OULUN AMMATTIKORKEAKOULU

Vitali Samurov

GAIT ANALYSIS AS A PROMINENT BEHAVIORAL BIOMETRIC MODALITY FOR CUSTOMER AUTHENTICATION PURPOSES

Master's Thesis

GAIT ANALYSIS AS A PROMINENT BEHAVIORAL BIOMETRIC MODALITY FOR CUSTOMER AUTHENTICATION PURPOSES

Vitali Samurov
Master's Thesis
Spring 2023
The Degree Programme in Data Analytics
and Project Management
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
The Degree Programme in Data Analytics and Project Managements

---

Author(s): Vitali Samurov
Title of the thesis: Gait analysis as a prominent behavioral biometric modality for customer authentication purposes
Thesis examiner(s): Ilpo Virtanen
Spring 2023 Pages: 94

As technology continues to advance, people are increasingly interacting with a variety of systems, ranging from mobile phones and wearables to payment terminals and vehicles. Although many of these systems still rely on traditional forms of authentication, such as PINs and passwords, these methods are not without limitations due to human errors and biases. As a result, there is a growing interest in the use of Behavioral Biometrics (BB) as a more seamless and passive authentication method. BB is typically used in combination with multifactor authentication (MFA). In this thesis, the focus is on gait analysis using accelerometer and gyroscope data as a biometric modality. The study aims to investigate the effectiveness of classical machine learning techniques in analyzing gait data for authentication purposes, as well as the potential of gait-based BB to enhance customer authentication through MFA.

The study is divided two parts: a theoretical exploration of BB, and an empirical analysis of gait data. A combination of qualitative and quantitative research methods was employed to investigate the effectiveness of various machine learning models and their applicability in the context of BB.

Keywords: Behavioral Biometrics, Machine Learning, Customer Authentication, Multifactor Authentication, Gait Analysis, Accelerometer, Gyroscope

CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

2FA	Two-Factor Authentication
BB	Behavioral Biometrics
CPU	Central Processing Unit
DTW	Dynamic Time Warping
FFT	Fast Fourier Transform
GBM	Gradient Boosting Machine
GPS	Global Positioning System
I2C	Inter-Integrated Circuit
LED	Light-Emitting Diode
LR	Logistic Regression
LiDAR	Light Detection and Ranging
MFA	Multi-Factor Authentication
NFC	Near-Field Communication
PIN	Personal Identification Number
RF	Random Forest
SCA	Strong Customer Authentication
SHAP	SHapley Additive exPlanations
SPI	Serial Peripheral Interface
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
TSC	Time Series Classification
UART	Universal Asynchronous Receiver-Transmitter

1. INTRODUCTION

Behavioral Biometrics (BB) is the measurement and statistical analysis of people's unique physical and behavioral characteristics. The technology is mainly used for identification and access control. The basic premise of biometric authentication is that every person can be accurately identified by intrinsic physical or behavioral traits. The term biometrics is derived from the Greek words *bio*, meaning "life", and *metric*, meaning "to measure" (Gillis, Loshin, and Cobb 2021).

Such unimodal systems have to deal with various challenges such as lack of secrecy, non-universality of samples, extent of user's comfort and freedom while dealing with the system, spoofing attacks on stored data, etc. BB is a vast area of computer science that is heavily dependent on machine learning and is about to determine when a significant deviation from patterns or trends established as a standard for users and entities is occurring. Authentication by biometric verification is becoming increasingly common in corporate and public security systems, consumer electronics and point-of-sale applications. In addition to security, the driving force behind biometric verification has been convenience, as there are no passwords to remember or security tokens to carry. Some biometric methods, such as measuring a person's gait, can operate with no direct contact with the person being authenticated (Gillis, Loshin, and Cobb 2021).

The uniqueness of the biometric features extracted from these traits, have allowed unimodal biometric systems to be widely used for identification and verification applications in a wide variety of scenarios (Soleymani et al. 2021), (Haghighat, Abdel-Mottaleb, and Alhalabi 2016).

2. SMARTPHONES

As one can infer from the name, smartphones are smart. Not the device itself, but because of its connection to a network, the sea of information. You can search for information using a smartphone and use the information via various applications and services. Behind its surface is the world of sensors, which is not normally of much interest. Using sensors, you can accurately and precisely monitor the movement and location of a three-dimensional device or detect changes in its environment. For instance, thanks to sensors, game applications can detect when the mobile phone is tilted or shaken and a weather application records the current environment through temperature, pressure, and humidity information (*What Kinds of Sensors are Embedded in Smartphones?* 2022).

DynaTAC 8000x launched in 1983, was the first commercially available smartphone in the world from Motorola. At that time, no one can even imagine that one day mobile phones will become as powerful as computers. Whether you want to click an amazing photo in low lighting conditions or you want to control your home appliances right from your phone. Today's smartphones can fulfill one's needs. Smartphones can detect light and adjust the screen brightness. Smartphones can count the user's steps, send the user a warning when the central processing unit (CPU) is overheated. The camera of a smartphone knows whether the user holds the smartphone in landscape or portrait (Tillu 2021).

2.1 Smartphone Sensors

2.1.1 Definition of Sensor

The sensor is a device that detects and measures the changes in the nearby environment and sends that data to the operating system or processor. They sense and collects data for which they are made. Like ambient light, the sensor is made for detecting light, so it is an expert in detecting the light. There are three main categories of sensors that smartphone having (Tillu 2021):

1. Motion Sensors
2. Environmental Sensors
3. Position Sensors

2.1.2 Motion Sensors

These sensors measure axis-based motion sensing, like acceleration forces and rotational forces, along with three axes. This category includes an accelerometer, gravity sensors, and gyroscopes sensors (Tillu 2021).

2.1.3 Environmental Sensors

These sensors measure environmental parameters like humidity, illumination intensity, surrounding pressure, and temperature sensors. Relative humidity, expressed as a percentage, indicates absolute humidity. Pressure and temperature are expressed in absolute values, and illumination is measured in lux. A thermometer inside a smartphone measures the temperature inside the device and prevents overheating. Some smartphone makers add either an altimeter or other sensors. With environment sensors, a smartphone can adjust the screen brightness to a comfortable level for the user, calculate the dew point of the day in a weather application, or gather

related information and deliver it to the user (*What Kinds of Sensors are Embedded in Smartphones?* 2022).

2.1.4 Position Sensor

These sensors are used to check a device's physical location. A location sensor uses a geomagnetic sensor and an accelerometer to indicate your relative location from the North Pole. Thanks to this sensor, you can use a device as a compass and check the device's location changes. A proximity sensor measures the distance between a certain object and the device. For instance, it can be used to measure how far a user's head is from a headset. That is how your smartphone's screen is off and does not respond to touch during a call. This sensor is usually located near the front camera, where your ear touches (*What Kinds of Sensors are Embedded in Smartphones?* 2022).

2.1.5 Standard Smartphone Sensors

Standard sensors that a modern smartphone is having (Tillu 2021):

- Accelerometer
- Ambient Light Sensor
- Ambient Temperature Sensor
- Air Humidity Sensor
- Barometer Sensor
- Fingerprint Sensor
- Gyroscope Sensor
- Harmful Radiation Sensor

- Magnetometer
- Near-Field Communication (NFC) Sensor
- Proximity Sensor
- Pedometer Sensor

2.1.6 New Types of Sensors

The standard types of smartphone sensors have been listed above. Smartphones released during the past two to three years have sensors like a heartbeat sensor previously used for medical devices or a light detection and ranging (LiDAR) sensor used for autonomous vehicles (*What Kinds of Sensors are Embedded in Smartphones?* 2022).

Heartbeat Sensor

A heartbeat sensor usually consists of two light-emitting diodes (LED) that emit light and a photodetector that measures the intensity of the reflected light. When the heart contracts and sends blood throughout the entire body, the pressure in the arteries increases, spurting blood corpuscles to capillary vessels. In other words, the number of blood corpuscles increases when the heart contracts. The number of blood corpuscles decreases when the heart relaxes. Usually, blood corpuscles tend to absorb LED light, so it looks dark if there are numerous blood corpuscles. If it looks bright, then it means there are fewer blood corpuscles. A photodetector measures heartbeat by identifying such changes (*What Kinds of Sensors are Embedded in Smartphones?* 2022).

Then, how is blood oxygen saturation (SpO2) measured? Oxygenated hemoglobin tends to absorb infrared light, whereas deoxygenated hemoglobin tends to absorb red light. A smartphone shines both infrared light and red light and measures how

oxygenated the blood is. This is why a smartphone's sensor sends off a red light (*What Kinds of Sensors are Embedded in Smartphones?* 2022).

LiDAR Sensor

A LiDAR sensor is used by autonomous vehicles to detect other vehicles or surroundings. As one can conclude from its name, a LiDAR sensor shines a light on a target, measures the features of the reflected signal, and maps the distance. The sensor can analyze reflected light patterns by the microsecond or even by the nanosecond. Based on this, a target's relative location can be drawn in a three-dimensional virtual space (*What Kinds of Sensors are Embedded in Smartphones?* 2022).

A LiDAR sensor is used because a smartphone camera is sensitive to the surrounding environment, including brightness, and an ultrasonic sensor cannot accurately measure distance and cannot be used if the distance is farther than a few meters. In other words, a LiDAR sensor can accurately locate a target even on rainy, foggy, or snowy days or in a dark environment. In addition, with a LiDAR sensor, you can place objects in a virtual space, develop a camera application that can instantly adjust the camera focus at night, and automatically measure the size of a three-dimensional space. Moreover, applications that can convert photos into 2D and 3D floor plans will soon be released (*What Kinds of Sensors are Embedded in Smartphones?* 2022).

Sensor Hub

Developers can leverage the rich sensing platform to enable thousands of mobile applications. Many of these applications require continuous sensing and monitoring for tasks ranging from simple step counting to more complex fall detection, sleep apnea diagnoses, dangerous driver monitoring and others. Unfortunately, continuous sensing applications are power hungry. Interestingly, it is neither the sensors nor

the computation that make these applications battery drainers. Instead, the main processor needs to be powered on frequently to collect sensor samples, which in turn increases the power consumption (Priyantha, Lymberopoulos, and Liu 2011), (Lin et al. 2012). Hardware manufacturers recognize that supporting low-power continuous sensing is crucial. To this end, companies such as Texas Instruments (TI), Intel, and Apple, are embedding a low power microcontroller called a *Sensor Hub* in their smartphones. The sensor hub continuously collects sensor data, keeping the higher power main processor idle. Sensor hub can perform continuous sensing while drawing a fraction of the power compared to the main processor. Figure 2.1 shows an example sensor hub architecture (Shen et al. 2015).

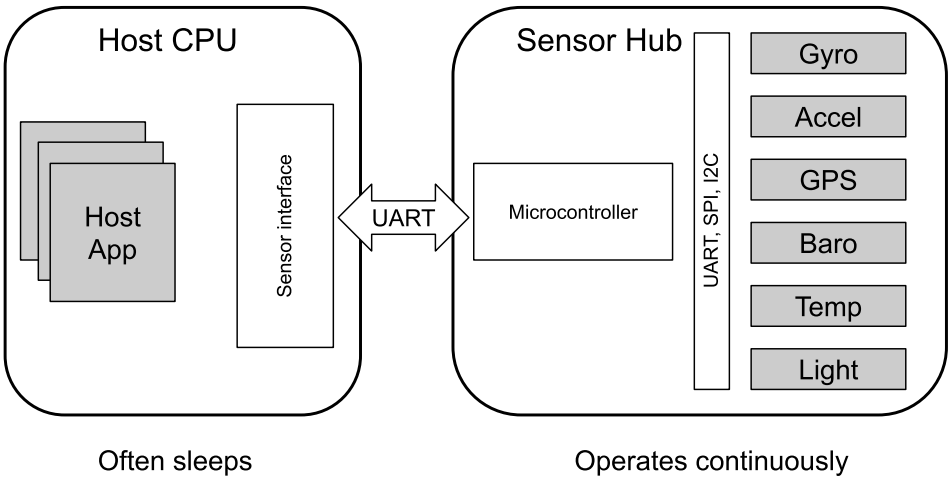

Figure 2.1 *A typical sensor hub architecture*

Communication between host CPU and microcontroller in sensor hub typically takes place through universal asynchronous receiver-transmitter (UART), and the microcontroller polls the sensors through various buses: UART, serial peripheral interface (SPI), and inter-integrated circuit (I2C) (Shen et al. 2015).

3. BIOMETRIC MODALITIES

A biometric modality refers to a system built to recognize a particular biometric trait. Face, fingerprint, hand geometry, palm print, iris, voice, signature, gait, and keystroke dynamics are examples of biometric traits. In the context of a given system and application, the presentation of a user's biometric feature involves both biological and behavioral aspects (Committee, Pato, and Millett 2010).

3.1 Types of Biometric Modalities

There are various traits present in humans, and these can be used as biometrics modalities. These modalities can be grouped into two main categories, namely physiological and behavioral (Piugie et al. 2021), like depicted in the Figure 3.1.

3.2 Reliability of Biometric Modalities

Biometric recognition systems are inherently probabilistic, and their performance needs to be assessed within the context of this fundamental and critical characteristic. Biometric recognition involves matching, within a tolerance of approximation, of observed biometric traits against previously collected data for a subject. Approximate matching is required due to the variations in biological attributes and behaviors both within and between persons. Consequently, in contrast to the largely binary results associated with most information technology systems, biometric systems provide probabilistic results (Committee, Pato, and Millett 2010).

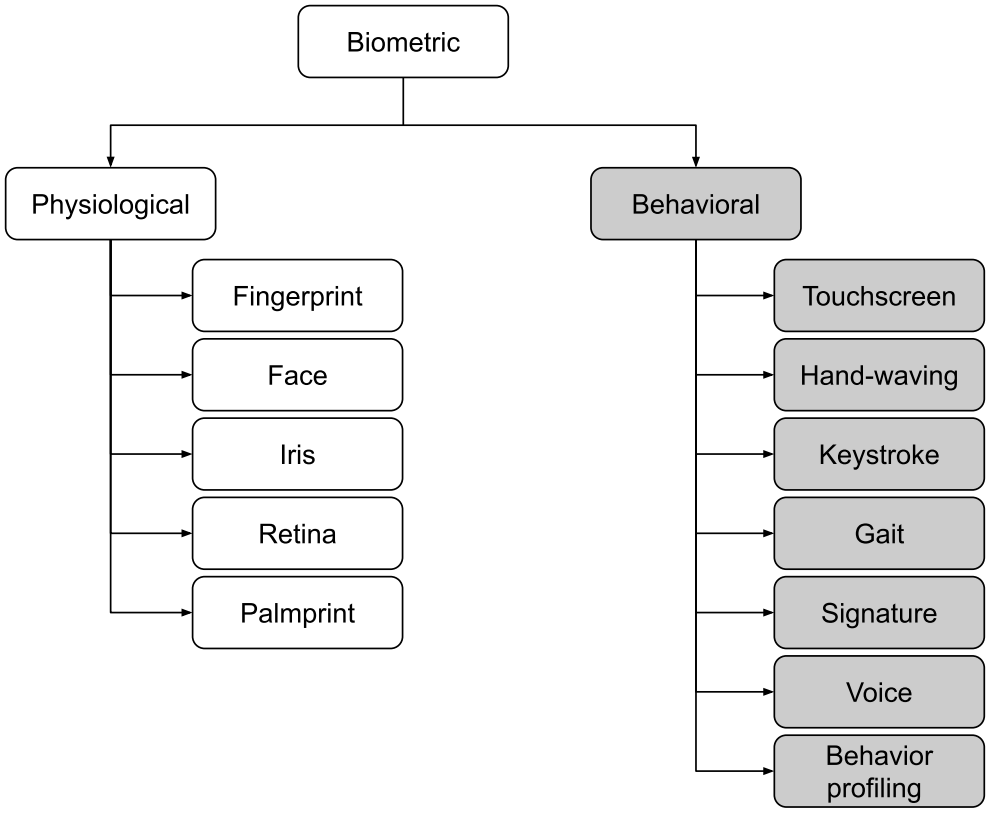

Figure 3.1 *Biometric modalities*

3.3 Behavioral Modalities

3.3.1 Touchscreen

Touchscreen dynamics, which is a behavioral biometry, refers to the unique patterns of rhythm and timing-based features that are created when a user types on a touchscreen in computing devices such as mobile devices (Krishnamoorthy et al. 2018), (Trojahn, Arndt, and Ortmeier 2013).

For example, pressure, size or exact coordinates of keystroke can be used. Basically, the rhythm is a characteristic which can be calculated by different aspects, but, in most cases, at least the time differences are used (Maiorana et al. 2011).

With the increased popularity of touchscreen mobile phones, touch gesture behavior is increasingly becoming important in comparison to its counterpart the

keystroke behavior, since almost all smartphones use the touchscreen as the main input method. Typically, there are three extra feature sets added in addition to the well-studies Keystroke biometrics: timing features, touchscreen features, and both timing and touchscreen features combined. The duration of each soft keystroke, and transitions between press-release and release-press events are taken, for the total features. The non-timing touchscreen features are calculated similarly to the timing features and include: the pressure and the position of screen-touch X- and Y-coordinates (Pandikumar et al. 2017).

3.3.2 Hand-waving

This is a relatively novel application authentication approach that can be used to protect a mobile device from unauthorized access. It captures user's intent to access the service via a lightweight hand waving gesture. This gesture is very simple, quick and intuitive for the user, but would be very hard for the attacker to exhibit without the user's knowledge. In this approach, an accelerometer is used to detect and classify hand waving patterns using neural network classifiers. This allows for a secure and user-friendly authentication experience on mobile devices (Mohana Priya and Alamelu 2018) (Ijartet, Greena, and Baveenther 2016).

For precisely characterizing user's shaking actions, selecting appropriate sensors is necessary. Typically, this technique uses the 3-axis accelerometer for detecting the hand shaking motion. The accelerometer allows smartphones to detect the motion performed on them. The accelerometer in smartphones measures the acceleration of the phone relative to free-fall. The accelerometer measures the acceleration of the phone in three different axes: X, Y, and Z. Based on these features, a system with neural network pattern classifiers decides is this a genuine user or an imposter (Ijartet, Greena, and Baveenther 2016).

3.3.3 Keystroke

Keystroke recognition has been defined by both industry and academics as the process of measuring and assessing a typing rhythm on digital devices, including computer keyboards, mobile phones, and touch screen panels. A noted typing measurement, keystroke recognition, often called "keystroke dynamics", refers to the detailed timing information, or the rhythm, that describes exactly when each key was pressed on a digital device and when it was released as a person types (Krishnamoorthy et al. 2018).

Basically, the rhythm is a characteristic which can be calculated by different aspects, but, in most cases, at least the time differences were used (Moskovitch et al. 2009).

3.3.4 Gait

With the growth in smartphone use as well as wearables (G. Bai and Sun 2019), a relatively new behavioral biometry is gaining popularity: a smartphone-based gait recognition. For this purpose, smartphone-based accelerometers are used to capture gait data continuously in the background, but only when an individual walks. Later, the system analyzes the recorded gait data and establishes the identity of an individual (Zeng et al. 2021), (Muaaz and Mayrhofer 2017).

An advantage of using the gait modality is that it can be linked with characteristics such as unobtrusiveness, effectiveness from a distance, and non-vulnerability, as it is difficult to continuously manipulate one's own gait (Khamsemanan, Nattee, and Jianwattanapaisarn 2017).

There are researches in the area of a lightweight user identification, exploiting both commodity sensing devices and up-to-date deep learning techniques. The technology is built on a key observation that footsteps carry "footprints" unique to individuals, and thus can be leveraged for effective user identification. These footprints can be passively captured by commodity acoustic sensing hardware, totally removing the

need for active user involvements (Cai et al. 2021).

3.3.5 Signature

As it was stated in (Piugie et al. 2021) the biometric features used in authentication process are divided into two categories: (a) physiological - related to the construction of the human body (e.g. fingerprint, iris, hand geometry, face) and (b) behavioral - related to the human behavior (e.g. signature, gait, keystrokes). A handwritten signature occupies a special place among behavioral characteristics, its acquisition is not controversial, and it is commonly socially acceptable (Cpalka, Zalasinski, and Rutkowski 2016).

A handwritten signature occupies a special place among behavioral characteristics, its acquisition is not controversial, and it is commonly socially acceptable (Cpalka, Zalasinski, and Rutkowski 2016).

There are two types of signature verification: (a) offline (static) verification and (b) online (dynamic) verification. In the offline setting, the system has the shape of the signature by capturing or scanning them from papers or extracted from the picture of the signature. Therefore, in an offline verification system, input data contains Y- Y- coordinates of signatures. However, in the online setting, the system uses devices for capturing additional information while the user is signing. Online signatures have extra information for extraction such as time, pressure, pen up and down, azimuth, etc. (Fayyaz et al. 2015).

3.3.6 Voice

Biometric voice recognition is the use of the human voice to uniquely identify biological characteristics to authenticate an individual, unlike passwords or tokens that require physical input. This modality includes both biometric categories - physiological and behavioral (Eshwarappa M and Latte 2012).

Voice authentication has become an integral part of security-critical operations, such as bank transactions and call center conversations. The vulnerability of Automatic Speaker Verification Systems (ASV) to spoofing attacks instigated the development of countermeasures, whose task is to tell apart bona fide and spoofed speech (Kassis and Hengartner 2021).

With the increased popularity of wearables, smart vehicles, and home automation systems, Voice Assistants, such as Siri, Google Now, Cortana and Alexa, have become the everyday fixtures, especially in scenarios where touch interfaces are inconvenient or even dangerous to use, when for example driving or exercising. However, with sound being an open channel, voice as an input mechanism is inherently insecure as it is prone to replay, sensitive to noise, and easy to impersonate. In order to improve the reliability, researchers are looking into additional methods, for example the novel system that provides usable and continuous authentication for voice assistant systems. As a wearable security token, it supports ongoing authentication by matching the user's voice with an additional channel that provides physical assurance. The system collects the body-surface vibrations of a user via a wearable-device accelerometer and continuously matches them to the voice commands received by the voice assistant (Feng, Fawaz, and Shin 2017).

3.3.7 Behavior Profiling

Lifestyle authentication has become a new research approach. A promising idea for it is to use the location history since it is relatively unique. Even when people live in the same area or they occasionally travel, it does not vary from day to day. For Global Positioning System (GPS) data, the previous work used the longitude, the latitude, and the timestamp as the features for the classification. In addition to this, researchers are investigating a new approach utilizing the distance coherence, which can be extracted from the GPS itself without the need to require other information (Thao 2020).

Another group of researchers proposed behavior profiling technique by using a mobile user's application usage to detect abnormal mobile activities. Whilst some users were difficult to classify, a significant proportion fell within the performance expectations of a behavioral biometric and therefore a behavior profiling system on mobile devices is able to detect anomalies during the use of the mobile device. Incorporated within a wider authentication system, this biometric would enable transparent and continuous authentication of the user, thereby maximizing user acceptance and security (Li et al. 2011).

As the popularity of wearable devices (smartwatches and fitness trackers) has been growing, researchers have begun to pay attention to new types of data, such as oxygen saturation, collected continuously using the oxygen saturation (SpO2) sensors and represent the percentage of oxygen-saturated hemoglobin compared to the total amount of hemoglobin in the blood. Such systems are becoming available to market wearables (*Suffocating Progress* 2020). This personalized data could be valuable to identify an individual and thereby could be useful for implicit user authentication (Muratyan et al. 2021).

4. DATA ANALYSIS

4.1 Dataset For Real-life Human Activity Recognition

In this thesis, a public domain dataset for Real-life Human Activity Recognition Using Smartphone Sensors has been used (*A Public Domain Dataset For Real-life Human Activity Recognition Using Smartphone Sensors* 2020).

The information in this dataset is the measurements from the accelerometer, gyroscope, magnetometer, and GPS of the smartphone. Additionally, each measure is associated with one of the four possible registered activities: inactive, active, walking and driving. This work also proposes an SVM model to perform some preliminary experiments on the dataset. Considering that this dataset was taken from smartphones in their actual use, unlike other datasets, the development of a good model on such data is an open problem and a challenge for researchers (*A Public Domain Dataset For Real-life Human Activity Recognition Using Smartphone Sensors* 2020).

Data was collected using an Android app from each of the 19 participant's smartphones in the study. The data was collected in a real-life environment, allowing each participant to use their device as they normally would for each specified action. As a result, the orientation and placement of the smartphones during data collection were not fixed.

There were four types of activities performed:

- *Inactive*: When the mobile phone is not being used. For example, the phone is kept on a desk while the person is doing some other activity.

- *Active*: When the mobile phone is being carried while performing any other activity such as cooking, attending a concert, shopping or doing the dishes.
- *Walking*: When the mobile phone is being carried while walking to a specific place. Running or jogging are also considered as "walking" activities.
- *Driving*: When the mobile phone is being carried while traveling in a vehicle powered by an engine. This includes cars, buses, motorbikes, trucks or any other similar vehicle.

The data was obtained from four distinct sensors, each capturing different aspects of the device's movement and location. These sensors include the accelerometer, gyroscope, magnetometer, and GPS. The accelerometer, gyroscope, and magnetometer provided tri-axial values, which were recorded. For the GPS sensor, detailed information on the device's location was collected, such as changes in latitude, longitude, and altitude, as well as the bearing, speed, and accuracy of the measurements. The resulting dataset is a comprehensive representation of the device's movement and location during the time it was being monitored.

It is important to note that the measures mentioned in the study cannot be set at a fixed frequency on Android devices. Furthermore, some participants did not have access to all the sensors on their smartphones, which resulted in certain sessions lacking a gyroscope or both a gyroscope and a magnetometer.

4.1.1 Experimental Setup

For all computer calculations presented in this thesis, a Python software environment was utilized, running on the author's personal computer. The following are the installed Python modules that were used:

- *Pandas*: a software library designed for the Python programming language, used for data manipulation and analysis.

- *Numpy*: a software library that is available as an open-source tool for several scientific and engineering fields. It is the most commonly used library in Python to work with numerical data and is an essential component of the scientific Python and PyData ecosystems.

- *SciPy*: provides algorithms for various classes of problems including optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, and statistics.

- *Scikit-learn*: a free software machine learning library for Python that features various classification, regression, and clustering algorithms.

- *Seaborn* and *Matplotlib*: Python data visualization libraries used for creating graphs and plots.

- *Jupyter Notebook*: Notebook documents, commonly referred to as "notebooks", are files that are created using the Jupyter Notebook App. These files contain both computer code, which is usually written in Python, and rich text elements, such as paragraphs, equations, figures, and links. Notebook documents serve a dual purpose - they can be read by humans as they contain descriptions of the analysis and its results (such as tables and figures), and they can also be executed to perform data analysis.

4.1.2 Sensor Selection

Sensors Set

Nowadays, smartphones are equipped with various sensors, as demonstrated in section 2.1.5. As these devices increasingly integrate into our daily lives, they have the potential to collect a wealth of personal information through their sensors. Therefore, it is essential to select a limited number of sensors that can reliably capture a user's unique characteristics.

The combination of data from two sensors indeed provides better authentication accuracy than using a single sensor. Another interesting finding is that using a combination of magnetometer and orientation sensors is worse than the other two pairs, which include an accelerometer. In fact, the combination of magnetometer and orientation sensors is not necessarily better than using just the accelerometer. And finally, the use of three sensors give the best authentication accuracy. Therefore, choosing good sensors is very important (W.-H. Lee and R. Lee 2017).

In the scope of this thesis, the following two sensors have been used: accelerometers and gyroscopes. They also represent different information about the user's behavior and environment: the accelerometer can detect coarse-grained motion of a user, like how the person walks (Nickel, Wirtl, and Busch 2012), the orientation (gyroscope) sensor can detect fine-grained motion of a user like how the person holds a smartphone (Xu, K. Bai, and Zhu 2012). Furthermore, these sensors do not need the user's permission to be used in Android applications, which is useful for continuous monitoring for implicit authentication. Also, these two sensors do not need the user to perform a sequence of actions dictated by a script - hence facilitating implicit authentication (W.-H. Lee and R. Lee 2017).

Combining Sensors Data

The main objective of this thesis is to identify the most common user activity patterns in gait by focusing on walking. However, some of the participants in the study did not have all the necessary sensors available on their smartphones. Additionally, some users interrupted or changed the recording of certain activities, which resulted in gaps in the continuous data. These gaps can hinder the frequency and statistical analysis, so it is recommended to split such data into separate records where sampling was done continuously.

The challenge was to integrate multi-sensor data, especially when the data is sep-

arated into different files. This situation is similar to real-world scenarios where sensor data is transmitted through different threads due to certain operating system designs.

Therefore, preliminary data preprocessing and analysis is crucial before conducting complex analysis to achieve objectives:

- Clean up the files with accelerometer and gyroscope raw data.
- Find out the number of users who participated in the recording of all datasets, and select the top 8 users with the longest duration of the "Walking" activity.
- Analyze the "Walking" activity of each user for data continuity, and if there are significant gaps in the continuous sequence of samples (for example, more than five seconds), break the activity into continuous fragments.
- Identify the beginning and end timestamps of each fragment and calculate the length of each fragment. Determine the discrepancy in timestamp values between the start and stop of user fragments in the corresponding dataset files. If the fragments match accurately, it is recommended to merge them into a single Data Frame.
- In cases of varying data recording frequencies among sensors, resampling is recommended to synchronize datasets for meaningful analyses.

This systematic approach will provide a reliable foundation for the precise analysis and interpretation of the data from multiple sensors.

4.1.3 Dataset Preprocessing

Reading the Dataset into Pandas DataFrames

Following code reads data from 2 sensors dataset files into Pandas Data Frames:

```
base_path = r'/Users/' + getpass.getuser() + '/real-life-HAR-
    dataset'
accel_raw_data_fn = base_path + '/data_raw/sensoringData_acc.csv'
gyro_raw_data_fn = base_path + '/data_raw/sensoringData_gyro.csv'
accel_raw_data_df = pd.read_csv(accel_raw_data_fn)
gyro_raw_data_df = pd.read_csv(gyro_raw_data_fn)
```

Figures 4.1 and 4.2 depict sensors Pandas Data Frames in the form of tables.

	id	username	timestamp	gyro_x_axis	gyro_y_axis	gyro_z_axis	activity_id	activity
0	4006559	11	1.570534e+09	-0.339005	-0.016215	-0.066681	-499	Walking
1	4006573	11	1.570534e+09	-0.257149	0.072972	0.117189	-499	Walking
2	4006584	11	1.570534e+09	0.094710	-0.013771	0.052437	-499	Walking
3	4006602	11	1.570534e+09	0.545528	0.071750	0.296783	-499	Walking
4	4006622	11	1.570534e+09	0.036067	0.150551	0.119632	-499	Walking
...	...	...	...	...	...	...	...	...
3242316	39691768	2	1.576784e+09	0.023458	0.022109	-0.012746	41	Driving
3242317	39691771	2	1.576784e+09	-0.015828	0.055907	0.023195	41	Driving
3242318	39691774	2	1.576784e+09	-0.022495	-0.012176	0.017558	41	Driving
3242319	39691777	2	1.576784e+09	-0.003558	0.031918	0.014676	41	Driving
3242320	39691780	2	1.576784e+09	-0.000646	0.028983	0.010752	41	Driving

3242321 rows × 8 columns Open in new tab

Figure 4.1 *Gyroscope raw data as Pandas DataFrame*

	id	username	timestamp	acc_x_axis	acc_y_axis	acc_z_axis	activity_id	activity
0	4006559	11	1.570534e+09	-0.388868	3.812294	6.222649	-499	Walking
1	4006561	11	1.570534e+09	-0.242817	3.838631	6.059839	-499	Walking
2	4006562	11	1.570534e+09	-0.259577	3.946373	6.023924	-499	Walking
3	4006563	11	1.570534e+09	-0.048881	3.807506	6.090964	-499	Walking
4	4006564	11	1.570534e+09	0.295895	3.795534	6.040684	-499	Walking
...	...	...	...	...	...	...	...	...
17378629	39691767	2	1.576784e+09	0.234983	0.153715	-0.208087	41	Driving
17378630	39691770	2	1.576784e+09	0.143519	0.216550	-0.204246	41	Driving
17378631	39691773	2	1.576784e+09	0.110278	0.207618	-0.285629	41	Driving
17378632	39691776	2	1.576784e+09	0.111215	0.097924	-0.070788	41	Driving
17378633	39691779	2	1.576784e+09	0.030945	-0.048773	0.089122	41	Driving

17378634 rows × 8 columns Open in new tab

Figure 4.2 *Accelerometer raw data as Pandas DataFrame*

The resulting gyroscope data frame consists of 3242321 rows and 8 columns and accelerometer data frame consists of 17378634 rows and 8 columns.

Cleaning Up Datasets

Data cleaning is a critical process that involves identifying and resolving any inaccuracies, inconsistencies, or errors in datasets. It typically involves correcting

or removing incorrect, corrupted, poorly formatted, duplicate, or incomplete data. Although there is no one-size-fits-all approach to data cleaning, it is essential to establish a standardized template to ensure that the process is consistent and reliable every time.

The following code removes rows with duplicate timestamps globally and drops any rows with missing values from the entire dataset:

```
accel_raw_data_df_len = len(accel_raw_data_df)
accel_raw_data_df_unique_len = len(pd.unique(accel_raw_data_df['timestamp']))
accel_dup_num = accel_raw_data_df_len - accel_raw_data_df_unique_len
accel_dup_percent = accel_dup_num/accel_raw_data_df_len * 100
gyro_raw_data_df_len = len(gyro_raw_data_df)
gyro_raw_data_df_unique_len = len(pd.unique(gyro_raw_data_df['timestamp']))
gyro_dup_num = gyro_raw_data_df_len - gyro_raw_data_df_unique_len
gyro_dup_percent = gyro_dup_num/gyro_raw_data_df_len * 100
accel_data_df = (accel_raw_data_df.drop_duplicates(subset="timestamp")).dropna()
gyro_data_df = (gyro_raw_data_df.drop_duplicates(subset="timestamp")).dropna()
```

Number of rows with duplicate "timestamp" column values in:

```
ACCEL raw data: 641583 (3.69%) from total length: 17378634
GYRO raw data: 8923 (0.28%) from total length: 3242321
```

Based on the data analysis, it seems that the duplicated "timestamp" values are most likely due to the low resolution of the time clock system, which results in measurement inaccuracies. To address this issue, the best approach would be to remove the duplicated rows without performing any interpolation. It is important to note that the overall percentage of duplicates is not significant.

4.1.4 Exploratory Data Analysis

Users IDs

The following code returns the count of unique user IDs, lists of usernames/user IDs from datasets, and a "common" list of users present in both sensor dataframes:

```
accel_list_of_usernames = accel_data_df['username'].unique()
gyro_list_of_usernames = gyro_data_df['username'].unique()
a = accel_list_of_usernames
b = gyro_list_of_usernames
userid_in_all = list(set.intersection(*map(set, [a, b])))
```

These are the usernames that are found in both sensor dataframes and are not duplicated:

```
ACCEL data: [11 1 15 18 17 9 6 7 3 13 0 10 14 8 4 5 16 12 2]
GYRO data: [11 1 9 6 7 3 13 0 10 8 5 16 12 2]
```

Here is a list of usernames that appear in both the accelerometer and gyroscope dataframes:

```
[0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 16]
```

The next step is to remove rows from the dataframe where the username is not on the "common list":

```
accel_users_df = accel_data_df[accel_data_df['username'].isin(
    userid_in_all)]
gyro_users_df = gyro_data_df[gyro_data_df['username'].isin(
    userid_in_all)]
```

```
ACCEL user list "clean up": 19 -> 14
GYRO user list "clean up": 14 -> 14
```

There is a file named *activityChanges.csv* in the public dataset folder that contains information about activity sessions, including the username, start and end times,

and activity types. The code below extracts the start time, end time, and duration of activities performed by users on the "common" list and sorts them by their duration:

```
activity_changes_fn = base_path + '/data_raw/activityChanges.csv'
changes_df = pd.read_csv(activity_changes_fn)
activity_changes_df = changes_df[changes_df['username'].isin(
    userid_in_all)]
walking_changes_df = activity_changes_df.query("activity == '
    Walking'")
walking_changes_sorted_df = walking_changes_df.sort_values(by='
    duration', ascending=False, inplace=False)
```

The "Walking" activity has been engaged by a total of 13 different users:

```
[ 8  7  1 16  5  9 11  6 10  2 13 12  3]
```

Top-10 users for further analysis

Below is a code that generates a list named *common_walk_users*. The list includes 10 users who have walked for at least 10 minutes and appear in both accelerometer and gyroscope datasets. Only the users who meet these two criteria are included in the list:

```
duration_sec = 600
walking_above_10m_df = walking_changes_sorted_df[
    walking_changes_sorted_df['duration'] >= duration_sec]
walk_unique_id = walk_above_10m_df["username"].nunique()
```

```
users_over_10min = walk_above_10m_df['username'].unique().tolist()
common_walk_users = list(set(users_over_10min).intersection(
    userid_in_all))
users_over_10min = walk_above_10m_df ['username '].unique ().
    tolist ()
2 common_walk_users = list(set( users_over_10min ). intersection (
userid_in_all ))
```

This is a list of user IDs for those with "Walking" activity duration exceeding 10 minutes:

```
[8, 7, 1, 16, 5, 9, 11, 6, 10, 2]
```

This is a list of user IDs who had a "Walking" activity in both sensor datasets:

```
[1, 2, 5, 6, 7, 8, 9, 10, 11, 16]
```

The dataframe should be filtered to include only the "Walking" activities for users on the *common_walk_users* list:

```
accel_walk_data_df = accel_users_data_df[accel_users_data_df['username'].isin(common_walk_users)].query("activity == 'Walking'")
gyro_walk_data_df = gyro_users_data_df[gyro_users_data_df['username'].isin(common_walk_users)].query("activity == 'Walking'")
```

The number of users in the "Walking" lists for accelerometer and gyroscope decreased from 14 to 10 after cleanup.

The format for dictionaries used in training and testing datasets

Dataframes containing accelerometer and gyroscope sensor data should be segmented based on "Walking" activity sessions listed in *activityChanges.csv* file. Each segment will be represented as an entry in a Python dictionary with the following fields:

- *idx*: Index of the dataframe row.
- *user*: Username.
- *data_idx_df*: Dataframe, cropped from the original Data Frame according to *init_timestamp* and *end_timestamp*, converted to "datetime" format and the timestamp column set as the index.

- *data_idx_df_resampled*: Dataframe, resampled from *data_idx_df* dataframe or *None* if not applicable.
- *init_ts*: Initial timestamp.
- *end_ts*: End timestamp.
- *dur*: Duration of the segment (data chunk).

The same code was applied to both sensor dataframes (e.g. *sensor_user_walk* -¿ *accel_user_walk* and *gyro_user_walk*):

```
sensor_user_walk = []
df = sensor_user_walk.copy()
for index, row in walk_above_10m_df.iterrows():
    user = row['username']
    start_t = row['init_timestamp']
    stop_t = row['end_timestamp']
    crops_df = df[(df['username'] == user) & (df['timestamp'] >= start_t) & (df['timestamp'] <= stop_t)]
    crops_df['timestamp'] = pd.to_datetime(crops_df['timestamp'], unit='s')
    crops_df.set_index('timestamp', inplace=True)

    crops_entry = {
        'idx': index,
        'user': row['username'],
        'init_ts': start_t,
        'end_ts': stop_t,
        'data_idx_df': crops_df,
        'data_idx_df_resampled': None,
        'dur': row['duration'],
        'activity_id': row['activity_id'],
    }
    sensor_user_walk.append(crops_entry)
```

Two Python dictionaries, *accel_user_walk* and *gyro_user_walk*, have been created after executing this code.

The code evaluates the duration of walking segments in two sensor datasets and outputs the top 10 durations:

```
def get_df_duration_in_sec(time_idx_df):
    time_idx_df_temp = time_idx_df.copy()
    duration =
        time_idx_df_temp.index.max() - time_idx_df_temp.index.min()
    return copy.deepcopy(duration.total_seconds())

the_range = 10
for i in range(the_range):
    sensor_dur = copy.deepcopy(sensor_user_walk[i].get("dur"))
    sensor_dur_actual = get_df_duration_in_sec(sensor_user_walk[i].
   get("data_idx_df"))
    sensor_user = copy.deepcopy(sensor_user_walk[i].get("user"))
    sensor_len = len(sensor_user_walk[i].get("data_idx_df"))
    sensor_activity_id = copy.deepcopy(sensor_user_walk[i].get("
   activity_id"))
```

The following list highlights the top 10 durations of "Walking" activity fragments among user IDs:

```
--- Top #0:
  ACCEL duration: 10160.0 sec, actual: 10159.6 sec, ID: 8, len:
    50118
  GYRO duration: 10160.0 sec, actual: 10159.8 sec, ID: 8, len:
    50241
--- Top #1:
  ACCEL duration: 2357.7 sec, actual: 2357.6 sec, ID: 7, len: 12047
  GYRO duration: 2357.7 sec, actual: 2357.2 sec, ID: 7, len: 11972
--- Top #2:
  ACCEL duration: 2093.2 sec, actual: 2093.1 sec, ID: 1, len:
    157591
```

```
  GYRO duration: 2093.2 sec, actual: 2092.7 sec, ID: 1, len: 10490
--- Top #3:
  ACCEL duration: 1812.1 sec, actual: 329.4 sec, ID: 16, len: 3145
  GYRO duration: 1812.1 sec, actual: 329.1 sec, ID: 16, len: 1579
--- Top #4:
  ACCEL duration: 1608.0 sec, actual: 1607.7 sec, ID: 8, len: 25965
  GYRO duration: 1608.0 sec, actual: 1607.7 sec, ID: 8, len: 7936
--- Top #5:
  ACCEL duration: 1560.0 sec, actual: 1560.0 sec, ID: 5, len: 77829
  GYRO duration: 1560.0 sec, actual: 1559.4 sec, ID: 5, len: 7805
--- Top #6:
  ACCEL duration: 1487.0 sec, actual: 1486.9 sec, ID: 1, len: 16155
  GYRO duration: 1487.0 sec, actual: 1486.5 sec, ID: 1, len: 7452
--- Top #7:
  ACCEL duration: 1464.3 sec, actual: 1464.3 sec, ID: 5, len: 69535
  GYRO duration: 1464.3 sec, actual: 1464.0 sec, ID: 5, len: 21307
--- Top #8:
  ACCEL duration: 1403.1 sec, actual: 1402.9 sec, ID: 9, len: 68309
  GYRO duration: 1403.1 sec, actual: 1402.6 sec, ID: 9, len: 7789
--- Top #9:
  ACCEL duration: 1357.9 sec, actual: 1357.8 sec, ID: 5, len: 67808
  GYRO duration: 1357.9 sec, actual: 1357.2 sec, ID: 5, len: 6794
```

Preliminary Conclusion

- Upon analysis, an issue with data fragments was found. Specifically, some accelerometer sensor data segments had more samples than the gyroscope sensor, while time duration matched.
- Fragment "3" has a discrepancy between the listed and actual duration for the "Walking" activity. The listed duration is 1812 seconds, but the actual duration is only 329 seconds based on the first and last timestamps.
- To ensure greater accuracy, it is recommended to exclude Fragment "3" from

further analysis and focus on examining the top nine fragments: [0,1,2,4,5,6,7,8,9]

Here is a list of the top 9 crop fragments for further analysis. Each value corresponds to an index from the dictionary of crops above:

```
top_9_fragments_for_analysis = [0,1,2,4,5,6,7,8,9]
```

Datasets Sampling Rate Analysis

Before conducting any analysis, it is crucial to evaluate the stability of the sensor signal samples for the "Walking" activity among the chosen users. Furthermore, it is essential to explore if there are any variations in the sampling frequency across datasets from different sensors and measure the degree of such differences.

One way to evaluate the sampling rate of an entire dataset is by constructing histograms of the time intervals (delta) between recorded timestamps of different sensor types. This practical method involves using histograms of time intervals to analyze the sampling rate. By visualizing the time intervals, it becomes easier to identify any inconsistencies or variations in the sampling frequency, which can be crucial for the success of the analysis efforts.

As described in Equation 4.1, the average sampling rate is the reciprocal of the Mean of the time intervals between recorded timestamps:

$$\text{Average Sampling Rate} = \frac{1}{\text{Mean of Time Differences}} \tag{4.1}$$

A histogram is a rudimentary estimation of a dataset's Probability Density Function (PDF). For a continuous random variable X, the PDF, $f(x)$, gives the likelihood of X taking on the value x. The mathematical representation for this is demonstrated in the following equation:

$$f(x) = \frac{P(x \leq X < x + \delta x)}{\delta x} \tag{4.2}$$

where δx is an infinitesimally small interval.

When creating a histogram, the data range is divided into intervals called bins. The number of observations in each bin can be described by the following equation:

$$h_i = \sum_{x \in B_i} 1 \tag{4.3}$$

where h_i is the height of the i-th bin and B_i represents the interval of the i-th bin.

The code provided below is used to calculate the difference between two timestamps and the average sampling rate:

```
for i in top_9_fragments_for_analysis:
    accel_idx_df = accel_user_walk[i].get('data_idx_df').copy()
    gyro_idx_df = gyro_user_walk[i].get('data_idx_df').copy()
    accel_original_actual_duration = get_df_duration_in_sec(
   accel_idx_df)
    gyro_actual_duration = get_df_duration_in_sec(gyro_idx_df)

    # Calculating the differences between timestamps
    accel_time_diffs = accel_idx_df.index.to_series().diff().dt.
   total_seconds()
    accel_avg_sampling_rate = 1 / accel_time_diffs.mean()
    gyro_time_diffs = gyro_idx_df.index.to_series().diff().dt.
   total_seconds()
    gyro_avg_sampling_rate = 1 / gyro_time_diffs.mean()
```

The code provided below is used to calculate histograms and plot them in a visual format:

```
    num_of_bins = 80
    edge_color = 'k'
    x_axis_limit = [0, 0.6]
    fig, axs = plt.subplots(1, 2, figsize=(10, 4), sharey=True)
    axs[0].hist(accel_time_diffs[1:], bins=num_of_bins, edgecolor=
   edge_color)
```

```
    axs[0].set_xlabel('Time Difference (s)')
    axs[0].set_ylabel('Frequency')
    axs[0].set_title(f'ACCEL. Average sampling rate: {accel_avg_sampling_rate:.1f} Hz')
    axs[0].grid(True)
    axs[0].set_xlim(x_axis_limit)
    acc_text_str = (
        f'Fragment #:{i}, user ID: {accel_user_id}\n'
        f'Length: {len(accel_idx_df)} rows\n'
        f'Duration (table): {accel_user_walk[i].get("dur"):.1f} sec\n'
        f'Duration (actual): {accel_original_actual_duration:.1f} sec\n'
    )
    axs[0].text(0.02, 0.98, acc_text_str, transform=axs[0].transAxes, verticalalignment='top', fontsize=9)

    axs[1].hist(gyro_time_diffs[1:], bins=num_of_bins, edgecolor=edge_color)
    axs[1].set_xlabel('Time Difference (s)')
    axs[1].set_title(f'GYRO. Average sampling rate: {gyro_avg_sampling_rate:.1f} Hz')
    axs[1].grid(True)
    axs[1].set_xlim(x_axis_limit)
    gyr_text_str = (
        f'Fragment #:{i}, user ID: {gyro_user_id}\n'
        f'Length: {len(gyro_idx_df)} rows\n'
        f'Duration (table): {gyro_user_walk[i].get("dur"):.1f} sec\n'
        f'Duration (actual): {accel_original_actual_duration:.1f} sec\n'
    )
    axs[1].text(0.02, 0.98, gyr_text_str, transform=axs[1].transAxes, verticalalignment='top', fontsize=9)
```

```
31
32      max_ylim = max(axs[0].get_ylim()[1], axs[1].get_ylim()[1])
33      axs[0].set_ylim(0, max_ylim)
34      axs[1].set_ylim(0, max_ylim)
```

Figures 4.3 to 4.11 display histograms of sampling periods for the "Walking" segments in the Top-9 list for both sensors datasets:

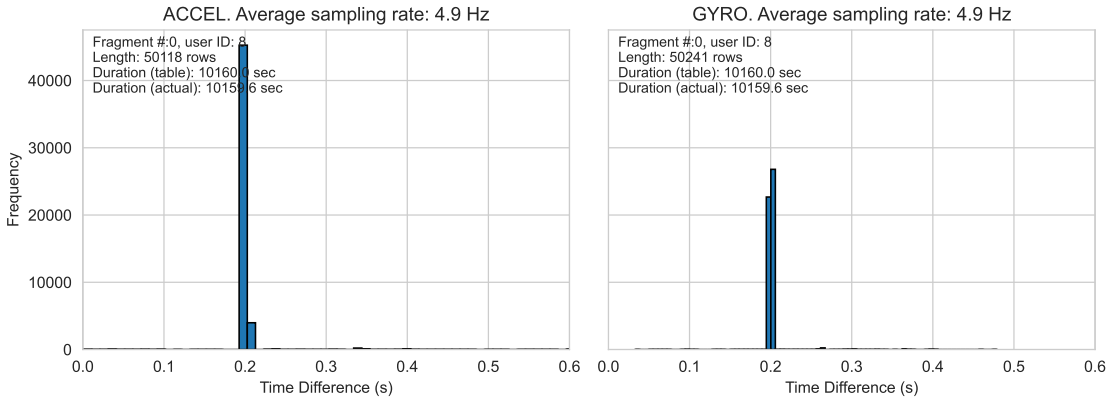

Figure 4.3 *Histograms of sampling periods, fragment "0", user ID: 8*

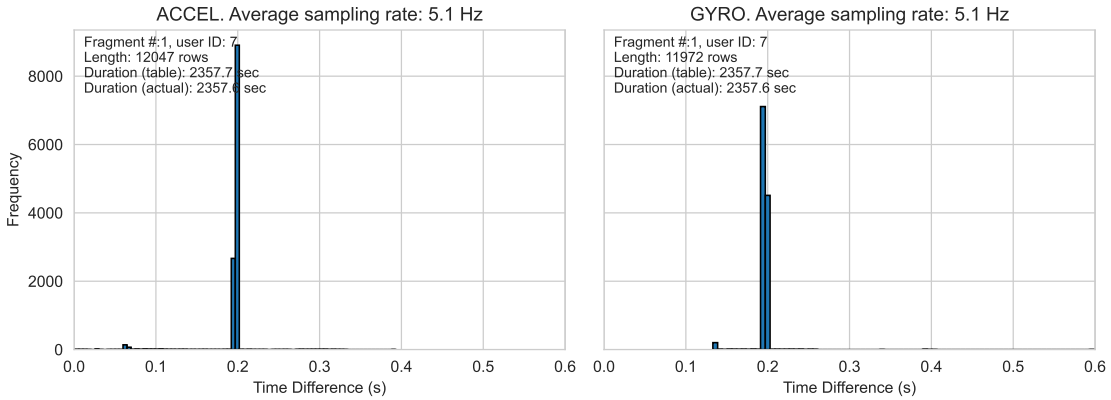

Figure 4.4 *Histograms of sampling periods, fragment "1", user ID: 7*

The readings from the accelerometer and gyroscope sensors in Android devices can sometimes be inconsistent. This is because the real-time performance of sensor data collection is not always guaranteed, depending on the manufacturer, model, background processes, and OS version of the Android device. There are several reasons for this variability:

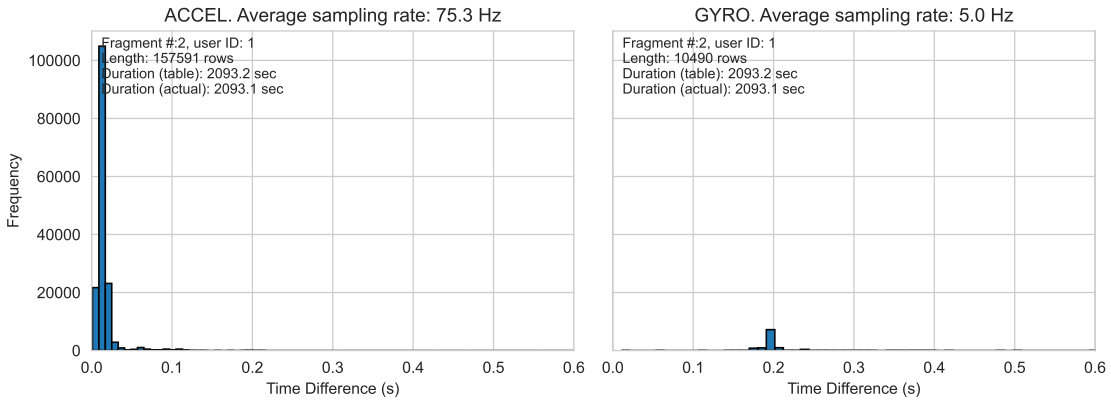

Figure 4.5 *Histograms of sampling periods, fragment "2", user ID: 1*

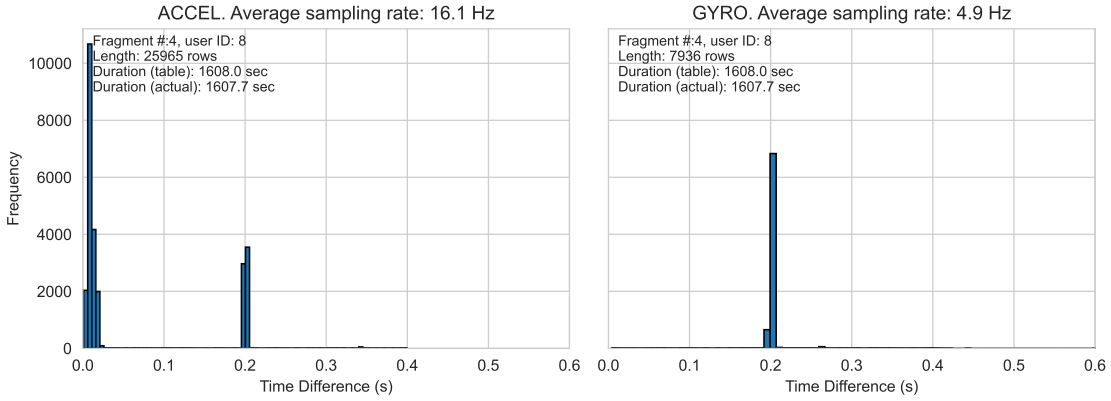

Figure 4.6 *Histograms of sampling periods, fragment "4", user ID: 8*

- Other processes and applications running in the background can interrupt or delay sensor data collection, causing inconsistencies in sampling rates. This happens because Android is not a real-time operating system.
- Sensor hardware differences can lead to varying actual data sampling rates, even if requested at a specific rate.
- In power-saving mode, Android's power management can affect sensor data sampling rates.

Upon analyzing the situation regarding the inconsistencies, it has become evident that aligning the datasets is a crucial step for successful machine learning, especially in the case of classification algorithms. To achieve optimal results, it is recommended

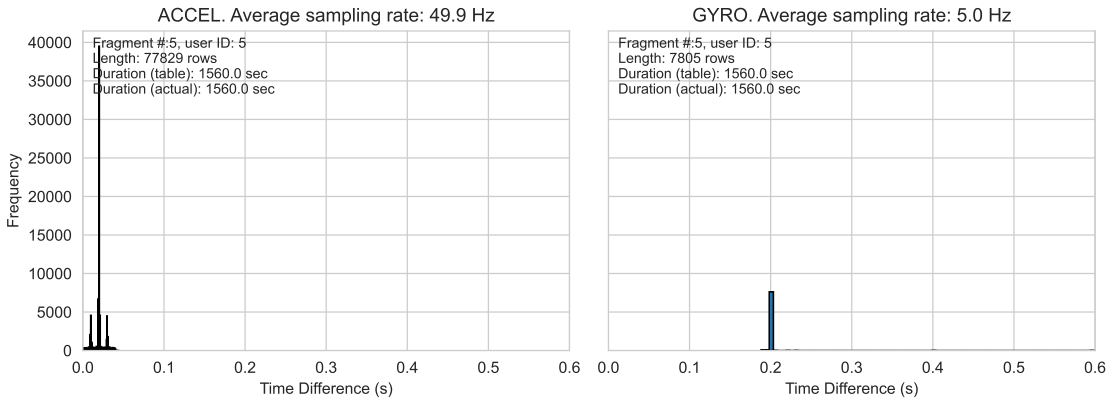

Figure 4.7 *Histograms of sampling periods, fragment "5", user ID: 5*

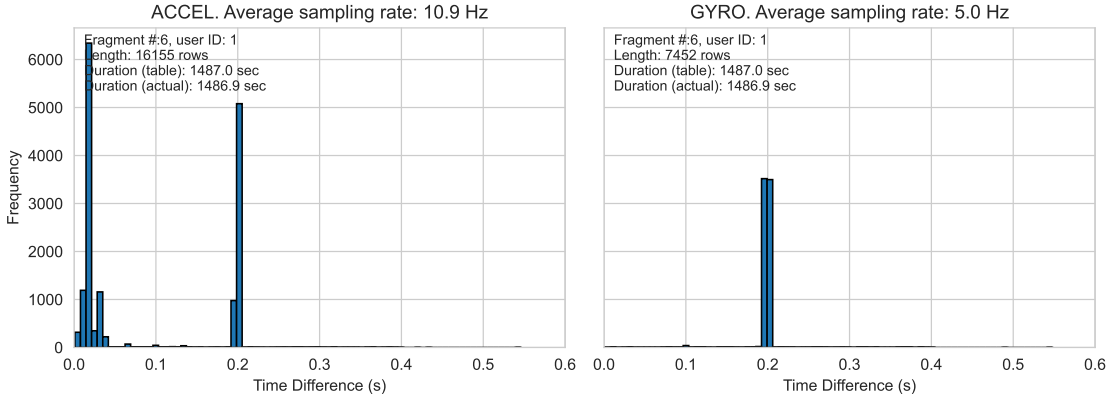

Figure 4.8 *Histograms of sampling periods, fragment "6", user ID: 1*

to either resample the accelerometer data or process it in a way that matches the same dimension and sampling rate as the gyroscope data. This alignment guarantees consistency and reliability when feeding the data into machine learning models, particularly when the features extracted from both datasets need to be comparable or combined for effective classification.

In order to accomplish this, the following steps were taken:

- First, the time differences between the timestamps in the gyroscope dataframe were calculated.
- Then, this time difference was divided by the total number of samples minus one, to derive the estimated sampling interval. This is because the time

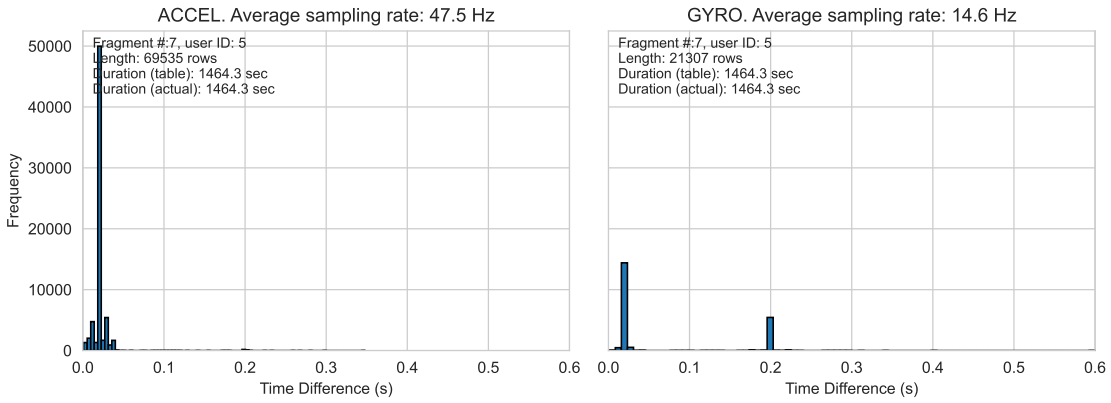

Figure 4.9 *Histograms of sampling periods, fragment "7", user ID: 5*

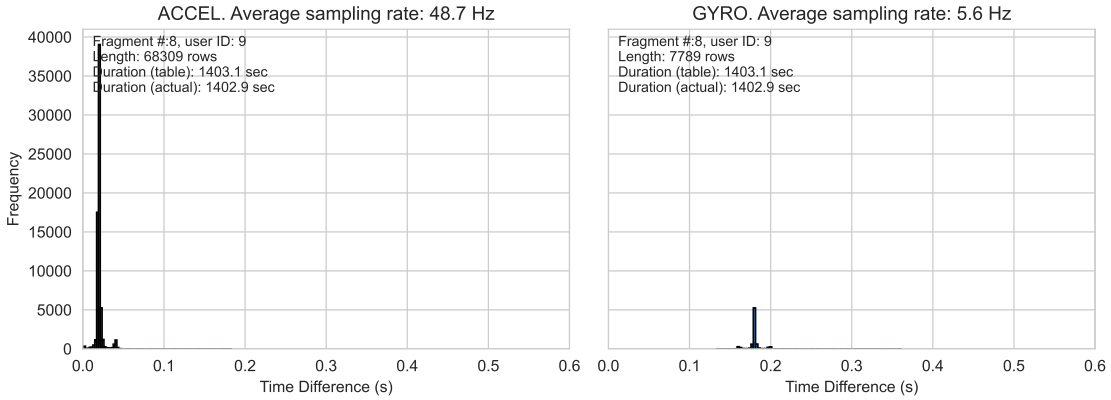

Figure 4.10 *Histograms of sampling periods, fragment "8", user ID: 9*

difference between n samples includes $n - 1$ intervals.

- Finally, the accelerometer data was resampled based on this interval.

It was discovered that there is an issue with a fragment of gyroscope sensor data, specifically labeled as "7". Upon analysis, it was found that the sampling period histogram shows several peaks, similar to the ones observed in the accelerometer data.

Resampling of accelerometer data

Resampling adjusts dataset's sample rate by adding/removing samples within fixed duration.

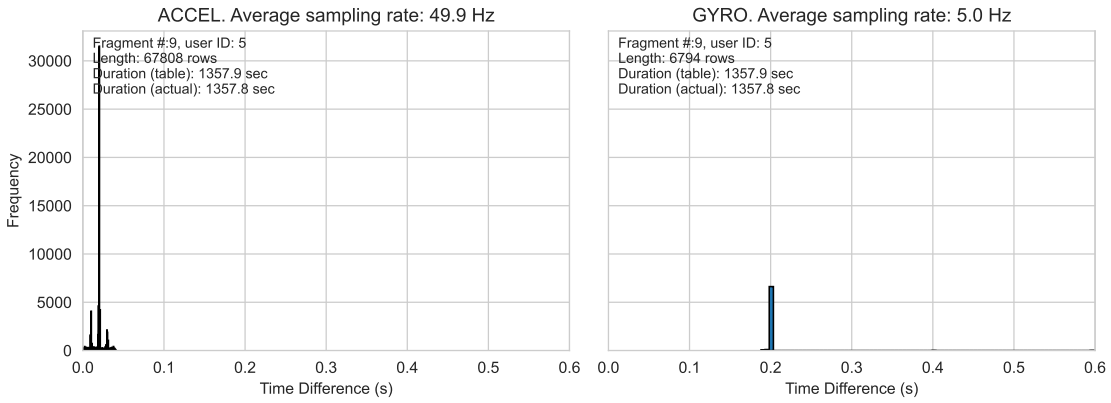

Figure 4.11 *Histograms of sampling periods, fragment "9", user ID: 5*

The sampling interval Δt, is the time duration between two consecutive samples. Its reciprocal gives the sampling frequency f_s, which is represented in equation 4.4:

$$f_s = \frac{1}{\Delta t} \tag{4.4}$$

Linear interpolation is a method used to estimate values between two known values by drawing a straight line between the two points (x_1, y_1) and (x_2, y_2).

For a value x in the interval $[x_1, x_2]$, the value y along the straight line is given by equation 4.5:

$$y = y_1 + \frac{(x - x_1) \times (y_2 - y_1)}{x_2 - x_1} \tag{4.5}$$

4.1.5 Resampling Technique

The goal of resampling is to change the sampling frequency of accelerometer data by interpolating it at new time intervals derived from the gyroscope data.

When a signal that was originally sampled at time intervals of Δt_1 needs to be resampled at different intervals of Δt_2, a common method is to use the aforementioned linear interpolation formula 4.5 for all new time points that fall between two consecutive original sample time points.

The Python code below resamples accelerometer data based on gyroscope data parameters. The code annotations provide additional insights for resampling accelerometer data based on gyroscope data parameters:

```
def get_resampled_accel_df(accel_time_idx_df, gyro_time_idx_df):
    acc_temp_idx = accel_time_idx_df.copy()
    gyr_temp_idx = gyro_time_idx_df.copy()

    # Sorting the dataframe by the index
    gyr_time_diff = gyr_temp_idx.index.to_series().diff().dt.
    total_seconds().dropna()

    # Calculate the sampling interval in microseconds
    gyr_time_diff_us = gyr_time_diff * 1_000_000
    gyro_avg_sampling_interval = gyr_time_diff_us.mean()

    # Resample the gyroscope data with a new sampling interval
    # with linear interpolation of missed values if any
    resampled_df = acc_temp_idx.resample(f'{round(
    gyro_avg_sampling_interval)}U').mean(numeric_only=True).
    interpolate(method='linear')

    # Convert needed columns back to integers
    resampled_df['id'] = resampled_df['id'].astype(int)
    resampled_df['username'] = resampled_df['username'].astype(int)
    resampled_df['activity_id'] = resampled_df['activity_id'].
    astype(int)
    resampled_df['activity'] = 'Walking'
    return resampled_df
```

```
for i in top_8_fragments_for_resampling:
    accel_idx_df = accel_user_walk[i].get('data_idx_df')
    gyro_idx_df = gyro_user_walk[i].get('data_idx_df')
    accel_original_actual_duration = get_df_duration_in_sec(
    accel_idx_df)
```

```
    accel_idx_df_resampled = get_resampled_accel_df(accel_idx_df,
gyro_idx_df)
    accel_resampled_actual_duration = get_df_duration_in_sec(
accel_idx_df_resampled)
    gyro_actual_duration = get_df_duration_in_sec(gyro_idx_df)

    # Update the dictionary with resampled data frame
    accel_user_walk[i]['data_idx_df_resampled'] =
accel_idx_df_resampled

    # Calculating the differences between timestamps
    accel_time_diffs = accel_idx_df_resampled.index.to_series().
diff().dt.total_seconds()
    accel_avg_sampling_rate = 1 / accel_time_diffs.mean()
    gyro_time_diffs = gyro_idx_df.index.to_series().diff().dt.
total_seconds()
    gyro_avg_sampling_rate = 1 / gyro_time_diffs.mean()
```

Figures 4.12 to 4.19 show histograms of sampling periods for the "Walking" segments in the Top-8 list for both sensor datasets after resampling the accelerometer data:

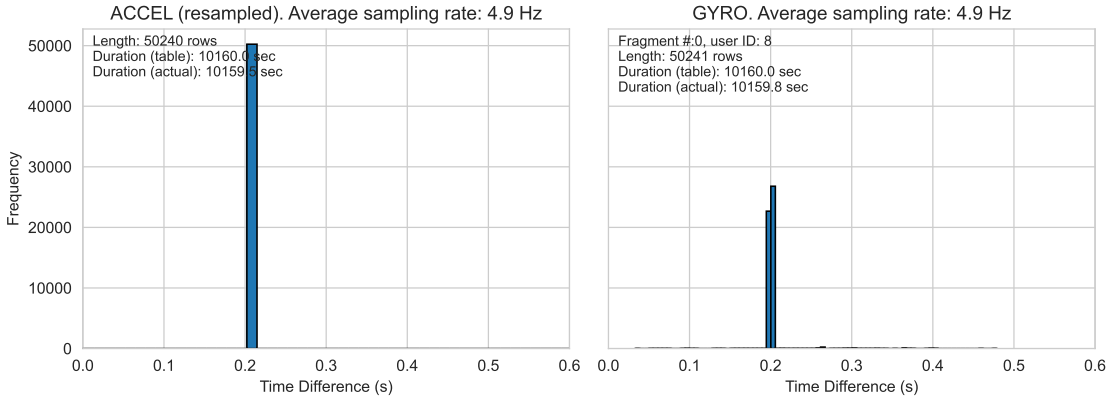

Figure 4.12 *Histograms of sampling periods, fragment "0", user ID: 8*

After resampling, the accelerometer data is now more aligned with the gyroscope data in terms of histogram distributions and sampling rate, which ensures uniformity

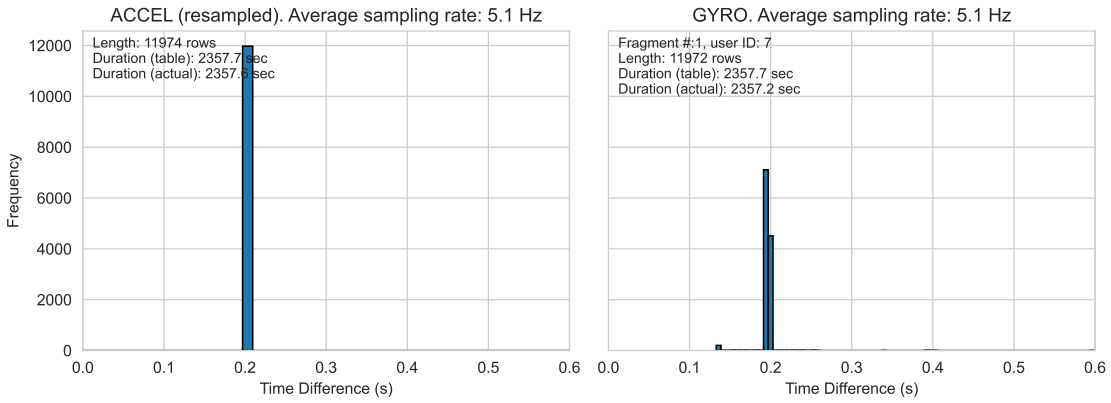

Figure 4.13 *Histograms of sampling periods, fragment "1", user ID: 7*

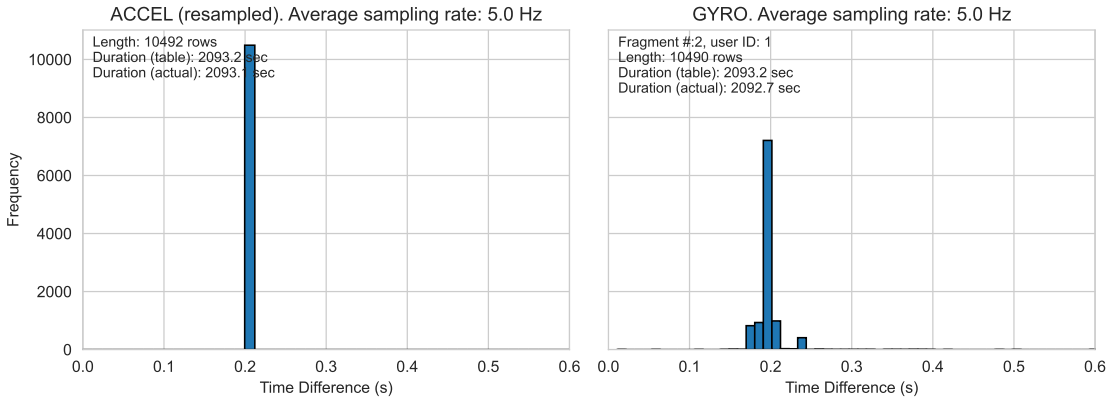

Figure 4.14 *Histograms of sampling periods, fragment "2", user ID: 1*

across the different sensor data dimensions.

4.2 Training and Testing Datasets

For this thesis project, the main objective was to create a method that could differentiate specific users from a larger pool of users. To ensure the machine learning model was robust and versatile, its performance was tested on data that was not used for training. As a result, the data was split into two categories - training data and testing data. The model was trained on 75% of the data, allowing it to identify patterns in the walking data of users. The remaining 25% was assigned for testing to evaluate the model's performance on unfamiliar data objectively. This approach ensures that the model can accurately identify users in real-world situations, instead

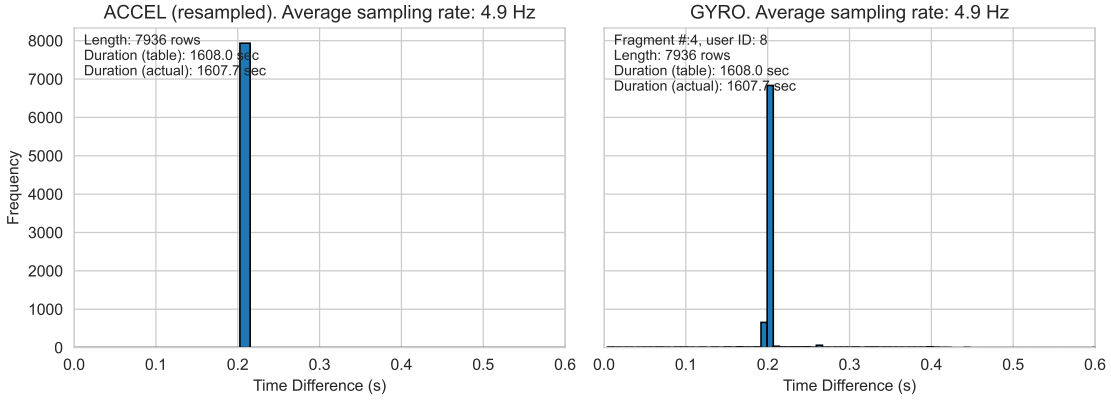

Figure 4.15 *Histograms of sampling periods, fragment "4", user ID: 8*

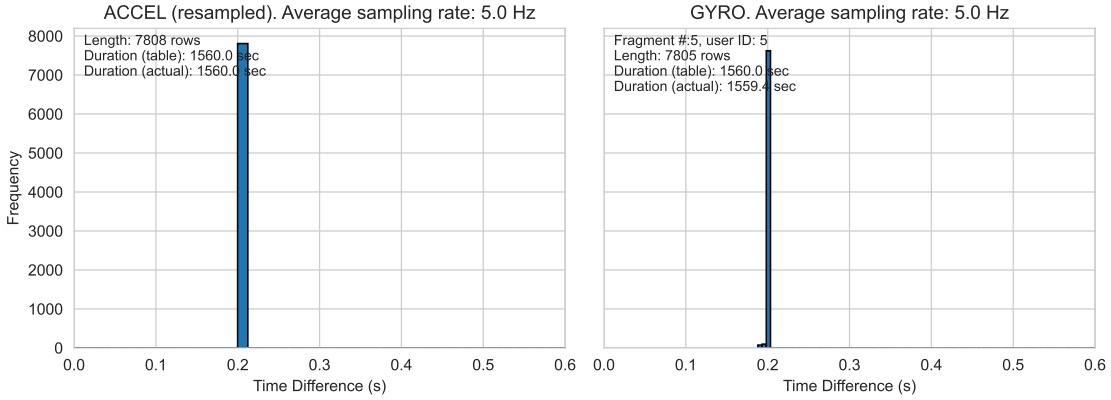

Figure 4.16 *Histograms of sampling periods, fragment "5", user ID: 5*

of adapting only to the training data's nuances.

The test dataset was kept entirely separate from the training dataset to ensure evaluations were conducted on previously unobserved data segments.

Figure 4.20 displays the arrangement of the training and testing datasets.

4.3 Feature Engineering on Time-Series Data

4.3.1 "Windowing" of Time-Series Data

Time-series data, which is collected from devices such as accelerometers and gyroscopes, is inherently sequential and continuous. However, analyzing such raw data

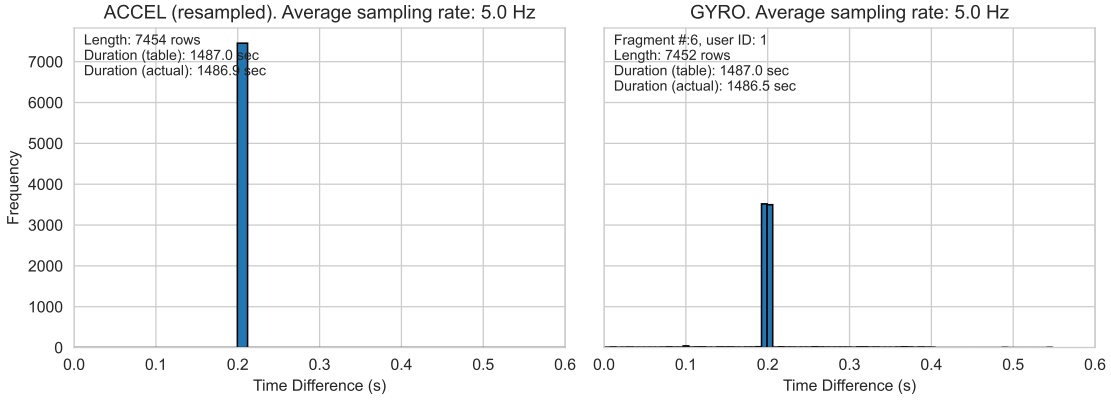

Figure 4.17 *Histograms of sampling periods, fragment "6", user ID: 1*

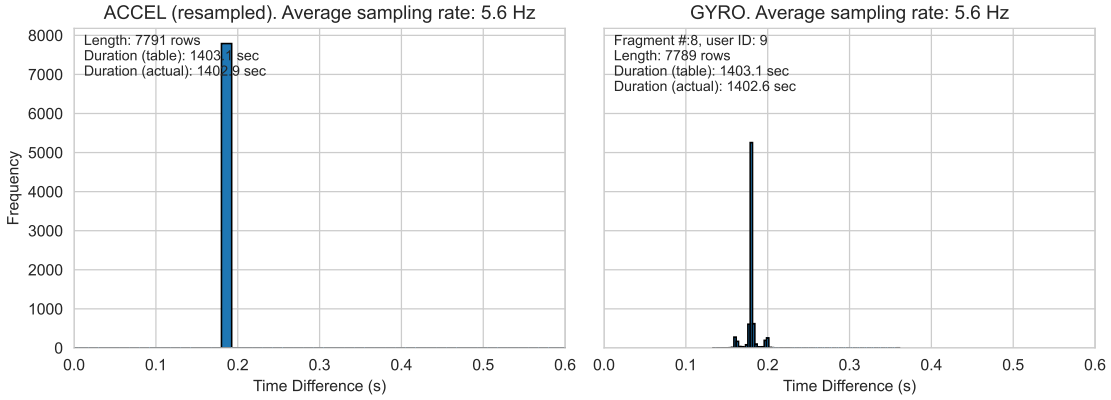

Figure 4.18 *Histograms of sampling periods, fragment "8", user ID: 9*

with machine learning models can pose challenges in terms of computational efficiency and significant pattern recognition. This is where feature engineering comes in. Through feature engineering, raw data is transformed into a more structured and representative format, which allows algorithms to identify patterns, trends, or anomalies with greater accuracy. This transformation makes it possible to extract more meaningful insights from the time-series data.

The process involves the segmentation of time-series data using a technique called "windowing". Each window spans a duration of 20 seconds, as shown in Figure 4.21. To ensure that there is a continuous flow of information and to account for the continuous nature of the data, overlapping windows are used instead of discrete ones. These overlapping windows have a 50% overlap and help to capture transitional

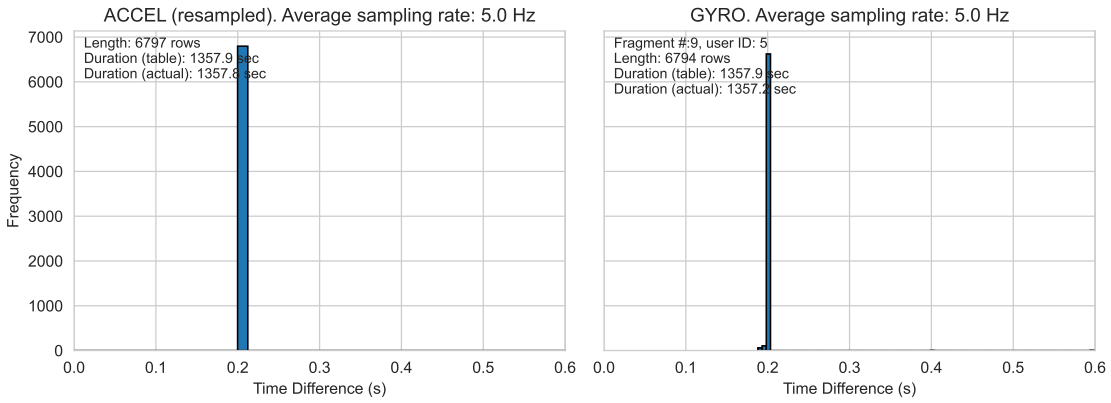

Figure 4.19 *Histograms of sampling periods, fragment "9", user ID: 5*

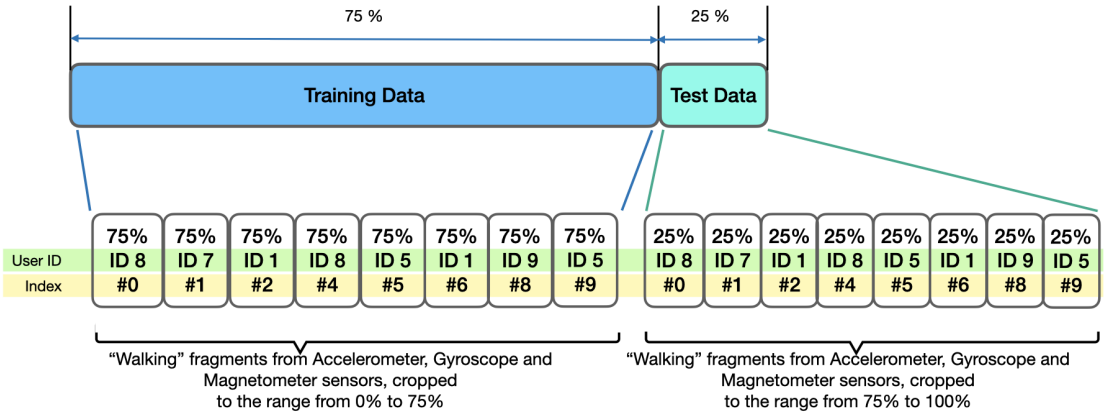

Figure 4.20 *Training and testing datasets layout*

patterns, thus reducing the risk of missing important information between discrete windows. This technique ensures that each subsequent row in the transformed dataset carries over some information from the previous window, resulting in a seamless flow of data.

The specific window duration chosen may be questioned by some. However, two reasons support this choice. Firstly, since the sensors' average sampling rate is five Hz, 100 samples amount to 20 seconds of data. Secondly, a 20-second duration is considered optimal for capturing the repetitive motions inherent to human walking. If the window is too short, it might not provide a complete snapshot of the motion. Conversely, an excessively large window could dilute the specific characteristics of the motion and reduce the number of data points in the transformed dataset. This

could compromise the training phase, potentially leading to suboptimal results.

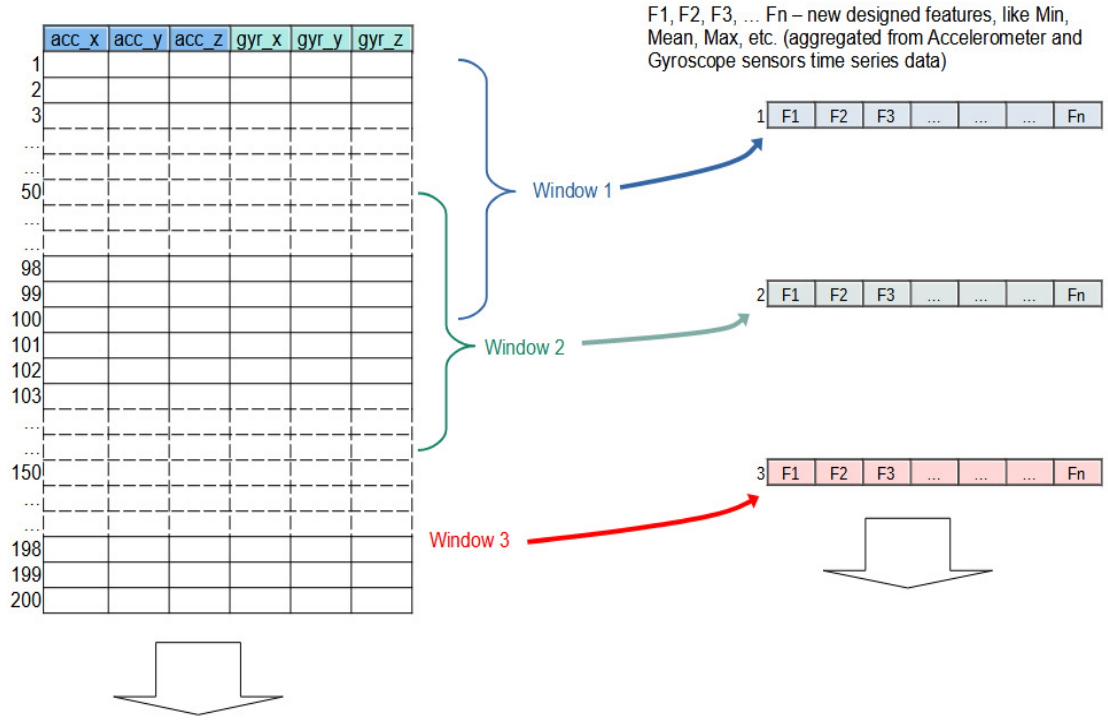

Figure 4.21 *Feature engineering of sensors time-series data*

By capturing 100 raw samples within each window, the aim is to extract relevant features that encapsulate the core characteristics of the data.

To assign class labels to each window, the most common user ID within the window is used. It is crucial to ensure that the extracted features are accurately correlated with the corresponding user IDs. This is particularly important for tasks that require user-specific analysis and classification.

The following Python code demonstrates a general approach to processing data from two sensor datasets, which includes the use of the windowing technique to analyze time-series data:

```
sensor_x = []
sensor_y = []
sensor_z = []
sensor_labels = []

```

```
for i in range(0, sensor_set_df.shape[0] - window_size_samples,
    step_size_samples):
     xs = sensor_set_df['sensor_x_axis'].values[i: i +
    window_size_samples]
     ys = sensor_set_df['sensor_y_axis'].values[i: i +
    window_size_samples]
     zs = sensor_set_df['sensor_z_axis'].values[i: i +
    window_size_samples]

     user_ids = sensor_set_df['username'].values[i: i +
    window_size_samples]
     label = np.argmax(np.bincount(user_ids))
     sensor_x.append(xs)
     sensor_y.append(ys)
     sensor_z.append(zs)
     sensor_labels.append(label)
```

4.3.2 List of Statistical Features

The complete list of statistical features employed in this feature engineering process includes:

- *Min*: Smallest value in the window.

$$Min(X) = \min_{i=1}^{n} x_i \tag{4.6}$$

  where X is the set of all data points in the window, and x_i is an individual data point.

- *Max*: Largest value in the window.

$$Max(X) = \max_{i=1}^{n} x_i \tag{4.7}$$

where X is the set of all data points in the window, and x_i is an individual data point.

- *Difference of maximum and minimum values*: The range of values within the window.

$$Range(X) = \max(X) - \min(X) \tag{4.8}$$

where X is the set of all data points in the window.

- *Mean*: Average of the values.

$$\mu(X) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.9}$$

where X represents the set of all data points in the window, and n is the total number of data points in X.

- *Average absolute deviation*: Measures the dispersion of data.

$$AAD(X) = \frac{1}{n}\sum_{i=1}^{n} |x_i - \mu(X)| \tag{4.10}$$

where X represents the set of all data points in the window, n is the total number of data points in X and $\mu(X)$ is the mean of the data points in X.

- *Standard deviation*: Measures how spread out data is from its average value.

$$\sigma(X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu(X))^2} \tag{4.11}$$

where X represents the set of all data points in the window, n is the total number of data points in X and $\mu(X)$ is the mean of the data points in X.

- *Median*: The middle value when data is arranged in order.

$$Median(X) = \begin{cases} x_{\frac{n+1}{2}} & \text{if } n \text{ is odd} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} & \text{if } n \text{ is even} \end{cases} \tag{4.12}$$

where X is the sorted version of the data in ascending order and n is the total number of data points in X.

- *Median absolute deviation*: This measure is calculated by finding the median of the absolute deviations from the median of the data.

$$MAD(X) = \text{med}(|x_i - \text{med}(X)|) \tag{4.13}$$

where X is the dataset, x_i is each data point in X, $\text{med}(X)$ is the median of X and $|x_i - \text{med}(X)|$ represents the absolute deviation of each data point from the median of X.

- *Interquartile range*: Range between the 25th and 75th percentile.

$$IQR = Q_3 - Q_1 \tag{4.14}$$

where Q_3 is the 75th percentile (or the third quartile) and Q_1 is the 25th percentile (or the first quartile).

- *Negative count*: Number of values below zero.

$$NegativeCount = \sum_{i=1}^{n} \mathbf{1}(x_i < 0) \tag{4.15}$$

where n is the total number of data points in the window, x_i is the value of the i^{th} data point and $\mathbf{1}$ is the indicator function that returns 1 if the condition inside the parenthesis is true, and 0 otherwise.

- *Positive count*: Number of values above zero.

$$PositiveCount = \sum_{i=1}^{n} \mathbf{1}(x_i > 0) \tag{4.16}$$

where n is the total number of data points in the window, x_i is the value of the i^{th} data point and $\mathbf{1}$ is the indicator function that returns 1 if the condition inside the parenthesis is true, and 0 otherwise.

- *Values above mean*: Count of values that are greater than the mean.

$$ValuesAboveMean = \sum_{i=1}^{n} \mathbf{1}(x_i > \bar{x}) \tag{4.17}$$

where n is the total number of data points in the window, x_i is the value of the i^{th} data point, $\bar{x}$ is the mean of all data points and $\mathbf{1}$ is the indicator function that returns 1 if the condition inside the parenthesis is true, and 0 otherwise.

- *Skewness*: Measures the asymmetry of the data distribution.

$$Skewness(X) = \frac{E[(X - \mu)^3]}{\sigma^3} \tag{4.18}$$

where E represents the expectation (the "average" or "expected value"), X represents the random variable (or the dataset), μ is the mean of X and σ is the standard deviation of X.

- *Kurtosis*: Quantifies the degree of heaviness or thickness of the tails of a probability distribution.

$$Kurt[X] = E\left[\left(\frac{X - \mu}{\sigma}\right)^4\right] - 3 \tag{4.19}$$

where X is the random variable, μ is the mean of X and σ is the standard deviation of X.

- *Number of peaks*: Indicates data's variability.

$$X_i \text{ is a peak if } X_i > X_{i-1} \text{ and } X_i > X_{i+1}$$
$$\text{Number of Peaks} = |\{i : X_i \text{ is a peak}\}| \tag{4.20}$$

- *Energy*: Total energy (mean of sum of squares of the values) of the signal.

$$E(X) = \frac{1}{100} \sum_{i=1}^{100} X_i^2 \tag{4.21}$$

- *Average resultant*: Average magnitude of the vector sum of its components.

$$R_i = \sqrt{X_i^2 + Y_i^2 + Z_i^2} \tag{4.22}$$
$$AvgResultant = \frac{1}{n} \sum_{i=1}^{n} R_i \tag{4.23}$$

  where X, Y and Z are the components of a 3-dimensional vector.

- *Signal magnitude area*: Represents the total area under a signal.

$$SMA = \frac{1}{100} \left(\sum_{i=1}^{n} |X_i| + \sum_{i=1}^{n} |Y_i| + \sum_{i=1}^{n} |Z_i| \right) \tag{4.24}$$

4.3.3 The Resulting Training and Testing Data Frames

After performing feature engineering, the datasets were divided into two distinct dataframes: *users_training_df* and *users_testing_df*.

Accelerometer and gyroscope sensors were used to collect data in both dataframes. The prefix "*acc_*" identifies accelerometer columns, while "*gyr_*" indicates gyroscope columns. The columns contain several statistical metrics, including minimum, maximum, energy, and signal magnitude area.

Both dataframes contain complete and well-preprocessed datasets as all the columns

are non-null, as illustrated below:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1656 entries, 0 to 1655
Data columns (total 98 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   acc_x_min           1656 non-null   float64
 1   acc_y_min           1656 non-null   float64
 2   acc_z_min           1656 non-null   float64
 3   gyr_x_min           1656 non-null   float64
 4   gyr_y_min           1656 non-null   float64
 5   gyr_z_min           1656 non-null   float64
 6   acc_x_max           1656 non-null   float64
 7   acc_y_max           1656 non-null   float64
 8   acc_z_max           1656 non-null   float64
 9   gyr_x_max           1656 non-null   float64
 10  gyr_y_max           1656 non-null   float64
 11  gyr_z_max           1656 non-null   float64
 12  acc_x_maxmin_diff   1656 non-null   float64
 13  acc_y_maxmin_diff   1656 non-null   float64
...
 93  gyr_z_energy        1656 non-null   float64
 94  avg_result_acc      1656 non-null   float64
 95  avg_result_gyr      1656 non-null   float64
 96  acc_sma             1656 non-null   float64
 97  gyr_sma             1656 non-null   float64
dtypes: float64(74), int64(24)
memory usage: 1.2 MB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 551 entries, 0 to 550
Data columns (total 98 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   acc_x_min           551 non-null    float64
```

```
 1   acc_y_min           551 non-null    float64
 2   acc_z_min           551 non-null    float64
 3   gyr_x_min           551 non-null    float64
 4   gyr_y_min           551 non-null    float64
 5   gyr_z_min           551 non-null    float64
 6   acc_x_max           551 non-null    float64
 7   acc_y_max           551 non-null    float64
 8   acc_z_max           551 non-null    float64
 9   gyr_x_max           551 non-null    float64
 10  gyr_y_max           551 non-null    float64
 11  gyr_z_max           551 non-null    float64
 12  acc_x_maxmin_diff   551 non-null    float64
 13  acc_y_maxmin_diff   551 non-null    float64
...
 93  gyr_z_energy        551 non-null    float64
 94  avg_result_acc      551 non-null    float64
 95  avg_result_gyr      551 non-null    float64
 96  acc_sma             551 non-null    float64
 97  gyr_sma             551 non-null    float64
dtypes: float64(74), int32(18), int64(6)
memory usage: 383.2 KB
```

After applying the windowing technique, the number of unique user IDs reduced to 5 as the window was labeled with the most frequently occurring ID:

```
acc_training_unique_labels: [1 5 7 8 9]
gyr_training_unique_labels: [1 5 7 8 9]
acc_testing_unique_labels: [1 5 7 8 9]
gyr_testing_unique_labels: [1 5 7 8 9]
```

Feature engineering is a crucial aspect of machine learning, especially when dealing with time-series data. In this process, raw data is transformed into meaningful features that can be more efficiently processed by machine learning models. By creating a dataset that is both representative and informative, the accuracy of learning, classification, and prediction is significantly improved. If this process is ignored, the

efficiency and precision of a machine learning model's output can be severely impacted.

4.4 Empirical analysis of classification methods for time-series data

In this concluding chapter, a systematic examination of three prominent classification models specifically tailored for time-series analysis is conducted. The importance of meticulously evaluating the effectiveness and precision of each model in differentiating among individual users is underscored, given the intricate nature of the dataset derived from accelerometer and gyroscope readings captured by a smartphone.

For an in-depth analysis, two distinct parameter tuning methods are applied to each classification model. This methodical procedure culminates in six comprehensive tests, aimed at determining the optimal configuration for achieving the highest level of accuracy and efficiency in user recognition.

The analytical endeavor is underpinned by two primary objectives. The first objective is to identify the best model and tuning combination specifically tailored for this dataset. Simultaneously, the second objective is to explore the wider implications and intricacies of parameter configuration variations and their subsequent impact on model efficacy.

4.4.1 Logistic Regression Model

Logistic Regression (LR) is an algorithm that is commonly used to classify data into two distinct labels. However, it can also be modified to work for multi-class classification. The way it works is by calculating the probability of a given input belonging to a specific class.

If the probability of a given input belonging to a specific class is greater than a

certain threshold, which is usually set to 0.5, then the algorithm will predict that the input belongs to that class. On the other hand, if the probability is less than the threshold, the algorithm will predict that the input belongs to the other class.

Feature Set Standardizing

To ensure accurate LR models, it is important to standardize features by centering them around zero and giving them a standard deviation of one, using the standard score formula:

$$z = \frac{x - \mu}{\sigma} \tag{4.25}$$

where μ is the mean of the training samples, and σ is the standard deviation of the training samples.

To maintain consistency in feature sets, the `StandardScaler()` method from `scikit-learn`, a well-known Python library for data analysis and data mining, is utilized before training the model. The fit method is used to calculate the mean and standard deviation of the training data, and the features are standardized using the transform method.

To ensure consistent scaling between the training and testing data, the scaler is fitted solely to the training data. The same scaling is subsequently applied to both datasets. Care is taken not to fit the scaler to the testing data, which can result in potential data leakage and inaccurate test results. The same transformation applied to the training data is then used on the testing data.

An example Python code illustrating the scaling process is provided below:

```
y_train = np.array(training_labels_list)
y_test = np.array(testing_labels_list)

```

```
scaler = StandardScaler()
scaler.fit(training_data_df)
x_train_data_lr = scaler.transform(training_data_df)
x_test_data_lr = scaler.transform(testing_data_df)
```

The variable `x_train_data_lr` mentioned earlier is a standardized version of the `training_data_df`. This standardized data will be used to train the LR model. Standardization of the training data makes sure that all features are on the same scale. This helps the LR model to converge faster and find a better solution.

Similarly, the `x_test_data_lr` is the standardized version of the `testing_data_df`. This standardized test data is crucial for evaluating the LR model. To ensure that the model is evaluated fairly, it is important to use the same scaling, including mean and standard deviation, that was applied to the training data.

Model Training

The provided Python code trains an LR model on standardized training data and uses the model to predict labels for standardized testing data:

```
logistic_regression = LogisticRegression(random_state = 21,
   max_iter=1000)
logistic_regression.fit(x_train_data_lr, y_train)
logistic_regression_default_y_pred = logistic_regression.predict(
   x_test_data_lr)
logistic_regression_default_report = classification_report(y_test,
   logistic_regression_default_y_pred, digits=4)
```

LR model training (with default parameters) classification report:

```
              precision    recall  f1-score   support

           1     0.7812    0.8427    0.8108        89
           5     0.8353    0.9861    0.9045        72
           7     1.0000    0.9833    0.9916        60
```

```
           8      0.9712    0.9278    0.9490       291
           9      1.0000    0.8462    0.9167        39

    accuracy                         0.9220       551
   macro avg      0.9176    0.9172    0.9145       551
weighted avg      0.9279    0.9220    0.9232       551
```

- **Accuracy**: This ratio reflects the number of correct predictions made by the model in comparison to the total number of observations. It is a good measure when the classes of the target variable in the data are balanced:

$$Accuracy = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Observations}} \tag{4.26}$$

- **Precision**: When it comes to measuring the accuracy of a classification, Precision is a commonly used metric. Essentially, it measures how many positive outcomes were predicted correctly out of all the predictions that were made. Another term for Precision is Positive Predictive Value. To calculate Precision, the following formula is used:

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{4.27}$$

- **Recall**: The *Recall*, also known as *Sensitivity*, *Hit Rate* or *True Positive Rate*, and it represents the proportion of correct positive predictions to all actual positive observations in a class. In order to calculate Recall, the following formula is used:

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4.28}$$

- **F1-Score**: The *F1-Score* is a metric that evaluates the performance of a classifier. It is calculated as the weighted average of Precision and Recall, taking into account false positives and false negatives. This score provides an accurate assessment of the classifier's value for both recall and precision. In order to calculate F1-Score, the following formula is used:

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.29}$$

- **Support**: When analyzing a dataset, the term *Support* refers to the number of instances of a particular class. In a classification report, the support for each class indicates the number of instances of that class in the actual data.

- **Macro Avg**: The *Macro Average* calculates the metric for each class and averages the results, treating all classes equally.

- **Weighted Avg**: The *Weighted Average* calculates the average of metrics by weighting the score of each class based on its presence in the true data sample.

Confusion Matrix

The confusion matrix is a valuable tool for assessing the accuracy of a classification model. It is commonly used when the true values of a dataset are known. The matrix is constructed with rows and columns, where each row represents the predicted instances within a specific class and each column represents the actual instances within a specific class. This organization helps evaluate the model's performance by

identifying true positives, false positives, true negatives, and false negatives.

In a binary classification task, a confusion matrix will be a 2x2 table as shown in table 4.1:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 4.1 *Confusion Matrix for Binary Classification*

- *True Positives (TP)*: True positives are cases where the actual and predicted class values are both 1.
- *True Negatives (TN)*: True negatives are values that are correctly predicted as negative, indicating that the actual and predicted class values are both 0
- *False Positives (FP)*: A false positive occurs when the predicted class is 1, but the actual class is 0.
- *False Negatives (FN)*: False negatives occur when the predicted class is 0, but the actual class is 1.

When dealing with tasks that involve multiple categories, the confusion matrix expands to accommodate the greater number of prediction categories. Despite the increase in size, the fundamental principle remains consistent. Each matrix entry specifies the count of predictions for a particular category versus its actual occurrences.

In the current classification task, there are five distinct categories labeled as one, five, seven, eight, and nine. Therefore, the confusion matrix will have a size of 5x5. The diagonal, running from the top left to the bottom right, displays the True Positives, which are instances where the model's predictions align with the actual categories. Conversely, off-diagonal entries indicate where the model has made inaccuracies.

Here is a Python code that uses default parameters to create a visualization of an LR model's prediction results, as shown in Figure 4.22.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
labels = np.unique(training_labels_list)
log_reg_confusion_mat = confusion_matrix(y_test,
    logistic_regression_default_y_pred)
sns.heatmap(log_reg_confusion_mat, xticklabels=labels, yticklabels=
    labels, annot=True, linewidths = 0.2, fmt='d', cmap = 'BuPu')
```

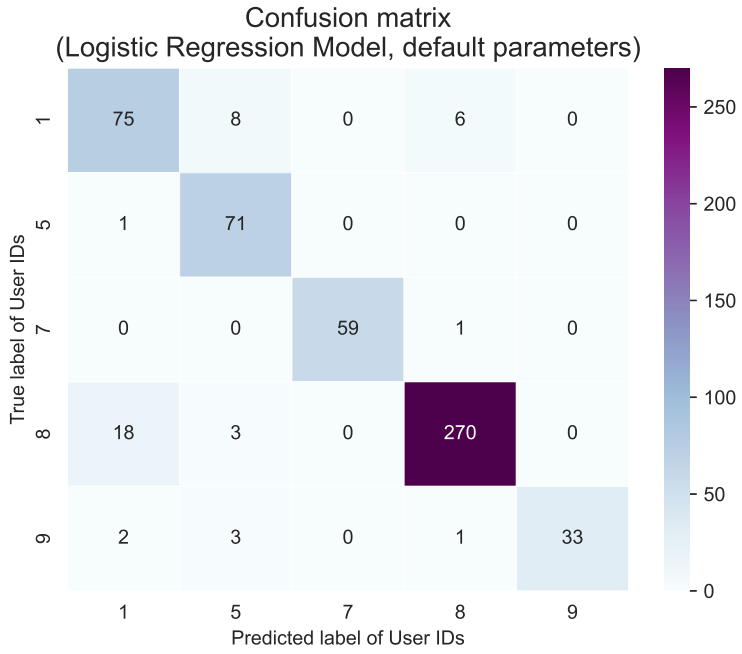

Figure 4.22 *Confusion Matrix of Logistic Regression (default parameters)*

A conclusion can be drawn from the provided confusion matrix that the LR model has performed relatively well in predicting user IDs. A high level of accuracy was observed across the diagonal of the matrix, which represents correct predictions or True Positives. User IDs 7, 5, and 9 were predicted with 59, 71, and 33 correct predictions, respectively, while User ID 8 had the highest number of correct predictions, with a total of 270. However, there are a few misclassifications that need to be taken into consideration.

User ID 1 was misclassified eight times as user ID 5 and 6 times as user ID 8.

Similarly, User ID 8 was mispredicted 18 times as user ID 1. User ID 9 had only minor misclassifications to user IDs 1, 5, and 8.

Although good accuracy was demonstrated for several user IDs, there is a struggle to differentiate between user IDs 1 and 8, which suggests that there is room for improvement. By tuning the model, its accuracy might be enhanced.

Optimizing the model parameters with GridSearchCV

Tuning hyperparameters is crucial for achieving the best model performance. *GridSearchCV* is a useful technique that facilitates the process of hyperparameter tuning. This method involves exhaustive search through a set parameter grid, where the model is trained for each parameter combination. The parameters that produce the best results are selected based on a cross-validated metric.

When working with an LR model, it is recommended to use GridSearchCV as it provides several advantages. Firstly, it helps to find the most suitable parameters for the model, resulting in a more accurate and robust model. This is particularly important for LR, which has several hyperparameters such as the regularization strength (C), penalty type (penalty), and solver algorithm (solver).

Additionally, GridSearchCV performs k-fold cross-validation during the search. This process helps in selecting parameters that generalize well to unseen data. This is crucial in avoiding overfitting the model to the training data.

Moreover, the parameter selection process is automated, saving significant time and effort, especially when there are multiple hyperparameters with many possible values.

Lastly, the flexibility of specifying any performance metric (accuracy, precision, recall, etc.) to select the best parameters allows for customization based on the specific requirements of the problem at hand.

GridSearchCV with an LR model selects optimal hyperparameters based on given performance metric, resulting in a more accurate and robust model.

Here is a Python code that utilizes GridSearchCV with the LR model:

```
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.linear_model import LogisticRegression

# Define the parameter grid to search over
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l2'],
    'solver': ['liblinear', 'lbfgs']
}

logistic_regression = LogisticRegression(max_iter=1000)
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5)

# Fit the GridSearchCV instance the original data
grid_search.fit(x_train_data_lr, y_train)
print("Best parameters:", grid_search.best_params_)

# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Predict using the best model
logistic_regression_grid_search_y_pred = best_model.predict(
    x_test_data_lr)
```

The outcome of the GridSearchCV optimization process is as follows:

```
Best parameters: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
```

```
logistic_regression_gridsearch_report = classification_report(
    y_test, logistic_regression_grid_search_y_pred, digits=4)
```

Here is the classification report for the model training after the optimization with

GridSearchCV:

```
              precision    recall  f1-score   support

           1     0.7429    0.8764    0.8041        89
           5     0.9103    0.9861    0.9467        72
           7     1.0000    0.9833    0.9916        60
           8     0.9710    0.9210    0.9453       291
           9     1.0000    0.8462    0.9167        39

    accuracy                         0.9238       551
   macro avg     0.9248    0.9226    0.9209       551
weighted avg     0.9314    0.9238    0.9257       551
```

Here is a Python code that uses default parameters to create a visualization of an LR model's prediction results after the optimization with GridSearchCV, as shown in Figure 4.23.

```
labels = np.unique(training_labels_list)
log_reg_confusion_mat = confusion_matrix(y_test,
   logistic_regression_grid_search_y_pred)
sns.heatmap(log_reg_confusion_mat, xticklabels=labels, yticklabels=
   labels, annot=True, linewidths = 0.2, fmt='d', cmap = color_map)
```

Analysis of Logistic Regression model results

Classification Report

The optimized parameter model using GridSearchCV slightly outperforms the default model with an accuracy improvement from 92.2% to 92.38%.

The model's ability to identify positive samples in Class 1 improved after optimization, with a recall increase from 84.27% to 87.64%. However, there was a slight decrease in precision from 78.12% to 74.29%, indicating an increase in false positives.

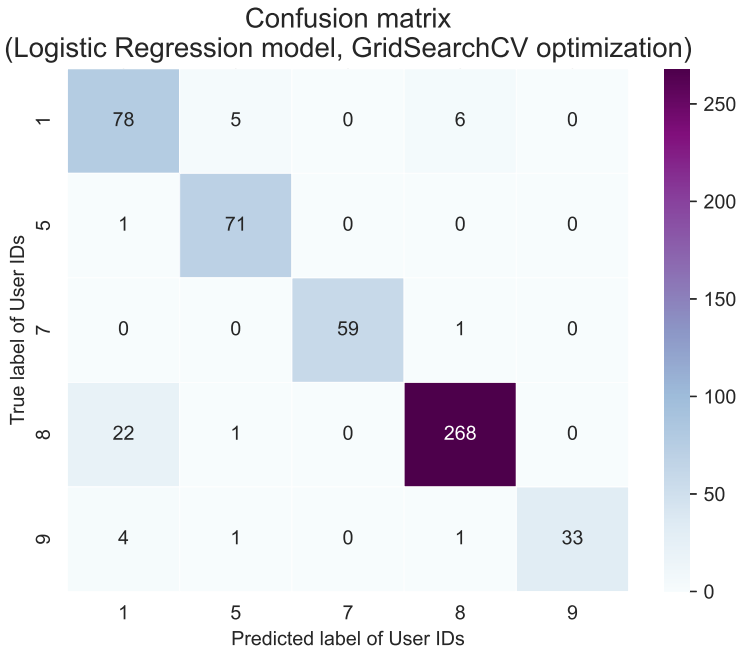

Figure 4.23 *Confusion Matrix of Logistic Regression (GridSearchCV)*

On the other hand, precision for Class 5 saw a significant increase from 83.53% to 91.03%, and there was a small increase in recall from 98.61% to 98.61%. Scores remained the same for Class 7 and Class 9. Class 8 saw a slight decrease in recall from 92.78% to 92.10%, but precision remained almost the same, dropping from 97.12% to 97.10%.

Confusion Matrix

There were 75 true positives in Class 1, which increased to 78. However, the number of false negatives went up from 18 to 22.

In Class 5, the number of false positives decreased from 8 to 5, which is a positive change. The confusion matrix remained unchanged for Class 7 and Class 9. In Class 8, the number of false negatives went up from 18 to 22, but the number of false positives decreased from 3 to 1.

Overall Performance

After optimization, an improvement was observed in the performance of the model.

However, in certain classes, there was an increase in precision but a decrease in recall, and vice versa. For instance, in Class 1, recall increased while precision decreased. Conversely, in Class 5, both precision and recall increased, which is considered a positive development.

It is crucial to determine which factor is more significant for the specific usage scenario, precision, or recall since there is often a trade-off between the two.

Conclusion

In conclusion, optimizing the LR model slightly improved its performance. However, it is important to consider the trade-offs in precision and recall for specific classes depending on the application.

4.4.2 Random Forest

Random Forest (RF) is a machine learning technique that employs a collection of decision trees to classify or predict results. Unlike LR, which is a linear model, RF has the ability to capture complex non-linear relationships in the data. Each tree in the Random Forest generates its own prediction, and the final output is determined by combining these predictions. For classification tasks, the most frequently predicted class among all the trees is taken as the final prediction, while for regression, the final prediction is typically the average of the predictions.

While LR models are robust when the relationship between the features and the target variable can be approximated by a linear function, RF models excel in cases where the data has intricate and non-linear patterns. It can be applied to a wide range of tasks, including both classification and regression. Another advantage of RF over LR is its ability to handle a mix of numerical and categorical features without requiring much preprocessing. However, due to its ensemble nature, RF could require more computational resources and may not be as interpretable as a

simple LR model.

The provided Python code trains an RF model on standardized training data and uses the model to predict labels for standardized testing data:

```
from sklearn.ensemble import RandomForestClassifier

scaler = StandardScaler()
scaler.fit(training_data_df)
x_train_data_rf = scaler.transform(training_data_df)
x_test_data_rf = scaler.transform(testing_data_df)

random_forest = RandomForestClassifier(random_state=21)
random_forest.fit(x_train_data_rf, y_train)
random_forest_default_y_pred = random_forest.predict(x_test_data_rf
    )
random_forest_default_report = classification_report(y_test,
    random_forest_default_y_pred, digits=4)
```

RF model training (with default parameters) classification report:

```
              precision    recall  f1-score   support

           1     0.7188    0.7753    0.7459        89
           5     0.8250    0.9167    0.8684        72
           7     1.0000    0.9833    0.9916        60
           8     0.9684    0.9485    0.9583       291
           9     0.9677    0.7692    0.8571        39

    accuracy                         0.9074       551
   macro avg     0.8960    0.8786    0.8843       551
weighted avg     0.9127    0.9074    0.9087       551
```

Here is a Python code that uses default parameters to create a visualization of an RF model's prediction results, as shown in Figure 4.24.

```
labels = np.unique(training_labels_list)
```

```
random_forest_default_confusion_mat = confusion_matrix(y_test,
    random_forest_default_y_pred)
sns.heatmap(random_forest_default_confusion_mat, xticklabels=labels
    , yticklabels=labels, annot=True, linewidths = 0.2, fmt='d',
    cmap = color_map)
```

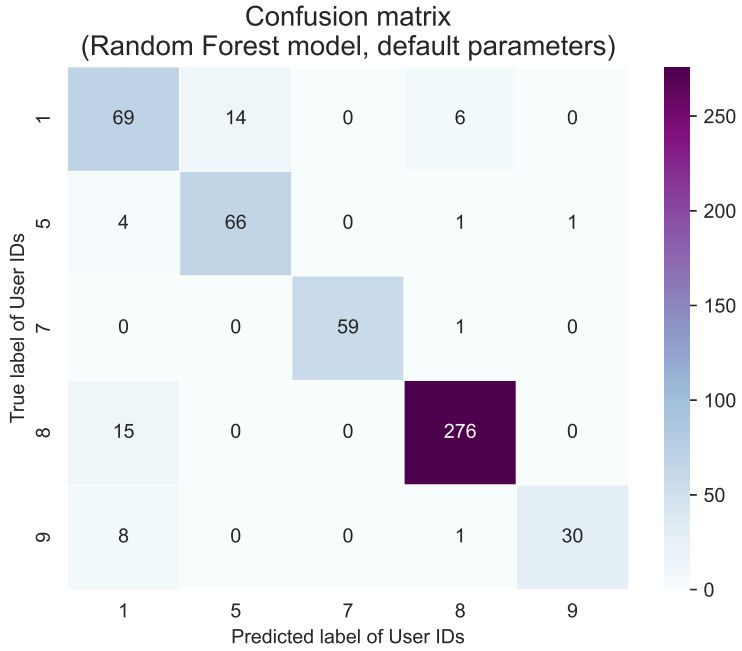

Figure 4.24 *Confusion Matrix of Random Forest (default parameters)*

Here is a Python code that utilizes GridSearchCV with the RF model:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search over
param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'max_features': ['sqrt'],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

```
13
14 random_forest = RandomForestClassifier(random_state=21)
15 grid_search = GridSearchCV(random_forest, param_grid, cv=5)
16
17 # Fit the GridSearchCV instance the original data
18 grid_search.fit(x_train_data_rf, y_train)
19 print("Best parameters:", grid_search.best_params_)
20
21 # Get the best model from the grid search
22 best_model = grid_search.best_estimator_
23
24 # Predict using the best model
25 random_forest_gridsearch_y_pred = best_model.predict(x_test_data_rf
   )
```

The outcome of the GridSearchCV optimization process is as follows:

```
1 Best parameters: {'bootstrap': False, 'max_depth': 20, '
   max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split
   ': 2, 'n_estimators': 50}
```

Here is the classification report for the model training after the optimization with GridSearchCV:

```
1 random_forest_gridsearch_report = classification_report(y_test,
   random_forest_gridsearch_y_pred, digits=4)
```

```
1                precision    recall  f1-score   support
2
3             1     0.7423    0.8090    0.7742        89
4             5     0.8608    0.9444    0.9007        72
5             7     1.0000    0.9833    0.9916        60
6             8     0.9718    0.9485    0.9600       291
7             9     0.9375    0.7692    0.8451        39
8
9      accuracy                         0.9165       551
10    macro avg     0.9025    0.8909    0.8943       551
```

```
11 weighted avg      0.9209     0.9165     0.9175        551
```

Here is a Python code example that uses default parameters to visualize an RF model's prediction results after the optimization with GridSearchCV, as shown in Figure 4.25.

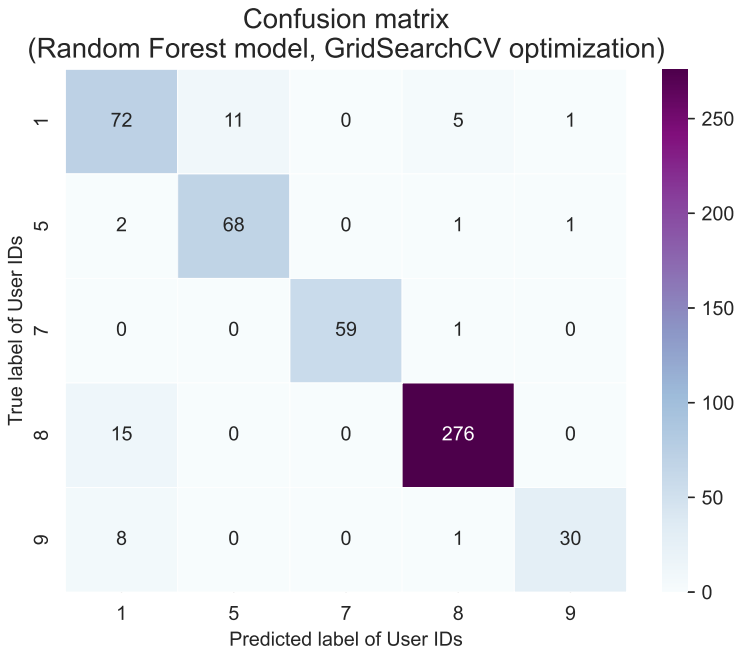

Figure 4.25 *Confusion Matrix of Random Forest (GridSearchCV)*

Analysis of Random Forest model results

Classification Report

After comparing the two reports, it was observed that the model with optimized parameters using GridSearchCV performed slightly better than the model with default parameters in terms of accuracy, precision, recall, and f1-score.

According to the test results, the model using default parameters had an accuracy rate of 90.74%. The weighted average of precision, recall, and f1-score were 0.9127, 0.9074, and 0.9087, respectively.

However, after optimizing the model using GridSearchCV, the accuracy improved to 91.65%, while the weighted average of precision, recall, and f1-score were respectively

0.9209, 0.9165, and 0.9175.

Confusion Matrix

The default model has a tendency to misclassify class 1 as either class 5 (14 times) or class 8 (6 times). Additionally, class 8 is often misclassified as class 1 (15 times) and class 9 is misclassified as either class 1 (8 times) or class 8 (1 time).

However, after undergoing optimization through GridSearchCV, there was a decrease in the number of misclassifications for class 1 as class 5 (11 instead of 14) and similar misclassifications for other classes compared to the default model.

Overall, the optimized model had slightly fewer misclassifications between certain classes.

Conclusion

In conclusion, implementing GridSearchCV optimization for the RF model led to a slight improvement in the accuracy, precision, recall, and f1-score of the model. These enhancements were evident in both the classification report and confusion matrix. It is important to note that even minor improvements can be critical, especially in cases where the cost of misclassification is significant.

4.4.3 Gradient Boosting Machine

Gradient Boosting Machine (GBM) is a powerful machine learning technique that can be used for both classification and regression tasks. As an ensemble learning method, GBM combines the predictions of multiple machine learning models to produce more accurate predictions than any individual model.

GBM operates iteratively and emphasizes the errors from previous iterations, or "weak learners," to enhance its performance. This step-by-step process enables GBM to improve its predictions over successive iterations, making it a flexible and

adaptable method that can optimize any given differentiable loss function.

Compared to LR, which is a linear classifier, GBM can capture complex non-linear relationships in data. This makes it a preferred choice in scenarios with intricate data structures. While LR works well when the relationship between features and the target can be delineated linearly, GBM is designed to navigate through and exploit more nuanced data patterns. However, this complexity comes at a cost. GBM models can be more computationally intensive to train than LR models and might require more careful tuning to avoid overfitting. Additionally, the interpretability of GBM models can be more challenging compared to the straightforward nature of LR coefficients.

The provided Python code trains a GBM on standardized training data and uses the model to predict labels for standardized testing data:

```
from sklearn.ensemble import GradientBoostingClassifier

y_train = np.array(training_labels_list)
y_test = np.array(testing_labels_list)

scaler = StandardScaler()
scaler.fit(training_data_df)
x_train_data_gbm = scaler.transform(training_data_df)
x_test_data_gbm = scaler.transform(testing_data_df)

gbm = GradientBoostingClassifier(random_state=21)
gbm.fit(x_train_data_gbm, y_train)
gbm_default_y_pred = gbm.predict(x_test_data_gbm)
gbm_default_default_report = classification_report(y_test,
   gbm_default_y_pred, digits=4)
```

GBM training (with default parameters) classification report:

```
              precision    recall  f1-score   support

```

```
           1      0.7917    0.8539    0.8216        89
           5      0.8537    0.9722    0.9091        72
           7      1.0000    0.9667    0.9831        60
           8      0.9684    0.9485    0.9583       291
           9      1.0000    0.7692    0.8696        39

    accuracy                         0.9256       551
   macro avg      0.9227    0.9021    0.9083       551
weighted avg      0.9305    0.9256    0.9262       551
```

Here is a Python code that uses default parameters to visualise GBM prediction results, as shown in Figure 4.26.

```
labels = np.unique(training_labels_list)
gbm_confusion_mat = confusion_matrix(y_test, gbm_default_y_pred)
sns.heatmap(gbm_confusion_mat, xticklabels=labels, yticklabels=
   labels, annot=True, linewidths = 0.2, fmt='d', cmap = color_map)
```

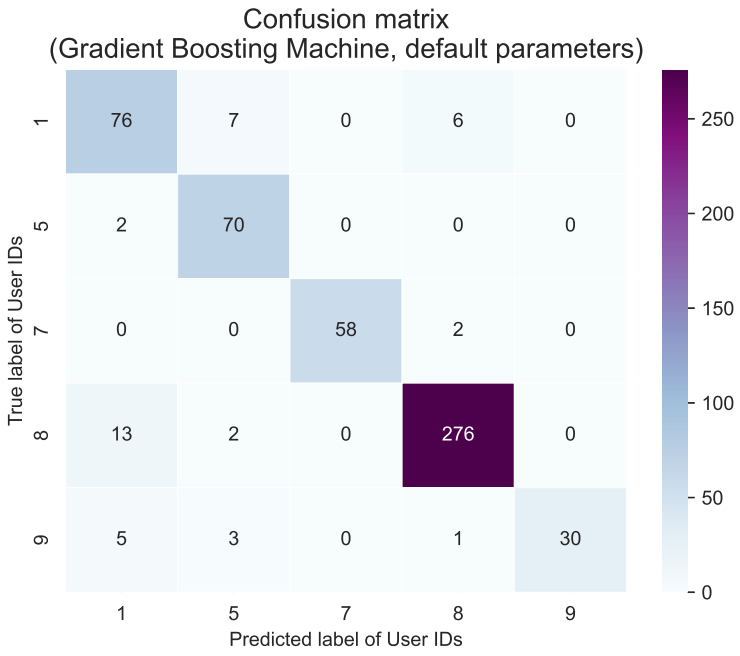

Figure 4.26 *Confusion Matrix of GBM (default parameters)*

Here is a Python code that utilizes GridSearchCV with GBM:

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search over
param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'learning_rate': [0.001, 0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

gbm = GradientBoostingClassifier(random_state=21)
grid_search = GridSearchCV(gbm, param_grid, cv=5)

# Fit the GridSearchCV instance the original data
grid_search.fit(x_train_data_gbm, y_train)
print("Best parameters:", grid_search.best_params_)

# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Predict using the best model
gbm_gridsearch_y_pred = best_model.predict(x_test_data_gbm)
```

The outcome of the GridSearchCV optimization process is as follows:

```
Best parameters: {'learning_rate': 0.1, 'max_depth': 5, '
    min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators':
    200}
```

Here's the classification report for GBM training after the optimization with GridSearchCV:

```
gbm_gridsearch_report = classification_report(y_test,
    gbm_gridsearch_y_pred, digits=4)
```

```
                precision    recall  f1-score   support

           1       0.8000    0.8090    0.8045        89
           5       0.8519    0.9583    0.9020        72
           7       0.9672    0.9833    0.9752        60
           8       0.9650    0.9485    0.9567       291
           9       0.9091    0.7692    0.8333        39

    accuracy                         0.9183       551
   macro avg       0.8986    0.8937    0.8943       551
weighted avg       0.9199    0.9183    0.9182       551
```

Here is a Python code that uses default parameters to visualise GBM prediction results, as shown in Figure 4.27.

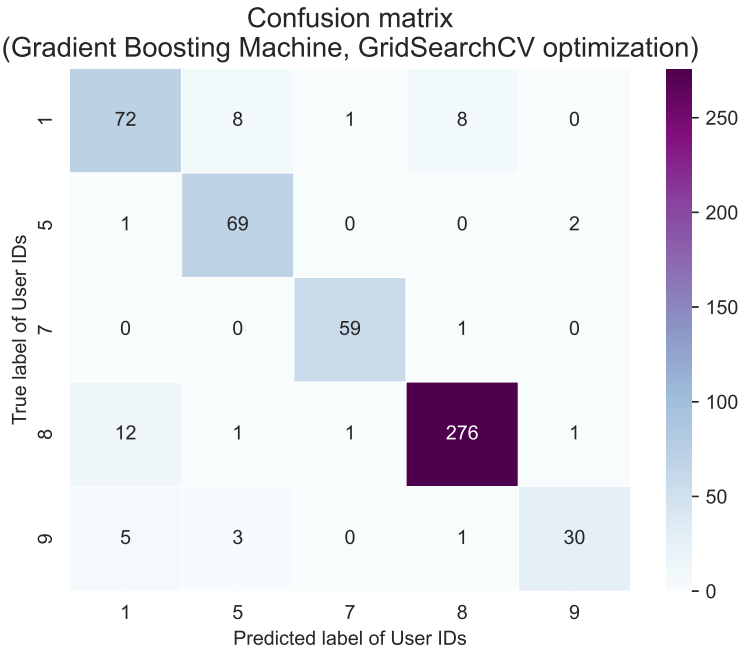

Figure 4.27 *Confusion Matrix of GBM (GridSearchCV optimization)*

Analysis of Gradient Boosting Machine results

Classification Report

Upon initial inspection, it was noted that the Gradient Boosting Machine (GBM)

model with default parameters performed marginally better in terms of accuracy, precision, recall, and f1-score than the optimized model.

The default GBM model had an accuracy score of 92.56%, while the weighted average precision, recall, and f1-score values were 0.9305, 0.9256, and 0.9262, respectively.

However, after optimizing the model using GridSearchCV, the accuracy score slightly decreased to 91.83%. The weighted average precision, recall, and f1-score values for the optimized model were 0.9199, 0.9183, and 0.9182, respectively.

Confusion Matrix

During the initial testing of the GBM with default parameters, significant misclassifications were observed between classes 1 and 8, occurring 6 times, and between classes 8 and 1, occurring 13 times. However, relatively good performance was observed with classes 7 and 9, where class 9 had a slightly higher misclassification rate compared to class 7.

After the model was optimized with GridSearchCV, a slight increase in misclassifications between class 1 and 8 was observed, which occurred 8 times, compared to the default model. The misclassification rate between classes 8 and 1 remained almost the same, with 12 instances. Additionally, a few new misclassifications between class 1 and 7 and between class 8 and 7 were observed, but they were minimal.

Conclusion

To conclude, upon comparing the metrics of the classification report, it appears that the GBM with default parameters outperforms the optimized model slightly. Furthermore, the optimized model seems to have a slightly higher rate of misclassification between certain classes.

It is important to note that GridSearchCV, which is used to find the best hyperparameters for a model, is limited to the predefined hyperparameter space given

during the search. In this case, the optimization performed by GridSearchCV did not result in an improved model when compared to the default parameters.

4.4.4 Applied machine learning models

Logistic Regression, *Random Forest*, and *Gradient Boosting Machine* models were used for both training and test datasets. The models were tested with default parameters and with optimized parameters using GridSearchCV. The table 4.2 summarizes the accuracy of machine learning models:

Model	Accuracy	Precision	Recall	F1-Score
LR (default)	0.9220	0.9176	0.9172	0.9145
LR (GridSearchCV)	0.9238	0.9248	0.9226	0.9209
RF (default)	0.9074	0.8960	0.8786	0.8843
RF (GridSearchCV)	0.9165	0.9025	0.8909	0.8943
GBM (default)	0.9256	0.9227	0.9021	0.9083
GBM (GridSearchCV)	0.9183	0.8986	0.8937	0.8943

Table 4.2 *Comparison of different models' performance metrics.*

Analyzing the performance of machine learning models, particularly for behavioral authentication that recognizes users through time-series data using accelerometer and gyroscope data, requires considering a combination of factors such as model performance metrics, underlying assumptions, and computational complexities. Below is a detailed analysis explaining these factors in detail:

Accuracy

After performing hyperparameter tuning on the models using GridSearchCV, it was found that LR had the highest accuracy among all the models, with an accuracy score of 0.9238. Following optimization, both the RF and GBM models showed an increase in accuracy, with RF achieving an accuracy of 0.9165.

Notably, the default GBM model, without hyperparameter tuning, performed better than its GridSearchCV counterpart, achieving the highest overall accuracy of 0.9256.

Precision and Recall

Among the models, LR with GridSearchCV optimization has the highest precision when identifying users. This means that the model can correctly identify users most of the time. LR also has a relatively high recall, which signifies the proportion of actual positive cases predicted correctly.

In comparison, RF with GridSearchCV and GBM with default settings have similar precision and recall. After optimization, RF has slightly higher precision, while GBM with default settings has better recall.

Overall, LR is the best performer when it comes to precision and recall. This showcases its robustness in correctly identifying users and not missing any actual users.

F1-Score

The F1-Score is a metric that assesses a model's accuracy and completeness by combining precision and recall.

Among the models optimized with GridSearchCV, LR has the highest F1-Score, which is 0.9209. After optimization, RF and GBM have similar F1-Scores.

Computational Complexities

Different machine learning models have varying levels of computational demands. LR is a lightweight model with linear complexity and uses minimal computational power and memory. However, if the dataset is too large, the training time may become an issue.

RF is more computationally intensive as it uses an ensemble of decision trees, which can lead to increased memory usage and training time, especially if the number of trees or depth of the trees is high. Additionally, when using GridSearchCV, the

computational demand of RF can increase significantly.

GBM is a computationally intensive algorithm. The boosting process involves building trees sequentially, which can take a significant amount of time. Consequently, when using GridSearchCV, the complexity of GBM can increase even further, resulting in a considerable amount of time required for the algorithm to execute. For instance, it took quite many hours to run on a PC that was employed for the calculations in this thesis.

Task-specific Considerations

When working with time-series data, it is essential to consider the intricate relationships and non-linearities that might exist. Models such RF and GBM may be advantageous in these situations because of their ability to handle such complexities.

However, in real-time applications like behavioral authentication, computation speed is a critical factor. Even if LR provides slightly lower accuracy, its efficiency may make it the preferred choice in such scenarios.

Conclusion

Recognizing users from time-series data using accelerometer and gyroscope data is a crucial task. In this regard, the best combination of performance (accuracy, precision, recall, and F1-score) and computational efficiency is offered by LR with GridSearchCV.

However, if abundant computational resources are available and small increases in accuracy are important, the GBM or RF with GridSearchCV may be considered.

When making the final decision, the deployment scenario, such as real-time versus batch processing, should be taken into account. Therefore, it is recommended to choose the appropriate model according to the specific use case.

4.5 Interpreting LR model result using SHAP

SHAP (SHapley Additive exPlanations) is a method that provides a comprehensive way to measure the importance of features within a model. The values are derived from game theory and are based on the concept of Shapley values. SHAP values allocate the discrepancy between a model's prediction and the average prediction to each feature in a consistent and equitable manner. This makes it easier to understand the importance of each feature in the model and how it contributes to the final outcome.

The provided Python code visualizes LR model (with GridSearchCV optimisation) result using SHAP:

```
import shap
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm

print(f'x_train_data_lr len: {len(x_train_data_lr)}')
print(f'x_test_data_lr len: {len(x_test_data_lr)}')

# Take a random sample of the training data
#   as the background dataset
background_data = shap.sample(x_train_data_lr, 400)
explainer = shap.KernelExplainer(best_model.predict_proba,
    background_data)

# Compute SHAP values for a subset of
#  the test data (to save computation time)
sample_test_data = shap.sample(x_test_data_lr, 200)
shap_values = explainer.shap_values(sample_test_data)

shap.initjs()
shap.force_plot(explainer.expected_value[0], shap_values[0][0],
    sample_test_data[0])
```

```
feature_names = training_data_df.columns
shap.summary_plot(shap_values, sample_test_data, feature_names=
    feature_names)
```

The figure 4.28 provides a summary of the features that influence the predictions of LR(GridSearchCV):

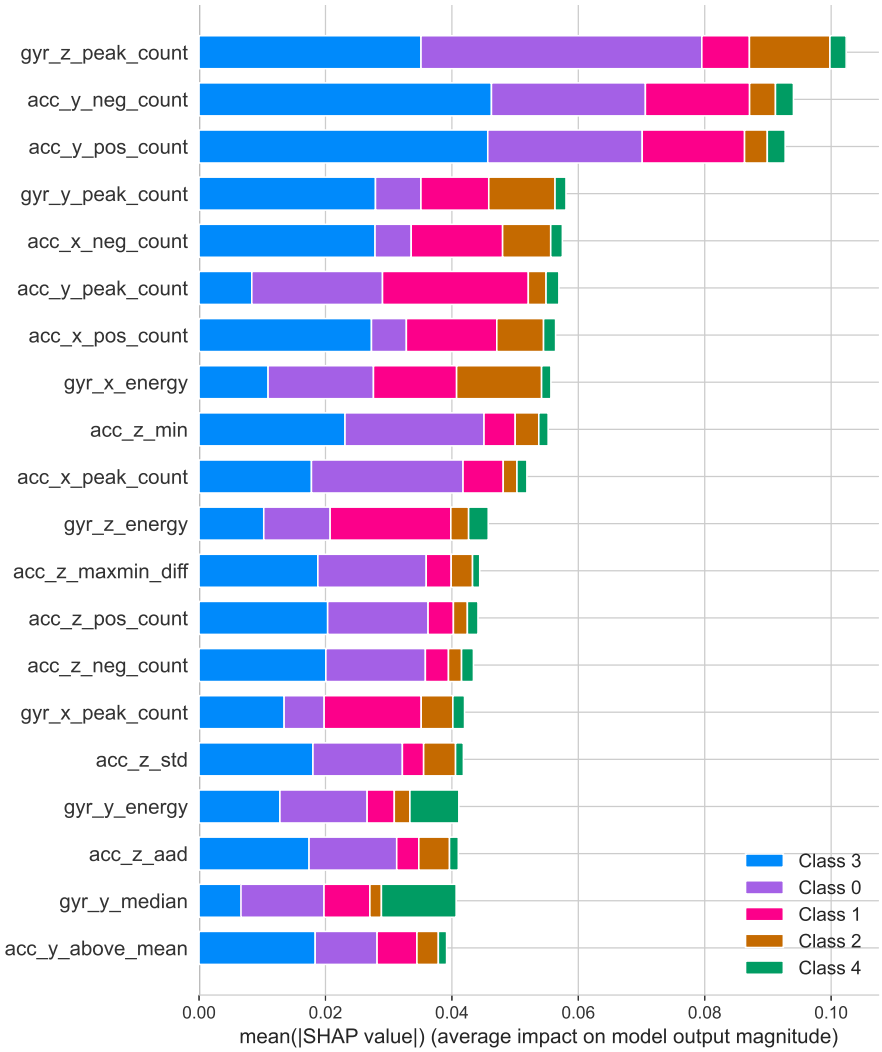

Figure 4.28 *Summary of the features that affect the LR(GridSearchCV) predictions*

4.5.1 Top 10 feature contributions

Top 10 feature contributions that highly influenced the model's prediction are listed below:

1. Gyroscope Z-axis peak count (`gyr_z_peak_count`).
2. Count of positive values on the Y-axis of accelerometer (`acc_y_pos_count`).
3. Count of negative values on the Y-axis of accelerometer (`acc_y_neg_count`).
4. Gyroscope Y-axis peak count (`gyr_y_peak_count`).
5. Accelerometer Y-axis peak count (`acc_y_peak_count`).
6. Count of positive values on the X-axis of accelerometer (`acc_x_pos_count`).
7. Count of negative values on the X-axis of accelerometer (`acc_x_neg_count`).
8. Minimum value on the Z-axis of accelerometer (`acc_z_min`).
9. Energy of X-axis gyroscope (`gyr_x_energy`).
10. Accelerometer X-axis peak count (`acc_x_peak_count`).

4.5.2 Conclusion

After analyzing the model's predictions, it was found that the Z-axis of the gyroscope and the Y-axis of the accelerometer have the greatest impact on peak counts, positive and negative value counts, and energy metrics. It is noteworthy that relying on these features improves the computational efficiency of the model.

Additionally, these features are relatively easy to obtain, which ensures that the model's performance is not compromised while enhancing its efficiency and speed. The balance between the model's accuracy and computational simplicity makes it highly effective for real-time applications or situations where computational resources are limited.

4.6 Data Collection and Analysis with Galaxy S21 Ultra

The quality and authenticity of data plays a pivotal role in understanding and modeling human activities through sensors. A project was undertaken to gather

raw sensor readings during various activities using the state-of-the-art Galaxy S21 Ultra smartphone. Specifically, the focus was on collecting data for the "Walking" activity of 5 users.

The data collection process was facilitated by the **Sensor Logger** Android app, which provided a seamless interface to record and export the necessary sensor data. In this chapter, the intricacies of this dataset, its characteristics, and the insights drawn from it will be discussed in detail.

4.6.1 Data Format Conversion

In order to simplify and standardize the processing of data, it was crucial that all datasets be in a consistent format. Therefore, the data files generated by the Sensor Logger application were converted to match the original data format used in this thesis. A generic Python code demonstrating this conversion process is presented below:

```
def convert_sensor_format_2_to_1(df_format_2, username):
    df_converted = df_format_2.copy()

    # Rename the columns
    df_converted.rename(columns={
        'time': 'timestamp',
        'z': 'sensor_z_axis',
        'y': 'sensor_y_axis',
        'x': 'sensor_x_axis'
    }, inplace=True)

    # Add the new columns
    # For "id", generate an incremental series of numbers starting
   from 0
    df_converted['id'] = range(len(df_converted))

```

```
    # Set "username", "activity_id", and "activity" to the
    specified values
    df_converted['username'] = username
    df_converted['activity_id'] = -499
    df_converted['activity'] = 'Walking'

    # Reorder columns to match Format 1
    column_order = [
        'id',
        'username',
        'timestamp',
        'sensor_x_axis',
        'sensor_y_axis',
        'sensor_z_axis',
        'activity_id',
        'activity']
    df_converted = df_converted[column_order]
    return df_converted
```

4.6.2 Sampling Rate Evaluation

Figures 4.29 to 4.38 display histograms of sampling periods for the "Walking" segments for both sensors datasets:

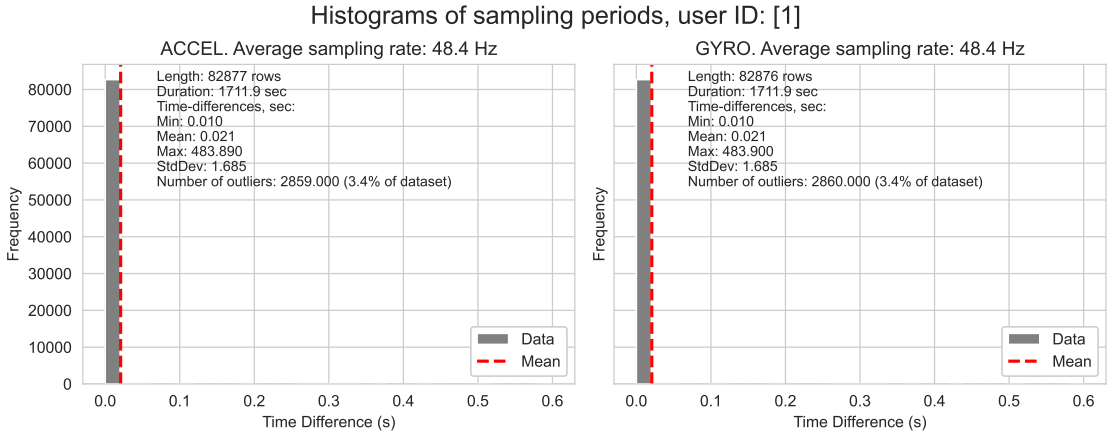

Figure 4.29 *Histograms of sampling periods, dataset: 1, user ID: 1*

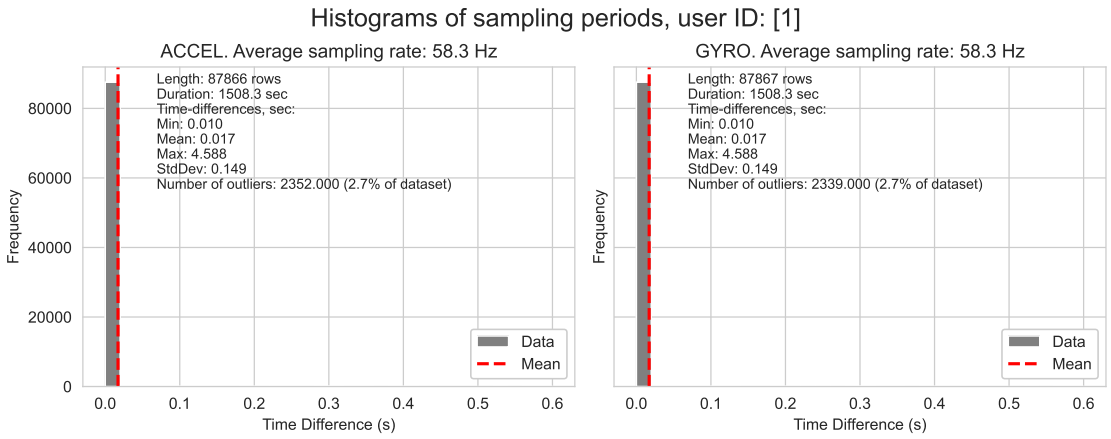

Figure 4.30 *Histograms of sampling periods, dataset: 2, user ID: 1*

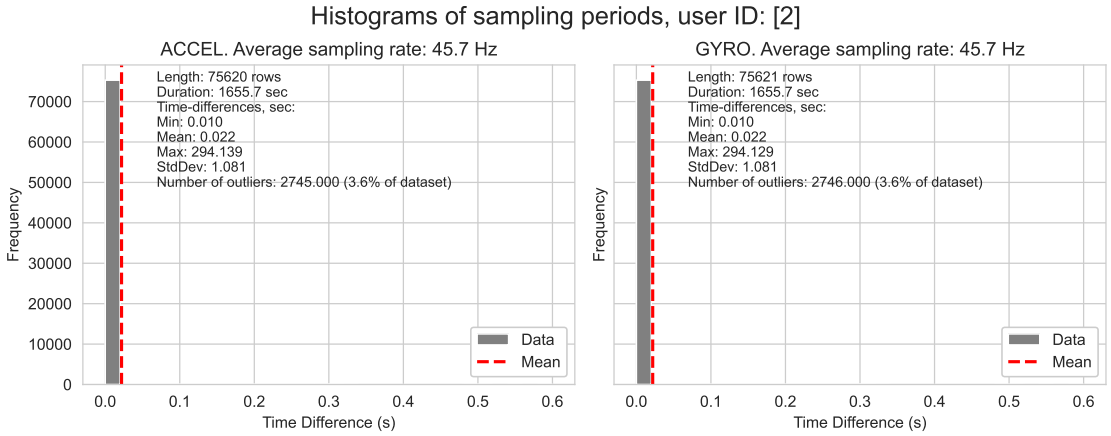

Figure 4.31 *Histograms of sampling periods, dataset: 1, user ID: 2*

Upon analyzing the sampling period histograms for the datasets collected using the Galaxy S21 Ultra smartphone, certain discrepancies in sampling rates were observed across different datasets. The statistics for each dataset are presented below:

- Dataframe 0: 3.4% outliers; average sampling rate: 48.4 Hz
- Dataframe 1: 2.7% outliers; average sampling rate: 58.3 Hz
- Dataframe 2: 3.6% outliers; average sampling rate: 45.7 Hz
- Dataframe 3: 4.2% outliers; average sampling rate: 33.1 Hz
- Dataframe 4: 2.5% outliers; average sampling rate: 67.8 Hz
- Dataframe 5: 3.6% outliers; average sampling rate: 57.0 Hz

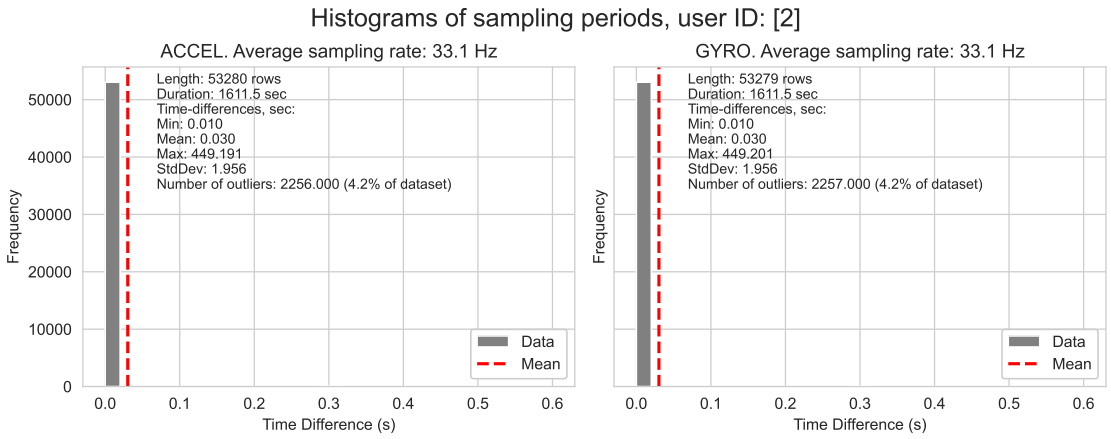

Figure 4.32 *Histograms of sampling periods, dataset: 2, user ID: 2*

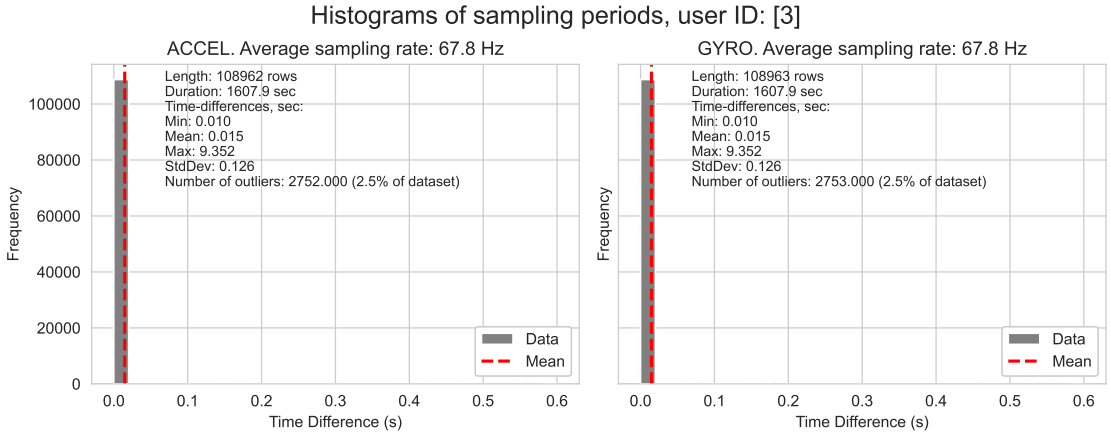

Figure 4.33 *Histograms of sampling periods, dataset: 1, user ID: 3*

- Dataframe 6: 3.3% outliers; average sampling rate: 56.0 Hz
- Dataframe 7: 2.5% outliers; average sampling rate: 62.3 Hz
- Dataframe 8: 3.7% outliers; average sampling rate: 54.5 Hz
- Dataframe 9: 3.6% outliers; average sampling rate: 55.8 Hz

Due to variations in sampling rates and the presence of outliers, it was decided to analyze the datasets "as is" using the LR model, without performing any resampling procedures. This approach aims to maintain the authenticity of the analysis and predictions, ensuring that they remain as close to real-life conditions as possible.

The primary objective is to determine how the model performs with raw data,

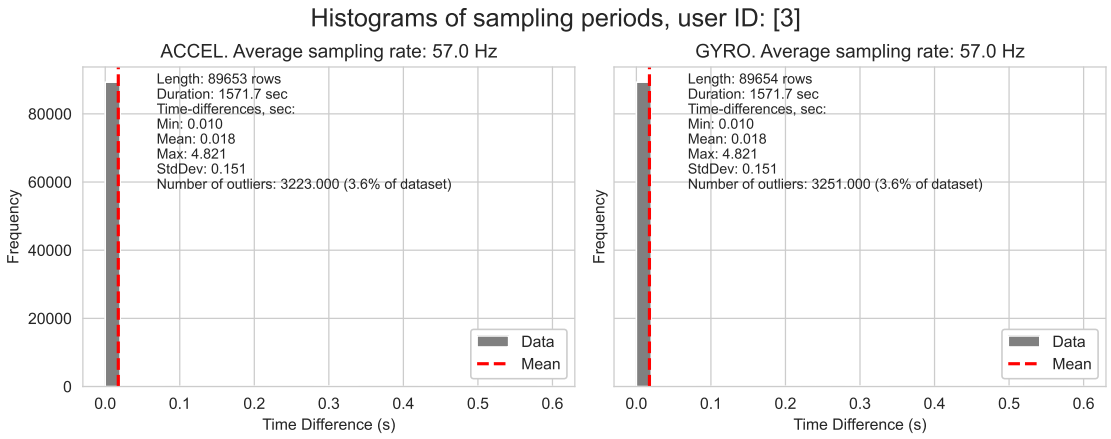

Figure 4.34 *Histograms of sampling periods, dataset: 2, user ID: 3*

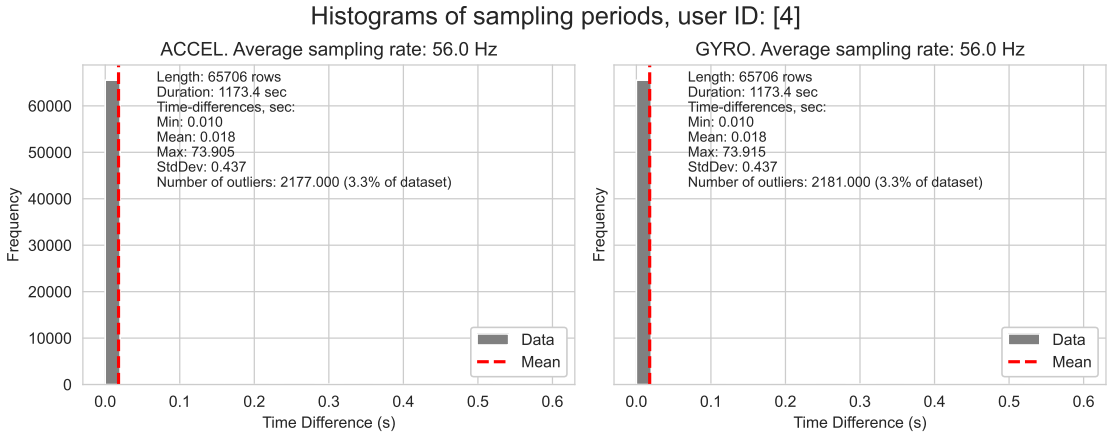

Figure 4.35 *Histograms of sampling periods, dataset: 1, user ID: 4*

embracing the inherent variability and real-world imperfections present in data collection.

4.6.3 Applying the LR model

Classification Report

LR model training (with default parameters) classification report:

```
              precision    recall  f1-score   support

           1     0.9414    0.8968    0.9186      1396
           2     0.6283    0.7380    0.6788       355
           3     0.6295    0.9524    0.7580       273
```

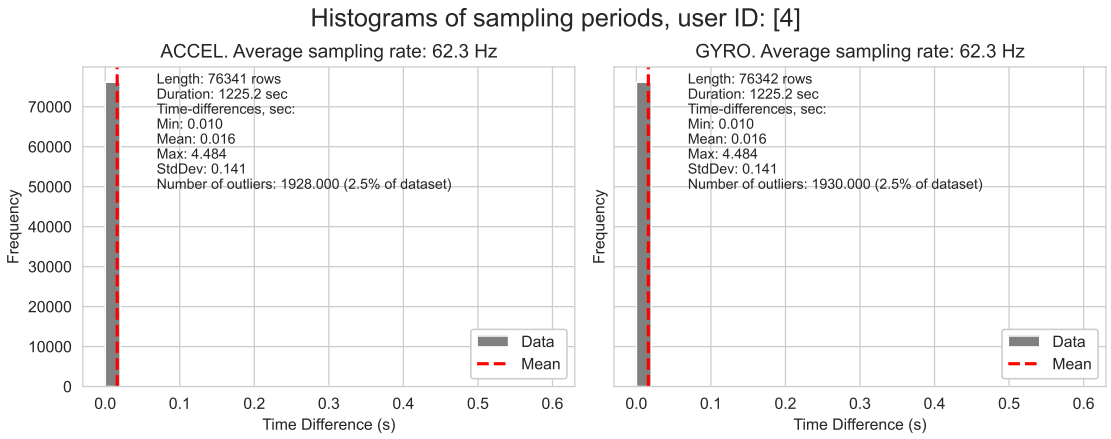

Figure 4.36 *Histograms of sampling periods, dataset: 2, user ID: 4*

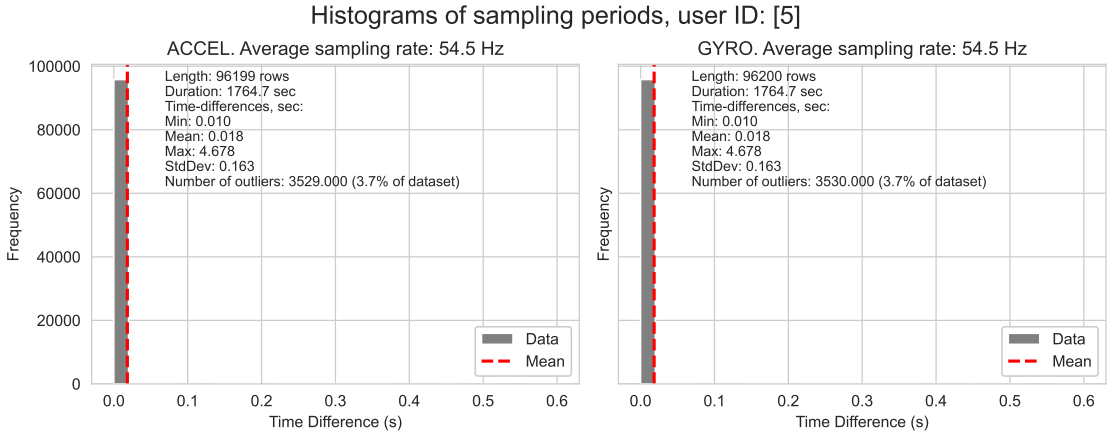

Figure 4.37 *Histograms of sampling periods, dataset: 1, user ID: 5*

```
           4      0.6297    0.8535    0.7247       273
           5      0.9389    0.8187    0.8747      1820

    accuracy                         0.8494      4117
   macro avg      0.7536    0.8519    0.7909      4117
weighted avg      0.8719    0.8494    0.8550      4117
```

The overall accuracy of the model on the dataset was found to be 84.94%. In terms of macro average, the model demonstrated a precision of 75.36%, recall of 85.19%, and F1-Score of 79.09%.

Furthermore, the weighted average values for precision, recall, and F1-Score are 87.19%, 84.94%, and 79.09%, respectively.

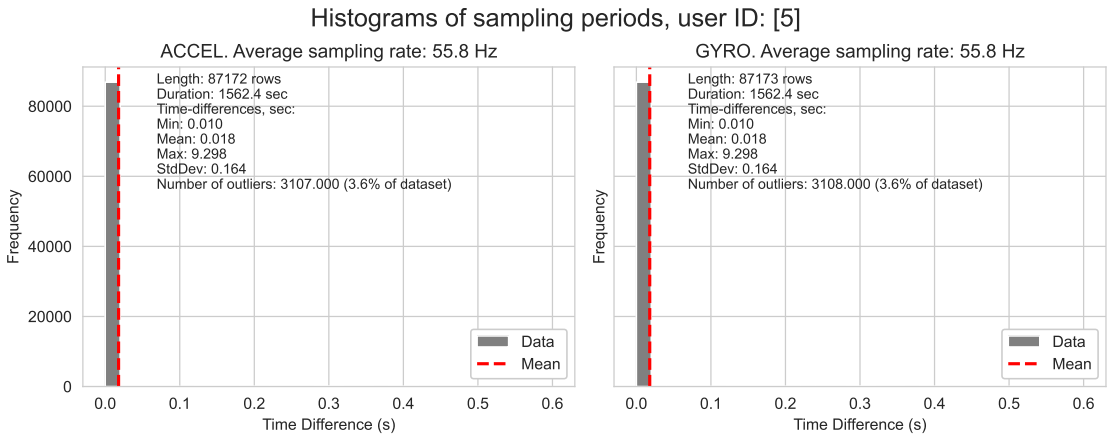

Figure 4.38 *Histograms of sampling periods, dataset: 2, user ID: 5*

Confusion Matrix

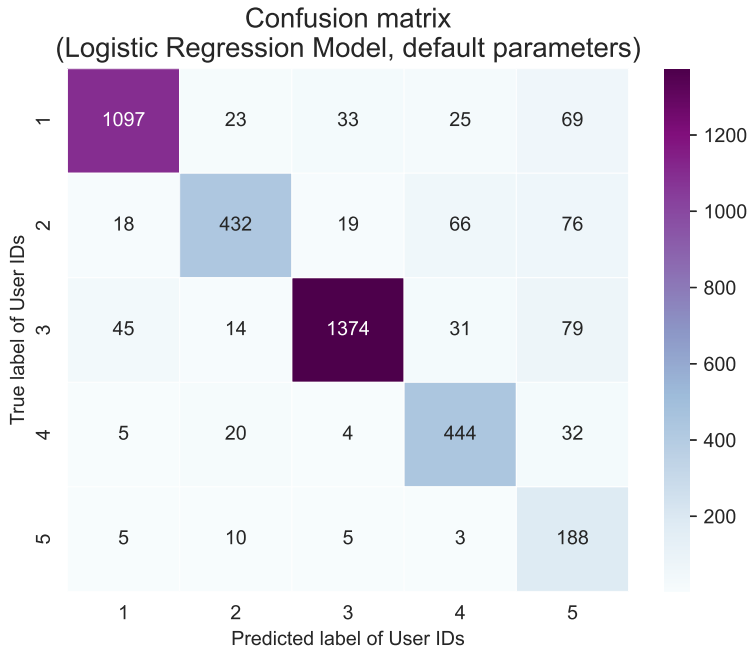

Figure 4.39 *Logistic Regression with default parameters*

Looking at the confusion matrix, it is evident that some of the classes were predicted accurately, while others, especially Class 5, had a high number of false positives. The LR model's performance is impressive, given the dataset's inherent challenges. However, the discrepancies in precision and recall values across the classes emphasize the impact of sampling rates and outliers in the data. This highlights the importance of having a well-prepared dataset or adapting machine learning models to handle

real-world, raw data effectively.

Conclusion

After thoroughly analyzing the results obtained from applying the LR model on a relatively short and unprepared dataset (without cleaning up the outliers and resampling), some conclusions can be made. Despite the challenges posed by the differences in sampling rates and outlier presence, the model performed well on multiple metrics. The model's accuracy rates were high, and it showed robust precision, recall, and F1-Score values for several classes, which demonstrated its resilience and adaptability.

It is important to note that many advanced models require carefully curated data to perform at their best. However, in real-world situations, obtaining such pristine data is often a luxury that is not always feasible. The LR model's ability to produce satisfactory results on unprocessed data highlights its suitability for practical applications where data preprocessing might be limited or even unwanted.

5. GLOBAL CONCLUSION ON LOGISTIC REGRESSION IN BEHAVIORAL BIOMETRICS

Biometrics is an evolving landscape, and gait analysis is a unique modality that uses the natural walking patterns of individuals. The data is captured through devices equipped with accelerometers and gyroscopes, and gait-based data is rich in behavioral traits that are inherently distinct for each individual.

Various machine learning models can be applied to analyze and distinguish between these walking patterns. However, it has been shown through research that LR is the most effective model for this task. The intrinsic dynamics of gait are efficiently captured and classified by LR, which involves intricate patterns of acceleration and angular rotations.

Computational Efficiency: Computational efficiency is one of the most significant advantages of LR. This model is not only capable of detecting subtle behaviors but also has a simple computing approach. This trait is especially crucial for applications that require real-time processing or have limited computational resources. While RF and GBM are alternative models that can capture complex patterns, they may not be the most efficient choice due to their high computational intensity. This can be a disadvantage, particularly when using GridSearchCV for exhaustive parameter tuning.

Task-specific Consideration: It is important to consider the type of biometrics when choosing an authentication mechanism. For BB, LR is the best choice for quick authentication due to its real-time processing speed. Although RF and GBM models

can analyze complex time-series patterns, they are not as fast as LR.

Feature Importance: After a SHAP analysis had been conducted, it was discovered that the model's decisions were most influenced by certain simple features. These features included peak counts, positive and negative value counts, and energy metrics from the Z-axis gyroscope and Y-axis accelerometer. It is noteworthy that these features are both informative and computationally efficient.

Model Performance on Personalized Data: Assessing the effectiveness of a behavioral biometric system requires evaluating its ability to adjust to individual datasets. The LR model has demonstrated remarkable adaptability and reliability, boasting an accuracy rate of approximately 85%. This makes it a viable and dependable option for personalized authentication.

In the future, the integration of security with user experience will be seamlessly achieved. To accomplish this, a balance between computational efficiency, model interpretability, and performance is crucially needed. LR has been recognized as a strong contender because of its combination of speed, simplicity, and effective performance. In particular, for real-time applications like BB, secure system maintenance and prevention of compromise requires quick and accurate user authentication. The user experience is enhanced while security is bolstered by this frictionless authentication method. Customers are not required to perform any specific actions as their natural walking behavior is sufficient to confirm their identity. A secure and user-friendly authentication system is provided by the fact that it is challenging to mimic or replicate another person's unique gait.

It is important to note that many sophisticated models often require well-curated data to function at their best. However, in real-world scenarios, such pristine data is a luxury that is not always feasible. The ability of the LR model to deliver appreciable results on unprocessed data makes it suitable for practical applications where data preprocessing might be limited or even undesirable.

In conclusion, the LR model has proven to be not only versatile but also robust, making it an excellent choice for applications that involve raw and unprepared data. Its performance amidst the dataset's imperfections highlights its potential as a reliable tool for real-world data analytics tasks, especially when striving for results that are as close as possible to real-life conditions.

REFERENCES

Moskovitch, Robert et al. (June 2009). "Identity theft, computers and behavioral biometrics". In: *2009 IEEE International Conference on Intelligence and Security Informatics*. IEEE, pp. 155–160. DOI: 10.1109/ISI.2009.5137288.

Committee, National Research Council (us) Whither Biometrics, Joseph N. Pato, and Lynette I. Millett (2010). "Introduction and Fundamental Concepts". In: *Biometric Recognition: Challenges and Opportunities*. National Academies Press (US). URL: https://www.ncbi.nlm.nih.gov/books/NBK219892.

Li, F. et al. (July 2011). "Behaviour Profiling for Transparent Authentication for Mobile Devices". In: *10th European Conference on Information Warfare and Security 2011, ECIW 2011*. URL: https://www.researchgate.net/publication/236001587_Behaviour_Profiling_for_Transparent_Authentication_for_Mobile_Devices.

Maiorana, Emanuele et al. (Jan. 2011). "Keystroke dynamics authentication for mobile phones". In: *Proceedings of the ACM Symposium on Applied Computing*, pp. 21–26. DOI: 10.1145/1982185.1982190.

Priyantha, Bodhi, Dimitrios Lymberopoulos, and Jie Liu (Mar. 2011). "LittleRock: Enabling Energy-Efficient Continuous Sensing on Mobile Phones". In: *IEEE Pervasive Comput.* 10.2, pp. 12–15. ISSN: 1536-1268. DOI: 10.1109/MPRV.2011.28.

Eshwarappa M, N. and Mrityunjaya V. Latte (2012). "Multimodal Biometric Person Authentication using Speech, Signature and Handwriting Features". In: *International Journal of Advanced Computer Science and Applications* 1.3. DOI: 10.14569/SpecialIssue.2011.010313.

Lin, Felix Xiaozhu et al. (Mar. 2012). "Reflex: using low-power processors in smartphones without knowing them". In: *SIGARCH Comput. Archit. News.* 40.1, pp. 13–24. ISSN: 0163-5964. DOI: 10.1145/2189750.2150979.

Nickel, Claudia, Tobias Wirtl, and Christoph Busch (July 2012). "Authentication of Smartphone Users Based on the Way They Walk Using k-NN Algorithm". In: *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE, pp. 16–20. DOI: 10.1109/IIH-MSP.2012.11.

Xu, Zhi, Kun Bai, and Sencun Zhu (2012). "TapLogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors". In: *WiSec'12 - Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 113–124. DOI: 10.1145/2185448.2185465.

Trojahn, M., F. Arndt, and F. Ortmeier (Jan. 2013). "Authentication with keystroke dynamics on touchscreen keypads-effect of different n-graph combinations". In: *MOBILITY 2013, the Third International Conference on Mobile Services, Resources, and Users*, pp. 114–119. URL: https://www.researchgate.net/publication/303006123_Authentication_with_keystroke_dynamics_on_touchscreen_keypads-effect_of_different_n-graph_combinations.

Fayyaz, Mohsen et al. (May 2015). "Feature Representation for Online Signature Verification". In: *arXiv*. DOI: 10.1109/AISP.2015.7123528. eprint: 1505.08153.

Shen, Haichen et al. (Sept. 2015). "Enhancing mobile apps to use sensor hubs without programmer effort". In: *ResearchGate*, pp. 227–238. DOI: 10.1145/2750858.2804260.

Cpalka, Krzysztof, Marcin Zalasinski, and Leszek Rutkowski (Oct. 2016). "A new algorithm for identity verification based on the analysis of a handwritten dynamic signature". In: *arXiv*. DOI: 10.1016/j.asoc.2016.02.017. eprint: 1610.01578.

Haghighat, Mohammad, Mohamed Abdel-Mottaleb, and Wadee Alhalabi (May 2016). "Discriminant Correlation Analysis: Real-Time Feature Level Fusion for Multimodal Biometric Recognition". In: *IEEE Trans. Inf. Forensics Secur.* 11.9, pp. 1984–1996. ISSN: 1556-6021. DOI: 10.1109/TIFS.2016.2569061.

Ijartet, Researcher, Ms. R. Greena, and Mr. U. Baveenther (Mar. 2016). "MOBILE USER AUTHENTICATION USING BEHAVIORAL HAND WAVING BIOMET-

RICS". In: *International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)* 3.Special 11, pp. 92–96. URL: https://www.researchgate.net/publication/304784604_MOBILE_USER_AUTHENTICATION_USING_BEHAVIORAL_HAND_WAVING_BIOMETRICS.

Feng, Huan, Kassem Fawaz, and Kang G. Shin (Oct. 2017). "Continuous Authentication for Voice Assistants". In: *MobiCom '17: Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. New York, NY, USA: Association for Computing Machinery, pp. 343–355. ISBN: 978-1-45034916-1. DOI: 10.1145/3117811.3117823.

Khamsemanan, Nirattaya, Cholwich Nattee, and Nitchan Jianwattanapaisarn (Aug. 2017). "Human Identification From Freestyle Walks Using Posture-Based Gait Feature". In: *IEEE Trans. Inf. Forensics Secur.* 13.1, pp. 119–128. ISSN: 1556-6021. DOI: 10.1109/TIFS.2017.2738611.

Lee, Wei-Han and Ruby Lee (Mar. 2017). "Multi-sensor authentication to improve smartphone security". In: *arXiv*. DOI: 10.48550/arXiv.1703.03378. eprint: 1703.03378.

Muaaz, Muhammad and René Mayrhofer (Mar. 2017). "Smartphone-Based Gait Recognition: From Authentication to Imitation". In: *IEEE Trans. Mob. Comput.* 16.11, pp. 3209–3221. ISSN: 1558-0660. DOI: 10.1109/TMC.2017.2686855.

Pandikumar, T. et al. (June 2017). "Enhancing Performance and Usability of Keystroke Dynamics Authentication on Mobile Touchscreen Devices using Features Extraction Scheme". In: *International Journal of Engineering Science and Computing* 07.06, pp. 13415–13421. URL: https://www.researchgate.net/publication/317955008_Enhancing_Performance_and_Usability_of_Keystroke_Dynamics_Authentication_on_Mobile_Touchscreen_Devices_using_Features_Extraction_Scheme.

Krishnamoorthy, Sowndarya et al. (May 2018). "Identification of User Behavioral Biometrics for Authentication Using Keystroke Dynamics and Machine Learning". In: *ResearchGate*, pp. 50–57. DOI: 10.1145/3230820.3230829.

Mohana Priya, A. and V. Alamelu (Apr. 2018). "Behavioral Hand Waving Biometrics". In: *International Journal of Engineering Research & Technology* 5.13. ISSN: 2278-0181. URL: https://www.ijert.org/behavioral-hand-waving-biometrics.

Bai, Guifeng and Yunqiang Sun (July 2019). "Application and research of MEMS sensor in gait recognition algorithm". In: *Cluster Computing* 22.3. ISSN: 1573-7543. DOI: 10.1007/s10586-018-2062-x.

A Public Domain Dataset For Real-life Human Activity Recognition Using Smartphone Sensors (July 2020). [Online; accessed 8. Feb. 2022]. URL: https://lbd.udc.es/research/real-life-HAR-dataset.

Suffocating Progress (Apr. 2020). [Online; accessed 21. Dec. 2021]. URL: https://www.city-journal.org/fda-blocks-apple-watch-blood-oxygen-feature.

Thao, Tran Phuong (Sept. 2020). "Location-based Behavioral Authentication Using GPS Distance Coherence". In: *arXiv*. eprint: 2009.08025. URL: https://arxiv.org/abs/2009.08025v1.

Cai, Chao et al. (Apr. 2021). "PURE: Passive mUlti-peRson idEntification via Deep Footstep Separation and Recognition". In: *arXiv*. eprint: 2104.07177. URL: https://arxiv.org/abs/2104.07177v1.

Gillis, Alexander S., Peter Loshin, and Michael Cobb (July 2021). "biometrics". In: *SearchSecurity*. URL: https://www.techtarget.com/searchsecurity/definition/biometrics.

Kassis, Andre and Urs Hengartner (July 2021). "Practical Attacks on Voice Spoofing Countermeasures". In: *arXiv*. eprint: 2107.14642. URL: https://arxiv.org/abs/2107.14642v1.

Muratyan, Alexa et al. (Sept. 2021). "Opportunistic Multi-Modal User Authentication for Health-Tracking IoT Wearables". In: *arXiv*. eprint: 2109.13705. URL: https://arxiv.org/abs/2109.13705v2.

Piugie, Yris Brice Wandji et al. (Sept. 2021). "How Artificial Intelligence can be used for Behavioral Identification?" In: *2021 International Conference on Cyberworlds (CW)*. IEEE, pp. 246–253. DOI: 10.1109/CW52790.2021.00049.

Soleymani, Sobhan et al. (Dec. 2021). "Quality-Aware Multimodal Biometric Recognition". In: *arXiv*. eprint: 2112.05827. URL: https://arxiv.org/abs/2112.05827v1.

Tillu, Jay (Dec. 2021). "Mobile sensors: The Components that make our smartphones smarter". In: *Medium*. URL: https://medium.com/jay-tillu/mobile-sensors-the-components-that-make-our-smartphones-smarter-4174a7a2bfc3.

Zeng, Xin et al. (July 2021). "Gait-Based Implicit Authentication Using Edge Computing and Deep Learning for Mobile Devices". In: *Sensors* 21.13, p. 4592. ISSN: 1424-8220. DOI: 10.3390/s21134592.

What Kinds of Sensors are Embedded in Smartphones? (Jan. 2022). [Online; accessed 7. Feb. 2022]. URL: https://www.samsungsds.com/en/story/What-Kinds-of-Sensors-are-Embedded-in-Smartphones.html.