models. An illustration of the what the overall process could look like is shown in Figure 10.1

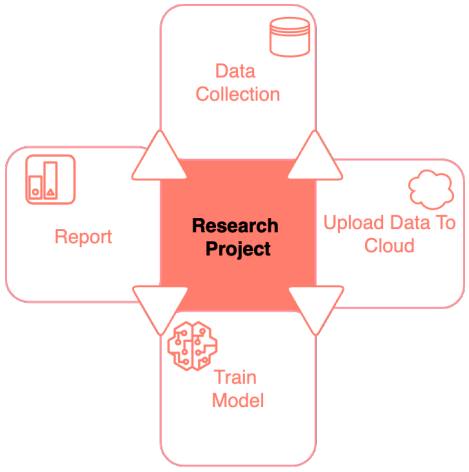

Figure 10.1: Potential flow of data collection app. The process shows how data could be collected from the app, sent to the online cloud database, where a stored model script is trained on the model and a report sent to the researchers on useful metrics such as model performance is given.

10.6 Machine Learning Libraries for Android Studio

In this project, the data collection app was built on Android Studio and the machine learning was done in Python. This was because no such current libraries similar to scikit-learn exist in Android Studio and Kotlin that allow ease of implementation of ML models. Building a library that allows the import of even simple models in Android studio would be a great leap towards building a personal informatics device to collect and present user data on their behaviours.

10.7 Using the App As A Fully Realised Personal Informatics Tool

The overall research goal is to produce an application to allow users to track their behaviour and provide user feedback in a manner that allows them to reflect on their habits and improve them. The ubiquity of the type of labelled data collected from the custom built app allows any number of activities to be tracked. This means that in theory the application can be used to monitor all activities that can be registered on sensors available in a smart watch. To produce a fully realised application to track such behaviour the ML models need to be trained in real - time on the smart watch device. Real time training would involve developing an app that initially for training purposes collects user data and trains itself by prompting the user when a certain activity such as “Drinking” is detected. Over time, the model will be able to predict the behaviour and provide useful metrics on habit tracking.

Appendix

A Feature Stat Calculation Class

```
import pandas as pd
import numpy as np
from scipy.signal import find_peaks
from scipy import stats

def average_accl(a_list, b_list, c_list):
    avg_result_accl = []
    for a, b, c in zip(a_list, b_list, c_list):
        sum_squares = np.array(a)**2 + np.array(b)**2 +
            np.array(c)**2
        sqrt = np.sqrt(sum_squares)
        average_accl = np.mean(sqrt)
        avg_result_accl.append(average_accl)
    return avg_result_accl

class FeatureStat:

    def __init__(self, df, x_list, y_list, z_list):
        self.df = df
        self.x_list = x_list
        self.y_list = y_list
        self.z_list = z_list

    def calculate_mean(self):
        self.df['x-mean'] = pd.Series(self.x_list).apply(np.mean)
        self.df['y-mean'] = pd.Series(self.y_list).apply(np.mean)
        self.df['z-mean'] = pd.Series(self.z_list).apply(np.mean)
        return self.df

    def standard_deviation(self):
        self.df['x-std'] = pd.Series(self.x_list).apply(np.std)
        self.df['y-std'] = pd.Series(self.y_list).apply(np.std)
        self.df['z-std'] = pd.Series(self.z_list).apply(np.std)
        return self.df

    def absolute_deviation(self):
        self.df['x-aab'] = pd.Series(self.x_list).apply(lambda x:
            np.mean(np.absolute(x - np.mean(x))))
        self.df['y-aab'] = pd.Series(self.y_list).apply(lambda y:
            np.mean(np.absolute(y - np.mean(y))))
```

```
38          self.df['z-aab'] = pd.Series(self.z_list).apply(lambda z:
                np.mean(np.absolute(z - np.mean(z))))
39          return self.df
40
41      def min(self):
42          self.df['x_min'] = pd.Series(self.x_list).apply(np.min)
43          self.df['y_min'] = pd.Series(self.y_list).apply(np.min)
44          self.df['z_min'] = pd.Series(self.z_list).apply(np.min)
45          return self.df
46
47      def median(self):
48          self.df['x_median'] =
                pd.Series(self.x_list).apply(np.median)
49          self.df['y_median'] =
                pd.Series(self.y_list).apply(np.median)
50          self.df['z_median'] =
                pd.Series(self.z_list).apply(np.median)
51          return self.df
52
53      def interquartile_range(self):
54          self.df['x_IQR'] = pd.Series(self.x_list).apply(lambda x:
                np.percentile(x, 75) - np.percentile(x, 25))
55          self.df['y_IQR'] = pd.Series(self.y_list).apply(lambda x:
                np.percentile(x, 75) - np.percentile(x, 25))
56          self.df['z_IQR'] = pd.Series(self.z_list).apply(lambda x:
                np.percentile(x, 75) - np.percentile(x, 25))
57          return self.df
58
59      def positive_count(self):
60          self.df['x_pos_count'] =
                pd.Series(self.x_list).apply(lambda x: np.sum(x > 0))
61          self.df['y_pos_count'] =
                pd.Series(self.y_list).apply(lambda x: np.sum(x > 0))
62          self.df['z_pos_count'] =
                pd.Series(self.z_list).apply(lambda x: np.sum(x > 0))
63          return self.df
64
65      def negative_count(self):
66          self.df['x_neg_count'] =
                pd.Series(self.x_list).apply(lambda x: np.sum(x < 0))
67          self.df['y_neg_count'] =
                pd.Series(self.y_list).apply(lambda x: np.sum(x < 0))
68          self.df['z_neg_count'] =
                pd.Series(self.z_list).apply(lambda x: np.sum(x < 0))
69          return self.df
70
71      def values_above_mean(self):
72          self.df['x-above-mean'] =
                pd.Series(self.x_list).apply(lambda x: np.sum(x >
                np.mean(x)))
73          self.df['y_above_mean'] =
                pd.Series(self.y_list).apply(lambda x: np.sum(x >
                np.mean(x)))
74          self.df['z_above_mean'] =
                pd.Series(self.z_list).apply(lambda x: np.sum(x >
                np.mean(x)))
75          return self.df
```

```
76
77      def peak(self):
78          self.df['x_peak_count'] =
                pd.Series(self.x_list).apply(lambda x:
                len(find_peaks(x)[0]))
79          self.df['y_peak_count'] =
                pd.Series(self.y_list).apply(lambda x:
                len(find_peaks(x)[0]))
80          self.df['z_peak_count'] =
                pd.Series(self.z_list).apply(lambda x:
                len(find_peaks(x)[0]))
81          return self.df
82
83      def skewness(self):
84          self.df['x_skewness'] =
                pd.Series(self.x_list).apply(lambda x: stats.skew(x))
85          self.df['y_skewness'] =
                pd.Series(self.y_list).apply(lambda x: stats.skew(x))
86          self.df['z_skewness'] =
                pd.Series(self.z_list).apply(lambda x: stats.skew(x))
87          return self.df
88
89      def kurtosis(self):
90          self.df['x_kurtosis'] =
                pd.Series(self.x_list).apply(lambda x:
                stats.kurtosis(x))
91          self.df['y_kurtosis'] =
                pd.Series(self.y_list).apply(lambda x:
                stats.kurtosis(x))
92          self.df['z_kurtosis'] =
                pd.Series(self.z_list).apply(lambda x:
                stats.kurtosis(x))
93          return self.df
94
95      def energy(self):
96          self.df['x_energy'] = pd.Series(self.x_list).apply(lambda
                x: np.sum(x**2)/100)
97          self.df['y_energy'] = pd.Series(self.y_list).apply(lambda
                x: np.sum(x**2)/100)
98          self.df['z_energy'] = pd.Series(self.z_list).apply(lambda
                x: np.sum(x**2)/100)
99          return self.df
100
101     def cal_average_accel(self):
102         self.df['average-accel'] = average_accl(self.x_list,
                self.y_list, self.z_list)
103         return self.df
104
105     def sma(self):
106         self.df['sma'] = pd.Series(self.x_list).apply(lambda x:
                np.sum(np.abs(x)/100)) + \
107                          pd.Series(self.y_list).apply(lambda y:
                             np.sum(np.abs(y)/100)) + \
108                          pd.Series(self.z_list).apply(lambda z:
                             np.sum(np.abs(z)/100))
109         return self.df
110     def capture_indicies(self):
```

```
111         self.df['x_argmax'] = pd.Series(self.x_list).apply(lambda
                x: np.argmax(x))
112         self.df['y_argmax'] = pd.Series(self.y_list).apply(lambda
                x: np.argmax(x))
113         self.df['z_argmax'] = pd.Series(self.z_list).apply(lambda
                x: np.argmax(x))
114
115         # index of min value in time domain
116         self.df['x_argmin'] = pd.Series(self.x_list).apply(lambda
                x: np.argmin(x))
117         self.df['y_argmin'] = pd.Series(self.y_list).apply(lambda
                x: np.argmin(x))
118         self.df['z_argmin'] = pd.Series(self.z_list).apply(lambda
                x: np.argmin(x))
119
120         # absolute difference between above indices
121         self.df['x_arg_diff'] = abs(self.df['x_argmax'] -
                self.df['x_argmin'])
122         self.df['y_arg_diff'] = abs(self.df['y_argmax'] -
                self.df['y_argmin'])
123         self.df['z_arg_diff'] = abs(self.df['z_argmax'] -
                self.df['z_argmin'])
124
125         # index of max value in frequency domain
126         self.df['x_argmax_fft'] =
                pd.Series(self.x_list).apply(lambda x:
                np.argmax(np.abs(np.fft.fft(x))[1:51]))
127         self.df['y_argmax_fft'] =
                pd.Series(self.y_list).apply(lambda x:
                np.argmax(np.abs(np.fft.fft(x))[1:51]))
128         self.df['z_argmax_fft'] =
                pd.Series(self.z_list).apply(lambda x:
                np.argmax(np.abs(np.fft.fft(x))[1:51]))
129
130         # index of min value in frequency domain
131         self.df['x_argmin_fft'] =
                pd.Series(self.x_list).apply(lambda x:
                np.argmin(np.abs(np.fft.fft(x))[1:51]))
132         self.df['y_argmin_fft'] =
                pd.Series(self.y_list).apply(lambda x:
                np.argmin(np.abs(np.fft.fft(x))[1:51]))
133         self.df['z_argmin_fft'] =
                pd.Series(self.z_list).apply(lambda x:
                np.argmin(np.abs(np.fft.fft(x))[1:51]))
134
135         # absolute difference between above indices
136         self.df['x_arg_diff_fft'] = abs(self.df['x_argmax_fft'] -
                self.df['x_argmin_fft'])
137         self.df['y_arg_diff_fft'] = abs(self.df['y_argmax_fft'] -
                self.df['y_argmin_fft'])
138         self.df['z_arg_diff_fft'] = abs(self.df['z_argmax_fft'] -
                self.df['z_argmin_fft'])
139
140
141
142     def all(self):
143         self.calculate_mean()
```

```
144         self.standard_deviation()
145         self.absolute_deviation()
146         self.min()
147         self.median()
148         self.interquartile_range()
149         self.positive_count()
150         self.negative_count()
151         self.values_above_mean()
152         self.peak()
153         self.skewness()
154         self.kurtosis()
155         self.energy()
156         self.cal_average_accel()
157         self.sma()
158         self.capture_indicies()
159         return self.df
```

B Power Automate Flow to Upload Accelerometer Data

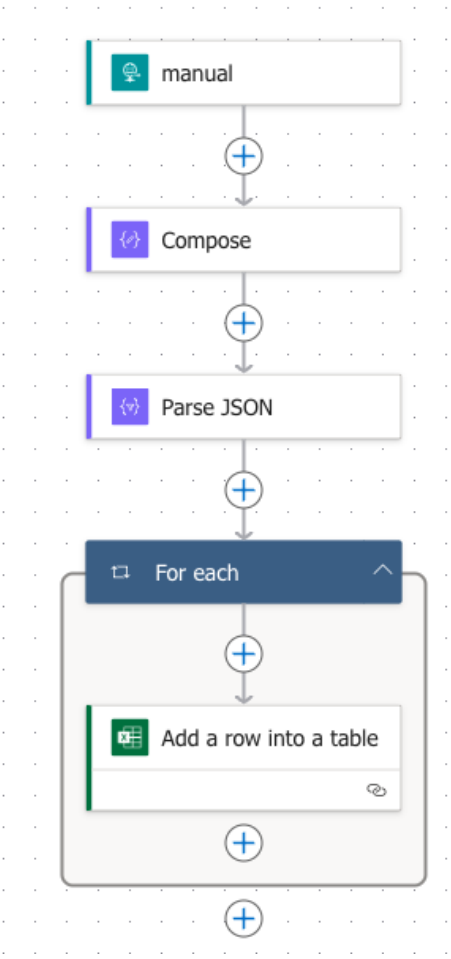

Figure 2: Power Automate Flow to Upload Accelerometer Data

C Power Automate Flow to retrieve activities list from excel when a Microsoft Form is submitted

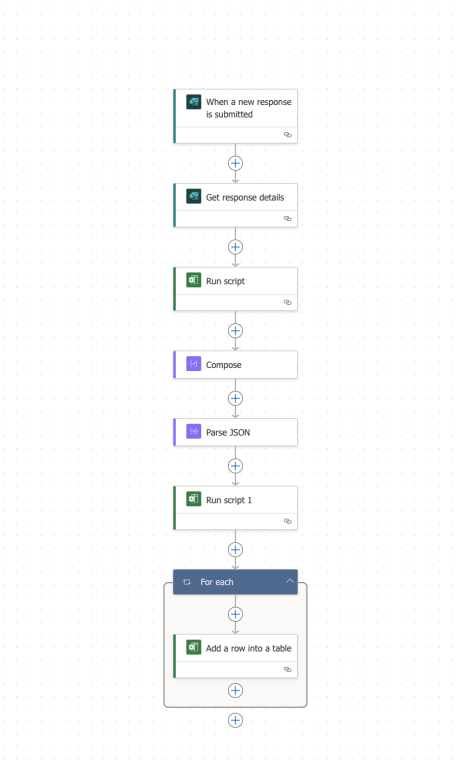

Figure 3: Power Automate Flow showing a Trigger (When a new form response is submitted), followed by an Action, to get response details from the form. This causes a custom Excel script to run which formats the list of activities in a string array from a "Activities List.xls" file. This is followed by a Data Processing action (Compose and ParseJSON) to process the string array containing activities into a JSON format, which allows the activities to be added to a second Excel file called "Processed Answers.xls" containing only one excel sheet with the activities list chosen from the form response

D Power Automate Flow to send selected Activities List from One Drive as a HTTP response to smart watch

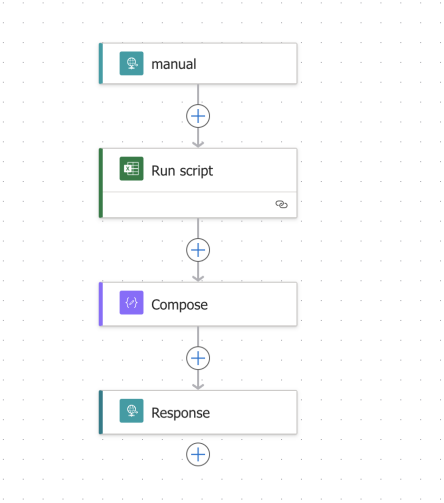

Figure 4: Power Automate Flow showing a Trigger (when a HTTP POST request is received), which triggers an action to execute an Excel Script which retrieves the selected activities from "Processed Answers.xls" file. This allows the data to be processed into a JSON format and sent as a the response body back to the smart watch

Bibliography

[1] Ong Chin Ann and Lau Bee Theng. "Human activity recognition: A review". In: *2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014)*. Penang, Malaysia: IEEE, Nov. 2014, pp. 389–393. ISBN: 978-1-4799-5686-9. DOI: 10.1109/ICCSCE.2014.7072750. URL: http://ieeexplore.ieee.org/document/7072750/ (visited on 10/27/2024).

[2] Xiaokang Zhou et al. "Deep-Learning-Enhanced Human Activity Recognition for Internet of Healthcare Things". In: *IEEE Internet of Things Journal* 7.7 (July 2020), pp. 6429–6438. ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2020.2985082. URL: https://ieeexplore.ieee.org/document/9055403/ (visited on 11/30/2024).

[3] Neha Gupta et al. "Human activity recognition in artificial intelligence framework: a narrative review". en. In: *Artificial Intelligence Review* 55.6 (Aug. 2022), pp. 4755–4808. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-021-10116-x. URL: https://link.springer.com/10.1007/s10462-021-10116-x (visited on 11/30/2024).

[4] *World Health Organization. Global status report on noncommunicable diseases 2023*. 2023. URL: https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases.

[5] Sangwon Bae et al. "Detecting Drinking Episodes in Young Adults Using Smartphone-based Sensors". en. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1.2 (June 2017), pp. 1–36. ISSN: 2474-9567. DOI: 10.1145/3090051. URL: https://dl.acm.org/doi/10.1145/3090051 (visited on 10/12/2024).

[6] Clare Strongman et al. "A Scoping Review of the Validity and Reliability of Smartphone Accelerometers When Collecting Kinematic Gait Data". en. In: *Sensors* 23.20 (Oct. 2023), p. 8615. ISSN: 1424-8220. DOI: 10.3390/s23208615. URL: https://www.mdpi.com/1424-8220/23/20/8615 (visited on 10/28/2024).

[7] Yuliia Kniazieva. *How to Label Data for Machine Learning Projects?* English. Jan. 2024. URL: https://labelyourdata.com/articles/label-data-for-machine-learning#:~:text=Most%20ML%20models%20use%

20supervised,about%20obstacles%20on%20the%20road. (visited on 12/02/2024).

[8] In: ().

[9] Hongwei Yin, Richard O. Sinnott, and Glenn T. Jayaputera. "A survey of video-based human action recognition in team sports". en. In: *Artificial Intelligence Review* 57.11 (Sept. 2024), p. 293. ISSN: 1573-7462. DOI: 10.1007/s10462-024-10934-9. URL: https://link.springer.com/10.1007/s10462-024-10934-9 (visited on 12/03/2024).

[10] Karen Simonyan and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos." In: *Advances in Neural Information Processing Systems* 27.Curran Associates (2014).

[11] Adrián Sánchez-Caballero, David Fuentes-Jiménez, and Cristina Losada-Gutiérrez. "Real-time human action recognition using raw depth video-based recurrent neural networks". en. In: *Multimedia Tools and Applications* 82.11 (May 2023), pp. 16213–16235. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-022-14075-5. URL: https://link.springer.com/10.1007/s11042-022-14075-5 (visited on 12/03/2024).

[12] Inyong Koo et al. "Contrastive Accelerometer–Gyroscope Embedding Model for Human Activity Recognition". In: *IEEE Sensors Journal* 23.1 (Jan. 2023), pp. 506–513. ISSN: 1530-437X, 1558-1748, 2379-9153. DOI: 10.1109/JSEN.2022.3222825. URL: https://ieeexplore.ieee.org/document/9961198/ (visited on 12/03/2024).

[13] Negar Golestani and Mahta Moghaddam. "Human activity recognition using magnetic induction-based motion signals and deep recurrent neural networks". en. In: *Nature Communications* 11.1 (Mar. 2020), p. 1551. ISSN: 2041-1723. DOI: 10.1038/s41467-020-15086-2. URL: https://www.nature.com/articles/s41467-020-15086-2 (visited on 12/03/2024).

[14] Gary Weiss. *WISDM Smartphone and Smartwatch Activity and Biometrics Dataset*. 2019. DOI: 10.24432/C5HK59. URL: https://archive.ics.uci.edu/dataset/507 (visited on 12/01/2024).

[15] Abeer Mostafa et al. "Multi-sensor Gait Analysis for Gender Recognition:" in: *Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics*. Lieusaint - Paris, France: SCITEPRESS - Science and Technology Publications, 2020, pp. 629–636. ISBN: 9789897584428. DOI: 10.5220/0009792006290636. URL: https://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0009792006290636 (visited on 12/03/2024).

[16] Tarun R. Katapally and Nazeem Muhajarine. "Towards uniform accelerometry analysis: a standardization methodology to minimize measurement bias due to systematic accelerometer wear-time variation". eng. In: *Journal of Sports Science & Medicine* 13.2 (May 2014), pp. 379–386. ISSN: 1303-2968.

[17] Strahinja Zivkovic. *Machine Learning – Decision three*. URL: https://datahacker.rs/011-machine-learning-decision-three/ (visited on 04/12/2024).

[18] James Chen. *What Is a Neural Network?* July 2024. URL: https://www.investopedia.com/terms/n/neuralnetwork.asp (visited on 12/04/2024).

[19] Song-Mi Lee, Sang Min Yoon, and Heeryon Cho. "Human activity recognition from accelerometer data using Convolutional Neural Network". In: *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*. Jeju Island, South Korea: IEEE, Feb. 2017, pp. 131–134. ISBN: 9781509030156. DOI: 10.1109/BIGCOMP.2017.7881728. URL: http://ieeexplore.ieee.org/document/7881728/ (visited on 12/04/2024).

[20] Kun Wang, Jun He, and Lei Zhang. "Attention-based Convolutional Neural Network for Weakly Labeled Human Activities Recognition with Wearable Sensors". In: (2019). DOI: 10.48550/ARXIV.1903.10909. URL: https://arxiv.org/abs/1903.10909 (visited on 12/04/2024).

[21] Gary Weiss and Jeffrey Lockhart. "The Impact of Personalization on Smartphone-Based Activity Recognition". In: *AAAI Workshop - Technical Report* (Jan. 2012).

[22] CS Xie Xie and Yu Zhou Zhou. *SZU HAD Basketball*. DOI: 10.21227/TVVV-0F08. URL: https://ieee-dataport.org/documents/szu-had-basketball-0 (visited on 12/04/2024).

[23] Mst Alema Khatun Khatun. *H-Activity*. DOI: 10.21227/QCFW-K581. URL: https://ieee-dataport.org/documents/h-activity (visited on 12/04/2024).

[24] Davide Anguita Jorge Reyes-Ortiz. *Human Activity Recognition Using Smartphones*. 2013. DOI: 10.24432/C54S4K. URL: https://archive.ics.uci.edu/dataset/240 (visited on 12/04/2024).

[25] Abdullah-Al Nahid. *KU-HAR: An Open Dataset for Human Activity Recognition*. Feb. 2021. DOI: 10.17632/45F952Y38R.5. URL: https://data.mendeley.com/datasets/45f952y38r/5 (visited on 12/04/2024).

[26] Margreet Riphagen et al. "LEARNING TOMORROW: VISUALISING STUDENT AND STAFF'S DAILY ACTIVITIES AND REFLECT ON IT". In: Sevilla, Spain, 2013.

[27] Shian-Ru Ke et al. "A Review on Video-Based Human Activity Recognition". en. In: *Computers* 2.2 (June 2013), pp. 88–131. ISSN: 2073-431X. DOI: 10.3390/computers2020088. URL: https://www.mdpi.com/2073-431X/2/2/88 (visited on 10/11/2024).

[28] Neil Robertson and Ian Reid. "A general method for human activity recognition in video". en. In: *Computer Vision and Image Understanding* 104.2-3 (Nov. 2006), pp. 232–248. ISSN: 10773142. DOI: 10.1016/j.cviu.2006.07.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S107731420600110X (visited on 10/11/2024).

[29] Keogh Eamonn J and Abdullah Mueen. “Curse of Dimensionality”. In: *Encyclopedia of machine learning and data mining 2017* (2017), pp. 314–315.

[30] I. C. Gyllensten and A. G. Bonomi. “Identifying Types of Physical Activity With a Single Accelerometer: Evaluating Laboratory-trained Algorithms in Daily Life”. In: *IEEE Transactions on Biomedical Engineering* 58.9 (Sept. 2011), pp. 2656–2663. ISSN: 0018-9294, 1558-2531. DOI: 10.1109/TBME.2011.2160723. URL: http://ieeexplore.ieee.org/document/5934365/ (visited on 10/11/2024).

[31] Pegah Esfahani and Hadi Tabatabaee Malazi. “PAMS: A new position-aware multi-sensor dataset for human activity recognition using smartphones”. In: *2017 19th International Symposium on Computer Architecture and Digital Systems (CADS)*. Kish Island: IEEE, Dec. 2017, pp. 1–7. ISBN: 978-1-5386-4379-2. DOI: 10.1109/CADS.2017.8310680. URL: http://ieeexplore.ieee.org/document/8310680/ (visited on 10/11/2024).

[32] Masud Ahmed, Anindya Das Antar, and Md. Atiqur Rahman Ahad. “An Approach to Classify Human Activities in Real-time from Smartphone Sensor Data”. In: *2019 Joint 8th International Conference on Informatics, Electronics & Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. Spokane, WA, USA: IEEE, May 2019, pp. 140–145. ISBN: 978-1-72810-786-8 978-1-72810-788-2. DOI: 10.1109/ICIEV.2019.8858582. URL: https://ieeexplore.ieee.org/document/8858582/ (visited on 10/12/2024).

[33] Rubén San-Segundo et al. “Robust Human Activity Recognition using smartwatches and smartphones”. en. In: *Engineering Applications of Artificial Intelligence* 72 (June 2018), pp. 190–202. ISSN: 09521976. DOI: 10.1016/j.engappai.2018.04.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197618300812 (visited on 10/12/2024).

[34] Philipp M. Scholl and Kristof 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*. Palermo, Italy: IEEE, July 2012, pp. 886–891. ISBN: 978-1-4673-1328-5 978-0-7695-4684-1. DOI: 10.1109/IMIS.2012.96. URL: http://ieeexplore.ieee.org/document/6296971/ (visited on 10/12/2024).

[35] Casey A. Cole et al. “Detecting Smoking Events Using Accelerometer Data Collected Via Smartwatch Technology: Validation Study”. eng. In: *JMIR mHealth and uHealth* 5.12 (Dec. 2017), e189. ISSN: 2291-5222. DOI: 10.2196/mhealth.9035.

[36] Stephen Cass. *Top Programming Languages 2024*. English. URL: https://spectrum.ieee.org/top-programming-languages-2024.

[37] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. "Activity recognition using cell phone accelerometers". en. In: *ACM SIGKDD Explorations Newsletter* 12.2 (Mar. 2011), pp. 74–82. ISSN: 1931-0145, 1931-0153. DOI: 10.1145/1964897.1964918. URL: https://dl.acm.org/doi/10.1145/1964897.1964918 (visited on 12/01/2024).

[38] Fang Wang. *Exploring The Effectiveness Of Different Machine learning Models For Smart-watch Based Drinking Detection*. Tech. rep. SCHOOL OF COMPUTER SCIENCE: University of Bristol, 2024.

[39] Pratik Nabriya. *Feature Engineering on Time-Series Data for Human Activity Recognition Transforming raw signal d*. English. June 2021. URL: https://towardsdatascience.com/feature-engineering-on-time-series-data-transforming-signal-data-of-a-smartphone-accelerometer-for-72cbe34b8a60 (visited on 10/28/2024).

[40] Muhammad Rehan Azhar. *Training CNN on Signal Data for Human Activity Recognition*. URL: https://medium.com/@rehanmbl/training-cnn-on-signal-data-for-human-activity-recognition-ca1eaddeac8f (visited on 07/12/2024).

[41] Tajammul Pangarkar. *Smartwatch Statistics 2024 By Wearables, Technology, Devices*. URL: https://scoop.market.us/smartwatch-statistics/#:~:text=The%20global%20smartwatch%20adoption%20rate,smartphone%20owners%20now%20use%20smartwatches..