



SAPIENZA  
UNIVERSITÀ DI ROMA

# Machine Learning for Human Movement Classification Based on Kinect Skeleton Data

Faculty of Information Engineering, Computer Science and Statistics  
Bachelor's Degree in Computer Science

**Lucian Dorin Crainic**  
ID number 1938430

Advisor  
Prof. Maurizio Mancini

Academic Year 2023/2024

Thesis not yet defended

---

**Machine Learning for Human Movement Classification Based on Kinect Skeleton Data**

Bachelor's Thesis. Sapienza University of Rome

© 2024 Lucian Dorin Crainic. All rights reserved

This thesis has been typeset by L<sup>A</sup>T<sub>E</sub>X and the Sapthesis class.

Author's email: crainic.lucian@gmail.com

## Abstract

This thesis conducts a detailed comparative study of several Machine Learning models, with a focus on their application to Kinect-based data for classifying human movements. The primary aim of this research is to evaluate these models to determine the most effective ones for accurately classifying movements recorded through Kinect sensors.

This study begins with a introduction to Kinect technology, highlighting its ability to capture detailed movement data. Following this, an examination of a range of Machine Learning models, such as Support Vector Machines, Random Forest, Linear Regression, and so on. Each model is tested to evaluate its accuracy, processing efficiency, and robustness in accurately classifying various movements.

The core of this comparative analysis is a diverse dataset consisting of several movements captured through a Microsoft Kinect. The research methodology involves several steps: processing the Kinect data, extracting key features that are characteristic of specific movements, and applying the selected models to this improved data. Performance evaluation of each model using standard metrics like accuracy, precision, recall, and the F1 score, which provide a complete picture of their effectiveness.

Over this study, valuable understandings are gained into the specific strengths and limitations of each model in the context of Kinect-based movement classification. The findings reveal that some models prove enhanced performance in certain situations, which is influenced by factors like the complexity of the captured movements and the characteristics of the dataset.

This thesis acts as a useful guide for researchers and professionals. It helps them pick the best models for similar work and sets the stage for more research in this area. This study contributes to the advancement of accurate and efficient Kinect-based data movement classification using Machine Learning methods, leading to more progress in this field.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem statement . . . . .	1
1.2	Literature review . . . . .	1
1.3	Dataset overview . . . . .	2
1.4	Objectives . . . . .	3
<b>2</b>	<b>Dataset analysis</b>	<b>4</b>
2.1	Data collection methodology . . . . .	4
2.1.1	Microsoft kinect . . . . .	4
2.1.2	Recording setup . . . . .	6
2.2	Data structure and attributes . . . . .	7
2.3	Participants characteristics . . . . .	8
2.4	Movements visualization . . . . .	10
2.5	Data processing . . . . .	11
2.5.1	Cleaning . . . . .	11
2.5.2	Normalization . . . . .	12
2.5.3	Transformation . . . . .	12
<b>3</b>	<b>Methodology</b>	<b>13</b>
3.1	Overview of the models . . . . .	13
3.1.1	Scikit-learn library . . . . .	13
3.1.2	Models selection . . . . .	13
3.2	Models analysis . . . . .	14
3.2.1	Random forest . . . . .	14
3.2.2	Gradient boosting . . . . .	15
3.2.3	Logistic regression . . . . .	16
3.2.4	Linear discriminant analysis . . . . .	16
3.2.5	Multi layer perceptron . . . . .	17
3.3	Data splitting methods . . . . .	19
3.3.1	Traditional . . . . .	19
3.3.2	Effective . . . . .	20
3.3.3	Sequential . . . . .	21
3.4	Feature engineering . . . . .	23
3.4.1	Overview . . . . .	23
3.4.2	Calculation Methods . . . . .	23

<b>4 Results and Discussion</b>	<b>27</b>
4.1 Models evaluation . . . . .	27
4.1.1 Validation . . . . .	27
4.1.2 Metrics . . . . .	29
4.2 Results . . . . .	30
4.2.1 Exploratory . . . . .	30
4.2.2 Effective . . . . .	31
4.3 Discussion . . . . .	36
<b>5 Conclusions</b>	<b>38</b>
5.1 Discoveries . . . . .	38
5.2 Limitations . . . . .	39
5.3 Future work . . . . .	39
<b>Bibliography</b>	<b>41</b>

# Chapter 1

## Introduction

This thesis is structured in the following way: **Chapter 1** presents the problem statement, literature review, dataset overview and the aims and objectives of the study. **Chapter 2** presents the data collection methodology, data structure and attributes, participants characteristics, movements visualization and data processing. **Chapter 3** presents the methodology used in this study, including the models used, data splitting and feature engineering. **Chapter 4** presents the evaluation metrics used, the results obtained and the discussion of the results. Finally, **Chapter 5** presents the conclusions of this study and future work.

### 1.1 Problem statement

Traditionally, movement classification requires high quality sensors and complicated computer vision algorithms. However, with the advent of the Microsoft Kinect sensor and the release of the Kinect SDK [15], it is now possible to obtain high quality 3D skeleton data with a relatively low cost device and with minimal effort. This opens up the possibility of using this data to classify movements performed by individuals, which can be used in a variety of applications, such as rehabilitation, sports, and fall risk assessment. In this thesis, the focus is on the latter, with the goal of using Kinect skeleton data to classify movements performed by elderly individuals.

### 1.2 Literature review

In recent times, the detection and classification of human activity have found a wide range of applications in various fields. Among various sensors used, the Kinect sensor stands out for its affordability and ease of use. S.A. Abdul Shukor et al. conducted a study utilizing the Kinect sensor to detect human falls. Their system demonstrated accurate results [25]. Tao Wang et al. conducted a study that involved gait analysis using the Kinect sensor for the automatic and real time detection of depression. The model developed achieved a classification accuracy of 93.75% [28]. Naveen Kumar Mangal et al. conducted a study that developed a filter to improve the quality of three dimensional coordinate data surrounding the body. With the aim of generating a movement signature crucial for the kinematic analysis of musculoskeletal disorders. Findings from the study reveal that the range of

motion values derived from the proposed filter significantly improved the monitoring accuracy of skeletal joints using the Kinect sensor [18]. Shalini Nehra et al. created a human activity recognition system designed for indoor monitoring and detection of daily activities using a Kinect sensor. The results demonstrated the system's consistently high accuracy across various dataset, as reported in [20]. Tan-Hsu et al. conducted a study that developed a detection system to identify both daily and abnormal activities in elderly individuals. The performance was evaluated using a fourfold cross validation approach, with a precision at 95,5%, recall at 95,6%, specificity at 99,8%, accuracy at 99,6%, and an F1 score of 95,3%.[27]. Weyan Ren et al. used the Kinect sensor to gather posture data from twenty individuals while lying in bed. Data was then subjected to a Machine Learning approach using the Support-Vector Machines architecture, resulting in a classification success rate of 97,1% [24]. Ömer Faruk İnce et al. introduced an innovative biometric system designed to identify human activities within three dimensional space. The study used K-Nearest Neighbor algorithm as part of a Machine Learning approach for classification, achieving an accuracy of 86,1%. [33]. Pramod Kumar Pisharady and Marting Saerbeck presented a multi class algorithm for human posture detection and recognition. This algorithm remained invariant to changes in position and scale by leveraging geometric properties from Kinect data. Tested in both offline and real time applications, it achieved a classification success of 95,78%. [23]. Wen-June Wang et al. introduced a method for accurately distinguish between various postures of five different individuals. A classification success rate exceeding 99% was achieved. [28].

After reviewing the literature, a consistent trend was observed: the effectiveness of motion classification tends to reduce as the number of classes increases. Therefore, there is a recognized need for further research to improve classification performance in multi class scenarios [3]. This study is specially focused on improving classification accuracy across 10 distinct classes.

### 1.3 Dataset overview

In this thesis, a dataset of Kinect skeleton data was used, provided by the PsyComp Lab. The dataset is composed of recorded movements performed by a group of 22 individuals. The movements were performed in front of a Kinect sensor, which recorded the movements and saved them as a series of 3D coordinates. The dataset contains 10 different movements, each performed a various number of times by each individual. The movements are listed in Table 1.1.

**Table 1.1.** Movements used in this study, along with a brief description.

No.	Movement Name	Description
1	Reach Overhead	In a standing position, the subject raises one of their arms above their head.
2	Chair to Chair	Starting from a sitting position, the subject stands up, then sits down on another chair.
3	Cross-Reach Left	In a standing position, the subject using their left arm reaches across their body to the right side.
4	Cross-Reach Right	In a standing position, the subject using their right arm reaches across their body to the left side.
5	Reach Forward	In a standing position, the subject reaches forward with one of their arms.
6	Hoop Walk	Starting from a standing position, the subjects walks inside a hoop placed on the floor and then walks out of it.
7	Right Leg Stand	In a standing position, the subject raises their left leg and holds it in the air for a few seconds.
8	Left Leg Stand	In a standing position, the subject raises their right leg and holds it in the air for a few seconds.
9	Mat Walk	Starting from a standing position, the subject walks over a mat placed on the floor and then off it.
10	TUG Walk	Starting from a sitting position, the subject is asked to stand up, walk 3 meters, turn around, walk back to the chair and sit down while being timed.

## 1.4 Objectives

In this thesis work the task that was set to be accomplished was to *classify movements using kinect skeleton data*, this task was divided into several objectives that would help to accomplish it. The objectives are described as follows:

1. Visualize and label the Kinect skeleton data using 3D plots animations.
2. Preprocess data to remove noise and outliers for better classification results.
3. Analyze different approaches for handling the data, such as using raw data or applying Feature Engineering techniques.
4. Implement and evaluate various Machine Learning models.
5. Conduct a comprehensive comparative analysis of the performance of the models based on evaluation metrics and execution time.
6. Provide insights into the interpretability of selected models, aiding in the understanding of the approaches used by the models to classify movements.

# Chapter 2

## Dataset analysis

In this chapter, the dataset used in this thesis is analyzed. The data collection methodology is described, along with the recording setup. The data structure and attributes are presented, along with the participants characteristics. Finally, the data processing steps are described.

### 2.1 Data collection methodology

Data used in this thesis was collected at the Waterford Hospital in Ireland as part of a Fear of Falling study conducted on a group of 22 elderly individuals. A Microsoft Kinect sensor was used to record the movements performed following a recording setup.

#### 2.1.1 Microsoft kinect

The first generation of the Kinect sensor, Kinect V1 in Figure 2.1, is a motion sensing input device developed by Microsoft and first released in 2010 for game consoles and Microsoft Windows PCs [31]. A new version of the Kinect sensor, Kinect V2, was released in 2014, with improved hardware and software [9].

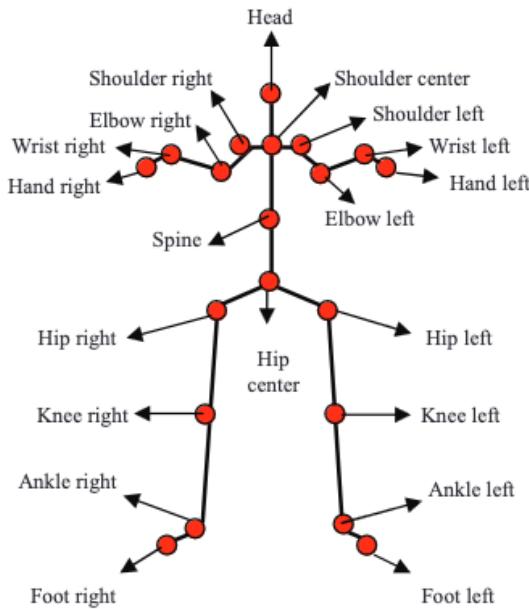


**Figure 2.1.** Microsoft Kinect Sensor.

#### Kinect sensor

The Kinect Sensor is a horizontal bar connected to a small based with a motorized pivot and is designed to be positioned lengthwise above or below the video display. The device features a color camera, an Infrared (IR) emitter, an IR depth sensor, an

engine for tilting, a microphone array, and an LED light [2]. The sensor is capable of sending three types of data: color images, 3D depth images, and bone information corresponding to the 3D imaging field [32][3]. Along with its open source libraries, the Kinect system has helped to develop a wide range of applications in the fields of computer vision, robotics, and human-computer interaction. This is because the Kinect offers a cost effective and broadly accessible method for capturing 3D human motion data, with the advantage of allowing users to interact with the system without the need for any physical devices [10].



**Figure 2.2.** Skeletal joints recognized by the Kinect sensor [14].

### PyKinect2

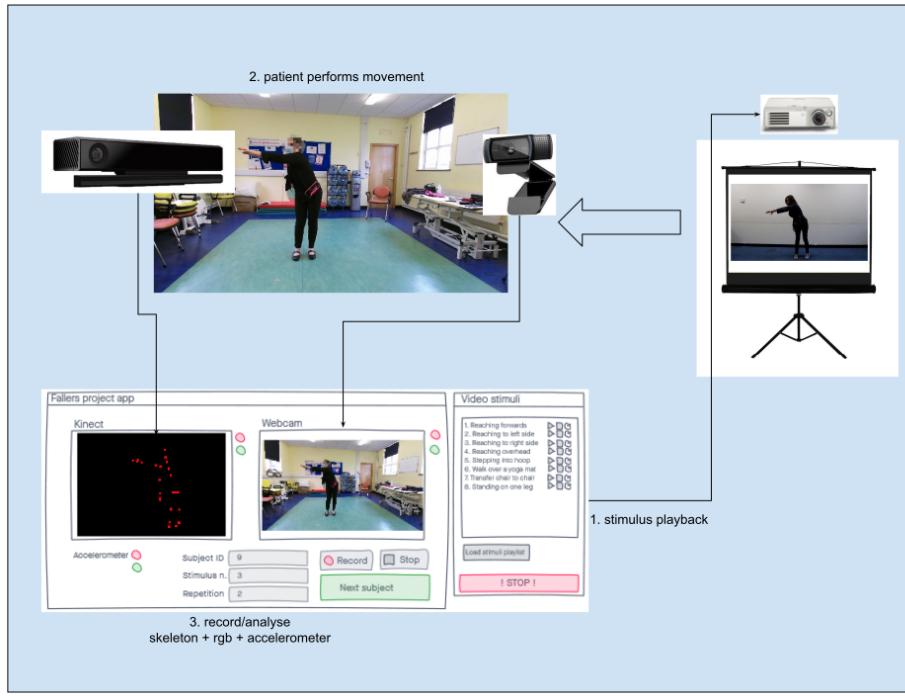
PyKinect2 is a Python library for Microsoft's Kinect v2 sensor. It provides a wrapper for the Kinect for Windows SDK 2.0, which allows for the use of the Kinect sensor in Python. The library abstracts the complex functionality of the Kinect v2 hardware into an easy-to-use API. Key features include:

- **Skeletal Tracking:** detects and tracks human bodies, providing joint positions and orientations.
- **Color, Depth, and Infrared Streams:** Accesses raw sensor streams for visual processing or analysis.
- **Coordinate Mapping:** translates between different spatial representations, such as mapping skeletal joints to color or depth images for overlay visualization.

The library is a great bridge between the Kinect sensor and the Python programming language, allowing for the use of the sensor in a variety of applications [1].

### 2.1.2 Recording setup

The patient's data recording setup illustrated in Figure 2.3 consists of consumer-level hardware (a laptop, a Kinect v2 depth camera, an external webcam, and a smartphone) and a dedicated application developed within the project. Once launched, the operator can display a sample stimulus on an external monitor to show the target movements to the patient (*1. Stimulus playback*) so they can repeat (*2. patient performs movement*) them by selecting one of them from a list in the application. Then, by pressing the "record" button (*3. record/analyse skeleton + rgb + accelerometer*), the recording of the patient's full-body movement can be started. The application stores the recorded patient's movement files in a separate folders, naming them based on their patient ID, movement ID, and repetition ID.



**Figure 2.3.** Kinect recording setup.

The full-body capture mainly relies on the PyKinect library [1]. The library provides functions for getting the patient's body segment's position and rotation 25 times per second. The application gets the data and stores it as a multi-dimensional time series (one per body segment and coordinate type) in CSV files like the one shown in Figure 2.4.

	A	B	C	D	E	F	G	H	I
1	timestamp	datetime	Head.state	Head.px	Head.py	Head.pz	Head.rx	Head.ry	Head.t
2	1.666.953.978.8	2022-10-28_11:4	2.00	2371.34.00	9561.15.00	11735.38.00	0.00	0.00	0.00
3	1.666.953.979.2	2022-10-28_11:4	2.00	2783.25.00	9490.54.00	11698.14.00	0.00	0.00	0.00
4	1.666.953.979.3	2022-10-28_11:4	2.00	2958.20.00	9435.11.00	11708.02.00	0.00	0.00	0.00
5	1.666.953.979.4	2022-10-28_11:4	2.00	3033.29.00	9422.21.00	11713.33.00	0.00	0.00	0.00
6	1.666.953.979.4	2022-10-28_11:4	2.00	3144.00.00	9419.49.00	11742.47.00	0.00	0.00	0.00
7	1.666.953.979.5	2022-10-28_11:4	2.00	3249.39.00	9413.10.00	11751.10.00	0.00	0.00	0.00
8	1.666.953.979.6	2022-10-28_11:4	2.00	3339.20.00	9410.33.00	11758.34.00	0.00	0.00	0.00
9	1.666.953.979.6	2022-10-28_11:4	2.00	3415.19.00	9406.38.00	11768.52.00	0.00	0.00	0.00
10	1.666.953.979.7	2022-10-28_11:4	2.00	3483.13.00	9407.19.00	11760.25.00	0.00	0.00	0.00
11	1.666.953.979.7	2022-10-28_11:4	2.00	3529.38.00	9401.21.00	11753.20.00	0.00	0.00	0.00
12	1.666.953.979.8	2022-10-28_11:4	2.00	3580.21.00	9208.26.00	11725.22.00	0.00	0.00	0.00

Figure 2.4. Example of a CSV file containing the Kinect skeleton data.

The accelerometer is captured via a smartphone running an app streaming 3D gyroscope data at 50 frames per second. Again, the application on the computer stores it as a multi-dimensional time series (one per rotation axis). The communication between the smartphone and the computer is based on a wireless network and the Open Sound Control (OSC) protocol [29].

## 2.2 Data structure and attributes

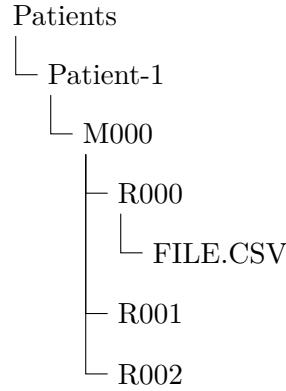
Data used in this thesis consists of a series of CSV files, each containing 3D coordinates of the joints of a participant performing a movement. The total number of csv files is **637**. In Figure 2.5 the distribution of the csv files between the movements is presented with a median of **68** files per movement.



Figure 2.5. CSV files distribution between the movements. *M009* is the only movement with less than 68 files due to it not being performed multiple times by the participants.

Dataset is organized in a directory structure. Each patient has a folder named with their ID (*Patient-ID*) and inside there are 10 folders for each movement, named with the movement ID (*M-XXX*). For every movement folder there is a folder for each repetition of the movement, named with the repetition ID (*R-XXX*). Inside

each repetition folder there is a CSV file that contains the Kinect skeleton data. In Figure 2.6 an example of the directory structure is displayed.



**Figure 2.6.** Directory structure example using the first patient and first movement in the dataset.

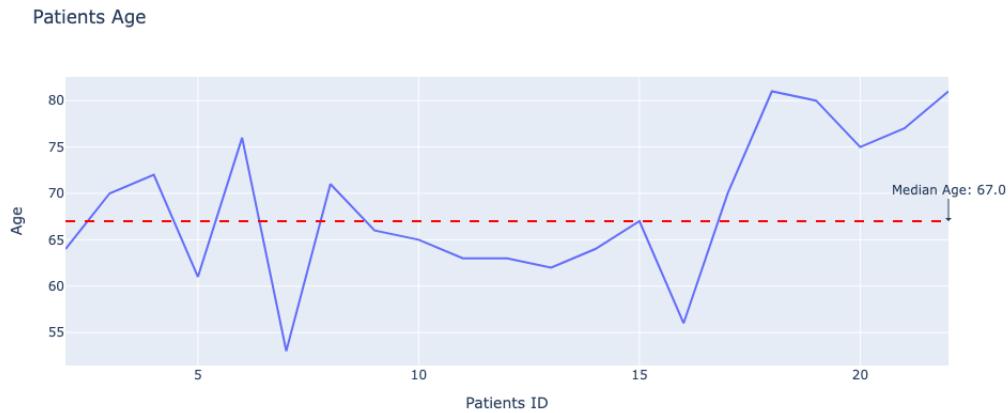
a CSV file is organized as a series of columns, each column represent a joint in Table 2.1 that the Kinect sensor records. Each joint is represented by 4 columns, one for each coordinate (x, y, z) and one for the state. The state column is used to indicate if the joint is tracked or not. Beside the joints columns, there are 2 columns for the timestamp and datetime of the recording.

**Table 2.1.** Joints processed with the PyKinect2 library.

<b>Joints</b>			
AnkleLeft	AnkleRight	ElbowLeft	ElbowRight
FootLeft	FootRight	HandLeft	HandRight
HandTipLeft	HandTipRight	Head	HipLeft
HipRight	KneeLeft	KneeRight	Neck
ShoulderLeft	ShoulderRight	SpineBase	SpineMid
SpineShoulder	ThumbLeft	ThumbRight	WristLeft
WristRight			

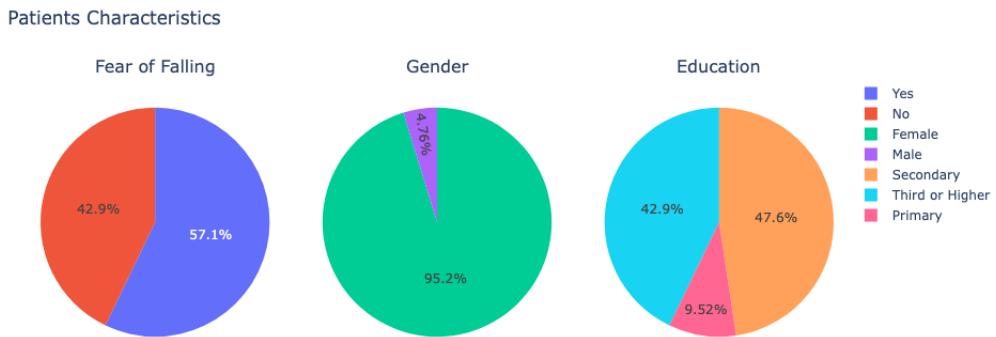
### 2.3 Participants characteristics

The participants that took part in the study were 22 elderly individuals, in Figure 2.7 the age distribution of the patients is presented, with a median age of 67 years.



**Figure 2.7.** Age distribution of the participants.

In Figure 2.8 patients characteristics are presented. Fear of Falling is present in **57%** of the patients, this is a relatively high percentage, due to the study being conducted on a Fear of Falling assessment group. The patients gender is dominated by females with a **95%** of the patients, this is also expected since most studies in the literature had mostly female participants (>50%) [17]. The patients education is also presented, with the majority of the patients having a **Secondary** or **Third Level** education.



**Figure 2.8.** Age distribution of the participants.

### What is fear of falling ?

The definition of **Fear of Falling** had various interpretations over the years. Initially, it was described as a phobic reaction to standing or walking. However, it was reclassified as a syndrome characterized by the aftermath of a fall. As understanding developed, this fear was seen as a loss of confidence in one's balance ability. It was also further defined as an ongoing concern about falling, which leads

to the avoidance of performing daily activities. Recently, it has been described as continuous avoidance of activities due to the concern of falling. [16]

## 2.4 Movements visualization

Kinect-based data comes as a series of 3D coordinates, which can be visualized in 3D space. In this section, the implementation of the movements visualization is described. The visualization was implemented using the Python programming language and the Plotly library [13].

The first step in the visualization process is to identify the set of joints to be used. In this approach, the dataset contains 25 joints but only 16 joints are used and are shown in Table 2.2.

**Table 2.2.** Selected kinect joints used for the visualization.

Joints		
Head	Spine Shoulder	Spine Mid
Spine Base	Shoulder Right	Elbow Right
Wrist Right	Shoulder Left	Elbow Left
Wrist Left	Hip Right	Knee Right
Ankle Right	Hip Left	Knee Left
Ankle Left		

Once the joints are selected, the next step is to transform the data. In its original state the data is organized incorrectly for the 3D visualization, the y and z coordinates are inverted. To fix this, the y and z coordinates are swapped.

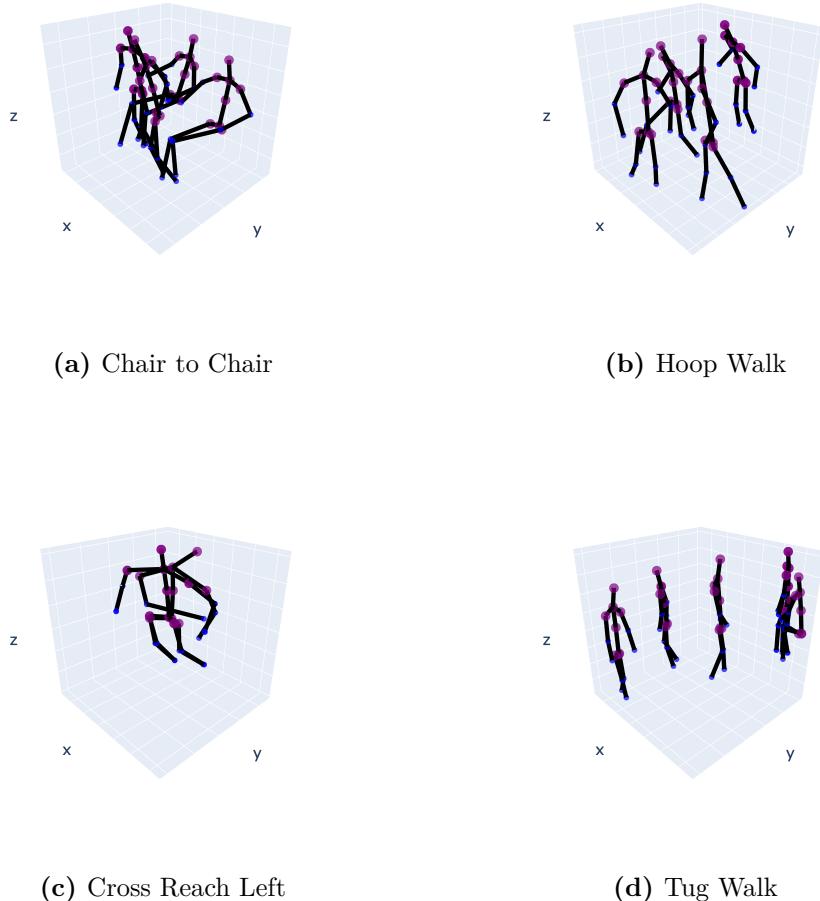
After the transformation is performed, the data is ready to be visualized. Snippet 2.1 shows the implementation of the visualization process, it begins by extracting joint coordinates and their connections, assigning colors and sizes to major joints, and configuring the 3D layout. Animation frames are generated by iteratively capturing snapshots of joint positions and connections over time. These frames are then combined and displayed in an interactive 3D plot, allowing the user to play the animation and rotate the plot to view the movement from different angles.

```

1   for index, row in data.iterrows():
2       x_values = [row[f"{joint}.px"] for joint in joints]
3       y_values = [row[f"{joint}.py"] for joint in joints]
4       z_values = [row[f"{joint}.pz"] for joint in joints]
5       lines = []
6       for connection in connections:
7           start, end = connection
8           lines.append(go.Scatter3d(
9               x=[row[f"{start}.px"], row[f"{end}.px"]],
10              y=[row[f"{start}.py"], row[f"{end}.py"]],
11              z=[row[f"{start}.pz"], row[f"{end}.pz"]]))
```

**Listing 2.1.** Code snippet creates connecting lines between joints using their 3D coordinates, enabling visualization of joint movements.

In Figure 2.9 a set of movements performed by the participants are presented. The movements are displayed in a 3D plot, with the x, y, and z axes representing the plot axes.



**Figure 2.9.** Visualization of movements performed by the participants. Each plot is a 3D visualization containing frames that display the animation.

## 2.5 Data processing

Original dataset containing the Kinect skeleton data was processed to remove noise and outliers. This process is needed to improve the classification results. The data processing steps are described in the following sections.

### 2.5.1 Cleaning

From the original dataset a process of cleaning the data was performed. Consisting of removing the columns that contained zero values and the ones that were not

needed for the classification task. The columns that were kept are listed in the Table 2.2, only the positional coordinates were kept, the state columns and rotation columns (x, y, z) were removed due to not giving any useful information for the classification task.

### 2.5.2 Normalization

Pose normalization was performed using the formula described in [19] as follow:

$$P_{n,i}(x, y, z) = P_{n,i}(x, y, z) - P_{spinebase,1}(x, y, z) \quad (2.1)$$

Equation 2.1 is used to normalize the pose of a participant performing a movement. The pose is normalized by subtracting the coordinates of the Spine Base joint in the first frame from the coordinates of all the joints in the dataframe. This is done to remove the effect of the position of the participant in the recording setup and align all the frames to the same position.

### 2.5.3 Transformation

Once the data cleaning and normalization was performed, the data was transformed into a format that can be used for a classification task. The data was transformed using the scikit-learn library [6], **MinMaxScaler** was used to scale the data between 0 and 1 then **StandardScaler** was used to standardize the data. After this process the data was ready to be used for a Machine Learning model.

# Chapter 3

## Methodology

In this chapter, the methodology used to split the data and train the models is presented. In addition, the fundamental concepts behind the models used are explained.

### 3.1 Overview of the models

In this comparative study a total of ten popular models are selected for analysis of their performance on Kinect-based data.

#### 3.1.1 Scikit-learn library

Scikit-Learn is a Python library designed for Machine Learning, it offers a wide range of *state of the art* algorithms for medium scale supervised and unsupervised problems. It emphasizes ease of use, performance, and API consistency, targeting non specialists with its high level approach. It stands out for its minimal dependencies and broad accessibility, being distributed under the simplified BSD license. It integrates well with the Python ecosystem, making it highly desirable for both academic and commercial applications [22].

#### 3.1.2 Models selection

The models presented in Table 3.1 are used for the classification task. Selected on a basis of popularity and performance, these models are widely used in the machine learning community. The models are implemented using the *scikit-learn* library and its functions for training, testing and evaluating[6].

Model Name	
1	Support Vector Machine
2	Gaussian Naive Bayes
3	Random Forest
4	Gradient Boosting
5	Logistic Regression
6	Linear Discriminant Analysis
7	Multi-Layer Perceptron
8	K-Nearest Neighbors
9	AdaBoost
10	Decision Tree

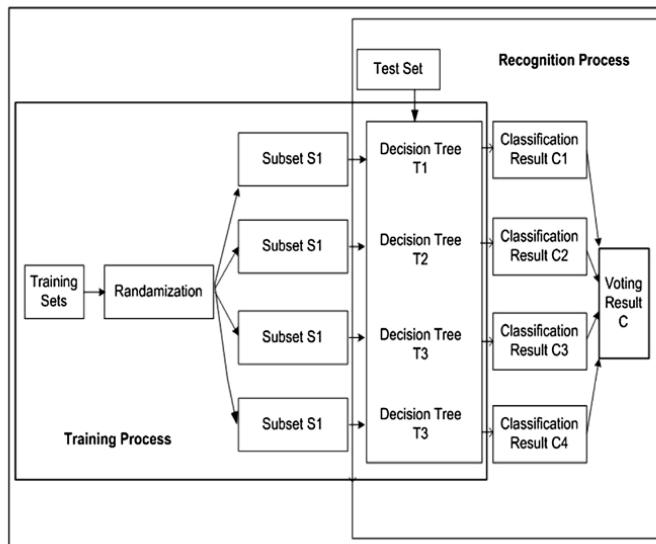
**Table 3.1.** Models selected for use in this thesis.

## 3.2 Models analysis

In this section the models that performed best in the Chapter 4 are analyzed. The models are analyzed in terms of their implementation.

### 3.2.1 Random forest

Also known as *random decision forests*, is a method of ensemble learning used for classification, regression, and various other tasks. It involves building numerous decision trees during the training phase. In classification tasks, the class chosen by the majority of trees is the output of the random forest [12].



**Figure 3.1.** The process starts with multiple training sets that undergo randomization to create several subsets S1. Each subset is used to train a separate decision tree (T1 to T3). The trained trees are then used to make predictions on a test set. The predictions (C1 to C4) from each tree are aggregated through a voting mechanism to produce a final classification result (C). This ensemble approach leverages multiple models to improve prediction accuracy and robustness. [21].

The main steps involved in building a random forest classifier are as follows:

1. Define  $M$  as the number of features in each subset.
2. Randomly select a feature subset  $\theta_k$  from the full set, distinct from preceding subset  $\theta_1, \dots, \theta_{k-1}$ .
3. Train decision trees on each  $\theta_k$  denoted as  $h(X, \theta_k)$ .
4. Iteratively select new  $\theta_k$  subsets and train until all trees are built.
5. Classify test data by majority vote of all trees in the forest.

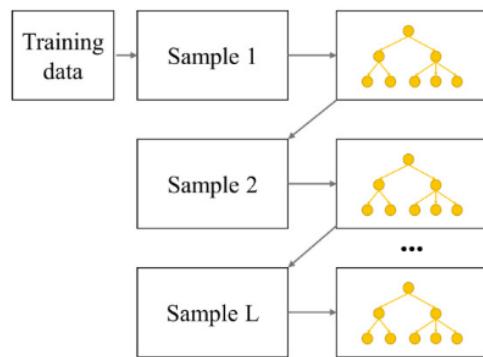
Random forests consist of numerous decision trees. Randomization in tree building—through sampling instances and feature subsets via bagging—enhances diversity, reducing overfitting and improving generalization. Feature subsets  $\theta_k$  are chosen by bagging, and the importance of features is ranked by their impact on the model’s accuracy. The “strength” and “correlation” of the forest are influenced by  $M$ , with optimal values providing a balance. The random forest’s efficiency is due to its parallel structure, accelerating classification significantly [21].

### 3.2.2 Gradient boosting

Gradient boosting is a machine learning method that refines predictions iteratively, combining the strengths of simple models, like decision trees, into a more accurate ensemble. Each iteration, represented by  $F_m(x)$ , improves upon the last by adding a weighted decision tree  $\rho_m h_m(x)$  that addresses the previous errors. The process follows the *principle of gradient descent*, where  $h_m(x)$  is trained to predict the negative gradient of the loss function, effectively reducing the residual between the predicted and true values. The ensemble begins with a single model  $F_0(x)$ , which is updated by the formula:

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x) \quad (3.1)$$

The aim is to minimize the loss function  $L(y, F_m(x))$  at each step, ensuring the model’s prediction becomes progressively more accurate [5].



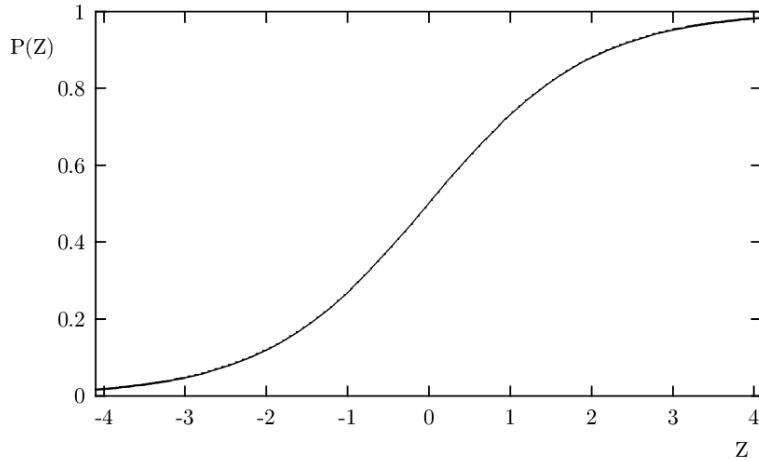
**Figure 3.2.** Starting with the training data, the algorithm iteratively trains decision trees (Sample 1 to Sample L). Each tree is trained on the errors of the previous ones, aiming to correct these mistakes. Over multiple iterations, each tree improves the model’s predictions, and the final output is the combined effort of all the trees, effectively reducing prediction errors. [7].

### 3.2.3 Logistic regression

Logistic regression is a statistical model used for binary classification that predicts the probability of a binary response based on one or more predict variables. It applies a logistic function to a linear combination of the input features to produce a value between 0 and 1, interpreted as the probability of the instance being in the positive class. Equation 3.2 shows the logistic function for binary classification.

$$P(Z) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \quad (3.2)$$

In multi class classification, the one-vs-rest approach involves training a separate logistic regression classifier for each class to distinguish that class from all other classes. For each classifier, the class it's designed to identify is treated as the positive class, and all others are lumped into a single negative class. The logistic function is the same as for binary classification, presented in Equation 3.2. It is applied multiple times, one for each class.



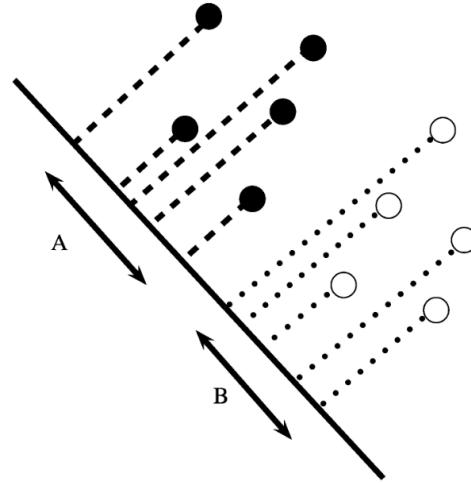
**Figure 3.3.** The horizontal axis labeled 'Z' represents the input variable (which is a linear combination of the features), and the vertical axis labeled 'P(Z)' represents the probability that the outcome is the positive class. The curve transitions smoothly from 0 to 1, with an inflection point at Z=0, where P(Z) = 0.5. This S-shaped curve allows logistic regression to convert continuous predictions into a probability between 0 and 1, facilitating binary classification [8].

### 3.2.4 Linear discriminant analysis

Linear Discriminant Analysis (LDA) is a method used in statistics and machine learning to find a linear combination of features that separates two or more classes of objects or events. It does so by maximizing the ratio of between-class variance to the within-class variance in any particular data set, thereby ensuring maximum separability.

In the binary class, the goal is to find a linear combination  $w$  that separates the classes. This involves computing the mean vectors  $m_1$  and  $m_2$  for each class, the

within class scatter matrix  $S_W$ , and the between class scatter matrix  $S_B$ . The linear discriminants are then the eigenvectors of  $S_W^{-1}S_B$  [30].



**Figure 3.4.** Intuition behind LDA. Data samples in two dimensions are projected in a lower dimension space. The line has to be chosen so that the projection maximizes the "separability" of the projected samples. [30].

For multi class problems, the same principle applies but extends to multiple classes. The within class scatter matrix  $S_W$  and between class scatter matrix  $S_B$  are computed considering all classes, and the objective is to find the linear discriminants that maximize the separation among all classes.

The simplicity and effectiveness of LDA, especially under the assumptions of normality and equal class covariances, make it a powerful tool for classification [4].

### 3.2.5 Multi layer perceptron

Multi-layer perceptron architecture include at least three layers, an input layer, an output layer, and one or more hidden layers, each composed of nodes with nonlinear activation functions [26]. They are referred to as "vanilla" neural networks [11].

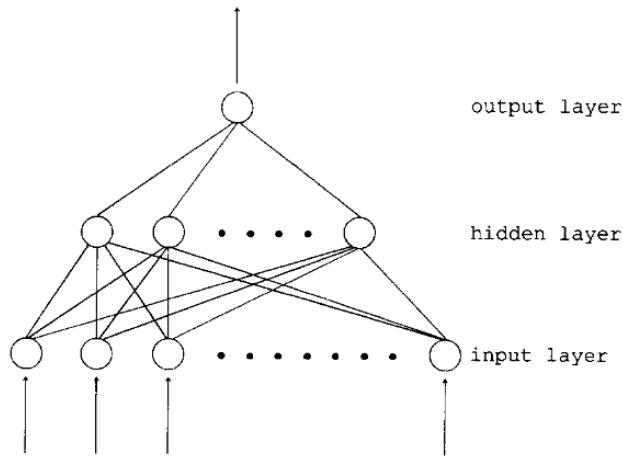
$$y(v_i) = \tanh(v_i) \quad (3.3)$$

$$y(v_i) = (1 + e^{-v_i})^{-1} \quad (3.4)$$

A linear function can simplify multiple layers to a two layer model, mapping weighted inputs to a neuron outputs. Nonlinear activation functions, like the hyperbolic tangent 3.3 ranging from -1 to 1, or the sigmoid function 3.4 ranging from 0 to 1, are used to introduce nonlinearity into the model. This allows the model to learn complex patterns in the data.

In the context of MLP, complete connectivity is maintained, every node within a given layer connects to every node in the subsequent layer via weighted connections.

The learning process involves the dynamic adjustment of connection weights following the processing of each data point. This adjustment is made in response to the disparity between the actual output and the expected outcome, with the goal of minimizing the error.



**Figure 3.5.** Feed forward network consisting of an input layer, one or more hidden layers, and an output layer. [26]

### 3.3 Data splitting methods

Due to the structure of the data, the traditional approach of splitting the data into training and testing sets is not effective. Two different approaches will be presented, one ineffective and one effective.

#### 3.3.1 Traditional

The data is split into 70% training and 30% testing following the traditional approach used in Machine Learning literature. The code snippet in [3.1](#) demonstrates this approach.

---

```

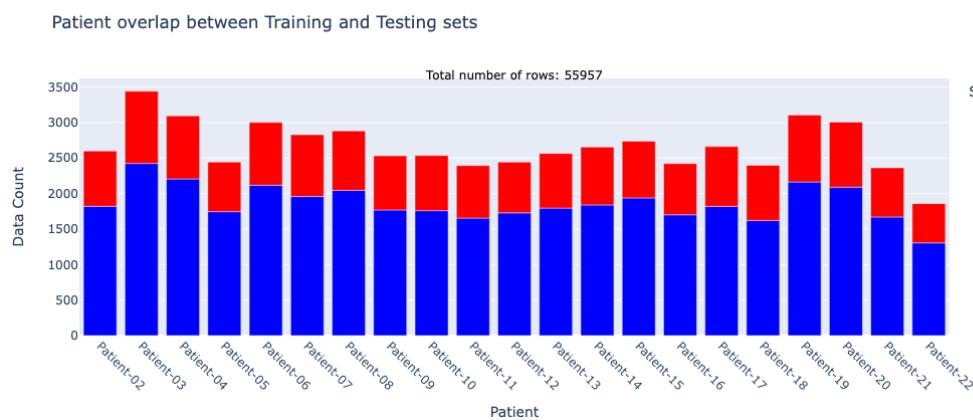
1  def split_data(data: pd.DataFrame) -> tuple:
2      X = data.iloc[:, :-1].values
3      y = data.iloc[:, -1].values
4
5      X_train, X_test, y_train, y_test = train_test_split(
6          X, y, test_size=0.33, random_state=42)
7
8      return X_train, X_test, y_train, y_test

```

---

**Listing 3.1.** Traditional approach to splitting the data into training and testing sets.

Figure [3.6](#) visualization demonstrates why this approach is ineffective. Every row in the dataset is associated with a specific patient. The data is split randomly, so there is a chance that the same patient will appear in both the training and testing sets. This means that the model will be trained on data that it will also be tested on, which will result in a high accuracy score. However, this is not a good indicator of the model's performance on unseen data.



**Figure 3.6.** Visualization of the ineffective data splitting approach.

### 3.3.2 Effective

Data is split into training and testing sets based on the patient's unique ids. The patients are split into training and testing sets, and then the data is split based on the patient's unique ids. The code snippet in [3.2](#) demonstrates this approach.

---

```

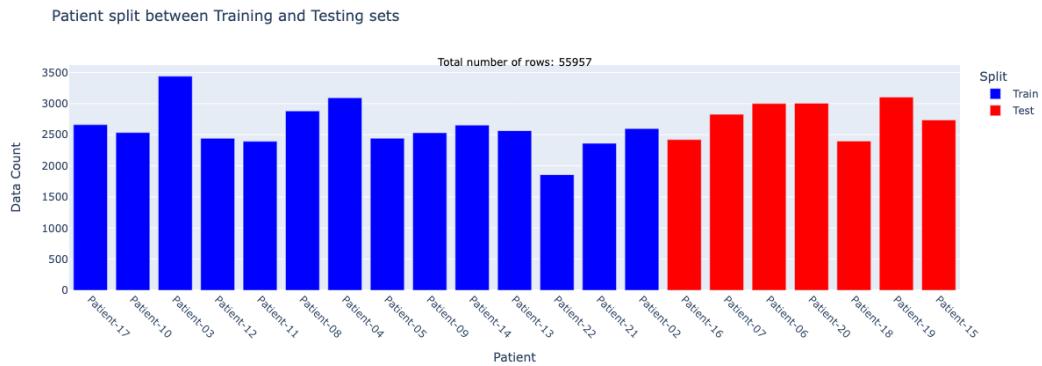
1  def split_data(data: pd.DataFrame) -> tuple:
2      unique_patients = data['patient'].unique()
3
4      train_patients, test_patients = train_test_split(unique_patients,
5          test_size=0.3, random_state=42)
6
7      train_data = data[data['patient'].isin(train_patients)]
8      test_data = data[data['patient'].isin(test_patients)]
9
10     X_train = train_data.drop(columns=['label', 'patient'])
11     y_train = train_data['label']
12
13     X_test = test_data.drop(columns=['label', 'patient'])
14     y_test = test_data['label']
15
16     return X_train, X_test, y_train, y_test

```

---

**Listing 3.2.** Effective approach to splitting the data into training and testing sets.

Figure [3.7](#) visualization demonstrates why this approach is effective. The data is split based on the patient's unique ids, so the model will be trained on data that it will not be tested on. This means that the model will be tested on unseen data, which is a good indicator of the model's performance.



**Figure 3.7.** Visualization of the effective data splitting approach.

### 3.3.3 Sequential

Data is split into training and testing sets based on the patient's unique ids, then the sets are split into sequences. Where each sequence represents the stack of frames that make up a movement. The code snippet in 3.3 demonstrates the splitting into sequences of the data.

---

```

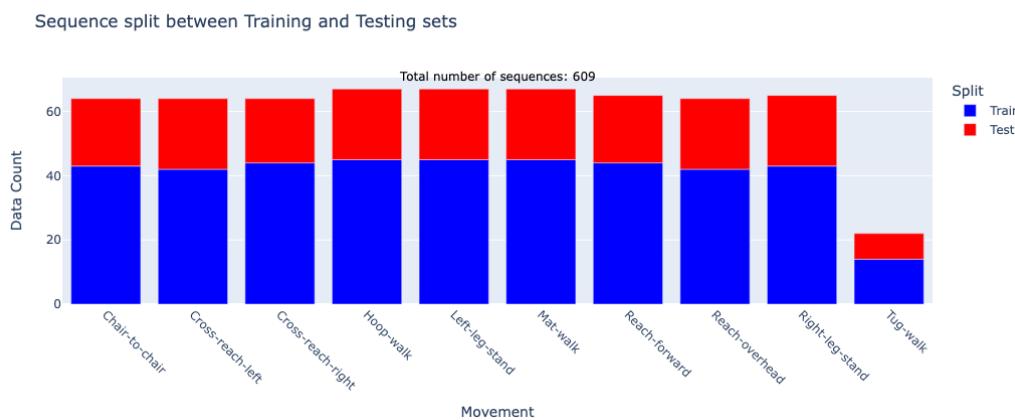
1  def sequences(df: pd.DataFrame, feature_columns: list,
2      sequence_column: str) -> tuple:
3      sequences = []
4      labels = []
5      current_sequence = []
6      current_check = None
7
8      for _, row in df.iterrows():
9          check = row[sequence_column]
10         label = row['label']
11         if check != current_check and current_sequence:
12             sequences.append(np.array(current_sequence))
13             labels.append(label)
14             current_sequence = []
15             current_sequence.append(row[feature_columns].to_numpy())
16             current_check = check
17
18         if current_sequence:
19             sequences.append(np.array(current_sequence))
20             labels.append(label)
21
22     return sequences, labels

```

---

**Listing 3.3.** Effective approach to splitting the data into training and testing sets.

Figure 3.8 visualization shows how for each movement the data is split into sequences for training and testing. However, using only this approach is not enough, as the sequences are of different lengths due to each movement having a variable number of frames. This means that the sequences cannot be used as input for the models since they require a fixed input size.



**Figure 3.8.** Visualization of the sequences splitting approach.

To solve the variable length problem, the sequences are aggregate into a single feature vector. The code snippet 3.4 demonstrates this approach. In Figure 3.9, the length of the sequences before and after aggregation is visualized. The aggregation is done by calculating the mean of each feature for each frame in the sequence. This results in a single feature vector for each sequence, which can be used as input for the models.

---

```

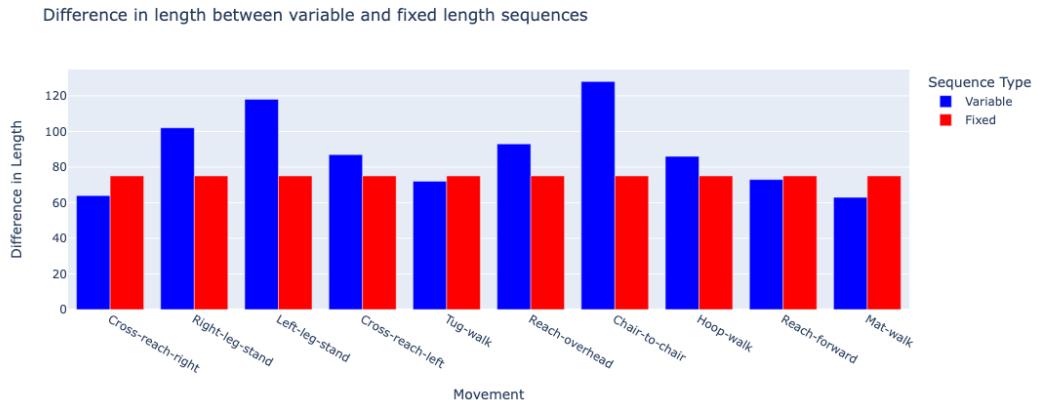
1  def aggregate_features(sequences: list) -> np.ndarray:
2      return np.array([np.mean(sequence, axis=0) if sequence.size
3                      != 0 else np.zeros(sequence.shape[1]) for sequence in
4                          sequences])

```

---

**Listing 3.4.** Sequences are aggregated into a single feature vector.

However, there are some drawbacks to this approach. The aggregation results in a loss of information, as the data is no longer represented as a sequence of frames. In addition, the aggregation results in a loss of the temporal information, as the order of the frames is lost. This means that the models will not be able to learn the temporal patterns in the data.



**Figure 3.9.** Visualization of the length of the sequences before and after aggregation.

## 3.4 Feature engineering

Feature engineering is the final approach used in this thesis. It is used to extract new features from the raw Kinect skeleton data, in order to improve the performance of the models.

### 3.4.1 Overview

This process is implemented in order to improve the performance of the models due to them not being able to differentiate well between the movements based on the raw data. This allows to obtain data that is more informative and easier to interpret. The features extracted from the Kinect Skeleton data are presented in Table 3.2.

Features			
1 Duration	2 Area		
3 Velocity	4 Distance		
5 Vertical Displacement	6 Horizontal Displacement		
7 Forward Displacement			

**Table 3.2.** Features extracted from the Kinect Skeleton data.

### 3.4.2 Calculation Methods

The features presented in Table 1.1 are calculated using the following methods. For each of the features, the method used to calculate it is presented, along with a brief description.

Body parts selected			
Head	ShoulderLeft	ShoulderRight	SpineShoulder
SpineMid	SpineBase	ElbowLeft	ElbowRight
WristLeft	WristRight	HipLeft	HipRight
KneeLeft	KneeRight	AnkleLeft	AnkleRight

**Table 3.3.** Selected body parts from the Kinect Skeleton data joints.

### Duration

The duration is defined as how long it takes for the movement to be performed from start to finish. It is calculated as the difference between the maximum and minimum datetime column values. The difference is calculated in seconds.

$$\text{Duration} = (\text{max\_datetime} - \text{min\_datetime}) \quad (3.5)$$

## Area

The area is defined as the aggregate area of convex hulls formed by the trajectories of selected body parts. It operates by extracting the (x,y,z) coordinates for each specified body part, constructing a convex hull for these points, and then calculating the hull's volume.

$$\text{Area} = \sum_{i=0}^n \text{Volume}(\text{Hull}(P_i)) \quad (3.6)$$

In Equation 3.6,  $P_i$  is the set of points representing the trajectory of body part  $i$  in 3D space, where  $i \in \{1, 2, \dots, n\}$  for  $n$  body parts. The convex hull of  $P_i$  is denoted as  $\text{Hull}(P_i)$ , is the smallest convex set that contains all points in  $P_i$ . The volume (area in 3D) of  $\text{Hull}(P_i)$  is calculated using the formula for the volume of a convex polyhedron, which depends on the vertices of the hull. The total area calculated is the sum of the volumes of these convex hulls for app specified body parts.

---

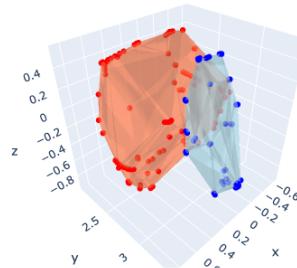
```

1  def area(df: pd.DataFrame, body_parts: list) -> float:
2      def calculate(points: np.ndarray) -> float:
3          hull = ConvexHull(points)
4          return hull.volume
5      trajectories = {}
6      for column in body_parts:
7          body_part = column.split('.')[0]
8          trajectory = df[[body_part + '.px', body_part + '.py',
9                            body_part + '.pz']].values
10         trajectories[body_part] = trajectory
11     temp = {}
12     for body_part, trajectory in trajectories.items():
13         temp[body_part] = calculate(trajectory)
14     return sum(temp.values())

```

---

**Listing 3.5.** Area calculation method using the ConvexHull class from the SciPy library.



**Figure 3.10.** Visualization of the area occupied by two movements, the red area represents *Chair to Chair* while the blue area represents *Right Leg Stand*.

## Velocity

The velocity is defined as the rate of change of the displacement over time. It is calculated as the square root of the sum of the squared displacement over time difference for each axis.

$$\text{velocity} = \sqrt{\left(\frac{\text{displacement}_x}{\text{time difference}}\right)^2 + \left(\frac{\text{displacement}_y}{\text{time difference}}\right)^2 + \left(\frac{\text{displacement}_z}{\text{time difference}}\right)^2} \quad (3.7)$$

---

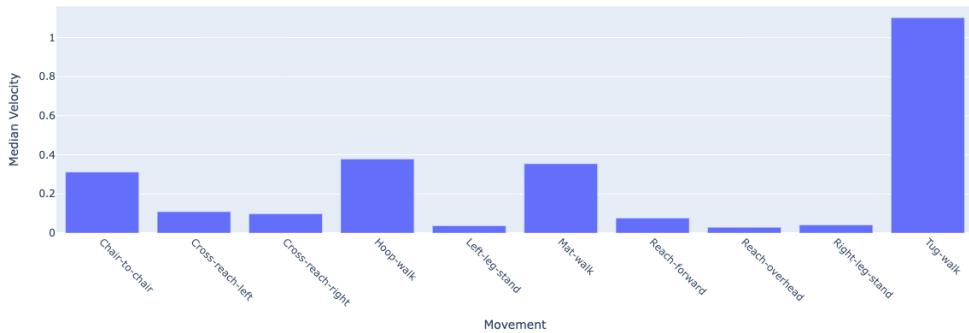
```

1  def velocity(df: pd.DataFrame) -> float:
2      first = df.iloc[0]
3      last = df.iloc[20]
4      start = first['datetime'].split('_')[1].split('.')[0]
5      end = last['datetime'].split('_')[1].split('.')[0]
6      diff = pd.to_datetime(end) - pd.to_datetime(start)
7      velx = last['Head.px'] - first['Head.px'] / diff.total_seconds()
8      vely = last['Head.py'] - first['Head.py'] / diff.total_seconds()
9      velz = last['Head.pz'] - first['Head.pz'] / diff.total_seconds()
10     return math.sqrt(velx**2 + vely**2 + velz**2)

```

---

**Listing 3.6.** Velocity calculation method.



**Figure 3.11.** Visualization of the median velocity of each movement in the dataset. This shows how the velocity varies between the movements and can be used to differentiate between them.

## Distance

The distance is defined the total 3D Euclidean distance between consecutive points representing the position of a body part, typically the head. It is calculated by taking each pair of consecutive rows in the dataset, computing the distance between their (x,y,z) head positions in 3D space using the Euclidean distance formula, and summing up these distances to find the overall total distance covered by the body part in the sequence.

$$\text{Distance} = \sum_{i=0}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2} \quad (3.8)$$

---

```

11     def calculate(row1: pd.Series, row2: pd.Series, point="Head") ->
12         float:
13             return np.sqrt((row2[f'{point}.px'] - row1[f'{point}.px'])**2 +
14                             (row2[f'{point}.py'] - row1[f'{point}.py'])**2 +
15                             (row2[f'{point}.pz'] - row1[f'{point}.pz'])**2)
16
17     def distance(df: pd.DataFrame) -> float:
18         return sum(calculate(df.iloc[i], df.iloc[i+1]) for i in range(len(df) - 1))

```

---

**Listing 3.7.** Distance calculation method.

### Time steps displacement

The following joints: *AnkleLeft*, *AnkleRight*, *WristLeft*, *WristRight*, *SpineMid* have been selected for the calculation of the displacement. These joints have been selected as they are the most informative for the movements in the dataset.

Time steps displacement is defined as the total change in the position of a body part from the start to the end of the movement. It is calculated by taking the absolute differences in the positions of the body part between consecutive time steps, and then summing up these differences over all time steps.

$$\text{Time steps displacement} = \sum_{t=1}^n |P(t) - P(t-1)| \quad (3.9)$$

In Equation 3.9,  $P$  is a placeholder for the axis positions of the body part, and  $t$  is the time step.

1. **Vertical** is calculated by taking the  $Y$  axis positions.
2. **Horizontal** is calculated by taking the  $X$  axis positions.
3. **Forward** is calculated by taking the  $Z$  axis positions.

---

```

18     def vertical(df: pd.DataFrame, joint: str) -> float:
19         df['vertical_diff'] = df[f'{joint}.py'].diff().abs()
20         total_vertical = df['vertical_diff'].sum()
21         return total_vertical

```

---

**Listing 3.8.** Vertical time steps displacement calculation method.

# Chapter 4

## Results and Discussion

This chapter presents the result obtained from the experiments conducted in the previous chapter. Classification models are evaluated using different approaches and metrics. The results are then discussed and compared to each other.

### 4.1 Models evaluation

Performed using the *Scikit-Learn* library [6]. It provides a wide range of validation methods and metrics to evaluate the performance of the models. The following sections will present the validation methods and metrics used in this thesis.

#### 4.1.1 Validation

Validation is the process of evaluating the performance of the models. The goal of validation is to estimate the performance of the model on new data, not used during the training process. The following validation methods are used:

##### **Hold-Out**

This method is widely used for its simplicity and speed. The dataset is split into two subsets. The Training set is used to train the model, Testing set is used to evaluate the performance of the model. Typically, a common split ratio is:

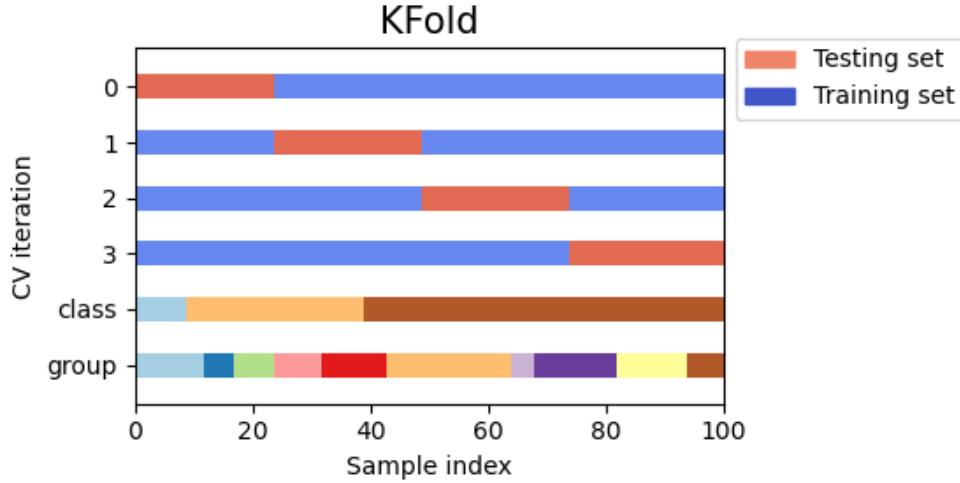
- **Training set:** 70% of the dataset.
- **Testing set:** 30% of the dataset.

##### **Cross-Validation**

This method is used in the literature for its effectiveness and robustness. It can be time consuming for large datasets, but it is the best method to evaluate the performance of the models.

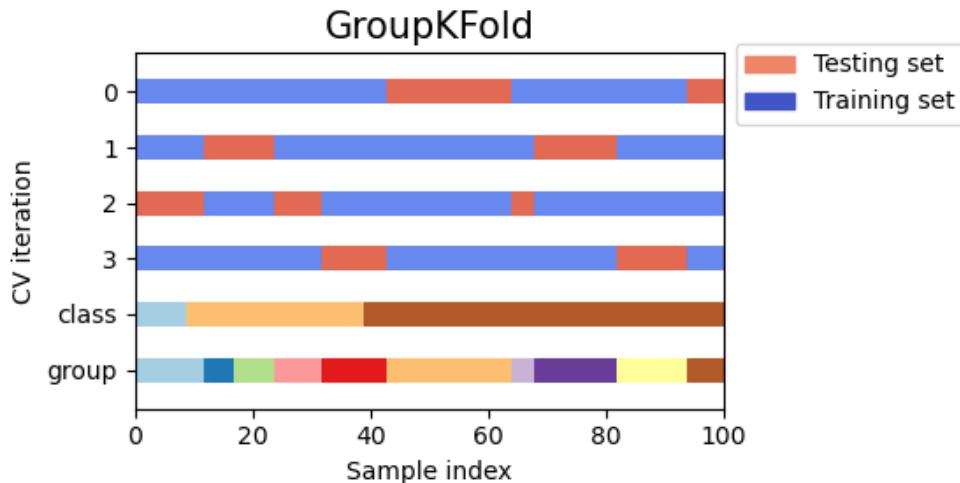
- **K-fold:** The data is divided into  $K$  folds, then  $K-1$  folds are used for training and the remaining fold is used for testing. This process is repeated  $K$  times,

with each fold being used exactly once for testing. The fig:kfold shows an example of a k-fold split. Fig 4.1 shows the KFold split.



**Figure 4.1.** KFold Visualization from the scikit-learn documentation [22].

- **Group-k-fold:** Variation of k-fold designed for situations where the data has inherent groupings or dependencies that should be preserved in the train/test split. In this method, the data is divided into  $K$  folds, then an additional constraint is imposed to ensure that data points from the same group are in the same fold. Fig 4.2 shows the GroupKFold split.



**Figure 4.2.** GroupKFold Visualization from the scikit-learn documentation [22].

The advantage of using *Cross-Validation* over *Hold-Out* is that all the samples are used for both training and testing, and each sample is used for testing exactly once. This method helps to reduce the variance of the estimated performance of

the model, by averaging the results over a number of trials. The disadvantage of using Cross-Validation is that it is computationally expensive for very large datasets.

In this study, both methods are used to evaluate the performance of the models. The Hold-Out method is used to evaluate the performance of the models for the *Wrong approach* and *Sequence approach* datasets due to their large dimension. The Cross-Validation method is used with the Hold-Out method to evaluate the performance of the models for the *Correct approach* and *Feature Engineering approach* dataset due to them scoring the best results and being the effective approaches. Confronting the results of the two methods will show the correctness of the evaluation, as the results should be similar.

#### 4.1.2 Metrics

This section will report the metrics used to benchmark the different models used in this study.

##### Accuracy score

The *accuracy* is the proportion of correct predictions, considering both true positives and true negatives, among the total number of samples. The formula used to calculate the accuracy is the following:

$$\frac{TP + TN}{TP + TN + FP + FN} \quad (4.1)$$

where **TP** is the number of true positives, **TN** is the number of true negatives, **FP** is the number of false positives and **FN** is the number of false negatives.

##### Precision score

The *precision* is the ability of the classifier not to label as positive a sample that is negative. The formula used to calculate the precision is the following:

$$\frac{TP}{TP + FP} \quad (4.2)$$

##### Recall score

The *recall* is the ability of the classifier to find all the positive samples. The formula used to calculate the recall is the following:

$$\frac{TP}{TP + FN} \quad (4.3)$$

##### F1 score

The *F1 score* is the harmonic mean of the precision and recall. The formula used to calculate the F1 score is the following:

$$\frac{2 \times (precision \times recall)}{precision + recall} \quad (4.4)$$

### Matthews correlation coefficient

The *Matthews correlation coefficient* (or  $\varphi$  coefficient) takes into account true and false positives and negatives and is regarded as a balanced measure which can be used even if the classes are of very different sizes. The formula used to calculate the  $\varphi$  coefficient is as follows:

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (4.5)$$

These metrics will be used to show the effectiveness of the approaches proposed in this thesis.

## 4.2 Results

Combining the validation methods and metrics presented above, the following tables will show the results obtained from the experiments conducted. The results are divided into two categories: **Exploratory** shows two approaches that were tested but did not obtain good results due to wrong implementation or loss of information. **Effective** shows two approaches that obtained good results and are suitable for this task. The results are presented in the following order: *Traditional approach*, *Sequence approach*, *Effective approach* and *Feature Engineering approach*.

### 4.2.1 Exploratory

The following approaches are included because they are a starting point in this thesis, and show how different implementations can affect the accuracy of the models.

#### Traditional approach

Pesented in Section 3.3.1, it was the first one to be tested and it got surprisingly good results. Such a simple approach and yet high accuracy raised doubts about the validity of the results, after further investigation it was discovered that the dataset was not properly split into training and testing sets.

**Table 4.1.** Evaluation results using **Hold-Out** validation method.

Model	Accuracy	F1	Recall	Precision	MCC
Random Forest	<b>0.99</b>	<b>0.99</b>	<b>0.99</b>	<b>0.99</b>	<b>0.99</b>
K-Nearest Neighbors	0.98	0.98	0.98	0.98	0.98
Decision Trees	0.96	0.96	0.96	0.96	0.96
Support-Vector Machines	0.87	0.86	0.85	0.86	0.86
Logistic Regression	0.82	0.80	0.80	0.80	0.80

In Table 4.1 the results obtained from the Hold-Out method are displayed. High values are obtained for all the metrics, with **Random Forest** obtaining the highest values with a score of **0.99** for accuracy. This confirmed the doubts about the validity of the results, a patient is both present in the training and testing set. This led to the models overfitting the data and obtaining high accuracy.

### Sequence approach

Presented in Section 3.3.3, it achieved the lowest results of all the approaches. Tested to see if by concatenating the frames of a movement into a sequence would help the models differentiate between movements and obtain a higher accuracy.

**Table 4.2.** Evaluation results using Hold-Out validation method.

Model	Accuracy	F1	Recall	Precision	MCC
K-Nearest Neighbors	<b>0.56</b>	<b>0.54</b>	<b>0.54</b>	<b>0.54</b>	<b>0.51</b>
Random Forest	0.55	0.52	0.53	0.53	0.49
Support-Vector Machines	0.52	0.47	0.49	0.46	0.47
Logistic Regression	0.44	0.41	0.42	0.43	0.38
Decision Trees	0.41	0.38	0.39	0.40	0.34

In Table 4.2 the results obtained from the Hold-Out method are displayed. Low values are obtained for all the metrics, with **K-Nearest Neighbor** obtaining the highest values with a score of **0.56** for accuracy. This results are considered low based on other approaches, however in the context of randomly guessing the movement of a patient the accuracy would be **0.10** as there are 10 movements. This means that the models are able to differentiate between movements, but the sequence implementation leads to a loss of information and a high accuracy cannot be obtained.

### 4.2.2 Effective

The following approaches are the ones that obtained the best results and are suitable for this task. The main difference between the two approaches is the data used to train the models. The *Correct Approach* uses the data as it is from the Kinect sensor, while the *Feature Engineering Approach* uses the data after applying feature engineering techniques.

#### Correct approach

Presented in Section 3.3.2, considered effective because the raw kinect data is able to obtain a high accuracy. The data is not modified in any way, beside the removal of rotational , state and pre-processing the data to remove noise and outliers.

In Table 4.3 the results obtain from the Hold-Out method are shown. **Random Forest** obtains the highest values for all the metrics, with a score of **0.74** for accuracy. Other models such as *Gradient Boosting*, *Linear Discriminant Analysis*, *Support-Vector Machines*, *K-Nearest Neighbors* obtained great results as well with a score greater than **0.70** for accuracy. This confirms that the data obtained from the Kinect sensor is suitable for the task of movement classification without any major tweaks.

**Table 4.3.** Evaluation results using Hold-Out validation method.

Model	Accuracy	F1	Recall	Precision	MCC
Random Forest	<b>0.74</b>	<b>0.73</b>	<b>0.73</b>	<b>0.73</b>	<b>0.71</b>
Gradient Boosting	0.73	0.72	0.72	0.72	0.69
Linear-Discriminant Analysis	0.72	0.71	0.71	0.74	0.68
Support-Vector Machines	0.71	0.71	0.71	0.72	0.68
K-Nearest Neighbors	0.71	0.69	0.69	0.70	0.67
Logistic Regression	0.66	0.64	0.64	0.64	0.62
Multi-Layer Perceptron	0.63	0.59	0.62	0.61	0.59
Naive Bayes	0.63	0.60	0.61	0.62	0.58
Decision Trees	0.63	0.60	0.62	0.61	0.58
Ada Boost	0.35	0.22	0.28	0.24	0.32

In Table 4.3 **Hold-Out** validation method was used for all the models, while in Table 4.4 **Cross-Validation** was used with only 3 models to compare the two validation methods and show that there is no major difference between them. The results obtained from the two methods are similar.

Hold-Out method was used for it's speed, with **10 minutes** of training time while cross-validation run for hours without finishing. This is due to the fact that this approaches use the raw Kinect data, that contains over **59000** rows and **100** columns.

**Table 4.4.** Comparison of obtained results with Cross-Validation and Hold-Out methods.  
The metrics reported are (from top to bottom): Accuracy, F1, Recall, Precision, MCC.

Model	Cross-Validation	Hold-Out
Linear Discriminant Analysis	0.73	0.72
	0.71	0.71
	0.70	0.71
	0.75	0.74
	0.70	0.68
K-Nearest Neighbors	0.71	0.71
	0.70	0.69
	0.70	0.69
	0.71	0.70
	0.68	0.67
Naive Bayes	0.66	0.63
	0.62	0.60
	0.63	0.62
	0.66	0.61
	0.62	0.58

In Table 4.5 are displayed the results of a final approach, where two movements *Mat-Walk* and *Hoop-Walk* are removed from the dataset one at a time. The results show that the accuracy of the models increased by **4%** to **5%**. This is due to the

fact that the two movements are very similar, and this caused the models to struggle to differentiate between them no matter the features used.

By removing either one of the movements, the models accuracy increased by the same amount. This leaves the decision to the user to choose which movement to remove based on the context of the application.

**Table 4.5.** Comparison of obtained results with Mat-Walk and Hoop-Walk removed from the dataset. The metrics reported are (from top to bottom): Accuracy, F1, Recall, Precision, MCC.

Model	Hoop Walk Removed	Mat Walk Removed
Random Forest	0.79	0.79
	0.79	0.79
	0.79	0.79
	0.79	0.79
	0.76	0.76
Gradient Boosting	0.78	0.78
	0.78	0.78
	0.78	0.78
	0.78	0.79
	0.74	0.75
Linear-Discriminant Analysis	0.76	0.76
	0.77	0.77
	0.76	0.76
	0.80	0.79
	0.73	0.73

### Feature engineering approach

Presented in Section 3.4, considered the most effective because it obtained the highest accuracy of all the approaches and it is the fastest to train. The data is modified by applying feature engineering techniques, this leads to the dataset having less rows and columns.

**Table 4.6.** Evaluation results using Cross-Validation method.

Model	Accuracy	F1	Recall	Precision	MCC
Multi-Layer Perceptron	<b>0.83</b>	<b>0.83</b>	<b>0.84</b>	<b>0.84</b>	<b>0.81</b>
Logistic Regression	0.82	0.83	0.83	0.84	0.80
Linear-Discriminant Analysis	0.81	0.82	0.83	0.84	0.80
Support-Vector Machines	0.81	0.82	0.83	0.83	0.80
Random Forest	0.79	0.80	0.80	0.82	0.77
Gradient Boosting	0.78	0.79	0.79	0.82	0.76
K-Nearest Neighbors	0.78	0.79	0.80	0.80	0.76
Decision Trees	0.73	0.73	0.74	0.77	0.70
Naive Bayes	0.63	0.63	0.66	0.66	0.60
Ada Boost	0.46	0.38	0.46	0.42	0.43

In Table 4.6 the results obtained from the Cross-Validation method are displayed. High values are obtained for all the metrics, with **Multi-Layer Perceptron** and **Logistic Regression** obtaining the highest values with a score of **0.83** and **0.82** for accuracy. Other models such as *Linear-Discriminant Analysis*, *Gradient Boosting*, *Random Forest*, *Support-Vector Machines*, *K-Nearest Neighbors* obtained great results as well with a score greater than **0.70** for accuracy.

**Table 4.7.** Comparison of obtained results with Cross-Validation and Hold-Out methods. The metrics reported are (from top to bottom): Accuracy, F1, Recall, Precision, MCC.

Model	Cross-Validation	Hold-Out
Linear Discriminant Analysis	0.81	0.82
	0.82	0.82
	0.83	0.83
	0.84	0.83
	0.80	0.80
Logistic Regression	0.82	0.80
	0.83	0.81
	0.83	0.82
	0.84	0.82
	0.80	0.78
Multi-Layer Perceptron	0.83	0.70
	0.83	0.71
	0.84	0.71
	0.84	0.78
	0.81	0.68

This confirms that the feature engineering techniques applied to the data are suitable for the task of movement classification. This approach is also the fastest to train, with a training time of **1 minute**.

Table 4.7 shows the results obtained from the Hold-Out validation method. Similar to the ones obtained with Cross-Validation method, with a lower training time of **10 seconds**. However, Cross-Validation is preferred over Hold-Out because it is more used in the literature and robust.

In Table 4.8 are displayed the results of the final approach used in the Correct approach. The results show that the accuracy of the models increased by **8%** to **12%**. This is a larger increase than the one obtained in the Correct approach, this is due to the fact that the feature engineering techniques applied are more informative than the raw Kinect data. Leading to the models being able to differentiate between the two movements more easily.

**Table 4.8.** Comparison of obtained results with Mat-Walk and Hoop-Walk removed from the dataset. The metrics reported are (from top to bottom): Accuracy, F1, Recall, Precision, MCC.

Model	Hoop Walk Removed	Mat Walk Removed
Multi-Layer Perceptron	0.90	0.91
	0.90	0.91
	0.90	0.92
	0.92	0.92
	0.89	0.90
Logistic Regression	0.90	0.91
	0.91	0.91
	0.91	0.91
	0.92	0.92
	0.89	0.90
Linear Discriminant Analysis	0.91	0.90
	0.91	0.91
	0.91	0.91
	0.92	0.92
	0.90	0.89

### 4.3 Discussion

This section will discuss the results obtained from the experiments conducted in the previous chapter. The difference between the approaches, the best performing models and the similarity between movements will be the topic of discussion.

#### Differences between approaches

Four approaches were tested in this thesis, each one with a different implementation. It was presented that *Feature engineering* obtained the best results in terms of accuracy and training time. However, the *Correct approach* also obtained great results but it was lacking in training time. These two approaches were considered the most effective and suitable for the task of movement classification. Nevertheless, their implementation is completely different, with one using the raw Kinect data containing a very large number of rows and columns, while the other uses the data after applying feature engineering techniques transforming the data into a more informative dataset. This demonstrates how the approach used to solve a problem can affect the results obtained. The other two approaches *Wrong approach* and *Sequence approach* prove that an incorrect data splitting method and a loss of information can lead to low accuracy. Their implementation is not suitable for this task but were a crucial step in the development phase to better understand why the models were not performing well.

#### Best performing models

The models listed below obtained the best results in the two approaches considered effective.

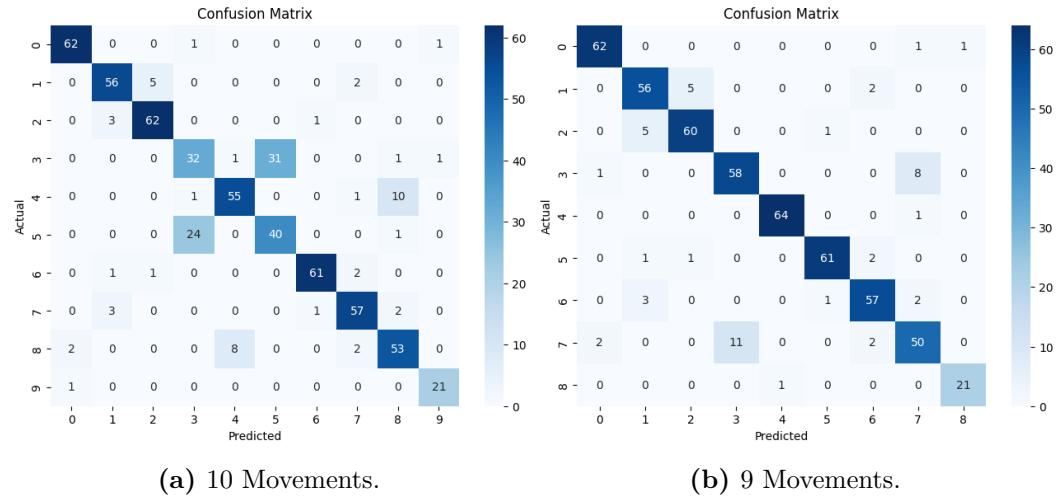
1. **Random Forest** and **Gradient Boosting** are the two best performing models in the *Correct approach* with a **0.74** and **0.73** accuracy score respectively.
2. **Multi-Layer Perceptron** and **Logistic Regression** are the two best performing models in the *Feature engineering approach* with a **0.83** and **0.82** accuracy score.

The results above demonstrate how the models perform differently depending on the approach used, due to the fact that the used to train the models is of different dimension and information.

#### Movements similarity

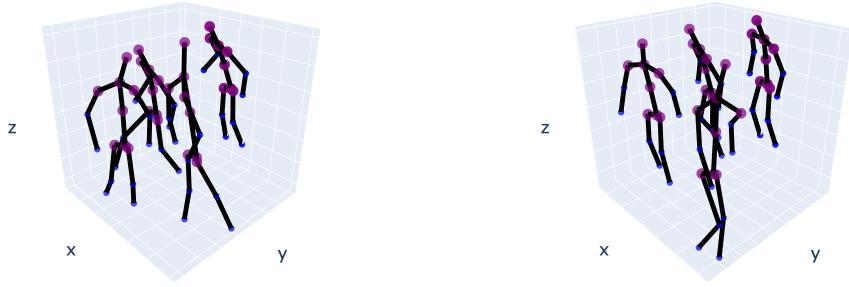
It was demonstrated in Table 4.5 and Table 4.8 that by removing *Mat-Walk* and *Hoop-Walk* from the dataset, the accuracy of the models increased.

This problem was first noticed when the confusion matrix was plotted for the *Feature engineering* approach using the *Multi-Layer Perceptron* model. In Figure 4.3a the Confusion Matrix of all 10 movements is displayed, the model is struggling to differentiate between movement 3 and 5 (*Mat-Walk* and *Hoop-Walk*). In Figure 4.3b one between *Mat-Walk* and *Hoop-Walk* is removed, the model does not struggle anymore to differentiate between the two movements.



**Figure 4.3.** Confusion Matrix of Multi-Layer Perceptron model using Feature Engineering approach.

To confirm this, *3D Visualization* of the movements used in Section 2.4 was used. A random sample of *Mat-Walk* and *Hoop-Walk* was plotted, in Figure 4.4a and Figure 4.4b the two movements are displayed. The two movements are very similar due to the fact that they are both walking movements, the only difference is that in *Mat-Walk* the patient is walking on a mat while in *Hoop-Walk* the patient is walking in a hoop. This is the reason why the models struggle to differentiate between the two movements.



**Figure 4.4.** 3D Visualization plots of the two similar movements.

# Chapter 5

## Conclusions

This chapter presents the key findings of the thesis, highlighting its limitations and review potential approaches for future research to improve the results and develop more effective models.

### 5.1 Discoveries

Presented below are the key findings of this thesis:

1. *Preprocessed raw data* obtained from the Kinect sensor is suitable for the task of movement classification. However, it alone does not provide enough information for the models to obtain a high accuracy.
2. *Feature engineering* is a crucial process of creating new features from the raw data with the goal of improving the accuracy and training time of the models.
3. *Multi-Layer Perceptron* is the best performing model for this task, with a score of **0.83** using 10 movements and **0.91** using 9 movements after removing one of the similar movements and using a feature engineering approach.
4. *Data splitting* techniques are an essential step in the process of training the models, an incorrect split can lead to overfitting and an incorrect evaluation. Using the "Patient-ID" as a split criteria is the best approach to avoid any data leakage between the training and testing sets.
5. *Sequence of frames* is not a good approach to take for this type of data, due to every movement having a variable number of frames and Machine Learning models need a fixed length input. Transforming the data into fixed length sequences will lead to a loss of information and a decrease in accuracy.
6. *3D visualization* of the movements allow to visually identify and label them. It was found that "Mat-Walk" and "Hoop-Walk" movements are very similar, with the only difference being the object that the patient is walking over. This led to the models struggling to differentiate between these two movements and by removing one of them from the dataset the accuracy of the models improved.

## 5.2 Limitations

Limitations encountered in this work will be presented, along with an exploration of their impact and the strategies used to overcome them.

A list of 10 movements names was provided with the dataset, however the movements were not labeled accordingly to the list and only a unique ID has been assigned to each one. This led to the need of updating the labels after visually identifying them with the help of the 3D visualization.

While the data was collected an unknown number of movements have not been performed correctly by the patients. It was not possible to develop a technique that would identify and remove them, so they have been kept in the dataset. This limitation may have affected the accuracy of the models due to the noise introduced.

The dataset dimensions is relatively small, with only 10 movements and 21 patients. This led to only using the Training and Testing sets for the evaluation of the models, as the dataset was too small to split it into Training/Validation/Testing sets. A larger dataset is needed to split it into these sets and evaluate the models better.

Features calculated in the feature engineering step are not accurate to literature due to only using positional data from the Kinect sensor. However, they still provide enough information for the models to obtain a high accuracy.

## 5.3 Future work

Kinect-based data is suitable for the task of movement classification, this leads to the possibility of implementing new techniques and approaches to improve the accuracy of the models.

Feature engineering approach obtains a high accuracy and reduce the training time of the models by reducing the dimension of the original dataset. It is recommended to use it if the dataset is going to be scaled up to include more movements and patients, as the training time will increase exponentially.

The number of features have been reduced but it was not possible to tell which ones contribute the most to the accuracy of the models. In future work it is recommended to calculate the importance of each feature and remove the ones that do not contribute to the accuracy of the models. With the help of domain experts it is possible to calculate new and more meaningful features that can help the models differentiate between movements.

As stated before two movements (*Mat-Walk* and *Hoop-Walk*) are very similar. It is suggested to remove one of them from the dataset to obtain a realistic evaluation of the models, this will help to differentiate between movements that are very similar.

This thesis only used Machine Learning models from the *Scikit-Learn* library. It is possible to implement new models from the *TensorFlow* library, such as *Convolutional Neural Networks* and *Long Short-Term Memory* networks. These models

are more complex and require a larger dataset to train on, but they can obtain a higher accuracy than the models used in this thesis with a correct implementation. It is crucial to acquire new data from the Kinect sensor, add new movements and patients. This will allow to scale up the dataset and study how the models perform on more classes and patients.

The possible approaches for future work on this task are endless, above are only a few suggestions that can be implemented. The goal of this thesis was to study the feasibility of using Kinect-based data for movement classification, and provide a baseline for future research with this type of data.

# Bibliography

- [1] GitHub - Kinect/PyKinect2: Wrapper to expose Kinect for Windows v2 API in Python — [github.com/Kinect/PyKinect2](https://github.com/Kinect/PyKinect2).
- [2] ABBASI, J., SALARIEH, H., AND ALASTY, A. A motion capture algorithm based on inertia-Kinect sensors for lower body elements and step length estimation. *Biomedical Signal Processing and Control*, **64** (2021), 102290. Available from: <https://www.sciencedirect.com/science/article/pii/S1746809420304110>, doi:10.1016/j.bspc.2020.102290.
- [3] AÇIŞ, B. AND GÜNEY, S. Classification of human movements by using Kinect sensor. *Biomedical Signal Processing and Control*, **81** (2023), 104417. Available from: <https://www.sciencedirect.com/science/article/pii/S1746809422008710>, doi:10.1016/j.bspc.2022.104417.
- [4] BALAKRISHNAMA, S. AND GANAPATHIRAJU, A. LINEAR DISCRIMINANT ANALYSIS - A BRIEF TUTORIAL.
- [5] BENTÉJAC, C., CSÖRGŐ, A., AND MARTÍNEZ-MUÑOZ, G. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, **54** (2021), 1937. Available from: <https://doi.org/10.1007/s10462-020-09896-5>, doi:10.1007/s10462-020-09896-5.
- [6] BUITINCK, L., ET AL. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122 (2013).
- [7] CHA, G.-W., MOON, H.-J., AND KIM, Y.-C. Comparison of Random Forest and Gradient Boosting Machine Models for Predicting Demolition Waste Based on Small Datasets and Categorical Variables. *International Journal of Environmental Research and Public Health*, **18** (2021), 8530. Available from: <https://www.mdpi.com/1660-4601/18/16/8530>, doi:10.3390/ijerph18168530.
- [8] CRAMER, J. S. The Origins of Logistic Regression (2002). Available from: <https://papers.ssrn.com/abstract=360300>, doi:10.2139/ssrn.360300.
- [9] CRUZ, L., LUCIO, D., AND VELHO, L. Kinect and RGBD Images: Challenges and Applications. In *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials*, pp. 36–49. IEEE, Ouro Preto, Brazil (2012). ISBN 978-0-7695-4830-2 978-1-4673-5091-4. Available from: <http://ieeexplore.ieee.org/document/6382717/>, doi:10.1109/SIBGRAPI-T.2012.13.

- [10] GOWING, M., AHMADI, A., DESTELLE, F., MONAGHAN, D. S., O'CONNOR, N. E., AND MORAN, K. Kinect vs. Low-cost Inertial Sensing for Gesture Recognition. In *MultiMedia Modeling* (edited by C. Gurrin, F. Hopfgartner, W. Hurst, H. Johansen, H. Lee, and N. O'Connor), Lecture Notes in Computer Science, pp. 484–495. Springer International Publishing, Cham (2014). ISBN 978-3-319-04114-8. doi:10.1007/978-3-319-04114-8\_41.
- [11] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, NY (2009). ISBN 978-0-387-84857-0 978-0-387-84858-7. Available from: <http://link.springer.com/10.1007/978-0-387-84858-7>. doi:10.1007/978-0-387-84858-7.
- [12] HO, T. K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20** (1998), 832. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. Available from: <https://ieeexplore.ieee.org/document/709601>, doi:10.1109/34.709601.
- [13] INC., P. T. Collaborative data science (2015). Available from: <https://plot.ly>.
- [14] JAIS, H. M., MAHAYUDDIN, Z. R., AND ARSHAD, H. A review on gesture recognition using kinect. In *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 594–599. IEEE, Denpasar, Bali, Indonesia (2015). ISBN 978-1-4673-6778-3 978-1-4673-7319-7. Available from: <http://ieeexplore.ieee.org/document/7352569/>, doi:10.1109/ICEEI.2015.7352569.
- [15] JANA, A. *Kinect for Windows SDK Programming Guide*. Packt Publishing, 1 edn. (2012). ISBN 978-1-84969-238-0. Available from: <https://www.perlego.com/book/389793/kinect-for-windows-sdk-programming-guide-pdf>.
- [16] JUNG, D. Fear of Falling in Older Adults: Comprehensive Review. *Asian Nursing Research*, **2** (2008), 214. Available from: <https://linkinghub.elsevier.com/retrieve/pii/S1976131709600037>, doi:10.1016/S1976-1317(09)60003-7.
- [17] MACKAY, S., EBERT, P., HARBIGE, C., AND HOGAN, D. B. Fear of Falling in Older Adults: A Scoping Review of Recent Literature. *Canadian geriatrics journal: CGJ*, **24** (2021), 379. doi:10.5770/cgj.24.521.
- [18] MANGAL, N. K. AND TIWARI, A. K. Kinect v2 tracked Body Joint Smoothing for Kinematic Analysis in Musculoskeletal Disorders. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference*, **2020** (2020), 5769. doi:10.1109/EMBC44109.2020.9175492.
- [19] MAUDSLEY-BARTON, S., MCPHEE, J., BUKOWSKI, A., LEIGHTLEY, D., AND YAP, M. H. A comparative study of the clinical use of motion analysis from Kinect skeleton data. In *2017 IEEE International Conference on*

- Systems, Man, and Cybernetics (SMC)*, pp. 2808–2813 (2017). Available from: <https://ieeexplore.ieee.org/abstract/document/8123052>, doi: 10.1109/SMC.2017.8123052.
- [20] NEHRA, S. AND RAHEJA, J. Unobtrusive and Non-Invasive Human Activity Recognition using Kinect Sensor. pp. 58–63 (2020). doi:10.1109/Indo-TaiwanICAN48429.2020.9181359.
- [21] PARMAR, A., KATARIYA, R., AND PATEL, V. A Review on Random Forest: An Ensemble Classifier. In *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018* (edited by J. Hemanth, X. Fernando, P. Lafata, and Z. Baig), Lecture Notes on Data Engineering and Communications Technologies, pp. 758–763. Springer International Publishing, Cham (2019). ISBN 978-3-030-03146-6. doi:10.1007/978-3-030-03146-6\_86.
- [22] PEDREGOSA, F., ET AL. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12** (2011), 2825.
- [23] PISHARADY, P. K. AND SAERBECK, M. Kinect based body posture detection and recognition system: 4th International Conference on Graphic and Image Processing, ICGIP 2012. *International Conference on Graphic and Image Processing, ICGIP 2012*, (2013). Available from: <http://www.scopus.com/inward/record.url?scp=84880152714&partnerID=8YFLogxK>, doi:10.1117/12.2009926.
- [24] REN, W., MA, O., JI, H., AND LIU, X. Human Posture Recognition Using a Hybrid of Fuzzy Logic and Machine Learning Approaches. *IEEE Access*, **PP** (2020), 1. doi:10.1109/ACCESS.2020.3011697.
- [25] SAIDIN, N. AND ABDUL SHUKOR, S. An Analysis of Kinect-Based Human Fall Detection System. pp. 220–224 (2020). doi:10.1109/ICSPC50992.2020.9305797.
- [26] SVOZIL, D., KVASNICKA, V., AND POSPICHAL, J. Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, **39** (1997), 43. Available from: <https://www.sciencedirect.com/science/article/pii/S0169743997000610>, doi:10.1016/S0169-7439(97)00061-0.
- [27] TAN, T.-H., GOCHOO, M., CHEN, H.-S., LIU, S.-H., AND HUANG, Y.-F. Activity Recognition Based on DCNN and Kinect RGB Images. pp. 1–4 (2020). doi:10.1109/iFUZZY50310.2020.9297815.
- [28] WANG, T., LI, C., WU, C., ZHAO, C., SUN, J., PENG, H., HU, X., AND HU, B. A Gait Assessment Framework for Depression Detection Using Kinect Sensors. *IEEE Sensors Journal*, **21** (2021), 3260. Conference Name: IEEE Sensors Journal. Available from: <https://ieeexplore.ieee.org/document/9187648>, doi:10.1109/JSEN.2020.3022374.

- [29] WRIGHT, M. AND FREED, A. Open SoundControl: A New Protocol for Communicating with Sound Synthesizers.
- [30] XANTHOPOULOS, P., PARDALOS, P. M., AND TRAFALIS, T. B. Linear Discriminant Analysis. In *Robust Data Mining* (edited by P. Xanthopoulos, P. M. Pardalos, and T. B. Trafalis), SpringerBriefs in Optimization, pp. 27–33. Springer, New York, NY (2013). ISBN 978-1-4419-9878-1. Available from: [https://doi.org/10.1007/978-1-4419-9878-1\\_4](https://doi.org/10.1007/978-1-4419-9878-1_4), doi: 10.1007/978-1-4419-9878-1\_4.
- [31] XU, X. AND MCGORRY, R. W. The validity of the first and second generation Microsoft Kinect™ for identifying joint center locations during static postures. *Applied Ergonomics*, **49** (2015), 47. Available from: <https://www.sciencedirect.com/science/article/pii/S0003687015000149>, doi: 10.1016/j.apergo.2015.01.005.
- [32] ZHENG, Z., WANG, Q., DENG, D., WANG, Q., AND HUANG, W. CG-Recognizer: A biosignal-based continuous gesture recognition system. *Biomedical Signal Processing and Control*, **78** (2022), 103995. Available from: <https://www.sciencedirect.com/science/article/pii/S1746809422004438>, doi:10.1016/j.bspc.2022.103995.
- [33] İNCE, . F., İNCE, I. F., YILDIRIM, M. E., PARK, J. S., SONG, J. K., AND YOON, B. W. Human activity recognition with analysis of angles between skeletal joints using a RGB-depth sensor. *ETRI Journal*, **42** (2020), 78. Publisher: John Wiley & Sons Inc. Available from: <https://research.bau.edu.tr/en/publications/human-activity-recognition-with-analysis-of-angles-between-skelet-2>, doi:10.4218/etrij.2018-0577.