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

This thesis has been typeset by LaTeX 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 skeleton 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 an 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 Forests, 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 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 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 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. The findings can be used to develop more accurate and efficient models for classifying human movements.

Contents

Chapter 1

Introduction

This thesis is structured in the following way: **Chapter** 1 presents the problem statement, literature review, dataset overview, aims, and objectives of the study. **Chapter** 2 presents the data collection methodology, data structure and attributes, patient's characteristics, movement visualization, and data processing. **Chapter** 3 presents the methodology used in this study, including the models, data splitting methods, and Feature Engineering approach. **Chapter** 4 presents the evaluation metrics used, results obtained, and discussion of the results. Finally, **Chapter** 5 presents 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 arrival 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, 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 and Nor Asilah Saidin. conducted a study utilizing a Kinect sensor to detect human falls. Their system demonstrated accurate results [25]. Tao Wang et al. conducted a study that involved gait analysis using a Kinect sensor for automatic and real-time detection of depression. The model developed achieved a classification accuracy of 93.75% [29]. Naveen Kumar Mangal and Anil Kumar Tiwari 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

kinematic analysis of musculoskeletal disorders. Findings from the study revealed that the range of motion values derived from the proposed filter significantly improved the monitoring accuracy of skeletal joints using a Kinect sensor [18]. Shalini Nehra and Jagdish Lal Raheja 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 datasets, as reported in [20]. Tan-Hsu Tan 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 precision at 95.5%, recall at 95.6%, specificity at 99.8%, accuracy at 99.6%, and an F1 score of 95.3% [27]. Weiyan Ren et al. used a Kinect sensor to gather posture data from twenty individuals while lying in bed. Data was then subjected to a Machine Learning approach using 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 the K-Nearest Neighbor algorithm as part of a Machine Learning approach for classification, achieving an accuracy of 86.1% [34]. Pramod Kumar Pisharady and Martin 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]. Tao Wang et al. introduced a method for accurately distinguishing between various postures of five different individuals. A classification success rate exceeding 99% was achieved [29].

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 is used. Composed of recorded movements performed by a group of 22 individuals in front of a Kinect sensor, data is saved as a series of 3D coordinates. It contains 10 different movements, each performed a various number of times by each individual. The movements are listed in Table 1.1.

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, and 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 subject 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.

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

1.4 Objectives

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

1. Visualize and label Kinect skeleton data using 3D plot animations.
2. Pre-process data to remove noise and outliers for better classification results.
3. Analyze different approaches for handling 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 understanding approaches used 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 a recording setup. Data structure and attributes are presented, along with patient's characteristics. Finally, data processing steps are described.

2.1 Data collection methodology

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

2.1.1 Microsoft kinect

The first generation of 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 [32]. A new version of a Kinect sensor, Kinect V2, was released in 2014, with improved hardware and software [9].

Figure 2.1. Microsoft Kinect Sensor.

Kinect sensor

Kinect Sensor is a horizontal bar connected to a small base with a motorized pivot and is designed to be positioned lengthwise above or below a 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 a 3D imaging field [33][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 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 a need for any physical devices [10].

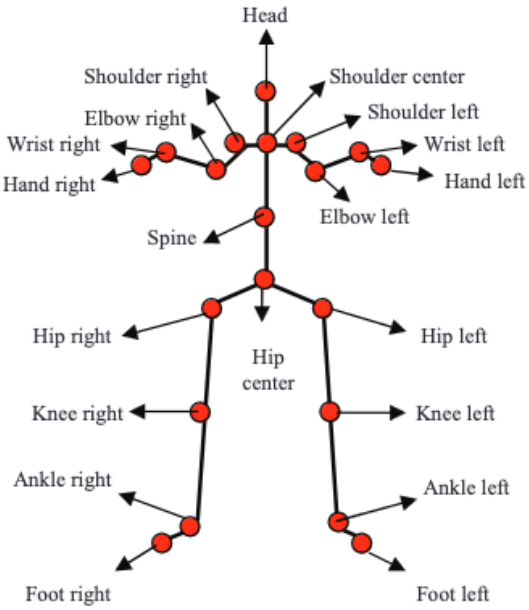


Figure 2.2. Skeletal joints recognized by a Microsoft Kinect sensor [14].

PyKinect2

PyKinect2 is a Python library for Microsoft's Kinect V2 sensor. It provides a wrapper for Kinect Windows SDK 2.0, which allows for the use of the sensor in Python. Library abstracts complex functionality of the 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.

PyKinect2 library is a great bridge between Kinect sensor and 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, an operator can display a sample stimulus on an external monitor to show target movements to patients (*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/analyze skeleton + RGB + accelerometer*), recording of the patient's full body movement can be started. The application stores recorded patient's movement files in a separate folder, naming them based on their patient ID, movement ID, and repetition ID.

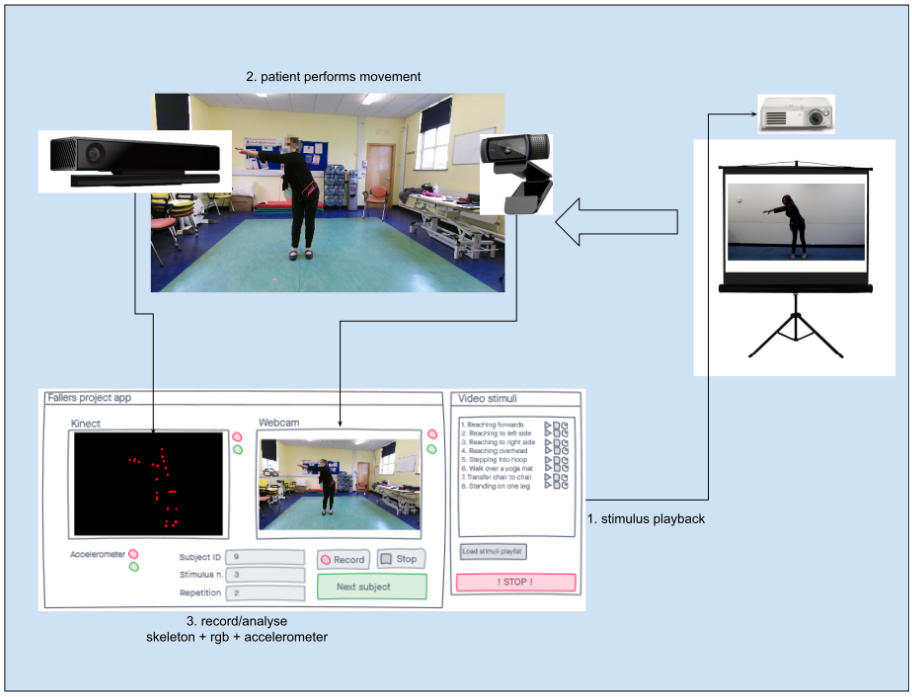


Figure 2.3. Setup used at Waterford Hospital for data collection.

Full body capture mainly relies on the PyKinect library [1], which 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 displayed in Figure 2.4.

	A	B	C	D	E	F	G	H
1	timestamp	datetime	Head.state	Head.px	Head.py	Head.pz	Head.rx	Head.ry
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
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
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
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
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
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
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
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
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
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
12	1.666.953.979.8	2022-10-28_11:4	2.00	3569.21.00	9398.26.00	11725.32.00	0.00	0.00

Figure 2.4. Example of a CSV file containing 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). Communication between smartphone and computer is based on a wireless network and Open Sound Control (OSC) protocol [30].

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 patient performing a movement. Total number of CSV files is **637**. In Figure 2.5 distribution of the CSV files between the movements is presented with a median of **68** files per movement.

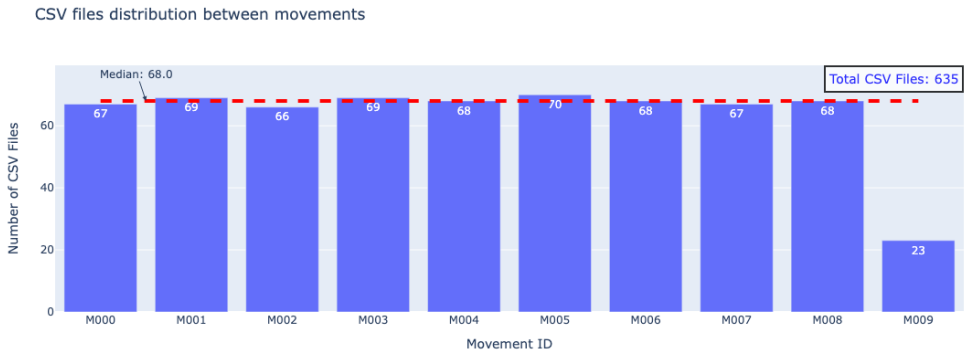


Figure 2.5. CSV files distribution between movements. *M009* is the only movement with less than 68 files due to it not being performed multiple times by the patients.

The 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 a repetition ID (*R-XXX*). Inside each repetition folder, there is a CSV file that contains Kinect skeleton data. In Figure 2.6 an example of a directory structure is displayed.

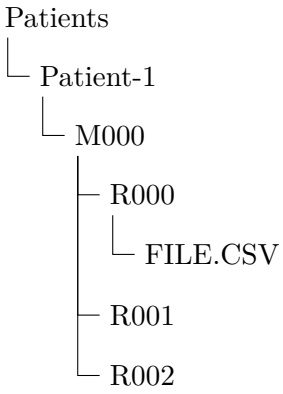


Figure 2.6. Directory structure example using first patient and first movement in the dataset.

a CSV file is organized as a series of columns, each column representing a joint in Table 2.1 that a Kinect sensor records. Each joint is represented by 7 columns, one for each position and rotation coordinate (x, y, z) and one for the state (used to indicate if the joint is tracked or not). Besides joints columns, there are 2 columns for timestamp and datetime of the recording.

Joints			
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			

Table 2.1. Joints processed with PyKinect2 library.

2.3 Patients characteristics

Patients that take part in the study are 22 elderly individuals, in Figure 2.7 age distribution is presented, with a median age of 67 years.

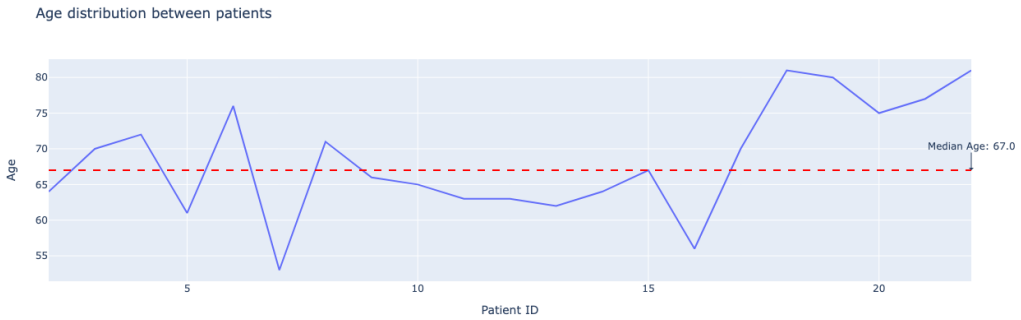


Figure 2.7. Age distribution of patients.

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

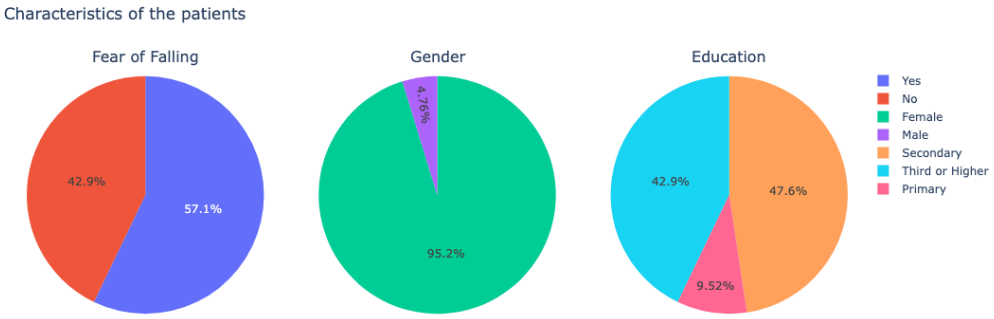


Figure 2.8. Characteristics of patients in the study.

What is fear of falling ?

The definition of **Fear of Falling** had various interpretations over the years. Initially, it is described as a phobic reaction to standing or walking. However, it is reclassified as a syndrome characterized by the aftermath of a fall. As understanding develops, this fear is seen as a loss of confidence in one's balance ability. It is also further defined as an ongoing concern about falling, which leads to avoidance of performing daily activities. Recently, it has been described as continuous avoidance of activities due to a concern of falling [16].

2.4 Movements visualization

Kinect skeleton data comes as a series of 3D coordinates, which can be visualized in 3D space. In this section, the implementation of movement visualization is presented. It is implemented using Python programming language and Plotly library [13].

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

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		

Table 2.2. Selected Kinect joints used for visualization.

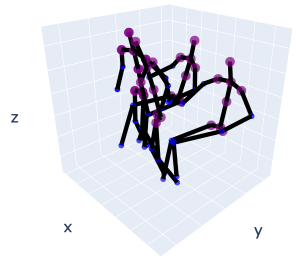
Once joints are selected, the next step is to transform the data. In its original state data is organized incorrectly for 3D visualization, and y and z coordinates are inverted. To fix this, the y and z coordinates are swapped.

After the transformation is performed, data is ready to be visualized. Snippet 2.1 shows an implementation of the visualization process, it begins by extracting joint coordinates and their connections, assigning colors and sizes to major joints, and configuring a 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 a user to play/stop the animation and rotate the plot to view a movement from different angles.

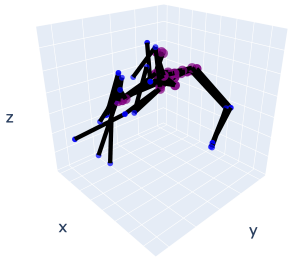
```
for index, row in data.iterrows():
    x_values = [row[f"{joint}.px"] for joint in joints]
    y_values = [row[f"{joint}.py"] for joint in joints]
    z_values = [row[f"{joint}.pz"] for joint in joints]
    lines = []
    for connection in connections:
        start, end = connection
        lines.append(go.Scatter3d(
                x=[row[f"{start}.px"], row[f"{end}.px"]],
                y=[row[f"{start}.py"], row[f"{end}.py"]],
                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.

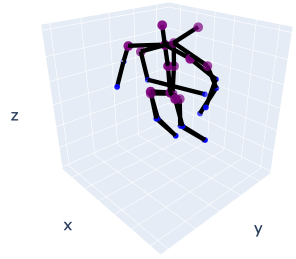
In Figure 2.9 a set of movements performed by the patients is presented. The movements are displayed in a 3D plot, with the x, y, and z axes representing plot axes.



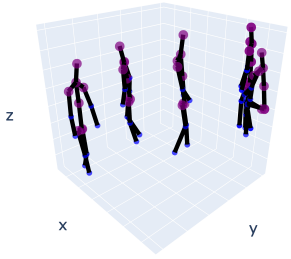
(a) Chair to Chair



(b) Chair to Chair corrupted



(c) Cross-Reach Left



(d) Tug Walk

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

Movement visualization allows for the identification of corrupted data, Figure 2.9b displays a movement that is corrupted, it is not possible to identify the movement due to the data being incorrect. This led to further investigation and the identification of corrupted data in the dataset, all of the data of *Patient-1* is corrupted and is removed from the dataset to not affect the classification task. This corruption is due to the patient not being in the field of view of the Kinect sensor, and the data is not recorded correctly.

2.5 Data processing

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

2.5.1 Cleaning

From the original dataset, a process of cleaning data is performed. Consisting of removing columns that contain zero values and the ones that are not needed for this classification task. Columns that are kept are listed in Table 2.2, only positional coordinates are kept, and state columns and rotation columns (x, y, z) are removed due to not giving any useful information. As mentioned in Section 2.4, all corrupted data that is identified with the visualization process is removed from the dataset.

2.5.2 Normalization

Pose normalization is performed using the formula described in [19] as follows:

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

Equation 2.1 is used to normalize the pose of a patient performing a movement. By subtracting the coordinates of the spine base joint in the first frame from the coordinates of all joints in the data. This is done to remove the effect of the position of a patient in the recording setup and align all frames to the same position.

2.5.3 Transformation

Once data cleaning and normalization are performed, it is transformed into a format that can be used for a classification task. Using the Scikit-Learn library [6], **MinMaxScaler** is used to scale data between 0 and 1 then **StandardScaler** is used to standardize it. After this process data is ready for a Machine Learning model.

Chapter 3

Methodology

In this chapter, the methodology used to split data and train 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 skeleton 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

Models presented in Table 3.1 are used for a classification task. Selected based on popularity and performance, these models are widely used in the Machine Learning community. Models are implemented using *Scikit-Learn* library and its functions for training, testing, and evaluating [6].

	Model Name		
1	Support-Vector Machines	6	Linear-Discriminant Analysis
2	Gaussian Naive Bayes	7	Multi-Layer Perceptron
3	Random Forests	8	K-Nearest Neighbors
4	Gradient Boosting	9	Ada Boost
5	Logistic Regression	10	Decision Trees

Table 3.1. Models selected for use in this thesis.

3.2 Models analysis

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

3.2.1 Random forests

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

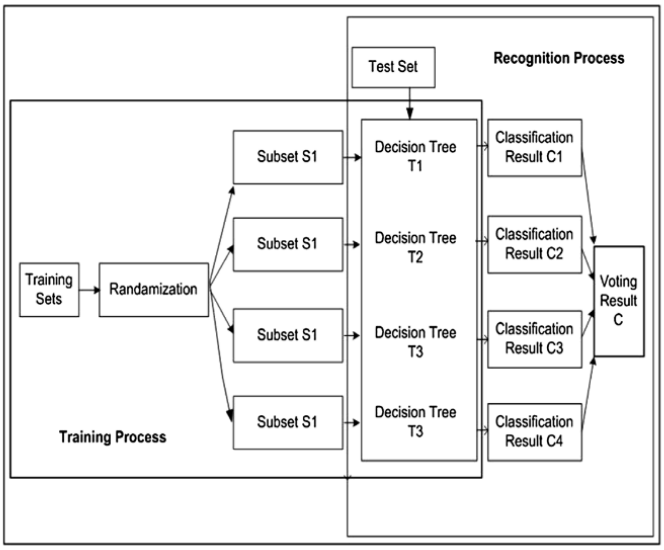


Figure 3.1. The process starts with multiple training sets that undergo randomization to create several subsets S1. Each subset is used to train a separate decision tree (T1 to T3). Trained trees are then used to make predictions on a test set. Predictions (C1 to C4) from each tree are aggregated through 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 Forests classifier are as follows:

1. Define M as the number of features in each subset.
2. Randomly select a feature subset θ_k from the full set, distinct from proceding subset $\theta_1, ..., \theta_{k-1}$.
3. Train decision trees on each θ_k denoted as $h(X, \theta_k)$.
4. Iteratively select new θ_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 θ_k are chosen by bagging, and the importance of features is ranked by their impact on the model's accuracy "strength" and "correlation" of the forest are influenced by M, with optimal values providing a balance. Random Forests 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 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 a loss function, effectively reducing residual between the predicted and true values. Ensemble begins with a single model $F_0(x)$, which is updated by the formula:

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

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

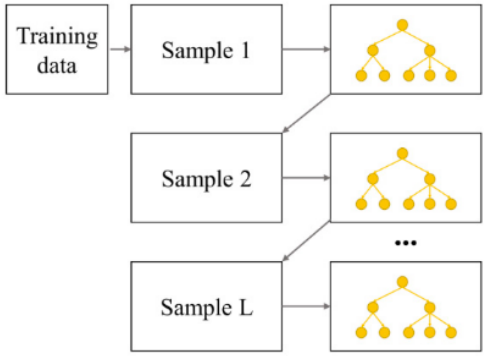


Figure 3.2. Starting with training data, an algorithm iteratively trains decision trees (Sample 1 to Sample L). Each tree is trained on errors of the previous ones, aiming to correct these mistakes. Over multiple iterations, each tree improves the model's predictions, and the final output is a combined effort of all 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 predictor variables. It applies a logistic function to a linear combination of input features to produce a value between 0 and 1, interpreted as a probability of the instance being in a positive class. Equation 3.2 shows the logistic function for binary classification.

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

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 is designed to identify 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.

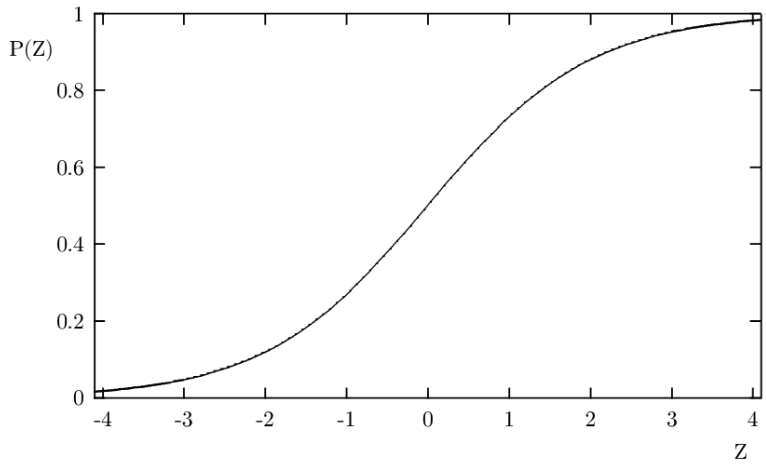


Figure 3.3. The horizontal axis labeled Z represents an input variable (which is a linear combination of the features), and the vertical axis labeled *P(Z)* represents a probability that an outcome is a positive class. The curve transitions smoothly from 0 to 1, with an inflection point at Z=0, where 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 a ratio of between-class variance to a within-class variance in any particular data set, thereby ensuring maximum separability.

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

class scatter matrix S_W, and between-class scatter matrix S_B. Linear discriminants are then the eigenvectors of $S_W^{-1} S_B$ [31].

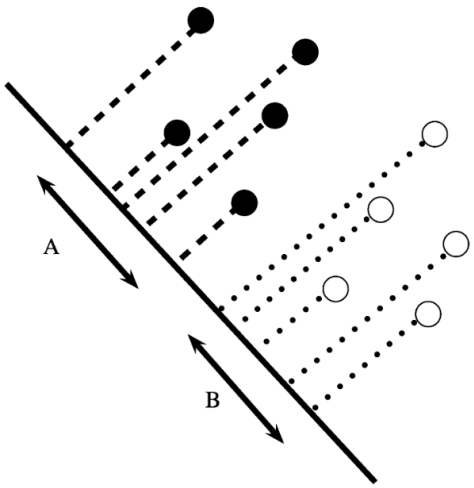


Figure 3.4. The intuition behind LDA. Data samples in two dimensions are projected in a lower-dimension space. The line has to be chosen so that the projection maximizes the "separability" of projected samples [31].

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

The simplicity and effectiveness of LDA, especially under 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 includes at least three layers, an input layer, an output layer, and one or more hidden layers, each composed of nodes with non-linear activation functions [26]. They are referred to as "vanilla" Neural Networks [11].

$$y(v_i) = \tanh(v_i) \tag{3.3}$$

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

A linear function can simplify multiple layers to a two-layer model, mapping weighted inputs to neuron outputs. Non-linear activation functions, like hyperbolic tangent 3.3 ranging from -1 to 1, or sigmoid function 3.4 ranging from 0 to 1, are used to introduce non-linearity into a model. This allows a 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 dynamic adjustment of connection weights following the processing of each data point. This adjustment is made in response to the disparity between actual output and expected outcome, to minimize error.

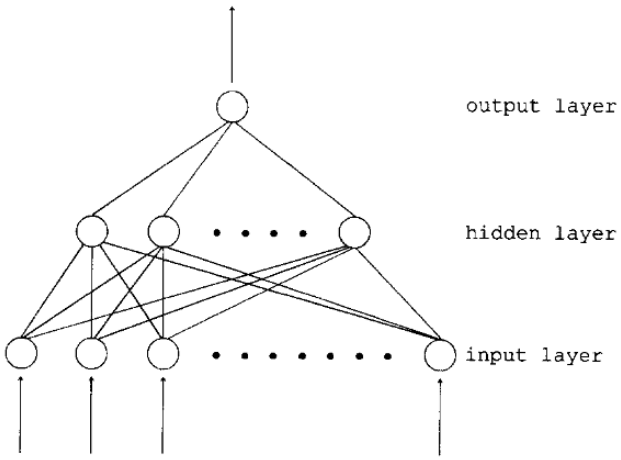


Figure 3.5. Feed-forward network consists of an input layer, one or more hidden layers, and an output layer [26].

3.3 Data splitting methods

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

3.3.1 Traditional

Data is split into 70% training and 30% testing following a traditional approach used in Machine Learning literature. The code snippet in 3.1 demonstrates this approach.

```
def split_data(data: pd.DataFrame) -> tuple:
    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.33, random_state=42)

    return X_train, X_test, y_train, y_test
```

Listing 3.1. Traditional approach to splitting 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. Data is split randomly, so there is a chance that the same patient will appear in both training and testing sets. This means that a 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 a model's performance on unseen data.

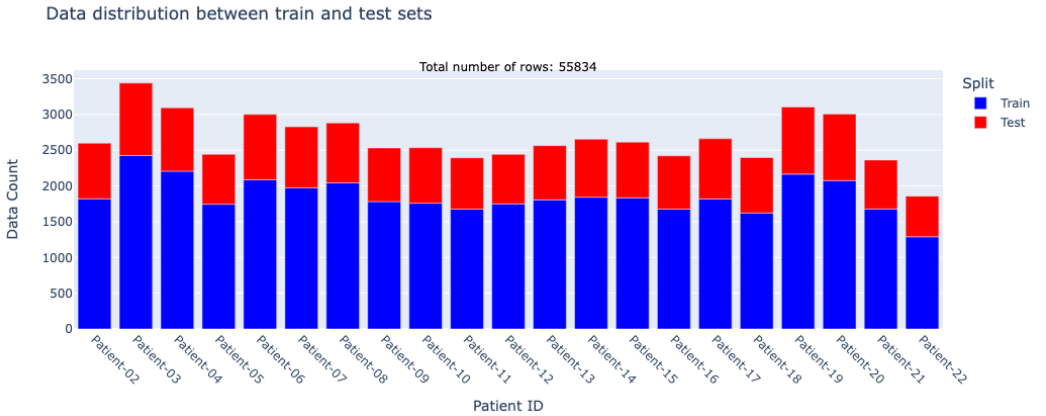


Figure 3.6. Patient presence in both training and testing sets visualization.

3.3.2 Effective

Data is split into training and testing sets based on a patient's unique IDs. They are split into training and testing sets, and then data is split based on patient's unique ids. The code snippet in 3.2 demonstrates this approach.

```
def split_data(data: pd.DataFrame) -> tuple:
    unique_patient = data['patient'].unique()

    train_patients, test_patients = train_test_split(unique_patients,
        test_size=0.3, random_state=42)

    train_data = data[data['patient'].isin(train_patients)]
    test_data = data[data['patient'].isin(test_patients)]

    X_train = train_data.drop(columns=['label', 'patient'])
    y_train = train_data['label']

    X_test = test_data.drop(columns=['label', 'patient'])
    y_test = test_data['label']

    return X_train, X_test, y_train, y_test
```

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

Figure 3.7 visualization demonstrates why this approach is effective. Data is split based on patient's unique IDs, so a model will be trained on data that will not be tested. This means that a model will be tested on unseen data, which is a good indicator of a model's performance.

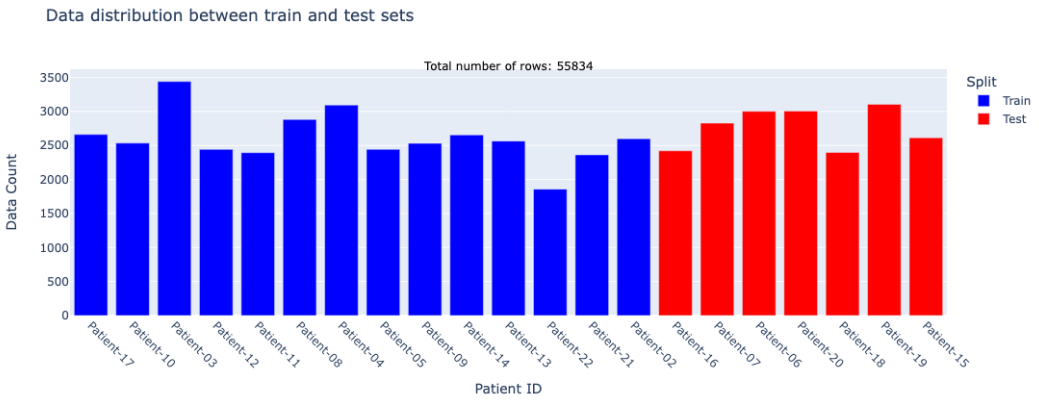


Figure 3.7. Patients split between training and testing sets visualization, ensuring that a patient is only present in one of the sets.

3.3.3 Sequential

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

```
def sequences(df: pd.DataFrame, feature_columns: list,
    sequence_column: str) -> tuple:
    sequences = []
    labels = []
    current_sequence = []
    current_check = None

    for _, row in df.iterrows():
        check = row[sequence_column]
        label = row['label']
        if check != current_check and current_sequence:
            sequences.append(np.array(current_sequence))
            labels.append(label)
            current_sequence = []
        current_sequence.append(row[feature_columns].to_numpy())
        current_check = check

    if current_sequence:
        sequences.append(np.array(current_sequence))
        labels.append(label)

    return sequences, labels
```

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

Figure 3.8 visualization shows how for each movement 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 sequences cannot be used as input for models since they require a fixed input size.

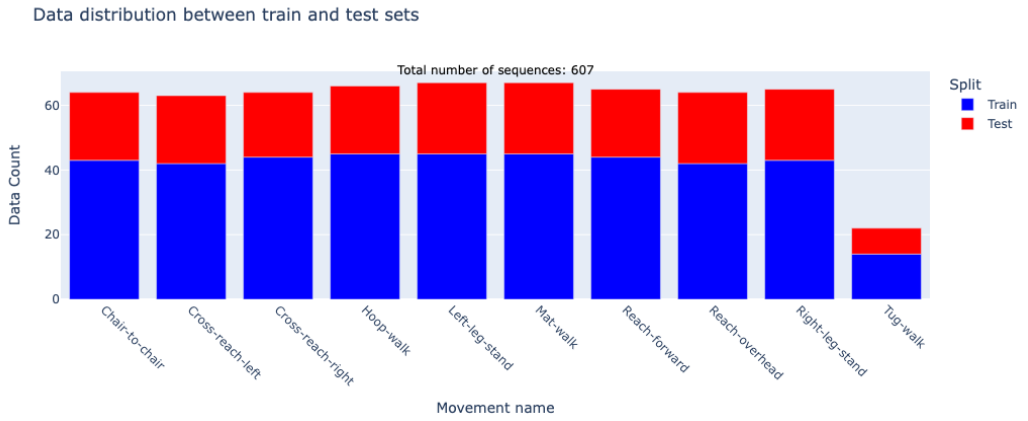


Figure 3.8. Visualization of the sequences splitting approach.

To solve this variable length problem, sequences are aggregated into a single feature vector. Code snippet 3.4 demonstrates this approach. In Figure 3.9, the length of sequences before and after aggregation is visualized. 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.

```
def aggregate_features(sequences: list) -> np.ndarray:
        return np.array([np.mean(sequence, axis=0) if sequence.size
            != 0 else np.zeros(sequence.shape[1]) for sequence in
            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 data is no longer represented as a sequence of frames. In addition, aggregation results in a loss of temporal information, as the order of frames is lost. This means that models will not be able to learn temporal patterns in data.

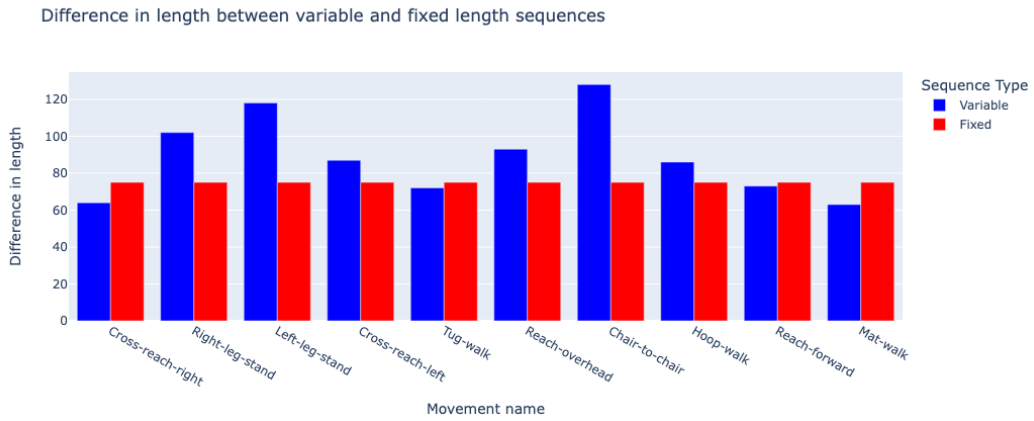


Figure 3.9. Visualization of the length of the sequences before and after aggregation.

3.4 Feature engineering

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

3.4.1 Overview

This process is implemented to improve the performance of the models due to them not being able to differentiate well between movements based on raw data. This allows us to obtain data that is more informative and easier to interpret. Features extracted from 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 Kinect skeleton data.

3.4.2 Calculation Methods

Features presented in Table 3.2 are calculated using the following methods. For each feature, 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 Kinect skeleton data joints.

Duration

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

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

Area

The area is defined as an aggregate area of convex hulls formed by trajectories of selected body parts. It operates by extracting (x, y, z) coordinates for each specified

body part, constructing a convex hull for these points, and then calculating the hull's volume.

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

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

```
def area(df: pd.DataFrame, body_parts: list) -> float:
    def calculate(points: np.ndarray) -> float:
        hull = ConvexHull(points)
        return hull.volume
    trajectories = {}
    for column in body_parts:
        body_part = column.split('.')[0]
        trajectory = df[[body_part + '.px', body_part + '.py',
            body_part + '.pz']].values
        trajectories[body_part] = trajectory
    temp = {}
    for body_part, trajectory in trajectories.items():
        temp[body_part] = calculate(trajectory)
    return sum(temp.values())
```

Listing 3.5. Area calculation method using ConvexHull class from SciPy library [28].

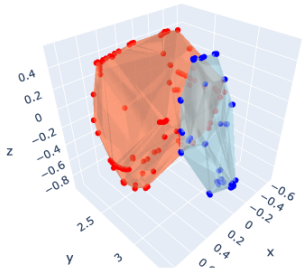


Figure 3.10. Visualization of the area occupied by two movements, red area represents *Chair to Chair* while blue area represents *Right Leg Stand*.

Velocity

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

axis.

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

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

Listing 3.6. Velocity calculation method.

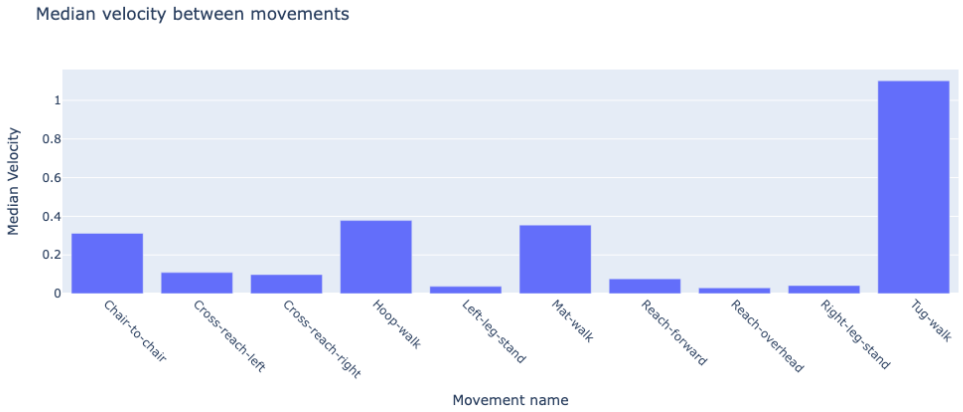


Figure 3.11. Visualization of the median velocity of each movement in the dataset. This shows how velocity varies between movements and can be used to differentiate between them.

Distance

Distance is defined as the total 3D Euclidean distance between consecutive points representing a 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 an overall total distance covered by a body part in the sequence.

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

```
def calculate(row1: pd.Series, row2: pd.Series, point="Head") ->
    float:
    return np.sqrt((row2[f'{point}.px'] - row1[f'{point}.px'])**2 +
                   (row2[f'{point}.py'] - row1[f'{point}.py'])**2 +
                   (row2[f'{point}.pz'] - row1[f'{point}.pz'])**2)

def distance(df: pd.DataFrame) -> float:
    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

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

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

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

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

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

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

Listing 3.8. Vertical time steps displacement calculation method.

Chapter 4

Results and Discussion

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

4.1 Models evaluation

Performed using *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 validation methods and metrics used in this thesis.

4.1.1 Validation

Validation is a 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 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**: 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. Fig 4.1 display KFold split.

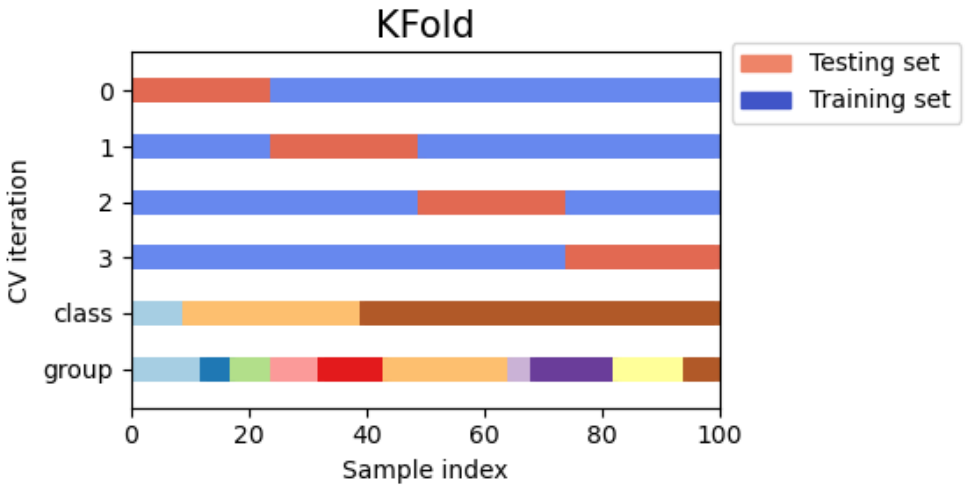


Figure 4.1. KFold Visualization from Scikit-Learn documentation [22].

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

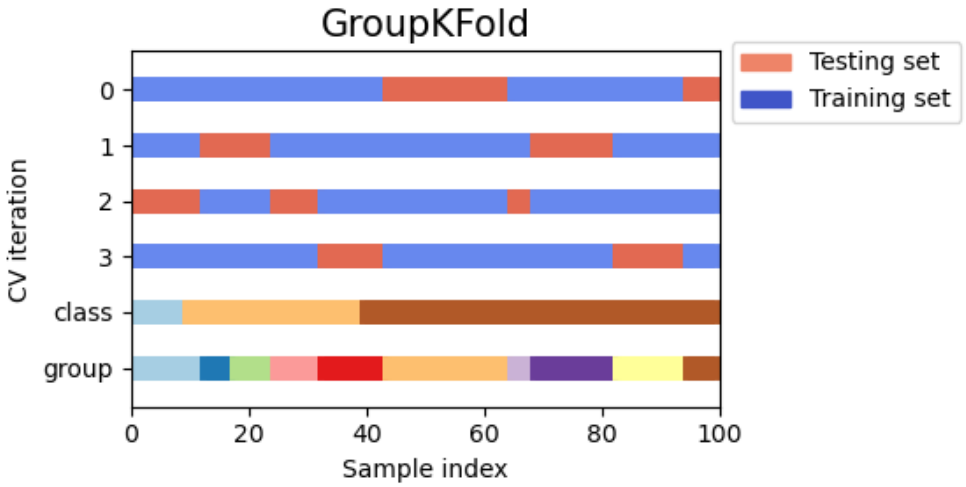


Figure 4.2. GroupKFold Visualization from Scikit-Learn documentation [22].

An advantage of using *Cross-Validation* over *Hold-Out* is that all 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 a model, by averaging results over several trials. A disadvantage of using Cross-Validation is that it is computationally expensive for very large datasets.

In this thesis, 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 *Wrong approach* and *Sequence approach* datasets due to their large dimensions. Cross-Validation method is used with the Hold-Out method to evaluate the performance of the models for *Correct approach* and *Feature Engineering approach* dataset due to them scoring the best results and being effective approaches. Confronting the results of two methods will show the correctness of an evaluation.

4.1.2 Metrics

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

Accuracy score

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

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{4.1}$$

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

Precision score

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

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{4.2}$$

Recall score

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

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{4.3}$$

F1 score

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

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

Matthews correlation coefficent

Matthews correlation coefficent (or φ coefficient) takes into account true and false positives and negatives and is regarded as a balanced measure that can be used even if the classes are of very different sizes. Formula used to calculate φ coefficient is as follows:

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

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

4.2 Results

Combining validation methods and metrics presented above, the following tables will show results obtained from experiments conducted. 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. 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

Presented in Section 3.3.1, it is 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 results, after further investigation, it has been discovered that the dataset is not properly split into training and testing sets.

Model	Accuracy	F1	Recall	Precision	MCC
Random Forests	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
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

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

In Table 4.1, results obtained from the Hold-Out method are presented. High values are obtained for all the metrics, with **Random Forests** obtaining the highest values with a score of **0.99** for accuracy. This confirmed doubts about the validity of results, a patient is both present in the training and testing set. This led to 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 concatenating frames of a movement into a sequence would help models differentiate between movements and obtain higher accuracy.

Model	Accuracy	F1	Recall	Precision	MCC
K-Nearest Neighbors	**0.56**	**0.54**	**0.54**	**0.54**	**0.51**
Random Forests	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

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

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

4.2.2 Effective

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

Correct approach

Presented in Section 3.3.2, considered effective because raw Kinect data can obtain a high accuracy. Data is not modified in any way, besides the removal of rotational and state data, and pre-processing to remove noise.

In Table 4.3 results obtained from the Hold-Out method are presented. **Random Forests** 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 data obtained from the Kinect sensor is suitable for the task of movement classification without any major tweaks.

Model	Accuracy	F1	Recall	Precision	MCC
Random Forests	**0.74**	**0.73**	**0.73**	**0.73**	**0.71**
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

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

In Table 4.3 **Hold-Out** validation method is used for all models, while in Table 4.4 **Cross-Validation** is used with only 3 models to compare 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 is used for its speed, with **10 minutes** of training time while Cross-Validation runs for hours without finishing. This is because this approach uses raw Kinect data, that contains over **59000** rows and **100** columns.

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

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

In Table 4.5 are displayed results of a final approach, where two movements *Mat Walk* and *Hoop Walk* are removed from the dataset one at a time. Results show that the accuracy of models increased by **4%** to **5%**. This is because the two movements are very similar, and this caused models to struggle to differentiate between them no matter the features used.
By removing either one of the movements, the model's accuracy increased by the same amount. This leaves a decision to the user to choose which movement to remove based on the context of the application.

Model	**Hoop Walk Removed**	**Mat Walk Removed**
Random Forests	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

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

Feature engineering approach

Presented in Section 3.4, considered most effective because it obtained the highest accuracy of all approaches and it is fastest to train. Data is modified by applying Feature Engineering techniques presented in 3.4, this leads to the dataset having fewer rows and columns.

Model	Accuracy	F1	Recall	Precision	MCC
Multi-Layer Perceptron	**0.83**	**0.83**	**0.84**	**0.84**	**0.81**
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 Forests	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

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

In Table 4.6 results obtained from the Cross-Validation method are displayed. High values are obtained for all 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 Forests*, *Support-Vector Machines*, *K-Nearest Neighbors* obtained great results as well with a score greater than **0.70** for accuracy.

This confirms that Feature Engineering techniques applied to 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 presents results obtained from the Hold-Out validation method. Similar to the ones obtained with the validation method, with a lower training time of **10 seconds**. However, Cross-Validation is preferred over Hold-Out because it is more used in literature.

Table 4.8 presents the results of a final approach used in the Correct approach. 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 Feature Engineering techniques applied to be more informative than raw Kinect data. Leading to models being able to differentiate between two movements more easily.

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

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, and 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

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

4.3 Discussion

This section discusses results obtained from experiments conducted in the previous chapter. Differences between approaches, best-performing models, and similarities between movements are topics of discussion.

Differences between approaches

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

Best performing models

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

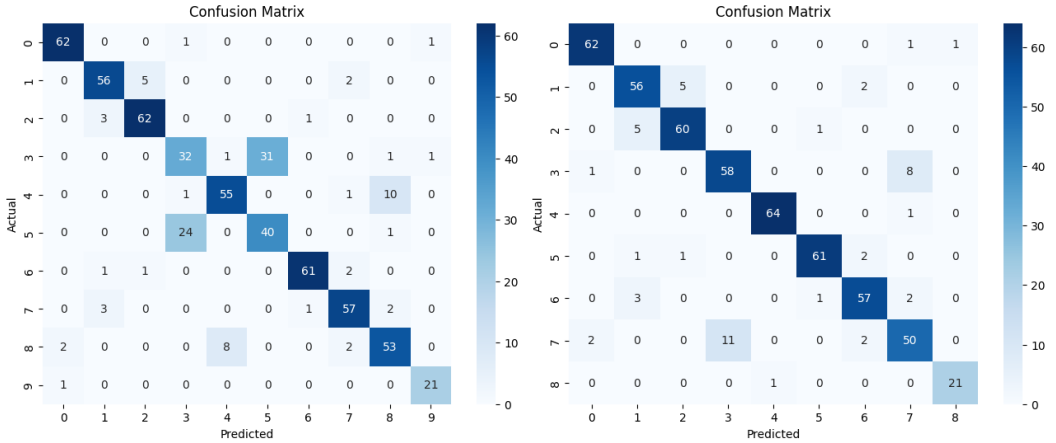
1. **Random Forests** and **Gradient Boosting** are two best performing models in *Correct approach* with a **0.74** and **0.73** accuracy score respectively.
2. **Multi-Layer Perceptron** and **Logistic Regression** are two best performing models in *Feature Engineering approach* with a **0.83** and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

The results above demonstrate how models perform differently depending on the approach used, due to data used to train models being of different dimensions and information.

Movements similarity

It is 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 is first noticed when a Confusion Matrix is plotted for *Feature Engineering approach* using *Multi-Layer Perceptron* model. In Figure 4.3a Confusion Matrix of all 10 movements is displayed, the model is struggling to differentiate between movements 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.



(a) 10 Movements. **(b)** 9 Movements.

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

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

(a) Hoop Walk **(b)** Mat Walk

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

Chapter 5

Conclusions

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

5.1 Discoveries

Presented below are the key findings of this thesis:

1. *Pre-processed raw data* obtained from the Kinect sensor is suitable for this 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 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 "Patient-ID" as a split criteria is the best approach to avoid any data leakage between 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 data into fixed-length sequences will lead to a loss of information and a decrease in accuracy.

6. *3D visualization* of movements allows us to visually identify and label them. It is discovered that "Mat Walk" and "Hoop Walk" are very similar, with the only difference being the object that the patient is walking over. This led to models struggling to differentiate between these two movements and by removing one of them from the dataset 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 movement names was provided with the dataset, however, movements were not labeled according to the list and only a unique ID was assigned to each one. This led to a need to update labels after visually identifying them with the help of a 3D visualization.
While 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 are relatively small, with only 10 movements and 21 patients. This led to only using Training and Testing sets for the evaluation of the models, as the dataset was too small to split 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 approach are not accurate to the literature due to only using positional data from the Kinect sensor. However, they still provide enough information for models to obtain a high accuracy.

5.3 Future work

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

Feature Engineering approach obtains a high accuracy and reduces 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 has been reduced but it was not possible to tell which ones contribute 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 *Scikit-Learn* library. It is possible to implement new models from *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 ac-

curacy than models used in this thesis with a correct implementation.
It is crucial to acquire new data from the Kinect sensor and add new movements and patients. This will allow us to scale up the dataset and study how 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 is to study the feasibility of using Kinect skeleton 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. 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] Açış, 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., Harbidge, 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 multilayer 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] Virtanen, P., et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, **17** (2020), 261. doi:10.1038/s41592-019-0686-2.

[29] 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.

[30] Wright, M. and Freed, A. Open SoundControl: A New Protocol for Communicating with Sound Synthesizers.

[31] 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, doi:10.1007/978-1-4419-9878-1_4.

[32] 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.

[33] 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.

[34] İnce, . F., Ince, 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.