Course: ELEC 292
Project Report

Group Number: 20

Names, Student Numbers, and Email Addresses:

XXXX,

XXXX,

Julien Chagnon, 20390465, 22pclt@queensu.ca

Date:

April 3rd, 2025

Table of Contents

Introduction 1

Data Collection 1

- Data collection protocol 1
- Data collection tools 1
- Challenges and Solutions 2
- Initial Data Organization 2

Data Storage 2

- Storage Requirements and Approach 3
- HDF5 File Structure Implementation 3
- Implementation Details 4
- Storing Preprocessed and Segmented Data 4
- Challenges and Solutions 4
- Advantages of Our Storage Approach 5

Visualization 5

- Accelerometer Data Visualization 5
- Raw vs. Smoothed Signal Comparison 6
- Gathered Understandings 7
- Future Improvements 7

Preprocessing 8

- Preprocessing Approach 8
- Forward Fill for Missing Values 8
- Moving Average Smoothing 9
- Parameter Selection Rationale 9
- Impact on Data Quality and Classification 9

Feature Extraction & Normalization 10

- Feature Extraction Approach 10
- Selected Features 11
- Feature Extraction Implementation 11
- Feature Normalization 12
- Justification of Feature Choices 12

Training and Testing the Classifier 13
Model Deployment 15

List of Figures

Figure 1: example of data collected within Phyfox 2
Figure 2: Hierarchical storage of HDF5 file 3
Figure 3: Graphical representation of walking and jumping samples for each team member 6
Figure 4: Graphical representation of raw and smoothed Z-axis acceleration 7
Figure 5: Confusion matrix table from data sample 14
Figure 6: Learning curve plotting model accuracy in regard to number of training samples 14
Figure 7: Graphical User Interface (GUI) of model highlighting walking and jumping segments 16

Introduction

The following report details the process of developing and implementing a machine learning model that accurately tracks the movement of its user. Specifically, when passed accelerometer data to read and interpret, the machine learning system distinguishes between two common movements – walking and jumping.

This project allowed for further understanding of data collection and processing, as well as allowed for an opportunity to apply the knowledge gained in ELEC 292 to a real-world implementation. Tracking the movement of humans can be integral for the study of health, fitness, and overall human understanding. Our work includes accurate collection and processing methods and demonstrates the ability to work with and interpret data from its raw form.

The project follows a structured methodology comprising seven key stages: data collection, storage, visualization, pre-processing, feature extraction, classifier training, and the development of a desktop interface using TKinter. Each stage builds upon the previous one, resulting in a system able to process new accelerometer data and provide real-time activity classifications per time segmented section of collected data.

By building a logistic regression model with the engineering techniques gathered in this course, group 20 has seen very high activity classification accuracy throughout the testing, while still considering code efficiency and simplicity.

To follow, a detailed description of each of the key components of the project will be discussed for further implementation understanding.

Data Collection

In this initial phase of our project, we collected accelerometer data from smartphones while performing two distinct physical activities: walking and jumping. This raw sensor data formed the foundation for our activity classification system.

Data collection protocol

Following the project requirements, each team member collected data to create a diverse compilation of collected data. This approach ensured that different walking gaits and jumping styles were accounted for in the sampling. Each team member collected three 50-second-long samples of each action, reorienting the smartphone to account for potential inconsistencies. These reorientations included front pocket, back pocket, and held in hand data collection. This ensured that the dataset was robust enough to accurately predict the action taken by the program's user. An equal number of samples were collected to evenly distribute the machine learning data and allow for a lesser bias towards one of the actions.

Data collection tools

The app selected for data collection during this process was Phyfox, seen below in Figure 1. It was easily installable on both apple and android devices, and included features such as the timed run, the start/stop noise control settings, and the ability for easy CSV exportation, which allowed for easy data collection and datasets that lacked unexpected noise. This was integral to the process, as noisy data lends to the "garbage in, garbage out" mantra valued throughout the course. Beginning with high quality data ensures quality results.

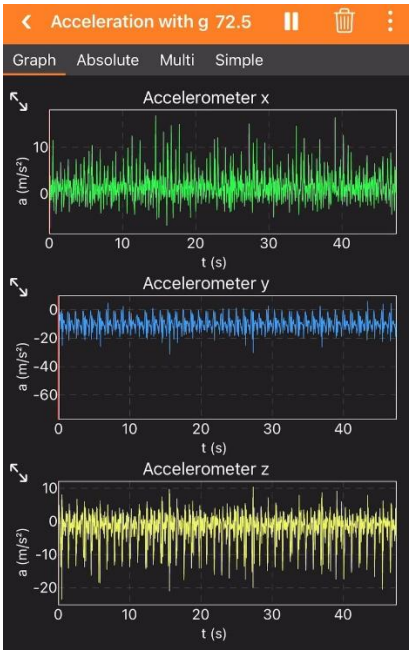

Figure 1: example of data collected within Phyfox

Challenges and Solutions

One of the main challenges seen throughout the collection process was limiting noise in the data. We did so by taking necessary precautions while collecting data: namely, we made sure to use a preset recording time to eliminate unpredictable start and stop movement. Moreover, we made use of the in-app noise filters provided. This allowed for cleaner raw data and sample consistency.

After examining the collected data, team members noticed certain obvious differences in the patterns formed. One prominent observation was the effect of the orientation of the phone collecting data. Juliens phone was oriented upwards in his pocket, whereas XXXX's phone lay sideways, impacting the axes changes seen by the accelerometer. To account for the lack of axis consistency, the program allows each axis to contribute equally to the dataset, as well as uses the absolute acceleration as a set of values useful in recognizing data patterns. The absolute acceleration is orientation invariant – it captures the overall intensity of the movement observed.

Initial Data Organization

After collection, we organized our CSV files using a systematic naming convention:

- [name]_walking_[trial_number].csv
- [name]_jumping_[trial_number].csv

As shown in our code, we loaded these files individually and labeled them with the corresponding activity and subject.

Data Storage

After the data collection process was complete, the next step was to find an effective data storage solution. This phase of the project involved organizing the multi-dimensional, axis-oriented data in a structured manner to facilitate the data processing, analysis, and model development to follow. Proper data storage is integral to the success of the project due to its immediate effects on the integrity,

accessibility, and proper management of the data. In the sections to follow, the process of deliberation, implementation, and decision-making are detailed to display the conscious choices required to come to a valuable solution.

Storage Requirements and Approach

For this project, it was required to find an efficient method for hierarchically organizing the data seen in multiple different accelerometer datasets. Due to these requirements, the HDF5 file type was ideal for the project organization, allowing for the storage of large, heterogeneous datasets in a structured manner. Some advantages of the HDF5 filetype include:

- Support for complex data hierarchies
- Efficient storage of numerical and string data
- Advanced data slicing capabilities for selective data access
- Ability to store metadata alongside datasets
- Cross-platform compatibility
- High-performance access to stored data

The capacity to data slice was particularly useful in the implementation of the project – easy access to subsets of data is integral for the data processing, and not having to load entire datasets into memory made access to data extremely simple.

HDF5 File Structure Implementation

Depicted below in Figure 2 is a hierarchical storage structure that organizes our data into logical groups. The storage hierarchy follows a clear organization pattern as shown in the (figure X) below:

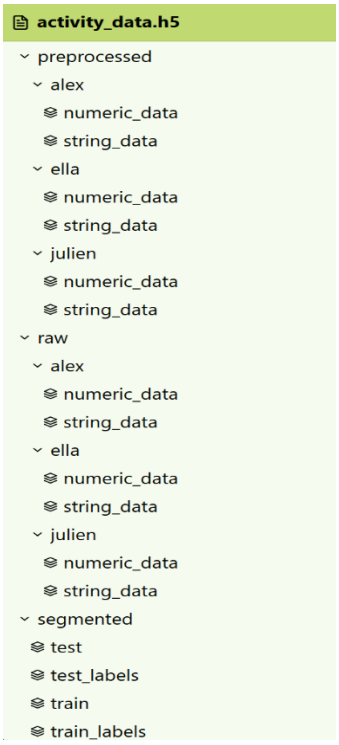

Figure 2: Hierarchical storage of HDF5 file

As is evidenced in the figure, when the created HDF5 file for this project is organized hierarchically, including raw, preprocessed, and segmented data easily defined in each of their categories.

Implementation Details

We implemented the data storage in our code using the h5py Python library. First, we created the HDF5 file and established the raw data group:

```
#Open HDF5 file and create raw group
hdf5_file = h5py.File("activity_data.h5", "w")
raw_group = hdf5_file.create_group("raw")
```

For each subject (Alex, Julien, and XXXX), the data was separated into numeric and string types using a helper function in order to maximize efficiency:

```
#Function to store subject data into HDF5
def store_subject_data(h5_group, subject_name, data):
    numeric = data.select_dtypes(include=[np.number])
    strings = data.select_dtypes(include=["object"]).astype("S")
    subj_group = h5_group.create_group(subject_name)
    subj_group.create_dataset("numeric_data", data=numeric.values)
    subj_group.create_dataset("string_data", data=strings.values)
    subj_group.attrs["numeric_columns"] = list(numeric.columns)
    subj_group.attrs["string_columns"] = list(strings.columns)
```

This approach provided the following positive attributes:

1. Separated numerical and string data for optimal storage efficiency
2. Preserved column names as metadata attributes
3. Created a clean hierarchy that mirrors our experimental design

Storing Preprocessed and Segmented Data

Later in the workflow, preprocessed data and segmented training/testing data was added to the HDF5 file:

```
with h5py.File("activity_data.h5", "a") as h5f:
    #Storing preprocessed data per subject
    preprocessed_group = h5f.require_group("preprocessed")
    for subject in preprocessed_data["Subject"].unique():
        subject_data = preprocessed_data[preprocessed_data["Subject"] == subject]
        numeric = subject_data.select_dtypes(include=[np.number])
        strings = subject_data.select_dtypes(include=["object"]).astype("S")
```

While this code is featured much later in the data management process, it is important to understand this piece of the code so that one may comprehend the complete data pipeline stored in a singular HDF5 file.

Challenges and Solutions

One of the primary difficulties with working with the collected data and storing it was storing diverse data types. More specifically: Pandas data frames frequently contain mixed types, and HDF5 has particular requirements for data types. The solution reached was to store column name information as metadata attributes after separating string and numeric data into distinct datasets. This approach allowed us to reconstruct the original data frames when needed.

Another challenge faced was maintaining data integrity throughout project phases. At every stage of the project, we had to guarantee reliable access to the data. To do so, it was important to maintain the connections between the subjects, activities, and preprocessing steps that create the hierarchical structure. This method preserved overall data integrity while enabling each part of our system to access the relevant data.

Advantages of Our Storage Approach

Several significant benefits were offered to the project by our HDF5 implementation:

- Data Organization: Navigating and accessing particular subsets of data was made simple by the hierarchical structure.
- Efficiency: Improved access speed and storage capacity was gained through separating string and numeric data.
- Integrity: Our preprocessing and machine learning pipelines were able to dependably access the data they required thanks to the consistent structure.
- Extensibility: As our project developed, we were able to incorporate new groups and datasets using the HDF5 format.

Ultimately, the project itself was very successful due to our carefully considered data storage structure. This method of data storage ensured dependable access to our data throughout the development process and served as a strong basis for the following stages of our activity classification system.

Visualization

An important part of the project was visualizing the accelerometer data, which provided insights that shaped the preprocessing, feature extraction, and classification strategies. by converting raw numerical data into graphical representations, it was possible to Assess the efficacy of the preprocessing techniques, confirm the accuracy of the classification results, and gain a better understanding of the distinctive features of walking and jumping. The main visualizations produced during the project are displayed below, along with the lessons learned throughout the project and how they will affect future data collection initiatives.

Accelerometer Data Visualization

Figure 3 seen below depicts the raw accelerometer data from each participant, displaying the graphs for both the walking and jumping activities. A multi-panel graph was plotted to compare the patterns in the acceleration for each subject in each activity.

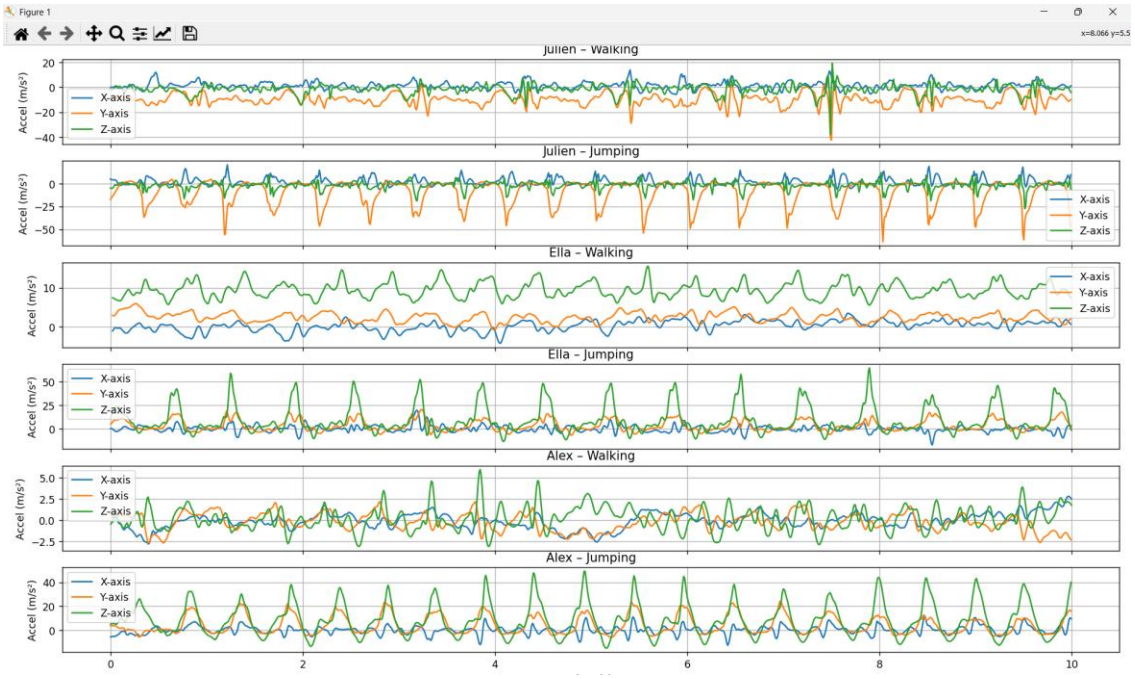

Figure 3: Graphical representation of walking and jumping samples for each team member

This visual representation of the collected data easily allows for the deduction of important data characteristics. During visualization of the accelerometer data, we observed a distinct trend in the axis that exhibited the largest acceleration spikes during jumping activities. For Julien, the most prominent spikes occurred along the Z-axis, whereas for XXXX, the spikes were more significant along the Y-axis. This discrepancy is attributed to differences in the orientation of the smartphone during data collection. Since accelerometers measure acceleration relative to the phone's internal coordinate system, changes in device placement—such as being flat in a pocket versus upright in a jacket—can shift which axis aligns with vertical motion. This variability is not only expected but also desirable, as it introduces real-world diversity into the dataset. Such diversity ensures that the classifier can generalize better for varying user behaviors and phone placements.

Some relevant observations concerning the patterns seen throughout the data were made to better understand the key differences between the two plots. Namely, walking produces regular, periodic oscillations with consistent amplitude; however, Jumping creates distinct, high-magnitude spikes corresponding to takeoffs and landings. These two observations allow for immediate differences to be drawn between the two plots, and understandings to be reached about which features are most relevant to the information held in each dataset.

Raw vs. Smoothed Signal Comparison

Figure 4 compares the raw and smoothed Z-axis acceleration data, showing the effectiveness of the preprocessing approach for the first 300 samples of the "Julien" data as an example.

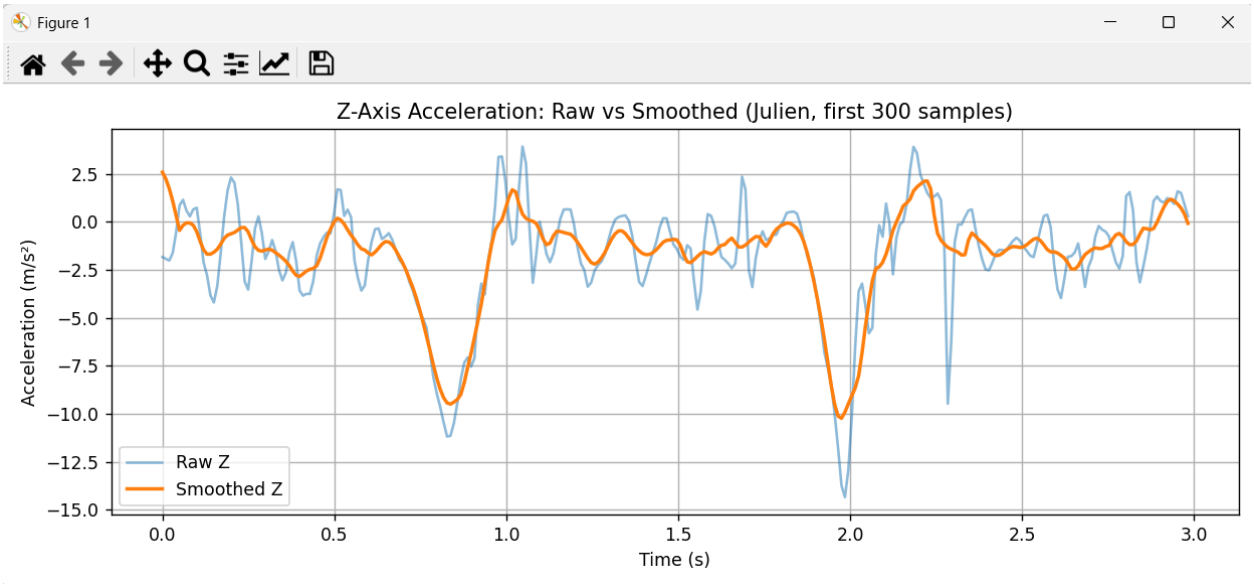

Figure 4: Graphical representation of raw and smoothed Z-axis acceleration

This visualization shows that the moving average filter effectively reduces random noise fluctuations, as well as that the fundamental patterns of the activities are preserved after smoothing. Signal peaks remain visible but are less jagged after preprocessing as well, and the smoothed signal provides a clearer representation of the underlying motion.

Gathered Understandings

Through the visualization process, several important insights were gained. The three axes (X, Y, Z) show different patterns depending on phone orientation, but the activity signatures stay recognizable despite that. However, using absolute acceleration in the training model allowed the data to evolve past axis dependent patterns, and begins to highlight the importance of varied data collection. Moreover, different participants showed varying acceleration patterns for the same activities, noting further the integral piece diverse training data plays in developing a strong prediction capability. Overall, visualizing the data helped identify which signal characteristics (periodicity, peak amplitude, etc.) were most useful for distinguishing between activities, and allowed for a well-trained model.

Future Improvements

In light of the insights gained from these visualizations, we would alter our data collection process in the following ways if we were to do it over:

- Standardized Phone Placement: To lessen participant variance, we would create a more uniform phone orientation protocol. Standardization would make the feature extraction process simpler, even though our current method is effective.
- Higher Sampling Rate: Since some of the jumping motions occur very quickly, a higher sampling rate would more accurately capture these quick movements.
- Extra Activities: The distinctions between walking and jumping were evident in our visualization. A more difficult and thorough dataset would be produced by including more subtle activity distinctions (jogging vs. running, for example).
- Controlled Environment: To lessen outside influences that could add noise to the data, we would gather it in a more controlled setting.

- Synchronized video recording: Adding Synchronized video recording during data collection would make it easier to link movements to matching accelerometer patterns.

These visual aids were essential for comprehending our data and creating a successful classification strategy. They helped identify the most pertinent characteristics for differentiating between walking and jumping activities and offered transparency into the efficacy of preprocessing.

Preprocessing

Preprocessing the accelerometer data is essential to ensuring the machine learning model receives clean, consistent input. In this section, the preprocessing methods are described, as well as their impact on the data, and the rationale behind the parameter choices.

Preprocessing Approach

Our raw accelerometer data presented two primary challenges that required preprocessing:

1. Occasional missing values in the time series
2. Noise that obscured the activity patterns

A two-step processing pipeline was implemented to address these issues:

```
#STEP 1 - Fill missing values using forward filling
preprocessed_data = combined_data.copy()
preprocessed_data.ffill(inplace=True)

# STEP 2 - Moving average filter
cols_to_smooth = [
    "Acceleration x (m/s^2)",
    "Acceleration y (m/s^2)",
    "Acceleration z (m/s^2)",
    "Absolute acceleration (m/s^2)"
]
for c in cols_to_smooth:
    if c in preprocessed_data.columns:
        preprocessed_data[c] = preprocessed_data[c].rolling(window=10, center=True,
min_periods=1).mean()
```

This preprocessing method first makes sure that the data is continuous through a forward filling approach. It then reduces the noise in the dataset through signal smoothing. By implementing the steps in this order, it is ensured that the dataset is continuous before applying the moving average filter – without this assurance, the code would not function.

Forward Fill for Missing Values

The first preprocessing step addressed the missing values in the dataset due to the inaccuracies within the accelerometer readings. Instead of discarding the rows that did not contain values, a forward fill method was implemented to maintain the time-dependent structure of the data.

The forward-fill technique takes the last valid observation and propagates it forward to fill any gaps in the dataset, maintaining the necessary continuousness of the time series data. Although the interpolation method is often seen as more accurate for value estimates, the forward fill method was ideal for this application. Initially, the forward fill method is ideal for its simple implementation as well as its memory

efficiency – the desktop app is required to process data in near-real time; more complex methods may slow down the program's internal processes. Moreover, in the context of physical movements, the transitions created by interpolated data may not be plausible due to their artificial nature. The forward fill method ensures that possible data transitions are seen at all points, and that the classifier isn't potentially confused by abstract acceleration plots.

Moving Average Smoothing

The second preprocessing step applied a centered moving average filter with a window size of 10 samples to reduce noise:

The moving average filter works by replacing each data point with the average of itself and neighboring points within the specified window. It was the obvious approach for this project for a few integral reasons. The moving average is intuitive to understand and implement, making the preprocessing pipeline maintainable. It also effectively removes high-frequency noise while preserving lower-frequency components that characterize human activities. Because this project is built for training a model that detects obvious, patterned movements, the moving average filter was ideal.

Unlike some filters that can reduce signal amplitude, the moving average preserves the overall magnitude of the acceleration signals, which is very important for finding the differences between walking and jumping. By using a centered window (center=True), phase shifts being introduced are avoided so that peaks and troughs in the smoothed data align timing-wise with the original signal.

The visualization seen in section X in the raw versus smooth Z-axis data plot shows that the smoothing approach reduces the effect of random data discrepancies while making sure the fundamental patterns of each action are still maintaining their integrity. The oscillations of walking and the obvious spikes of jumping stay visible, but with significantly less noise.

Parameter Selection Rationale

The moving average filter's window size of 10 was chosen because it strikes a balance between reducing random fluctuations and maintaining significant activity features. Choosing a window size depends on a variety of factors related to the data collection process and is highly specific to the data's implementation. The accelerometer readings were taken at around 100 Hz, according to the time column of the data. Therefore, a 10-sample window corresponds to roughly 0.1 seconds of activity, which is both long enough to average out noise and short enough to preserve significant motion details.

To prevent phase shifts in the signal and guarantee that the peaks and troughs in the smoothed data correctly match the original signal, the centered window approach (center=True) was selected. To make sure the filter would function even at the edges of our data, where a full window might not be available, we also set min_periods=1.

Impact on Data Quality and Classification

The classification system's performance and visual inspection both demonstrate the effects of the preprocessing steps. Specifically, the preprocessing successfully decreased random fluctuations and maintained the patterns observed throughout each activity, as demonstrated in the visualization comparing raw and smoothed Z-axis data. For walking, the preprocessed signal exhibits more distinct

periodicity, and for jumping, it displays more distinct peaks. Furthermore, the forward fill operation made sure that there were no gaps in the time series, supplying continuous data for the feature extraction procedure that followed. Most significantly, the feature extraction procedure, which computed statistical measures like mean, standard deviation, and range, benefited from the cleaner signals. When these features are computed from smoothed signals instead of raw data, they are more indicative of the tasks being carried out. Ultimately, the preprocessing strategy simulated clean, continuous data that supported the next steps in our classification pipeline.

Feature Extraction & Normalization

The next crucial step in the process was to convert the continuous time series accelerometer data into useful features that could be applied to classification. Raw sensor data was transformed into a collection of descriptive statistics through feature extraction, this transformed data allowing for identification of the unique patterns of walking and jumping. This section describes how features were extracted, which features the team chose, and these features were normalized to achieve the best classification results.

Feature Extraction Approach

After preprocessing the accelerometer data, the next step was extracting meaningful features that could reveal the difference between the walking and jumping activity plots. We extracted statistical features from identical time windows to capture the characteristic patterns of each activity.

The feature extraction process followed these steps:

1. Segmenting the preprocessed data into 5-second windows
2. Calculating multiple statistical features for each window
3. Organizing these features into a structured format for the classifier

It is to note that an issue that arose when examining the data collection process was the differences in the frequencies used by the in-smartphone accelerometer. The discrepancies in this frequency meant that the samples taken per 5 second slice of each piece of collected data had different sampling rates per unit time. To take this issue into consideration the implemented code calculates the number of samples per unit time, then calculates how many samples are present per second, and finally multiplies it by 5, ensuring that each spliced instance of data contains exactly 5 seconds of content.

The following code implements our feature extraction approach:

```
time_diffs = data["Time (s)"].diff().dropna()
sampling_rate = 1 / time_diffs.median()
samples_per_window = int(5 * sampling_rate)

feature_cols = [
    "Acceleration x (m/s^2)",
    "Acceleration y (m/s^2)",
    "Acceleration z (m/s^2)",
    "Absolute acceleration (m/s^2)"
]
```

```
def extract_features(window):
    feats = {}
    for col in feature_cols:
        if col not in window.columns:
            continue
        col_data = window[col]
        feats[f"{col}_mean"] = col_data.mean()
        feats[f"{col}_std"] = col_data.std()
        feats[f"{col}_min"] = col_data.min()
        feats[f"{col}_max"] = col_data.max()
        feats[f"{col}_skew"] = skew(col_data)
        feats[f"{col}_kurtosis"] = kurtosis(col_data)
        feats[f"{col}_range"] = col_data.max() - col_data.min()
        feats[f"{col}_median"] = col_data.median()
        feats[f"{col}_var"] = col_data.var()
        feats[f"{col}_mad"] = np.mean(np.abs(col_data - col_data.mean()))
    return feats
```

The 5-second window length was selected as this duration is sufficient to capture multiple cycles of walking and at least one complete jumping action, providing enough context for classification.

Selected Features

For each 5-second window and each acceleration axis (X, Y, Z, and Absolute), 10 different statistical features were calculated, resulting in a total of 40 features per window. These are considered common features and are necessary for activity tracking [1]:

1. Mean
2. Standard Deviation
3. Minimum
4. Maximum
5. Skewness
6. Kurtosis
7. Range
8. Median
9. Variance
10. Mean Absolute Deviation (MAD)

Feature Extraction Implementation

The feature extraction was completed by segmenting the data and applying the feature extraction function to each window:

```
X_list = []
y_list = []
```

```
# Group by subject, activity
for (subj, act), group_df in data.groupby(["Subject", "Activity"]):
    group_df = group_df.reset_index(drop=True)
    for start in range(0, len(group_df) - samples_per_window, samples_per_window):
        window = group_df.iloc[start:start + samples_per_window]
        if len(window) == samples_per_window:
            feats = extract_features(window)
            X_list.append(feats)
            y_list.append(act)
```

Approaching it this way allowed the builders to make sure that that each window contained data from only one subject and one activity, preventing confusion in the training data. By grouping the data by subject and activity before windowing, clean separation is maintained between different activities and individuals.

Feature Normalization

After extracting features, they were normalized using standardization (Z-score normalization) to ensure all features contributed equally to the classification despite their original scales potentially having varying values.

Standardization transforms each feature to have a mean of 0 and a standard deviation of 1 by applying the formula [2]:

$$X\_standardized = (X - \mu) / \sigma \quad (1)$$

Where:

- X is the original feature value
- μ is the mean of the feature
- σ is the standard deviation of the feature

This normalization method was selected for several important reasons. First, it guarantees scale independence, which is crucial because certain features, like kurtosis, can have much smaller values than others, like maximum acceleration. In the absence of normalization, the model may be disproportionately affected by these scale discrepancies. Second, standardized features help many machine learning algorithms, such as logistic regression, converge more quickly and produce predictions that are more accurate. Third, standardization encourages equal feature importance, making sure that a feature's magnitude doesn't erroneously exaggerate its model contribution. Finally, by avoiding problems caused by widely disparate feature scales, it improves numerical stability during training, which in turn improves the behavior of optimization algorithms.

Justification of Feature Choices

Several significant factors influenced our feature selection. By choosing characteristics that characterize important elements of the acceleration signal, like central tendency, variability, and distribution shape, we first sought to achieve thorough signal characterization. This guarantees a comprehensive depiction of motion patterns. Second, by computing features independently for each axis, we added multi-axis

information. This makes the model resilient to changes in the phone's orientation while data is being collected. Third, we gave computational efficiency top priority by selecting statistical features that are quick and easy to calculate, making them appropriate for real-time applications. Lastly, we followed established procedures, utilizing generally recognized characteristics frequently employed in accelerometer-based human activity recognition studies.

This feature set works particularly well for the activities. Features like mean, standard deviation, and kurtosis are good at capturing the regular, periodic acceleration patterns with the moderate variability that walking tends to produce. On the other hand, features like maximum, range, and skewness reflect the sharper, more extreme acceleration peaks produced by jumping. Our classification algorithm can successfully distinguish between walking and jumping thanks to the rich representation of the recorded movements that this feature combination offers.

Training and Testing the Classifier

We trained a logistic regression model using feature vectors extracted from our preprocessed dataset. Initially, we created training and test splits (90% and 10%, respectively) to ensure no data leakage occurred. As for parameters, we implemented logistic regression using the LogisticRegression class from scikit-learn. The model was configured with max_iter=1000 (maximum number of iterations) to ensure convergence. The default solver (lbfgs) was used due to its suitability for smaller datasets. We used the default penalty='l2', which adds regularization to reduce the risk of overfitting and improve generalization. No additional tuning was necessary due to the model's strong performance and low complexity.

The resulting model demonstrated strong performance, correctly identifying every instance in our test set as either walking or jumping, leading to a 100% test accuracy. A key visualization of this success is the confusion matrix, shown below in Figure 5, where each of the 22 test samples are correctly plotted by its true class versus the predicted class.

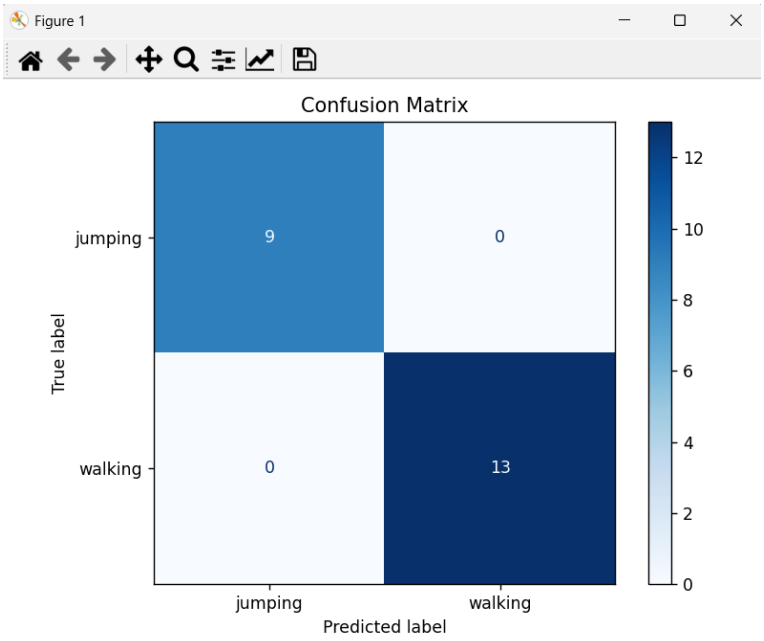

Figure 5: Confusion matrix table from data sample

Next, we evaluated whether the model benefits from additional training samples or risks overfitting by plotting the learning curve, shown in Figure 6 below. In this figure, the training accuracy remains near 100% even when only a small subset of data is used. Meanwhile, the validation accuracy rises quickly to match the training accuracy, indicating that the model generalizes well. This behavior implies that the logistic regression classifier is both a suitable and efficient choice for our feature set.

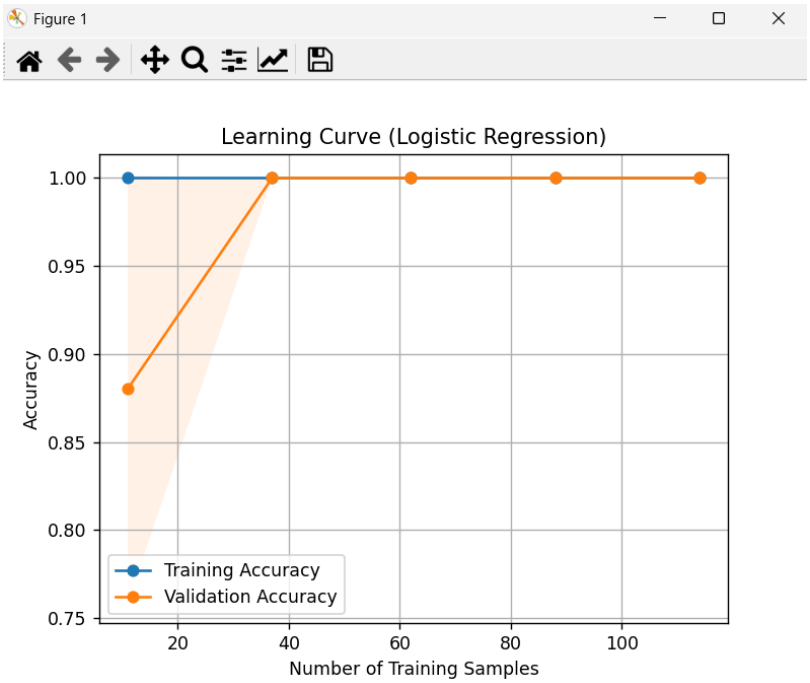

Figure 6: Learning curve plotting model accuracy in regard to number of training samples

Model Deployment

After training our logistic regression model on the extracted features, we deployed it as a standalone desktop application using Python's Tkinter library (GUI.py). To achieve this, we first saved our trained model (logreg_model.pkl) and the associated scaler (scaler.pkl) using the joblib library, which allowed us to load these objects directly into our graphical user interface (GUI) for classification. Within the GUI's main function, we structured the interface by creating a top frame for basic navigation elements. For instance, a "Browse CSV" button was added for loading input data and the "Quit" button for exiting the application. Below that, we placed a scrolled text widget to display classification results, which ensures that even lengthy outputs can be navigated easily without overwhelming the user. In a separate frame, we embedded a matplotlib figure canvas that dynamically plots the acceleration data, highlighting predicted "walking" windows in green and "jumping" windows in red. This color-coded approach offers immediate visual feedback, making it clear where different activities occur in the time series.

When a user selects a CSV file, our classify_csv function is triggered. This function reads the accelerometer data (Time and Acceleration columns), fills in any missing values, applies smoothing to reduce noise, and then segments the data into five-second windows. Each window is transformed into the appropriate feature vector via the same pipeline used during training (i.e., the exact methods in main.py). Once scaled by our saved StandardScaler, the windows are passed into the loaded logistic regression model for prediction. The GUI subsequently displays a textual summary of each window's time range and predicted label, while the matplotlib plot overlays a subtle green or red rectangle over each window region to intuitively convey the classification results. This design choice was made to let users quickly interpret both raw signal behaviors and classification outcomes in a single visual. Additionally, we included a text box for written output where the user can see the model's predicted output for every time increment of 5s.

Once deployed, we tested the system through our previously unused CSV files taken from 10% of our data collection, which proved successful. In addition, to test and demonstrate GUI capability we recorded new data that included both walking and jumping in the same snippet to see how the model would interpret it, which was also successful, seen in the Figure 7 below.

Figure 7: Graphical User Interface (GUI) of model highlighting walking and jumping segments

List of References

[1] "Sensors Article – Accelerometer Feature Extraction," PubMed Central. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC9572087/pdf/sensors-22-07482.pdf. [Accessed: Apr. 2, 2025].

[2] "Z-Score Normalization," Statology. [Online]. Available: https://www.statology.org/z-score-normalization/. [Accessed: Apr. 2, 2025].

Participation Report

Individual	Tasks Completed	Total Time Spent
XXXX	Coding: Data Collection, Data Storage, Preprocessing, Segmentation, Training and Testing the classifier Report: Participation Table, Pre-processing, Data collection, Data Storage, Editing and refining the entire report, refining all sections	14
XXXX	Coding: Feature Extraction & Normalization Report: Introduction, Feature Extraction & Normalization, Training and Testing the Classifier	12
Julien	Coding: Visualization, Refining and comments overall, Graphical User Interface Video: Coordinated video, Editing, Narration Report: Visualization, Model Deployment	12

Figure 8: Participation Table for Work Distribution.