University of Genoa

Computer Engineering

A comparative analysis across algorithmic, machine learning, and visual paradigms for the automatic detection of the perceived origin of full-body human movement

Candidates

Fausto Martina, Romano Gabriele

Advisors

Prof. Volpe Gualtiero, Prof. Oneto Luca

Academic year 2022/2023

Table of Contents

List of Figures

List of Tables

Acronyms

AUC	Area Under the Curve
BF	Brute Force Algorithm
B-SMOTE	Borderline Synthetic Minority Over-sampling Technique
DT	Decision Tree
FPR	False Positive Rate
HM	Hungarian Matching Algorithm
KNN	K-Nearest Neighbors
LOOCV	Leave-One-Out Cross-Validation
ML	Machine Learning
MoCap	Motion Capture
MaxWPM	Maximum Weight Perfect Matching
MinWPM	Minimum Weight Perfect Matching
RF	Random Forest
ROC	Receiver Operating Characteristic
TNR	True Negative Rate
TPR	True Positive Rate
TSS	True Skill Statistic
WDC	Weighted Degree Centrality

Chapter 1

Introduction

The thesis consists of a summary report and a pipeline written in Python. The dataset was sourced from the archives of Casa Paganini, the same location where artists' performances were recorded. All the figures and the tables included in this thesis are our own creation.

1.1 Context

In sports, dance, and physical activities, motion capture technology stands as a game-changer. It offers immense benefits for athletes and artists by revolutionizing their training methods and performance outcomes. The use of motion capture technology goes beyond traditional training approaches. It provides athletes with precise insights and tools to refine their techniques and improve performance. Whether it's analyzing running styles or perfecting the fluid movements in dance routines, these technologies play a crucial role in skill enhancement. Moreover, these tools aren't solely focused on improving performance; they also help prevent injuries by identifying potential stress points or incorrect body movements that could lead to harm. In rehabilitation, motion capture accelerates the recovery process by monitoring movements accurately. This allows for

tailored rehabilitation programs, ensuring a quicker return to peak physical condition after an injury. What's impressive is that this technology encourages self-improvement. It allows individuals to monitor their performances, pinpoint areas for improvement, and make necessary adjustments independently. This fosters a culture of continuous self-improvement without always relying on external expertise. Additionally, motion capture isn't limited to professionals; it's accessible for enthusiasts and newcomers. It encourages independent learning and exploration of movement analysis, biomechanics, and performance enhancement. In essence, motion capture technology redefines training, performance, and recovery by not only enhancing performance and preventing injuries but also empowering individuals to take charge of their own progress.

In kayaking, trunk motion stands as a critical factor influencing both injury prevention and performance enhancement [1]. Previous kinematic studies within this domain have predominantly occurred in controlled laboratory settings employing paddling simulators and ergometers. However, these setups fail to authentically emulate the complexities of kayaking in a competitive water environment. To address this limitation, a video camera-type kayak motion capture system was introduced, utilizing action cameras affixed to a kayak to capture markers placed on an athlete's body.

The use of Motion Capture is also employed to identify which motion features can effectively distinguish between performances of professional violinists and those of novice students, without relying on the produced sound [2]. In traditional music education, the predominant approach revolves around a teacher-student relationship, where the delay between a student's performance and the instructor's feedback causes a disconnect between the teacher's input and the student's auditory perception, as discussed in [3]. Given that this interaction commonly takes place during weekly classes, this aspect becomes even more significant [4]. This critical aspect of music instruction is frequently overlooked, leaving students accustomed to practicing independently without comprehending how to structure and assess their progress during solitary rehearsals.

It's important to acknowledge that prolonged periods of self-study among students can be challenging, leading to a solitary experience that frequently results in a high dropout rate [5]. To effectively tackle these challenges in music education, it is highly beneficial to consider reflective thinking and the cognitive aspects of learning. As we can see in [6], there exist four innate forms of thinking within the humnan mind and the fourth type emphasized is known as reflective thinking, a foundational aspect of what we refer to as metacognition. Reflection isn't just a series of thoughts; it's a sequence where each thought leads to the next as its logical consequence, and each subsequent thought relates back to or builds upon its predecessors. This underlines the importance of fostering reflective and metacognitive thinking in music education, enabling students to assess their own learning process. Employing self-regulation strategies and metacognition can enhance the outcomes of music learning. In regard to self-regulation strategies, Nielsen [7] investigated their application by studying how two college-level musicians monitored their learning progress. This involved observing practice behaviors, collecting verbal reports during practice sessions, and conducting retrospective debriefing reports post-practice to delve into the facets of self-directed learning. The [2] proposes a system for automatically classifying recordings of performances by professional musicians and violin students. This analysis will focus on two distinct scenarios, aiming to expand the findings obtained concerning different exercises and diverse violinists. Additionally, efforts will be made to identify the most relevant movement characteristics for assessing a violinist's abilities, thereby reinforcing the significance of the obtained results. These outcomes, besides validating the model's accuracy, will be pivotal in gaining a deeper understanding of the challenge and advocating for the utilization of this technique as a valuable support tool for students, aiding them in tracking and enhancing their individual practice at home.

In [8] it is introduced a computational model and a system designed to automatically recognize emotions based on full-body movements. The three-dimensional motion data

capturing these movements is sourced from professional optical motion-capture systems (*Qualisys*) or more affordable RGB-D sensors (*Kinect* and *Kinect2*). The resulting model and system have been successfully applied in the development of serious games for helping autistic children learn to recognize and express emotions by means of their full-body movement. Traditionally, theories of emotion have primarily focused on facial and vocal expressions, sidelining the significance of full-body movements and expressive gestures until recent years. Limited explorations in psychology and computer systems analyzing full-body movement for emotional and expressive content existed. Recent studies highlight advantages in expressing and recognizing emotions through full-body gestures. For instance, from a distance, bodily cues are more perceptible than subtle facial changes. A growing body of research in affective neuroscience underscores the role of the entire body in expressing and even subconsciously recognizing emotions. This shift is evident in multimedia technology utilizing full-body movement across various applications.

One of the Sustainable Development Goals suggested by the United Nations Organization (ONU) is geared toward achieving universal and comprehensive healthcare coverage while reducing related inequalities, aiming to ensure that everyone enjoys good health [9]. In line with this, it is acknowledged that inequalities contribute to millions of people with disabilities facing challenges in performing their basic daily activities. This disparity is more pronounced among individuals from communities with fewer opportunities and resources, typically located in geographically distant areas from the necessary rehabilitation services [10]. Among various types of disabilities, motor impairment is considered one of the primary hindrances in executing daily activities for humans, significantly impacting the individual's quality of life and those around them [11]. In recent years, telemedicine and telerehabilitation have been enhanced through the implementation of diverse technologies supporting rehabilitation processes. These innovations aim to provide necessary services to patients, reducing the need for

frequent trips to major cities, where specialists, hospitals, clinics, and technologically equipped therapy centers are typically located. [12] focuses on supporting the physical rehabilitation of upper limbs using video games and a motion capture system, primarily employing the Kinect sensor and inertial sensors for motion capture.

Cycling is one of the most widely practiced sports globally, both professionally and recreationally. Covering long distances during a training session imposes significant metabolic and biomechanical stress on the body. Epidemiological studies have demonstrated that cycling carries a high incidence of overuse injuries, making it one of the sports with a considerable number of yearly incidents, following basketball and soccer. Analyzing a cyclist's posture during pedaling is crucial not only to identify biomechanical factors limiting performance but also to uncover injury predispositions. Abnormal pelvic movements during cycling correlate with poor bike fit and related pathologies. Employing a Motion Capture System, considered the gold standard for assessing biomechanical parameters in sports performance, our aim is to investigate variations in pelvic kinematics during different phases of the pedaling cycle. Ten cyclists were engaged in the study, pedaling at varied cadences. Upon data collection, pelvic kinematic parameters were analyzed. Through statistical analysis, the three different pedaling conditions were compared. This study [13] revealed that the lowest pelvic oscillation occurs at 90 revolutions per minute (rpm). Despite not being the lowest pedaling frequency, this rate proves to be the safest in preventing injuries and the most comfortable for cyclists.

1.2 Starting point and problem statement

Currently the method for automated analysis of body movement consists of an approach involving transferable-utility cooperative games on graph [14]. A Motion Capture dataset is originated by recording with 13 infra-red cameras, two professional dancers that were equipped with 64 infra-red reflective markers, 5 accelerometers and 1 microphone. Then

the perceived Origin of Movement are manually annotated by experts in the field. By using the native software "Qualisys Track Manager" have been computed the trajectories of each point and tracked across the whole timeframe of the sample.

The output of the software is a highly precise description of the trajectories of either 64 or 41 markers based on the version of the capture system; thus, for each marker we obtain a timeseries of x, y and z positions. For each sample, only the time frame in which there is a clear origin of the movement is considered. From this result the process is moved on the MATLAB programming language. The number of markers has been compressed by clustering and mapping the markers based on a scheme that uniquely maps the human skeletal structure, resulting in 20 points, that from now will be called *joint*. By iterating this process over each sample of the dataset a list of 36 labeled timeseries is obtained.

From here three kinds of movements features are calculated for each joint and for each time instant: velocity, acceleration and angular momentum. A first graph is created by linking each joint that is physically connected with another, simulating the real connections of each joint. A weight for each arc (that from now will be called *edge*) is assigned based on a custom made similarity measure based on the inverse of the norm of the selected feature calculated between two joints. Here, the similarity function used in the previous work is reintroduced.

- In the case where the feature (referred to here as x) is scalar, as with velocity and acceleration, the following formula is applied:

$$\begin{cases} \frac{1}{\|x_j - x_i\| + \text{tol}}, & \text{if an edge exists between joint } i \text{ and } j \\ \frac{1}{5 \cdot (\|x_j - x_i\| + \text{tol})}, & \text{if no edge exists between joint } i \text{ and } j \end{cases} \tag{1.1}$$

- Instead, if the feature is vectorial ($\vec{x}$), such as angular momentum, the following

formula is applied:

$$\begin{cases} \frac{\vec{x}_i \times \vec{x}_j}{\|x_j\| \cdot \|x_i\| + 1}, & \text{if an edge exists between } i \text{ and } j \\ \frac{\vec{x}_i \times \vec{x}_j}{5 \cdot (\|x_j - x_i\| + 1)}, & \text{if no edge exists between } i \text{ and } j \end{cases} \tag{1.2}$$

Where the factor *tol* is a small number to prevent divisions by zero.

A clustering algorithm was applied to the graph, resulting in an auxiliary graph where the only edges present connect nodes belonging to different clusters. Upon this auxiliary graph, a mathematical game was constructed, and the solution was found by calculating the Shapley values for each joint.

Once the Shapley values were obtained, an online survey was used to validate their reliability in accurately predicting the perceived origin of movement. Users were asked to indicate their confidence level and select one or two vertices from the skeletal structure as the movement's origin. They were provided hints in three different ways by highlighting joints based on either the highest Shapley value, the two highest Shapley values (potentially also the third), or randomly chosen joints, shuffled for each participant. This methodology was applied across three distinct features: velocity, acceleration, and angular momentum.

The study revealed that the most confident users consistently chose the suggested movement origin indicated by the two highest Shapley values, specifically those associated with the velocity feature.

This work has been reevaluated due to certain critical aspects and has been expanded with our contribution.

Specifically, the number of samples in the dataset was too limited, and the movements were overly complex, making the identification of the movement origin challenging, especially in cases where it presented multiple origin of movements in the same fragment.

The similarity measure used for calculating the features had not been compared with others available.

Moreover, it appears that there there is not a clear contribution in terms of added value derived from nodes cooperation, thus reducing Shapley values and game theory to a weighted degree centrality. Additionally, the entire pipeline was developed using MATLAB's proprietary software, which, however, is quite limited in terms of data analysis libraries.

Finally, a manual annotation performed by only one individual may be prone to biases.

1.3 Thesis's structure

This paragraph outlines the structure of our thesis.

Chapter 2 introduces our proposed approach, highlighting the key differences from the starting point, emphasizing the added value and innovative proposition that our work brings.

Chapter 3 covers theoretical concepts and relevant notions essential for comprehending our thesis, including graph theory, theoretical aspects of machine learning, and definitions from smoothing theory and clustering stabilization.

Chapter 4 introduces the dataset used in our work, composed of movement samples sourced from two distinct datasets: *Wholodance* and the *Montpellier - UniGe* dataset. As the datasets were recorded in different years, the technologies used to capture movements differs, leading to distinct data formats based on the number of marker sets on the dancers' suits. We will also demonstrate the process of achieving consistency within our dataset by establishing a standardized structure with reduced marker sets. Additionally, the annotation part will provide classification for each sample.

Chapter 5, stands as the most extensive segment of our work. It precisely outlines the

procedures employed to achieve our three objectives: algorithmic prediction, temporal stabilization of clusters and prediction using machine learning approaches.

Lastly, Chapter 6 presents and discuss the results obtained from each major task. It also presents some ideas for the future researches.

Chapter 2

Proposed Approach

Our work commenced with an analysis of the paper by Kolykhalova et al. [14] and its subsequent implementation in the publication by Matthiopoulou et al. [15], retaining some aspects of the previous research while introducing a different and more comprehensive approach for predicting movement origins using machine learning.

We developed for visualization purposes a method to prevent inconsistent coloring behaviours among clusters over time, by introducing a novel cluster stabilization algorithm. This can aid in visualization to identify any potential origins of movement.

Another improvement made was to retrieve and expand the previous dataset using movement fragments. We chose to utilize a subset of the previous dataset, and also increase the number of samples by manually annotating videos from other internal resources provided by Casa Paganini [16].
This choice has been motivated by the necessity to have many short samples that clearly present an origin of movement instead of few samples that are hard to infer in terms of origin of movement. This approach aimed to achieve high-quality data suitable for

implementation in machine learning techniques for pure classification purposes.

The step forward we bring compared to the previous work is that, starting from a robust ground truth, we explored two different types of classification algorithms. Compared to the previous work, we will be able to perform a classification that is not mediated by the consensus of multiple participants but rather focuses on automating the process.

Roadmap From the starting point of the previous work, we re-implemented and modified the approach based on graphs and improved the whole framework introducing a novel machine learning-based pipeline.

The first part of our work was done from an algorithmic perspective to enhance the existing method. A modification we made to the previous work was to compare two similarity measurement methods: the one implemented in the original work and Cosine Similarity (explained in Section 3.1).
Furthermore, we simplified the Game Theory to a simple WDC to expedite the pipeline and avoid calculating 20! coalitions.
We shifted the validation of fragments to the beginning of the pipeline, using a multi-step validation mechanism between expert and non-expert annotators. This was done to minimize the potential effects of biases that could have been introduced into the model we would subsequently use.

The second part of our work concerns the development of a machine learning approach to achieve classification of the origin of movement, not just perceived but actual. We focused extensively on the dataset to attain a robust method for automatic recognition.

The roadmap of this thesis can be outlined as follows:
The manual annotation was performed through an agreement between us, two non-expert

annotators, followed by an initial filtering by a third independent non-expert annotator. Finally, the last filtering was applied by an expert performer. So the final dataset was composed by only those samples that received consensus among all the annotators.

Our work encompass the reaching of three main objectives.

A **Cluster Transistion Smoothing Algorithm** which ensure that clusters representing groups of related markers or joints, maintain their consistency and coherence over time. This first one is just for visualization purposes and does not implement a classification method that can be compared with the others.

A **Graph-based Procedure** to recognize the Origin of Movement by identifying key nodes that play pivotal roles in the motion analysis using an algorithmic method that has been handcrafted for this dataset.

A **Machine Learning Model** that perform the same classification task as the previous, but automatically recognizes patterns within the data, without explicitly coding them by hand.

Considering the multifaceted objectives, two parallel pipelines were developed, with the compression procedure of the markers being the only common aspect inherited from the previous work.

From now we will refer to the Figure 2.0.1 for the steps. The algorithmic pipeline (on the left) encompasses several more steps than the other (on the right).

Initially, we smooth the time series of each sample of movement in order to clean the data from impurities such as noise, missing data for markers occlusion or tracking errors that could end up creating unnatural accelerations of some markers (Timeseries Smoothing).

Subsequently, we extract physical derived measurements from the trajectories, like the velocity, acceleration and angular momentum from the smoothed data (*Features Extraction*).

Then, employing a model of human body joints, we apply cosine similarity to connected pairs of joints (*Cosine Similarity*). A graph is created with joints as nodes and physical

connections as edges (*Graph*). A clustering algorithm is employed to create a certain number of group of nodes (*Clustering*).

The primary aim at this stage is to ensure the stability of color changes within these clusters.

Additionally, we construct a secondary graph, where only nodes at cluster borders are interconnected (*Auxiliary Graph*). These nodes are weighted based on their connections and ranked across all frames (*Weighted Degree Centrality*).

The machine learning pipeline (on the right) follows a different approach on the data. We start by normalizing the dataset concerning segment length, the performer's body structure, and the complete movement trajectory (*Timeseries Normalization*).

Following this, we extract a diverse set of pertinent features from this set of normalized timeseries (*Features Extraction*).

These extracted features are then utilized to train a machine learning model (*Machine Learning Model*), whose performance is thoroughly evaluated using diverse metrics (*Evaluation Metrics*).

Finally, the results from each pipeline are compared and contrasted to evaluate their individual effectiveness.

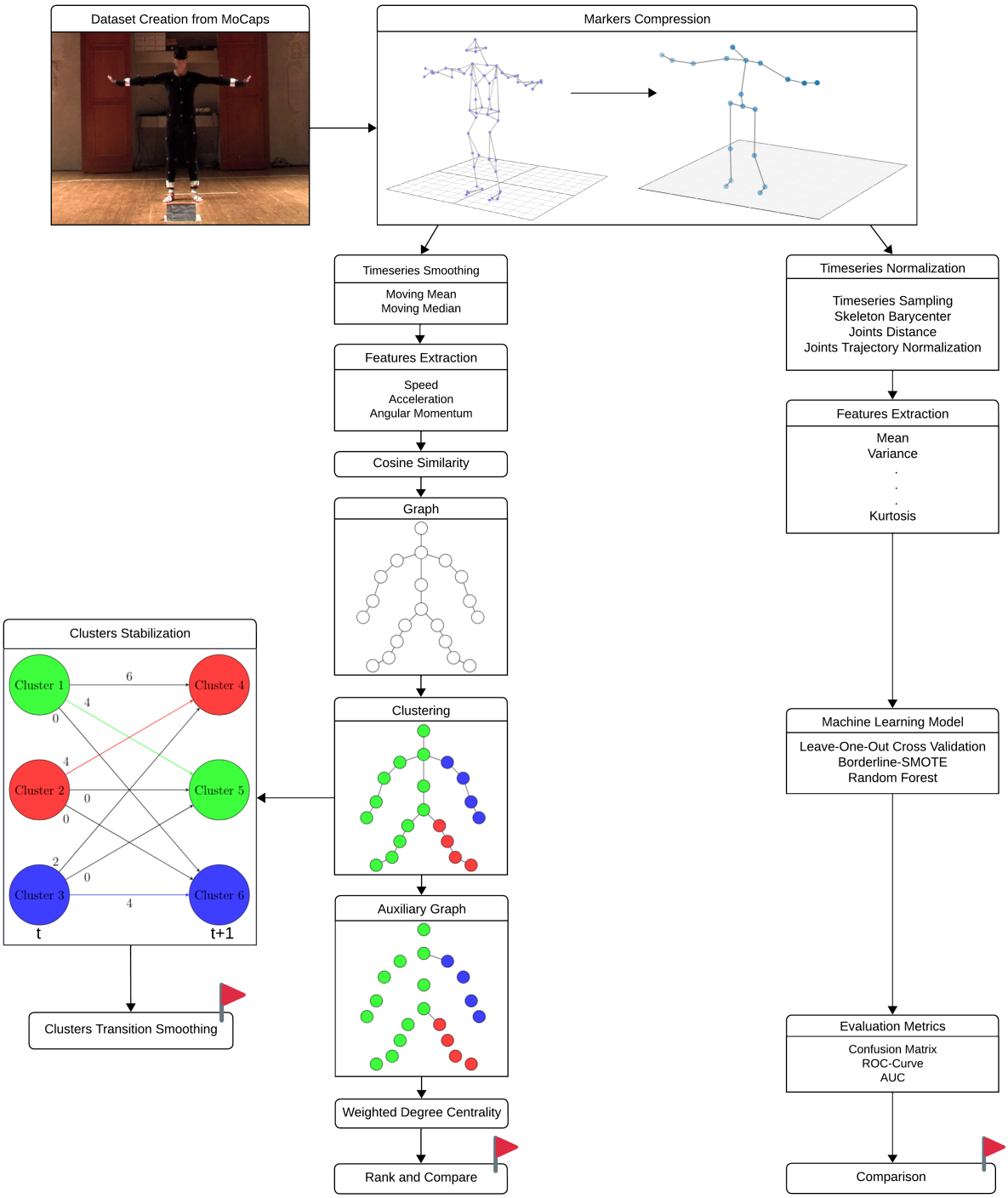


Figure 2.0.1: Roadmap of this Thesis

Chapter 3

Theoretical framework

This Chapter introduces concepts and notions about various domains crucial to understanding the research methodology.

It comprises sections focusing on Cosine Similarity, Smoothing theory (including Median Filtering and Rolling Mean), Graph theory (covering concepts like simple graphs, adjacency, weighted degree centrality, matrix representations, bipartite graphs, algorithms for matching, and others), and Machine Learning Theory.

Specifically, the section on Machine Learning Theory discusses classifiers, algorithms like K-Nearest Neighbors, Borderline Synthetic Minority Over-sampling Technique, and the Random Forest Classifier. Furthermore, it introduces methodologies such as Cross Validation and Evaluation Metrics to assess model performance.

This comprehensive framework covers foundational theories and methodologies pertinent to signal processing, network analysis, and machine learning, essential for understanding and implementing subsequent approaches discussed in the thesis.

3.1 Cosine Similarity

Cosine similarity is a measure of similarity between two vectors, frequently used to gauge how aligned vectors are in space.

It returns a value ranging from -1 to 1. The concept is that similar vectors will have a small angle between them, resulting in a cosine similarity value close to 1.

Conversely, dissimilar vectors will have a wide angle and a cosine similarity close to 0. A higher value indicates greater similarity, where 1 represents identical vectors (perfect similarity), 0 means no similarity, and -1 indicates perfect dissimilarity (opposite directions). We iteratively calculate the similarity between all the joints connected by an edge. The cosine similarity between two vectors x and y is defined as:

$$cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \cdot \|\vec{y}\|} = \frac{\sum_{i=1}^{3}(x_i \cdot y_i)}{\sqrt{\sum_{i=1}^{3} x_i^2} \cdot \sqrt{\sum_{i=1}^{3} y_i^2}} \tag{3.1}$$

3.2 Smoothing theory

This section contains definitions about the smoothing algorithms used on data to mitigate the impact of noise in data.

3.2.1 Median Filtering

Median filtering is a digital signal processing technique used to reduce noise in a signal by replacing each data point with the median value within a specified window or neighborhood around that point. This method is effective in removing outliers or random fluctuations while preserving the signal's overall structure.

For every i to n, the number of the observations the value of x_i is set to the median of the values within the window i and $i + k$.

3.2.2 Rolling Mean

The rolling mean involves computing the average of a specific number of consecutive observations while shifting this window along the time series. The centering of the window in the middle of it. This means that for every i to n, the number of the observations:

$$rolling(x_i, k) = \frac{1}{k} \sum_{j=i}^{i+k} x_j \tag{3.2}$$

On extreme index of the observations (i.e when $j < 1$ or when $j > n$) the k is reduced such that $1 \leq i - k \leq i + k \leq n$ to keep the centering of the window. In the following Figure 3.2.1 it is plotted a sinusoidal signal with holes that have been replaced with 0 and the different behaviours of the *Median Filtering* function and the *Rolling Mean*. We can clearly see that while the second method tends to fit the holes, the first one better reconstruct the original signal.

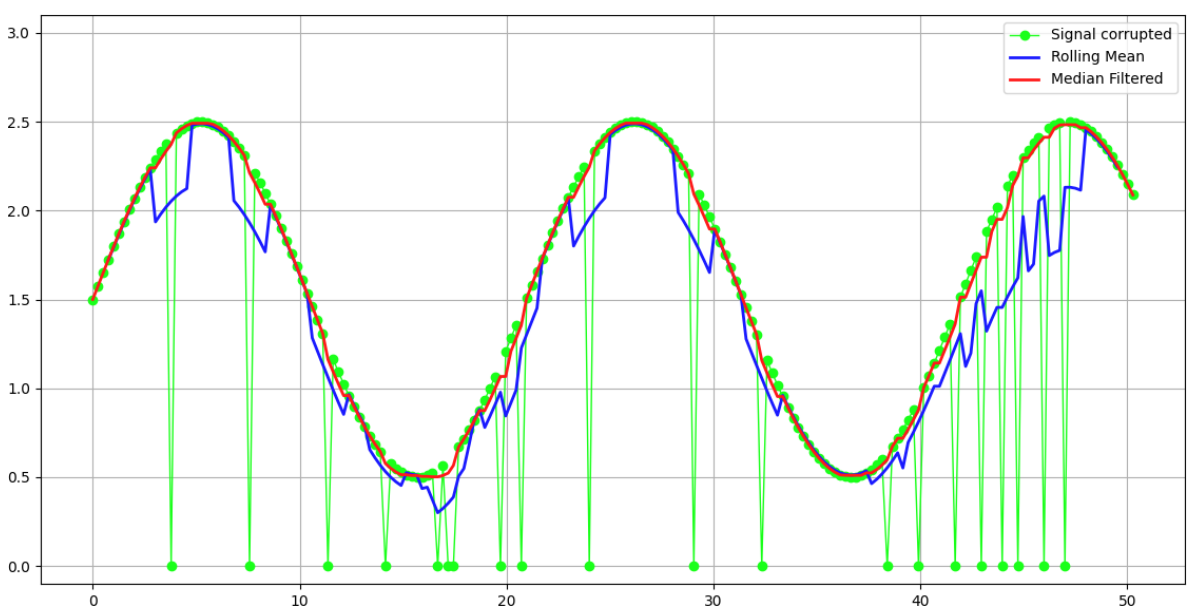


Figure 3.2.1: A sinusoidal signal with corrupted data and smoothing methods applied

3.3 Graph theory

This section contains definitions, notation and concepts related to Graph Theory according to [17]. Graph theory deals with connection amongst vertices by edges. Graphs are the foundation of many day to day processes and concepts, as they provide a convenient and intuitive way of representing objects.

3.3.1 Simple graphs

A **simple graph** G consists of a non-empty finite set *V(G)* of elements called **vertices** (or **nodes**), and a finite set *E(G)* of distinct unordered pairs of distinct elements of *V(G)* called **edges**. We call *V(G)* the **vertices set** and *E(G)* the **edge set** of G. An edge $\{v, w\}$ is said to join the vertices v and w, and is usually abbreviated to vw. For example, Figure 3.3.1 represents the simple graph G whose vertex set *V(G)* is $\{u, v, w, z\}$, and whose edge set *E(G)* consists of the edges uv, uw, vw and wz.

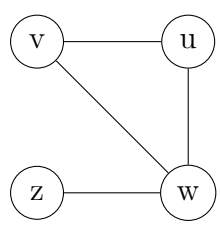


Figure 3.3.1: A simple graph

3.3.2 Adjacency

We say that two vertices v and w of a graph G are **adjacent** if there is an edge vw joining them, and the vertices v and w are then **incident** with such an edge.
Similarly, two distinct edges e and f are **adjacent** if they have a vertex in common (see Figure 3.3.2).

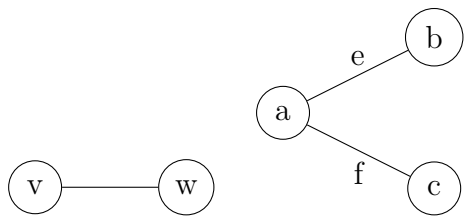


Figure 3.3.2: Adjacent vertices and adjacent edges

3.3.3 Weighted Degree Centrality

Weighted degree centrality is a measure used in network analysis to quantify the importance or centrality of a node in a network, taking into account the weights of the edges connected to that node. Unlike traditional degree centrality, which only considers the number of connections (edges) a node has, weighted degree centrality considers the strengths or weights associated with those connections. For a given node, calculate its weighted degree by summing up the weights of all the edges connected to that node. In mathematical terms, it can be expressed as:

$$WDC(v) = \sum_{i=1}^{N} w_{vi} \tag{3.3}$$

Where v is a vertex in the graph and N is the number of vertices.

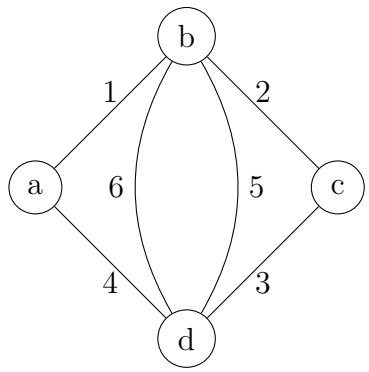
3.3.4 Matrix representations

One approach to representing a graph is through the use of matrices, especially when dealing with large graphs that may not be well-suited for diagram-based representations. Let's consider a graph G with n vertices and m edges.

An **adjacency matrix A** is the $n \times n$ matrix whose ij-th entry is the number of edges joining vertex i and vertex j.

If, in addition, the edges are labelled $\{1, 2, \ldots, m\}$, its **incidence matrix M** is the $n \times m$ matrix whose ij-th entry is 1 if vertex i is incident to edge j, and 0 otherwise.

An example of this is given in Figure 3.3.3.



$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 3.3.3: Graph G with its adjacency A and incidence M matrices

3.3.5 Bipartite graphs

If the vertex set of a graph G can be split into two disjoint sets A and B so that each edge of G joins a vertex of A and a vertex of b, then G is a **bipartite graph**. Alternatively, a bipartite graph is one whose vertices can be coloured red and blue in such a way that each edge joins a red vertex (in A) and a blue vertex (in B).

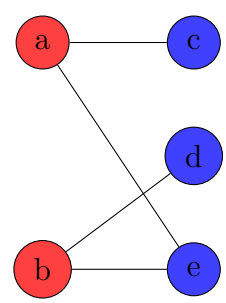


Figure 3.3.4: A simple bipartite graph

3.3.6 Bipartite graphs with matching

A **bipartite graph with matching** is a bipartite graph in which there is a set of edges selected in such a way that no node is shared among the edges of the matching. In other

words, each node is involved in at most one edge of the matching.

This means that if the number of vertices in A and B is different, at least one vertex will have no connection to another vertex (see Figure 3.3.5a).

If there is the same number of vertices in both A and B, every vertex is connected to another vertex (see Figure 3.3.5b).

Figure 3.3.5: Bipartite graph

3.3.7 Maximum Weight Perfect Matching

The Maximum Perfect Weight Matching is an optimization problem concerning graphs. The graphs in question are bipartite, meaning the vertices can be divided into two distinct sets in such a way that all edges of the graph connect vertices from different sets, with no edges connecting vertices within the same set. Additionally, these graphs are undirected, meaning the relationship between two nodes is bidirectional, in other words, the edge connecting node A to node B automatically implies the existence of the edge connecting node B to node A.

In the MaxWPM problem, the vertices represent elements to be paired, and the edges represent possible connections between these elements. Each edge is associated with a weight that represents the value or importance of the pairing between the connected vertices.

The goal is to find a perfect matching, which is a set of edges in which every node

is connected to exactly one other node, with no overlaps or isolated nodes, while maximizing the total weight of the edges in the matching. The total weight is the sum of the weights of the edges included in the matching. The weights of the edges are collected in a matrix called a *utility matrix*.

The utility matrix will be a square matrix because the number of nodes in the two distinct sets is the same. The rows could represent the nodes in the first set, while the columns could represent the nodes in the second set, or vice versa. The element (i, j) of the matrix represents the weight of the edge connecting vertex i to vertex j.

A MaxWPM problem using an utility matrix can be transformed into a **Minimum Weight Perfect Matching** problem. This can be achieved by introducing an auxiliary matrix, referred to as the *cost matrix*. The cost matrix is essentially a duplicate of the utility matrix, but with all its elements negated.

3.3.8 Brute Force Algorithm

The BF can be a possible approach to solve the MaxWPM problem for bipartite graphs. However, it is important to note that the BF has exponential time complexity, specifically, its time complexity grows exponentially with the number of vertices in the graph. This is because the BF examines all possible combinations of matchings within the bipartite graph to find the one with the maximum weight.

If the graph has n vertices in both the first and second partitions, there are $n!$ possible matchings. Each matching requires $O(n)$ time to be computed because you need to check that it is a perfect matching and calculate its total weight. Therefore, the overall complexity of the algorithm is on the order of $O(n! \cdot n)$. Since the complexity grows rapidly as n increases, it quickly becomes impractical for large-sized graphs.

The operation of the BF for MinWPM problems can be summerized in these 3 steps:

1. Compute the cost matrix for the considered bipartite graph with n vertices in each partition.
2. Compute the total cost of all the $n!$ permutations.
3. Choose the assignment with the lowest total cost.

3.3.9 Hungarian Matching Algorithm

The HM, also called the Kuhn-Munkres algorithm, is a $O(n^3)$ algorithm that can be used to find MaxWPM in bipartite graphs with n vertices for each partition, which is sometimes called the assignment problem. As previously demonstrated in the Brute Force algorithm, maximum-weight perfect matching problems using the utility matrix can be converted into MinWPM problems using the cost matrix. We will describe the application of the HM to address Minimum Weight Perfect Matching problems.
The operation of the HM for MinWPM problems can be summerized in these 6 steps:

1. Compute the cost matrix for the considered bipartite graph with n vertices in each partition.
2. Subtract the smallest entry in each row from all the other entries in the row. This will make the smallest entry in the row now equal to 0.
3. Subtract the smallest entry in each column from all the other entries in the column. This will make the smallest entry in the column now equal to 0.
4. Draw lines through the row and columns that have the 0 entries such that the fewest lines possible are drawn.
5. If there are n lines drawn, an optimal assignment of zeros is possible and the algorithm is finished. If the number of lines is less than n, then the optimal number of zeroes is not yet reached. Go to the next step.

6. Find the smallest entry not covered by any line. Subtract this entry from each row that isn't crossed out, and then add it to each column that is crossed out. Then, go back to Step 3.

Let's take an example of a cost matrix and try to apply the HM for MinWPM problems as in Table 3.3.1:

(a) From the table that describes the cost associated for every assignment

(b) Subtract the smallest value in each row from the other values in the row

(c) Now, subtract the smallest value in each column from all other values in the column

(d) Since there are 2 lines drawn, less than 3, so there is not yet the optimal number of zeroes.

(e) Find the smallest entry not covered by any line. Subtract this entry from each row that isn't crossed out, and then add it to each column that is crossed out. Then, go back to step (c). 2 is the smallest entry. First, subtract from the uncovered rows.

(f) Now add to the covered columns

(g) Now go back to step (c), drawing lines through the rows and columns that have 0 entries

(h) There are 3 lines, so we are done. The assignment will be where the 0's are in the matrix such that only one 0 per row and column is part of the assignment.

(i) And this results in the following minimum total cost for the assignment problem.

108	125	150
150	135	175
122	148	250

(a)

0	17	42
15	0	40
0	26	128

(b)

0	17	2
15	0	0
0	26	88

(c)

0	17	2
15	0	0
0	26	88

(d)

-2	15	0
15	0	0
-2	24	86

(e)

0	15	0
17	0	0
0	24	86

(f)

0	15	0
17	0	0
0	24	86

(g)

0	15	0
17	0	0
0	24	86

(h)

108	125	150
150	135	175
122	148	250

(i)

Table 3.3.1: Resolution of the Hungarian Algorithm

3.4 Machine Learning Theory

ML deals with the ability of a system to improve its performance through experience, without being explicitly programmed. It also serves as the foundation for numerous everyday applications and concepts, as it provides a powerful and efficient way to address complex challenges, make data-driven decisions, and develop intelligent systems that can learn and adapt autonomously.

Diving deeper into this concept, one critical aspect is the role of hyperparameters. Hyperparameters are settings or configurations that guide the learning process of ML algorithms. They act as the levers that fine-tune the behavior of these algorithms, influencing their performance, accuracy, and generalization capabilities.

Hyperparameters encompass a wide range of choices, such as the learning rate in gradient descent, the depth of a decision tree, the number of hidden layers in a neural

network, or the number of clusters in a k-means clustering algorithm. Selecting the right hyperparameters can be a challenging and often iterative task, as they significantly impact the model's ability to capture underlying patterns in data.

The process of hyperparameter tuning involves experimenting with different combinations of values, assessing the model's performance using techniques like cross-validation, and iteratively adjusting the hyperparameters to optimize the model's accuracy and generalization. This fine-tuning is crucial to ensure that the ML model not only fits the training data well but also performs effectively on unseen data. Moreover, the choice of hyperparameters is problem-dependent, and there's no one-size-fits-all solution. It requires a deep understanding of the algorithm, the dataset, and the specific problem at hand.

3.4.1 Classifier

A mathematical **Classifier** is a model that takes an input dataset and assigns each data point to a class or category. The mathematical formulation of a classifier can vary depending on the algorithm used. Suppose we have a training dataset represented as pairs $(\overline{x}, y)$, where $\overline{x}$ is a vector of data features, and y is the associated class label. The y is a categorical (can take a fixed, number of possible values) variable that categorizes between two or more classes (e.g.: dog, cat, bird), while in a regression problem it is a continuous variable. Our goal is to find a function $f(\overline{x})$ that maps the feature vectors $\overline{x}$ to class labels y.

A **Binary Classification Problem** is a specific type of classification problem where the goal is to categorize input data into one of two distinct classes or categories. These two classes are typically referred to as the positive class and the negative class, and the task is to assign each data point to one of these two classes based on its features or attributes.

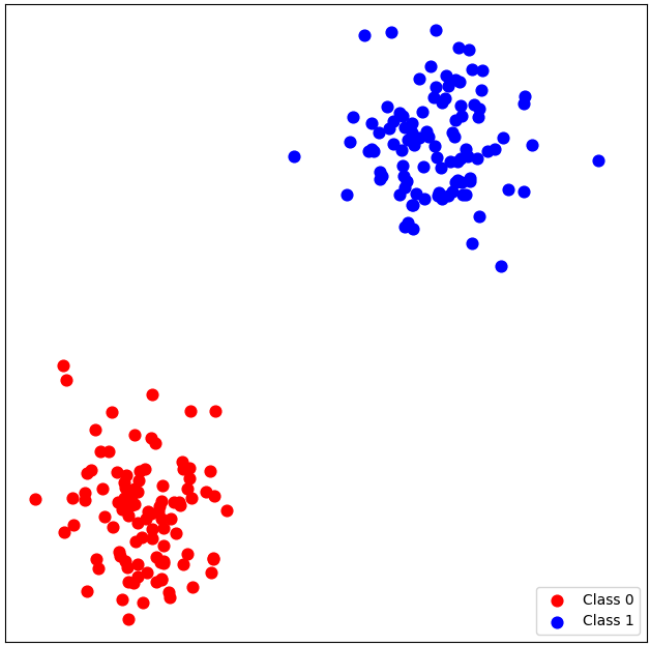


Figure 3.4.1: A binary classification example

3.4.2 K-Nearest Neighbors

The **KNN** algorithm is a supervised ML algorithm used for classification and regression tasks. It is a simple and intuitive algorithm that makes predictions based on the similarity between a data point and its k-nearest neighbors in a training dataset. The k in k-nearest neighbors refers to the number of items that the algorithm uses to make its prediction whether its a classification problem or a regression problem. In Figure 3.4.2 we can see an example of a test sample represented by a gray dot and we want to classify it either as a blue circle or a red circle based on a KNN algorithm, the decision depends on the value of k, which determines how many nearest neighbors we consider to take the decision in a majority voting mechanism.

When $k = 3$ (solid line circle), the gray dot is assigned to the category that has the majority among its three nearest neighbors. In this case, if there are 2 red circles and only 1 blue circle inside the inner circle, the gray dot is classified as a red circle.

When $k = 5$ (dashed line circle), the gray dot is assigned to the category with the

majority among its five nearest neighbors. If there are 3 blue circles and 2 red circles inside the outer circle, the gray dot is classified as a blue circle.

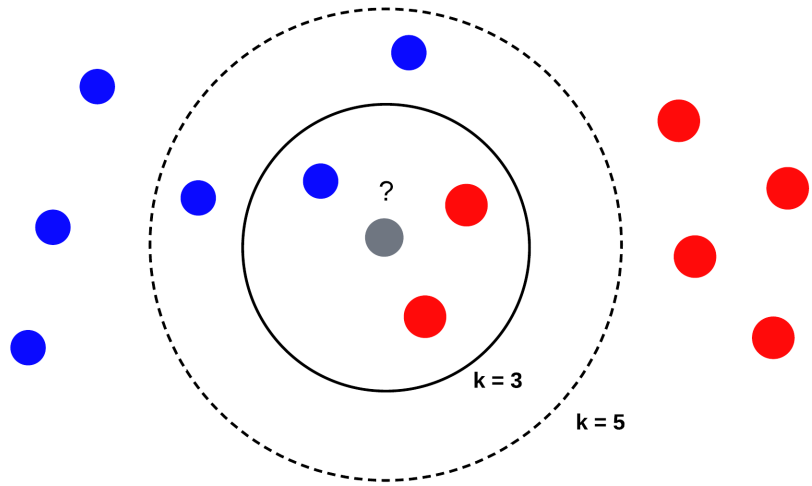


Figure 3.4.2: KNN algorithm with $k = 3$ and $k = 5$

The choice of distance metric is a critical aspect of the KNN algorithm, and while it is not typically considered a hyperparameter, it significantly influences the model's performance and outcomes. The selection of a specific distance metric can have a substantial impact, especially when dealing with datasets of varying sizes and dimensions.

If we consider for example two data points p, q in n-dimensional space, there are various distance metrics that can be used to find the neighbors.

Among the different most common is the Euclidean distance:

$$Euclidean(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{3.4}$$

3.4.3 Borderline Synthetic Minority Over-sampling Technique

Oversampling is a technique used in the field of imbalanced ML to address the problem of class imbalance. Class imbalance occurs when one class in a classification problem has significantly fewer instances than another class. This imbalance can lead to biased

models that perform poorly on the minority class (the class with fewer instances) because the model tends to be biased towards the majority class.

The concept of oversampling involves increasing the number of instances in the minority class by generating synthetic or duplicate samples. The goal is to balance the class distribution, making the minority class more comparable in size to the majority class. A synthetic sample can be generated duplicating randomly another sample of the minority class or considering its underlying data distribution (ADASYN [18]). This could result in a performance improvement of ML models by providing them with more information about the minority class.

The SMOTE generates synthetic samples by interpolating between neighboring minority class instances using an underlying KNN model.

The **B-SMOTE** [19], is a vairant of the SMOTE that focuses on generating synthetic samples only for the boundary instances, which are the minority class instances close to the majority class. This approach aims to concentrate on the regions of the dataset that are near the decision boundary between classes.

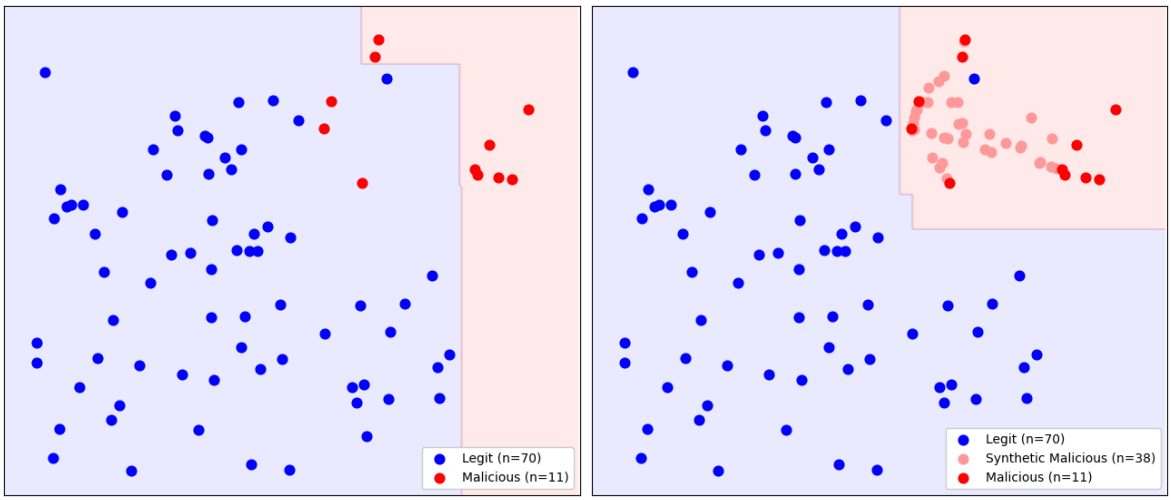
The model internally fits two instances of KNN with different parameters *k-neighbors* and *m-neighbors*: one instance defines the neighborhood of samples to generate the synthetic samples; the other, instead, is used to determine if a minority sample is in *danger*.

A *sample in danger* is one near to the boundary between two or more classes.

In the illustrated scenario depicted in Figure 3.4.3a, the task involves categorizing two distinct categories of network traffic: one being malicious (depicted in red), the other representing legitimate traffic (shown in blue), and the decision boundaries are the same color meshes. The distribution of malicious traffic is more sparse and rare compared to legitimate traffic, this means that the classifier has been trained on an imbalanced dataset. The primary objective of the classification is to maximize the detection of

malicious traffic, even if it entails misclassifying some instances of legitimate traffic. B-SMOTE is a possible solution for this scenario, where the two classes are significantly imbalanced, and the weight of legitimate traffic in the classification process could be overwhelming.

The application of the B-SMOTE algorithm facilitates class rebalancing by generating synthetic data points in the boundary regions between the classes concentrated in proximity of the decision boundary. In Figure 3.4.3b, the depicted image showcases the ultimate outcome achieved following the retraining of the classifier. The retraining process maintained the same parameters but employed a dataset that has been more uniformly balanced. The synthetic samples generated by B-SMOTE are represented as light-red dots.



(a) **(b)**

Figure 3.4.3: Visual representation of B-SMOTE algorithm applied on a dataset

However the B-SMOTE algorithm, has its limitations and drawbacks:

- **Risk of Overfitting:** In the provided example, the malicious data was generated using a normal distribution, while the legitimate data was generated using a uniform distribution. This means that the second classification fits better the underlying distribution without any a-priori assumption. But in some cases

depending on the choice of borderline examples, B-SMOTE may introduce a bias towards certain patterns or classes in the synthetic samples, which can affect the model's generalization. It can potentially lead to overfitting, especially when it generates a large number of synthetic samples in the borderline region. This may cause the model to perform exceptionally well on the training data but poorly on unseen data.

- **Computationally intensive:** It can require high hardware resources, especially in high-dimensional spaces, as it involves calculating distances between data points. This can lead to longer training times, making it less suitable for large datasets.
- **Designed for Clearly Separable Datasets:** It is designed for datasets with clear class boundaries and is most effective when the borderline instances are well-defined. In cases where class separation is not distinct, it may not provide substantial benefits.

So in general, even if it aims to improve the classification of the minority class, there is no guarantee that it will always lead to better results.

3.4.4 Random Forest Classifier

Decision Tree A (DT) is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a internal nodes, leaf nodes and branches. From the root node, splitting criterias are applied on the samples creating branches and child nodes that lead to leaf nodes which represent the final classification of the tree. The most common impurity measure used to quantify the impurity of a dataset is the Gini criterion: For a set of items with J classes and relative frequencies p_i, $i \in 1, 2, ..., J$ the probability of choosing

an item with label i is p_i, and the probability of miscategorizing that item is:

$$\sum_{k \neq i} p_k = 1 - p_i \tag{3.5}$$

The Gini impurity is computed by summing pairwise products of these probabilities for each class label:

$$Gini(p) = 1 - \sum_{i=1}^{J} (p_i^2) \tag{3.6}$$

In the example shown in Table 3.4.1, the objective is to determine whether to grant a loan based on two pieces of information: the applicant's monthly income and the requested loan amount. The second figure illustrates the DT that has been trained on this dataset. The underlying principle of the DT is that at every non-terminal node, if there exist samples that need further separation, the following algorithm is applied:

1. It starts by calculating the impurity of the current node before making any splits. This initial impurity serves as a baseline for evaluating feature splits.

2. For each feature in the dataset, calculate a measure of impurity or information gain for potential splits. This typically involves examining the feature values and considering different threshold values.

3. Calculates the reduction in impurity (or information gain) achieved by the potential split. This is done by comparing the impurity of the current node with the impurity of the child nodes created by the split.

4. Choose the feature that results in the highest information gain. This feature becomes the one on which to apply the threshold for the split.

5. It applies the threshold that separates the data into two or more subsets.

6. Continue the process recursively for each child node created by the split. This means repeating steps 1 to 5 for each child node until a stopping criterion is met

(e.g., a maximum tree depth is reached).

The goal is to create child nodes that are as homogeneous as possible with respect to the target variable, thereby improving the predictive accuracy of the tree.

Income	**Loan_Amount**	**Loan_Approved**
2000	300000	Not Approved
3000	300000	Approved
4000	400000	Not Approved
5000	400000	Not Approved
10000	400000	Approved
15000	400000	Approved

Table 3.4.1: Dataset of the DT

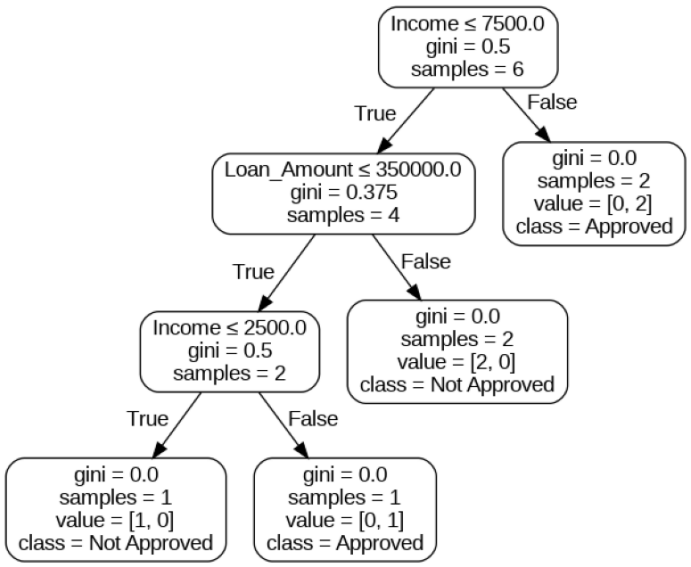


Figure 3.4.4: DT applied on the dataset of Table 3.4.1

In the example provided the term "samples" represent the total number of training samples used to create the specific tree node. In other words, it is the number of data examples that were considered in that node during the tree construction process.

Note that due to the absence of constraints on hyperparameters like tree depth or the

minimum number of features required to split nodes, it becomes evident that each sample in the dataset follows a distinct path within this tree. The term *value* represents the distribution of classes or labels of the samples within a node. It is a vector that indicates how many training samples in that node belong to each class or category. For instance, if we see $value = [0, 2]$, it means that there are 0 samples belonging to class *Not Approved* and 2 samples belonging to class *Approved* in that node. This example is simply used for showing the behaviour of a DT

Random Forest is an ensemble learning method that builds multiple DTs during training and combines their predictions to make more accurate and robust classifications or regressions. A RF is typically constructed in this way:

1. **Bootstrapped Data:** A random subset of the training data (with replacement) is used to train each DT. This results in each tree being trained on a slightly different dataset.

2. **Random Feature Selection:** At each node of each DT, only a random subset of the available features is considered for splitting. This helps introduce diversity among the trees and reduces the risk of overfitting.

3. **Independent Training:** Each DT is trained independently. There is no communication or shared information between the trees during training.

4. **Voting or Averaging:** For classification tasks, the RF combines the individual tree predictions by applying a Majority Voting (each tree votes for a class and the final prediction reflects the class that receives the most votes overall)

The key idea behind the RF method is that by aggregating the predictions of multiple DTs, the ensemble can reduce overfitting, improve generalization, and provide more robust and accurate results. For example the Figure 3.4.5 is shown a RF classifier made

of 3 DTs has been created applying the bootstrap method on the dataset and correctly classifies the first sample of the dataset in Table 3.4.1.

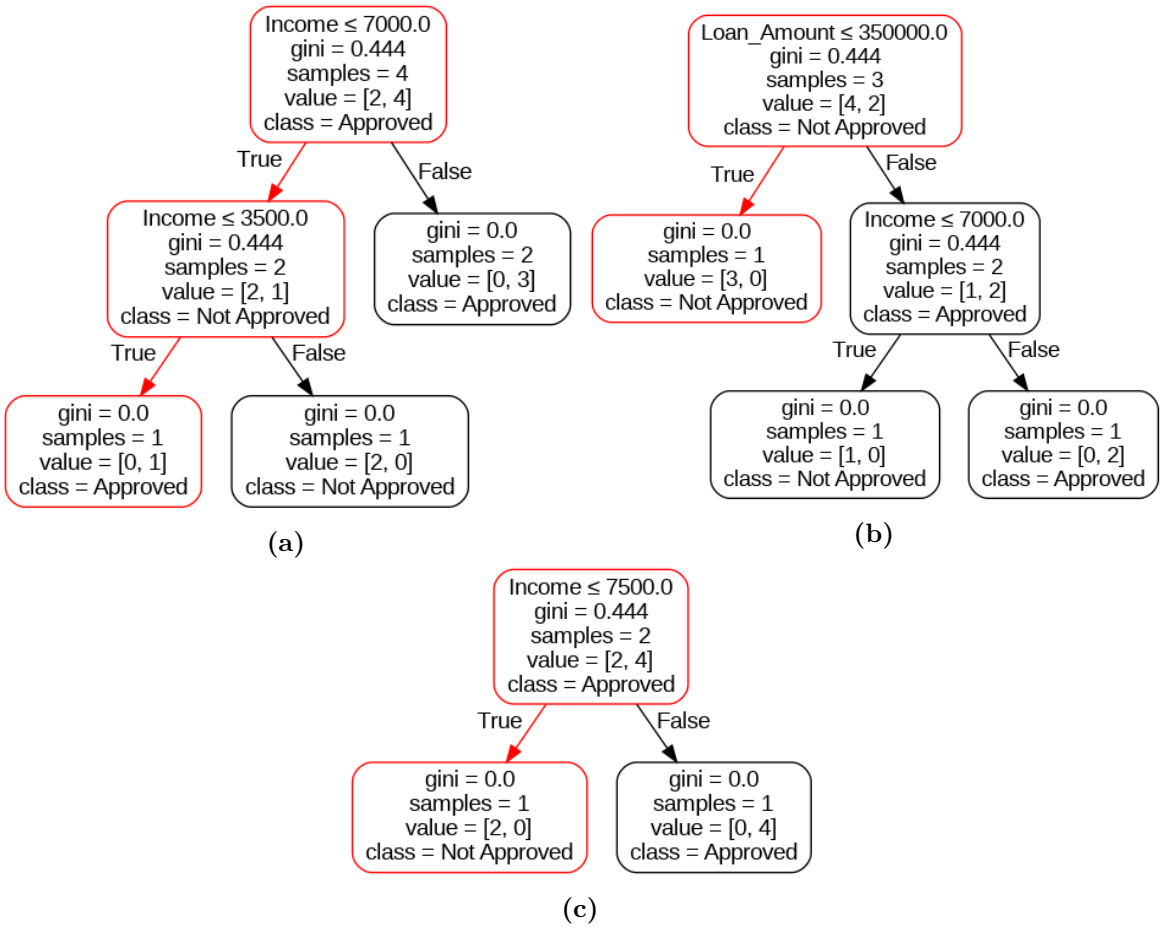


Figure 3.4.5: Path taken by the first sample through the three DTs of a RF classifier

3.4.5 Cross Validation

Cross Validation is a technique used to estimate the effectiveness of a ML model based on a data sample. It involves dividing the dataset into multiple training and testing subsets called *folds* and then training and evaluating the model on each fold. The process of training and testing is typically repeated multiple times and the results from each iteration are often averaged to provide an overall estimate of the model's performance. The most common approach in basic cross-validation is to use a 80-20 or 70-30 split between the training and test sets.

When we partition our dataset into folds for techniques like cross-validation, it is crucial to ensure that each fold accurately represents the diversity of the entire dataset. This entails maintaining the same proportion of different classes or categories within each fold.

While random splitting is often sufficient, there are instances, particularly in intricate datasets, where we must deliberately enforce the correct distribution of classes within each fold to maintain data integrity.

Leave-One-Out Cross-Validation (LOOCV) is a specific type of cross-validation where the number of folds k is set equal to the number of samples in the dataset n. In this approach, the model undergoes training k times, with k being precisely the total number of samples n. During each of the k iterations, a distinct sample is singled out as the test set, while all the remaining samples are utilized for model training. This procedure ensures that every single sample gets its turn as the test set, comprehensively assessing the model's performance across the entire dataset. In Figure 3.4.6 is shown how it works, note that training and testing is performed on the same model at each iteration. The evaluation metric is based on the consistency between the predicted values and the actual values across all iterations.

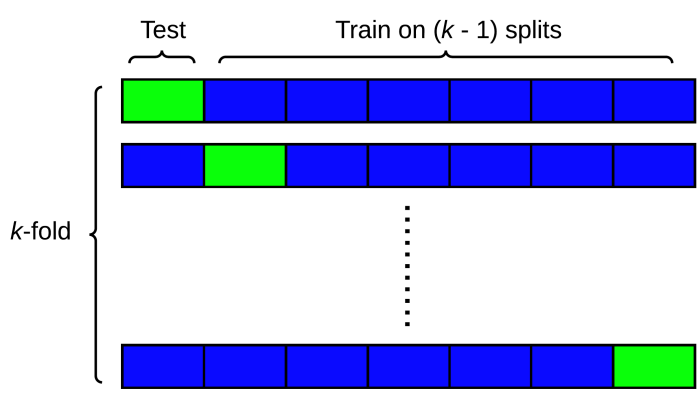


Figure 3.4.6: LOOCV algorithm example on k folds

However, LOOCV has some downsides because it can be computationally expensive, especially on large datasets, as it requires a significant number of separate model trainings. Training the model repeatedly for each sample can be time-consuming and resource-intensive. Instead, it's important to emphasize that although it's the preferable choice for small datasets, as it allows an unbiased estimation of the model's accuracy, in some cases, it can lead to high variance. This phenomenon has been observed especially in situations with the presence of outliers in the dataset and unstable models across multiple specific studies [20, 21, 22].

Moreover, utilizing RF as a model for high-dimensional, small-sample, and high-complexity datasets has been shown to further increase instability in repeated runs [23], adding an additional layer of averaging of results across multiple runs to obtain more truthful outcomes.

3.4.6 Evaluation Metrics

Evaluation metrics in ML are fundamental tools for measuring the effectiveness and accuracy of predictive models. These metrics provide a clear overview of the model's performance and assist in making informed decisions during the development and optimization process.

The **confusion matrix** is an essential tool used in the evaluation of classification models in ML. It provides a detailed overview of a model's performance by comparing the model's predictions with the actual class labels of the test data.
The confusion matrix consists of four main elements:

- **True Positives (TP)**: The model correctly predicted a positive class. For example, if we are trying to identify spam emails, and the model correctly classified an email as spam, it is a True Positive.

- **True Negatives (TN)**: The model correctly predicted a negative class.
- **False Positives (FP)**: The model incorrectly predicted a positive class when it was actually negative. This is also referred to as *Type I error*.
- **False Negatives (FN)**: The model incorrectly predicted a negative class when it was actually positive. This is also referred to as *Type II error*.

Real \ Pred	Negative	Positive
Negative	TN	FP
Positive	FN	TP

Figure 3.4.7: Example of confusion matrix

The confusion matrix is an important resource for calculating various evaluation metrics such as:

- **Accuracy** is a measure of the overall percentage of correct predictions made by a classification model.

$$Accuracy = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.7}$$

- **TSS** or True Skill Statistic is a evaluation metric excellent for imbalanced datasets. In these cases, performs better than Accuracy, which might appear high despite the poor prediction of TP.

$$TSS = \frac{\text{TN}}{\text{TN} + \text{FP}} + \frac{\text{TP}}{\text{TP} + \text{FN}} - 1 \tag{3.8}$$

- **Precision** measures the percentage of positive predictions made by the model that are actually correct.

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.9}$$

- **Recall** (also known as Sensitivity) measures the percentage of actual positive cases that were correctly predicted by the model.

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.10}$$

- **F1-score** is a measure that combines precision and recall into a single value to evaluate the overall performance of the model.

$$F1\text{-}score = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} \tag{3.11}$$

The **Receiver Operating Characteristic** (ROC) curve is a fundamental tool in ML and statistics for evaluating and visualizing the performance of binary classification models. It provides valuable insights into how well a model can distinguish between two classes (binary classification) across different classification thresholds. For a more comprehensive evaluation of a model's performance across various thresholds, it is advisable to utilize a model that can provide probability scores for both classes rather than relying solely on categorical classifications. Components of the ROC Curve are:

- **True Positive Rate** (TPR) also known as *Sensitivity* or *Recall* (3.10).
- **True Negative Rate** (TNR), also known as $Specificity$, is the ratio of true negative instances to the total actual negative instances:

$$TNR = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{3.12}$$

- **False Positive Rate** (FPR) is the ratio of negative instances misclassified as positive to the total actual negative instances:

$$FPR = \frac{\text{FP}}{\text{FP + TN}} \tag{3.13}$$

- **Classification Thresholds:** ROC curves are created by varying the classification threshold (the point at which the model decides whether a sample belongs to the positive or negative class) and observing how TPR and FPR change at each threshold.

Plotting the ROC Curve consists of:

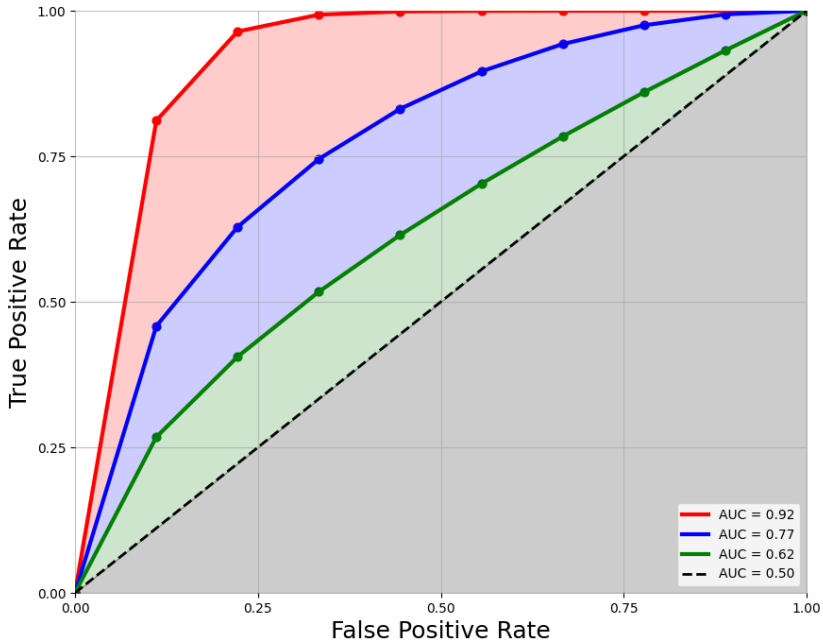
1. Start by sorting the instances by their predicted probabilities or scores.
2. Begin with a threshold that classifies all instances as positive, resulting in a point at (0, 0) on the ROC curve.
3. Gradually lower the threshold, moving along the sorted instances, and calculate TPR and FPR at each step.
4. Plot TPR on the y-axis and FPR on the x-axis, creating the ROC curve.

A random classifier (the same as flipping a coin for every prediction) would produce a diagonal line from (0,0) to (1,1), so a good model should have an ROC curve that is above this line.

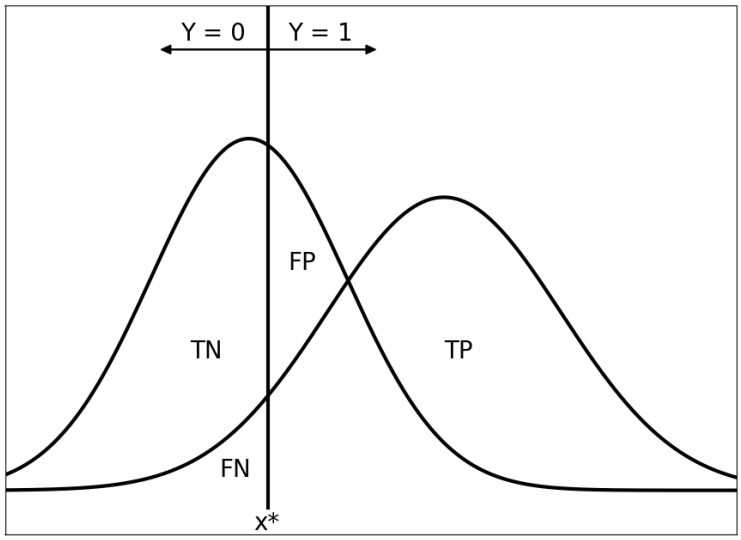
Area Under the ROC Curve (AUC-ROC): The AUC-ROC quantifies the overall performance of a binary classification model. It represents the area under the ROC curve. A perfect model has an AUC-ROC of 1, indicating it can perfectly distinguish between positive and negative cases. A random or poorly performing model has an AUC-ROC of approximately 0.5, resulting in a diagonal ROC curve. Typically, the

higher the AUC-ROC, the better the model's ability to discriminate between classes. ROC curves provide a visual depiction of a model's trade-off between *Sensitivity* and *Specificity* (1 - FPR) across different classification thresholds. In Figure 3.4.8a, it is shown an illustration of three distinct curves, each accompanied by its corresponding area value.

In binary classification, the class prediction for each instance is often made based on a continuous random variable X, which is a *score* computed for the instance (e.g. the estimated probability in logistic regression). Given a threshold parameter T, the instance is classified as *positive* if $X > T$, and *negative* otherwise. Changing the threshold $x*$ along a curve corresponds to shifting the position of the black vertical line in Figure 3.4.8b. The underlying idea is that a perfect classifier achieves a distinct separation between the two classes (0 for negative and 1 for positive), enabling an ideal thresholding operation. In the case of an imperfect classifier, there exists a tradeoff between FP and FN.



(a)



(b)

Figure 3.4.8: ROC curves, AUC and thresholds applied on the data distribution

Chapter 4

Dataset and Marker Set

A critical phase in our research is the creation of a relevant and high-quality dataset, as the quality of the dataset can significantly impact the outcomes of our research efforts. Here, we detail the methodology used to collect the dataset and its constituents. Our primary objective is to develop an automated method for detecting the initiation of full-body human movements and identifying the most effective predictive movement features. Central to this research is capturing motion dynamics accurately. The participation of these skilled dancers is crucial because their deep knowledge of body mechanics leads to precise and high-quality motion data. This collaboration yields expressive movements with distinct starting points, aligning with research goals. Their expertise increases dataset quality, providing a resource that encapsulates human motion intricacies accurately and artistically.

4.1 Raw Data Sources

The dataset contains two categories of expressive movements, each associated with a different recording session. The details of both types of movements are provided below. These segments represents various participants performing simple movement sequences,

which are captured from different angles using some cameras. The videos are synchronized through SMPTE timecode and each video is accompanied by corresponding motion capture data that records the positions of body parts. The systems used for the recording are different versions of the same software, the *Qualisys Motion Capture system*.

4.1.1 WhoLoDance Dataset

This specific portion of the dataset was collected within the framework of the H2020-ICT-2015 EU Project named *WhoLoDance*. during the month of March in 2016. These recording sessions are primarily characterized by their multimodal nature, meaning that they involve the use of multiple data recording methods. The primary focus during these sessions was on capturing expressive aspects of movement, specifically qualities related to movement dynamics. These qualities encompassed various factors, including the origin of movement, lightness, fluidity, and more. The participants in these sessions are professional dancers who prepared for their recording sessions by designing a series of exercises in advance. The recorded movements primarily revolved around contemporary dance techniques. Importantly, these movements were performed without any accompanying music to ensure that the dancers' performances were not influenced by external auditory cues.

4.1.2 Montpellier - UniGe Dataset

This specific subset of the dataset was captured during a collaborative project involving the University of Genova and the University of Montpellier in July 2020. These recording sessions are divided into two segments: individual actions and dual actions. The primary aim of these action categories is to identify the source or initiator of motion in purposeful movements.

<u>Action: Grasping</u>

From the trials that entail individual actions we took some that involves an individual

grasping a object. In this scenario, participants are positioned in a way that they are facing an object situated at a distance equal to the length of their shoulder and at an angle of 20 degrees in the outward direction from the front of their dominant shoulder. Participants are instructed to reach for the object with their dominant hand in a natural and spontaneous manner and then place it in front of their non-dominant side.

Action: Throwing

Participants are instructed to perform a two-handed throw, where in one scenario, the participant throws the ball and retrieves it themselves, while in the other scenario, two participants throw the ball to each other. This was done to introduce greater movement diversity. It is worth noting that, for the purpose of motion analysis, the receiver and the thrower are considered separately.

4.2 Original Marker Set

Due to a four-year gap between the recording of the two distinct subsets of the dataset, some differences in marker technology have emerged. In particular, the WhoLoDance recordings feature a full marker set consisting of 64 reflective markers. In contrast, the Montpellier-Unige recordings utilize the newer Qualisys sport marker set, comprising 41 markers. Nevertheless, the fundamental principles guiding the marker placement and grouping remain consistent for both datasets. The following figures (Figure 4.2.1, Figure 4.2.2) represent the body mapping of the different markers set.

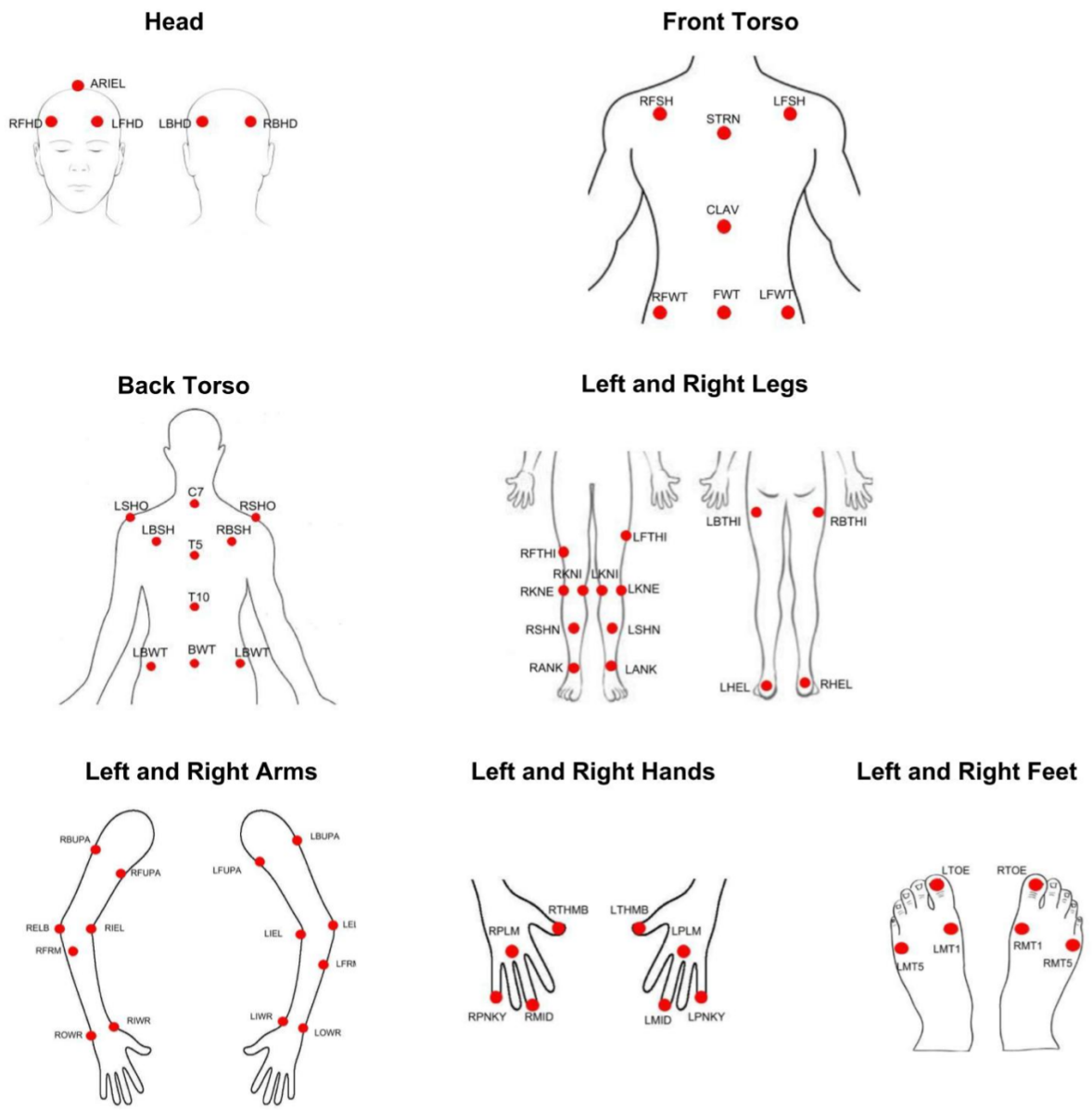


Figure 4.2.1: Markers Set with 64 Labels

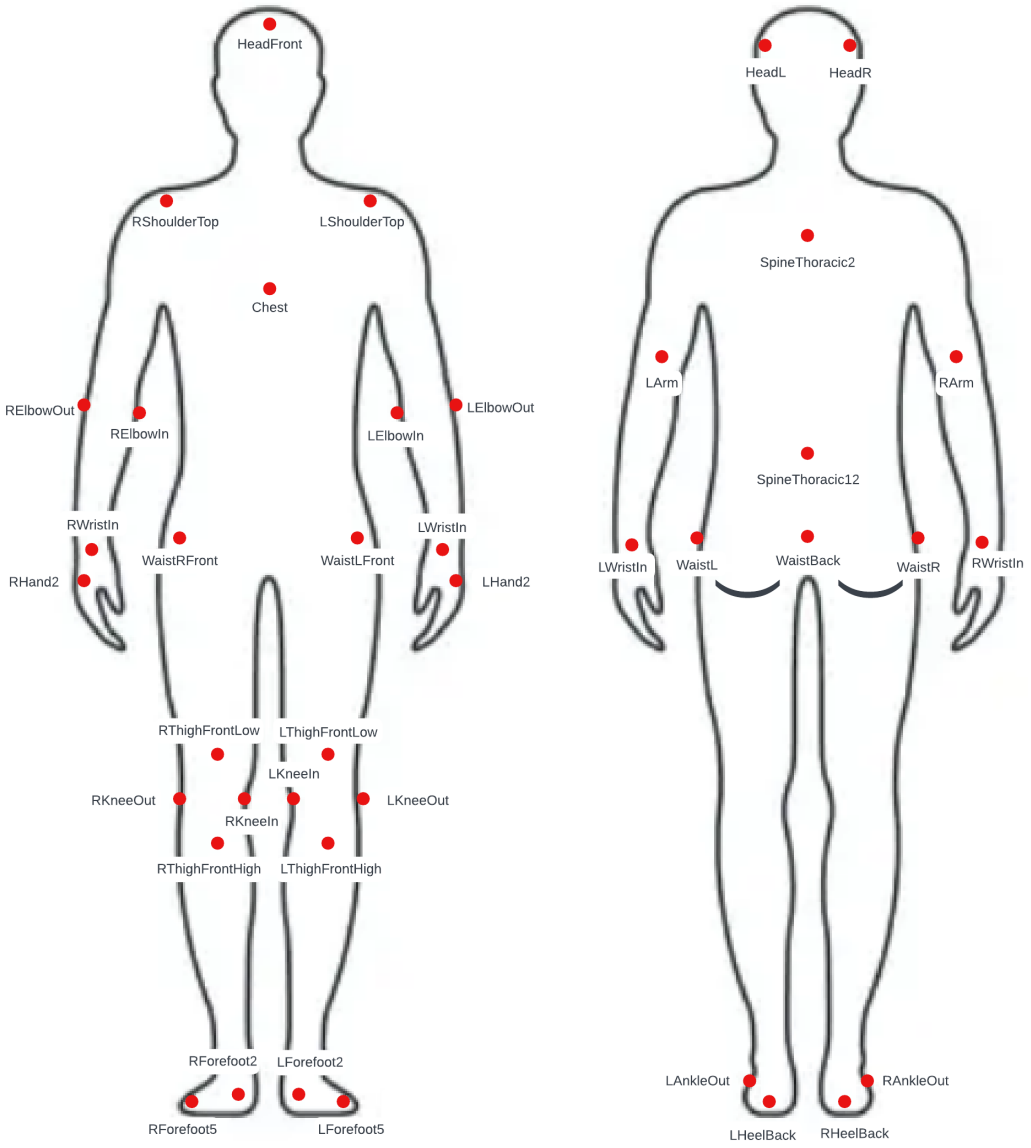


Figure 4.2.2: Markers Set with 41 Labels

4.3 Reduced Marker Set

Firstly, we standardized the marker labels following the 64-marker scheme, we renamed some of the 41-marker labels to align with the full version (Table 4.3.1).

The reduced marker set is achieved by reducing the count of markers employed from the full marker set. This because a simplified skeletal framework can adeptly communicate essential insights about expressive movements. The reduction process involved computing the *barycenter* of grouped joints, following the specified grouping scheme provided in Table 4.3.2. Initially, the coordinates (x, y, z) for each marker within the cluster were extracted. Subsequently, the averages of each cohordinate was calculated. The joints are considered all equally weighted.

41 Sport Marker Set Labels	**64 Marker Set Labels**
RKneeIn, LKneeIn	RKNI, LKNI
HeadFront, HeadL, HeadR	ARIEL, LFHD, RFHD
Chest	STRN
WaistLFront, WaistRFront	LFWT, RFWT
LShoulderTop, RShoulderTop	LSHO, RSHO
SpineThoracic2, SpineThoracic12	C7, T10
WaistBack, WaistL, WaistR	BWT, LBWT, RBWT
LKneeOut, RKneeOut	LKNE, RKNE
LHeelBack, RHeelBack	LHEL, RHEL
LAnkleOut, RAnkleOut	LANK, RANK
LForefoot2, LForefoot5, RForefoot2, RForefoot5	LTOE, LMT5, RTOE, RMT5
LHand2, RHand2	LPLM, RPLM
LWristIn, LWristOut, RWristIn, RWristOut	LIWR, LOWR, RIWR, ROWR
LElbowIn, LElbowOut, RElbowIn, RElbowOut	LIEL, LELB, RIEL, RELB

Table 4.3.1: 41 Markers renaming convention

Reduced Marker Set Labels	Partition of the 64 Marker Set Labels
right_foot	RHEL, RMT1, RMT5, RTOE
left_foot	LHEL, LMT1, LMT5, LTOE
right_ankle	RANK
left_ankle	LANK
right_knee	RKNE, RKNI
left_knee	LKNE, LKNI
right_hip	RFWT, RBWT
hip_center	RFWT, LFWT, RBWT, LBWT
left_hip	LFWT, LBWT
spine	C6, C7, T5, T10, BWT, STRN, CLAV, FWT
right_hand	RPLM, RTHMB, RINDX, RMID, RPNKY
left_hand	LPLM, LTHMB, LINDX, LMID, LPNKY
right_wrist	ROWR, RIWR
left_wrist	LOWR, LIWR
right_elbow	RELB, RIEL, RFRM
left_elbow	LELB, LIEL, LFRM
right_shoulder	RSHO
shoulder_center	LSHO, RSHO
left_shoulder	LSHO
head	RBHD, LBHD, LFHD, RFHD, ARIEL

Table 4.3.2: Mapping of the full marker set to the reduced marker set

4.4 Annotations

Starting from the recordings of movements from these sources, we selected about one hundred segments in which we could clearly identify the origin of the movement.

Each video fragment was given a label corresponding to the edge between two joints that was deemed to be the origin of the movement.

The labelling process was carried out by individually labelling half of the segments. Then we compared our labels and considered for our dataset only those on which we agreed without any a-priori knowledge on the other's opinion. We proposed those labels to our professors and considered for the dataset only those on which all of us agreed. The final selection was made merging our labels with the opinion of an expert dancer. This process was used to include only the segments where the labelling was more certain and evident.

The ending result comprises an amount of 60 fragments, divided between 58 recordings coming from the "WhoLoDance" and 2 from the "Montpellier - Unige" session. It spaced over 15 of the 20 joints available, the others are discarted during refinement process. Unfortunately, the dataset obtained ended up being unbalanced, meaning that there are some labels that are very frequent, some that are infrequent, and some labels that are completely absent. In Figure 4.4.1 we can see the resulting distribution of each edge as the origin of movement.

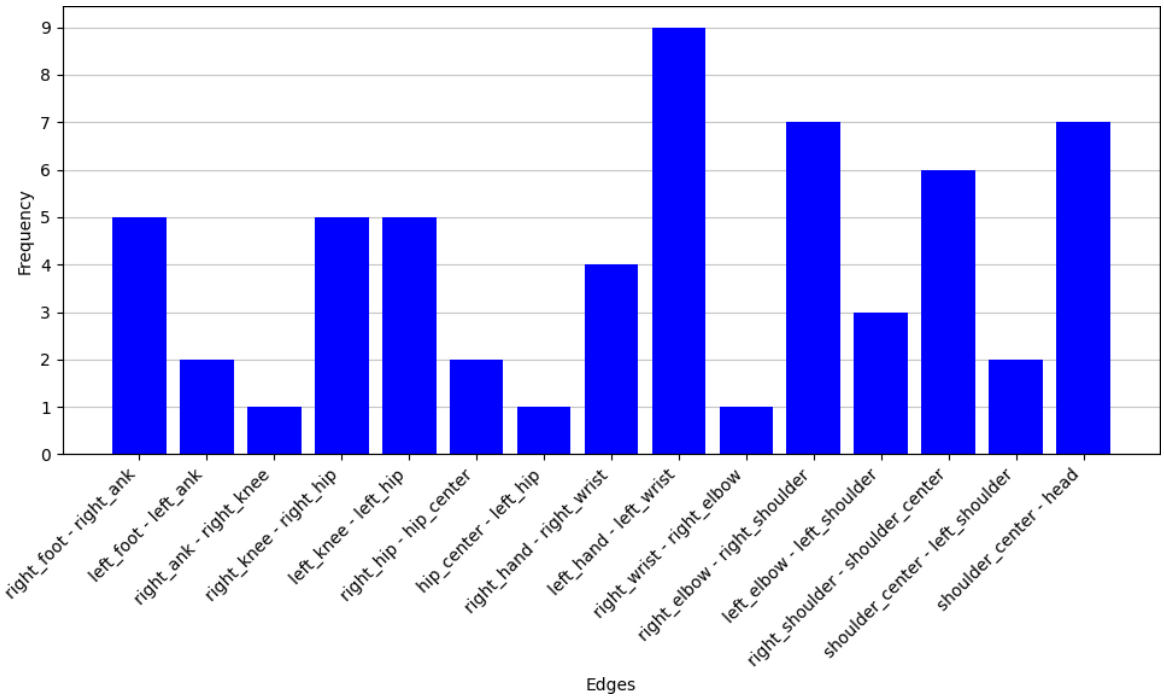


Figure 4.4.1: Edges frequencies in the Dataset

Chapter 5

Methodology

This chapter provides a more detailed explanation of the pipeline previously introduced in Chapter 2, specifically elucidating the steps leading to our three main objectives: algorithm clasification with WDG, clustering stabilization and ML classification.

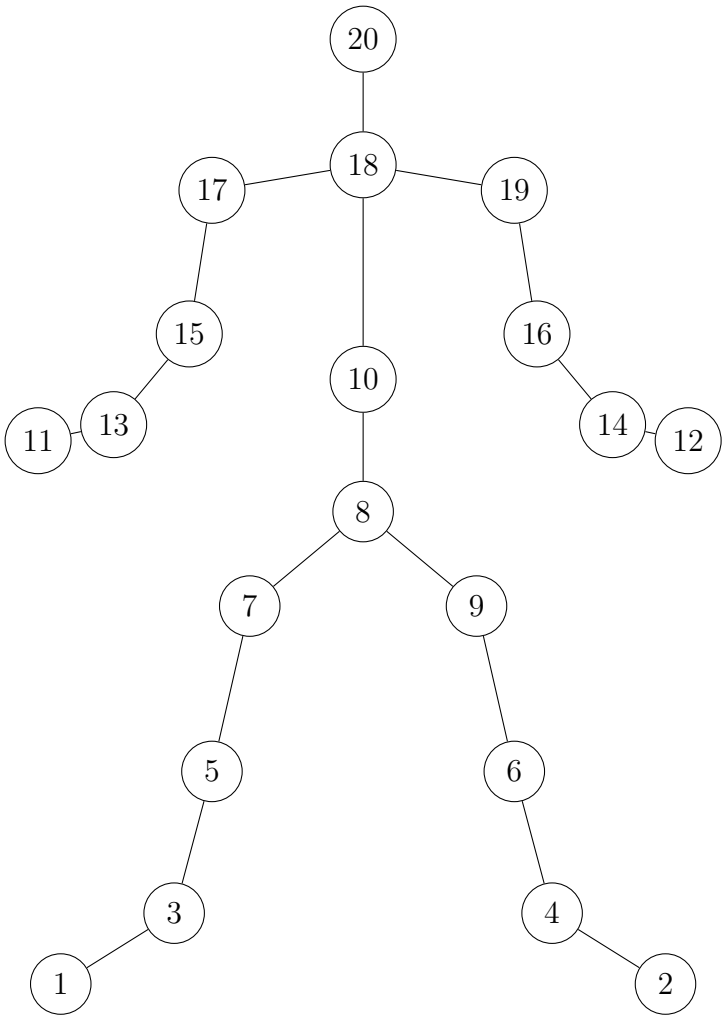
5.1 Graph-based Framework

From the *reduced marker set*, obtained after the reduction from the *full marker set* which has been discussed previously, we model the human skeleton as an undirected graph, where the vertices are the joints and the edges represent connections between consecutive physical body joints.

Each edge is associated with a specific weight, the value of which depends on a feature extracted from motion capture data.

From this *graph skeletal structure*, we define a mathematical game in which the vertices are the players and the edges model the communication channels (through which movement can propagate) between these players.

In figure we can see the *graph skeletal structure* without and a table with the joints' labels. For simplicity, the links are without weights.



Node	Label
1	right_foot
2	left_foot
3	right_ankle
4	left_ankle
5	right_knee
6	left_knee
7	right_hip
8	hip_center
9	left_hip
10	spine
11	right_hand
12	left_hand
13	right_wrist
14	left_wrist
15	right_elbow
16	left_elbow
17	right_shoulder
18	shoulder_center
19	left_shoulder
20	head

Figure 5.1.1: Skeletal structure and nodes names

5.1.1 Movement features extraction

The low-level features extracted for our purpose are *velocity*, *tangential acceleration*, and *angular momentum*.

While velocity indicates how quickly an object is moving, acceleration indicates how rapidly its velocity is changing. Angular momentum provides information about the

rotation of an object around a fixed point. By incorporating all three quantities as features, a more comprehensive and detailed view of the overall motion is obtained.

Average Velocity is calculated by finding the change in positions with respect to the delta time, which is the sampling time of the MoCap system and then applying a moving average to obtain a more accurate estimate that is less affected by noise. We calculate this velocity as the norm of the velocity vector.

$$\vec{v}(t) = \frac{\Delta \vec{r}}{\Delta t} = \left(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}, \frac{\Delta z}{\Delta t} \right) \tag{5.1}$$

The final feature consists of the magnitude of the average velocity.

Tangential Acceleration is obtained in the same way as the average velocity.

$$\vec{a}(t) = \frac{\Delta \vec{v}}{\Delta t} \tag{5.2}$$

5.1.2 Timeseries Smoothing

After computing the features, we implement two smoothing techniques on the time series data.

Firstly, we apply a *Median Filtering* algorithm centered in the middle of the window with $k_1 = 5$. On the resulting vector we apply the *Rolling Mean* again centered in the middle of the window with $k_2 = 25$.

Formally with respect to the Formula 3.2, centering the window means:

$$rolling(x_i, k) = \frac{1}{k} \sum_{j=i-(k-1)/2}^{i+(k-1)/2} x_j \tag{5.3}$$

The two final features are obtained by applying this procedure and then extracting the magnitude.

Angular Momentum has been calculated with respect to the center of mass in this way:

1. The relative position with respect to the center of mass as:

$$\vec{s}_{CM} = \frac{\sum_{j=1}^{20} \vec{s}_j}{20} \qquad \vec{s}_r = \vec{s} - \vec{s}_{CM} \tag{5.4}$$

2. The relative velocity:

$$\vec{v}_{CM} = \frac{d\vec{s}_{CM}}{dt} \qquad \vec{v}_r = \vec{v} - \vec{v}_{\mathrm{CM}} \tag{5.5}$$

3. Lastly the angular momentum relative to the center of mass:

$$\vec{L} = \vec{s}_r \times (\vec{v}_r \cdot m) \quad with\ m = 1\ for\ every\ joint \tag{5.6}$$

Analyzing angular momentum can reveal specific patterns of rotation or changes in orientation. These patterns can be used to identify distinctive gestures or movements in activities such as gymnastics or choreography.

Similarity measure To calculate the weights of the edges connecting the different joints, we utilized *Cosine similarity* as a better similarity measure compared to the one presented in the previous work, after testing its higher performance. However, since it returns a value between -1 and 1, we added 1 to shift the values and obtain positive weights. By doing this, it is possible to compare the directions of the two considered nodes, and thus evaluate if two nodes are associated with (nearly) the same direction. After obtaining the weights for each edge of the graph, we then have, for each instanta-

neous frame, a graph $G := (V, E, w)$, where V is the set of *vertices*, E is the set of *edges* and w is the set of *edgeweights*.

5.1.3 Spectral Clustering

Clusterization aims to identify groups of nodes that exhibit significant similarity based on the features or data they represent. This technique is often employed when data is represented as vectors and you want to uncover similarity relationships among these vectors.

The Shi-Malik algorithm is a clustering technique based on analyzing the eigenvalues and eigenvectors of the Laplacian matrix of a graph constructed from data.

This type of algorithm can be used to partition data into clusters based on their connectivity structure. Clusterization works to find natural divisions in the graph so that nodes within each cluster are similar to each other based on cosine similarity.

The objective is to unveil hidden structures within data so that similar objects are grouped together to aid in data understanding and analysis. The algorithm has some limitations due to the initialization phase, such as sensitivity to the initial choice of centroids and the requirement to specify the number of clusters beforehand.

This is why there will be a subsequent phase of cluster stabilization over time.

Shi-Malik Algorithm The algorithm can be summarized in these 8 steps:

1. **Calculation of Laplacian Matrix:** The Laplacian matrix is the difference between the diagonal matrix of node degrees D and the similarity matrix A:

$$L = D - A \tag{5.7}$$

2. **Calculation of Laplacian's Eigenvectors and Eigenvalues:** Eigenvectors and eigenvalues of the Laplacian matrix are calculated using the `np.linalg.eig()`

function. Eigenvectors represent new transformed data features in the eigenvector space, while eigenvalues provide information about the overall variance of data along the corresponding eigenvectors.

3. **Selection of Eigenvector to use:** Eigenvectors are ordered based on their associated eigenvalues. The eigenvector corresponding to the second-largest eigenvalue is selected (the first eigenvalue corresponds to a constant eigenvector representing scale and is ignored).

4. **Binarization of Eigenvector:** The selected eigenvector is binarized by assigning the value 1 to data points with values greater than 0 in the eigenvector and 0 otherwise.

   This process separates the data into two initial groups.

5. **Cluster Splitting:** A recursive cluster splitting algorithm, known as `split_a_cluster`, is applied to further divide clusters into smaller subclusters. The option *ncut* is selected as optimization option. Computes a weighted cut metric on the matrix of the weights of each arc for evaluating the split quality, and the optimal cutting point that minimizes the weight cut is identified. If two clusters that satisfy the *nmin* condition, the resulting two subclusters are treated separately, and the splitting algorithm is applied to each of them.

6. **Iterative Cluster Splitting:** The splitting algorithm is applied iteratively until a sufficient number of clusters are created to satisfy the desired cluster count k. At each iteration, points to be split are selected based on the weights of the current clusters. The eigenvector associated with each subcluster is computed and used for the next iteration.

7. **Final Cluster Assignment:** At the end of the algorithm, data points are assigned to clusters based on their membership in the obtained subclusters.

8. **Returning Results:** The output of the algorithm is an array of cluster assignments, where each data point is associated with a cluster number.

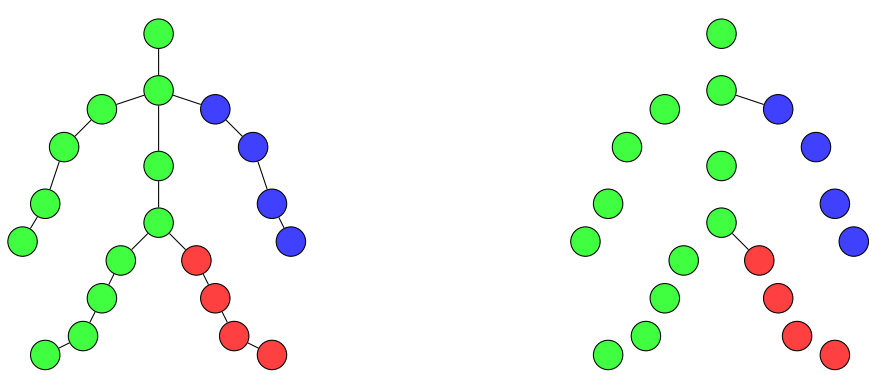
In essence, the algorithm starts by dividing data into two initial clusters, then proceeds with iteratively splitting smaller clusters until the desired number of clusters is achieved. This method leverages the spectral information of the data graph to obtain a data partition into clusters with at least 5 nodes for each one.

5.1.4 Auxiliary Graph

From the clustered graph a weighted auxiliary graph is defined for each frame, denoted as $G^{aux} := (V, E^{aux}, w^{aux})$.

The vertices of this auxiliary graph are the same as in the original graph G, but the edge set is comprised of a subset of edges that only connect vertices belonging to different clusters. The weight of every edge is calculated by applying Euclidean norm (Formula 5.9) between each pair of feature values of the nodes that are connected by an edge. One of these edges in the auxiliary graph will be the edge associated with the origin of motion.

Considering, for example, a graph G with 3 clusters (see Figure 5.1.2a), we can visualize which edges will be involved in the auxiliary graph (Figure 5.1.2b).



(a) Graph G **(b)** Auxiliary graph G^{aux}

5.1.5 Weighted Degree Centrality

To identify the edge involved in the OoM, we use the WDC, as explained in Section 3.3.3. The calculation of the WDG is done on nodes, not on edges. However, we will show how from a WDG calculated on nodes, we can derive the perceived movement origin on an edge. Starting from G^{aux}, each edge belonging to E^{aux} will have a new weight assigned to it. This weight will be divided equally between the two nodes as a fair contribution from both.

These edges will be ranked in descending order based on their weights. The top two nodes will be chosen as the origins of the movement.

For each frame, we track the two nodes with the highest WDC. Among the collected nodes, the two most frequent ones are selected across all frames.

When compared to the ground truth, it will be considered correct if one of the two nodes is at one of the two ends of the edge identified as OoM.

5.2 Visualization Framework

This section aims to enhance the coherence of cluster visualization. To achieve this goal, it is necessary to retrace the pipeline from Section 5.1 to Section 5.1.3.

5.2.1 Temporal stabilization of clusters

After clustering, when visualizing their evolution over time, we observe a lack of consistency in subsequent assignments of colors.
As mentioned before, the order in which clusters are determined at each moment is random, and the selection order of clusters is represented by a color. Therefore, from the initial random color assignment, we aim to transition to a color assignment where two clusters of the same color share as many nodes (joints) as possible over time.

This is because the random assignment of colors poses visualization issues for the graph. It is possible that from one moment to the next, two clusters with many shared joints might have different colors, resulting in a visualization over time that appears as a continuous switching of colors. This could lead to a loss of continuity in the visual representation. To stabilize clusters over time, ensuring that pairs of clusters colored from one moment to another share the highest number of common joints, we turn to the MaxWPM.

We have implemented two algorithms: a BF approach, where we compute the total weights of all possible permutations of color assignments and select the assignment with the maximum total weight, and the Hungarian Algorithm, a combinatorial optimization method that solves the assignment problem in polynomial time.

Since in our case the number of clusters is usually small, there are instances where the BF is more efficient.

Max Weight Perfect Matching Instead of the previously introduced nodes, we will have clusters, which are sets of nodes representing the joints of the skeleton, connected to each other by edges.

Consider the clusters at time t and the clusters at time $t+1$ as two distinct sets, so there will be no edges connecting clusters that belong to the same set. Each cluster in one set must be connected by an edge to exactly one cluster in the other set, with the goal of maximizing the total weight of the edges. The weight of an edge is equal to the number of nodes that the two clusters it is connecting have in common.

As previously mentioned, at this stage of the pipeline, we already have information about the joints belonging to various clusters and the colors of these clusters. However, we want to reassign colors to stabilize the visualization over time. The previous clustering groups the joints into clusters and assigns them random colors, while in this phase of the pipeline, we optimize the color assignment.

We will start by analyzing the colors of the clusters at the initial frame and choose the best color assignment for the next frame. Then will repeat this process iteratively until the last frame.

For example, we can consider a case that can we have frequently encountered in which the use of MaxWPM improved the visualization. Let's suppose we have 20 nodes representing joints at time t and 3 clusters labeled with different colors, as shown in Figure 5.2.1a. At time $t+1$, there are the same 20 nodes as before but with 3 clusters changed but labeled using the same lexicographic order for the nodes (see Figure 5.2.1b). Certainly, this clustering-based assignment cannot be regarded as the optimal assignment, as we still need to assess the $3!$ possible color assignments.

We know the colors of the clusters at time t, and we want to assign colors to the clusters at time $t+1$. In Figure 5.2.2, you can visually see all the possible color assignments, while in Table 5.2.1, all the values of common nodes for each color and the total weight of the matching are gathered.

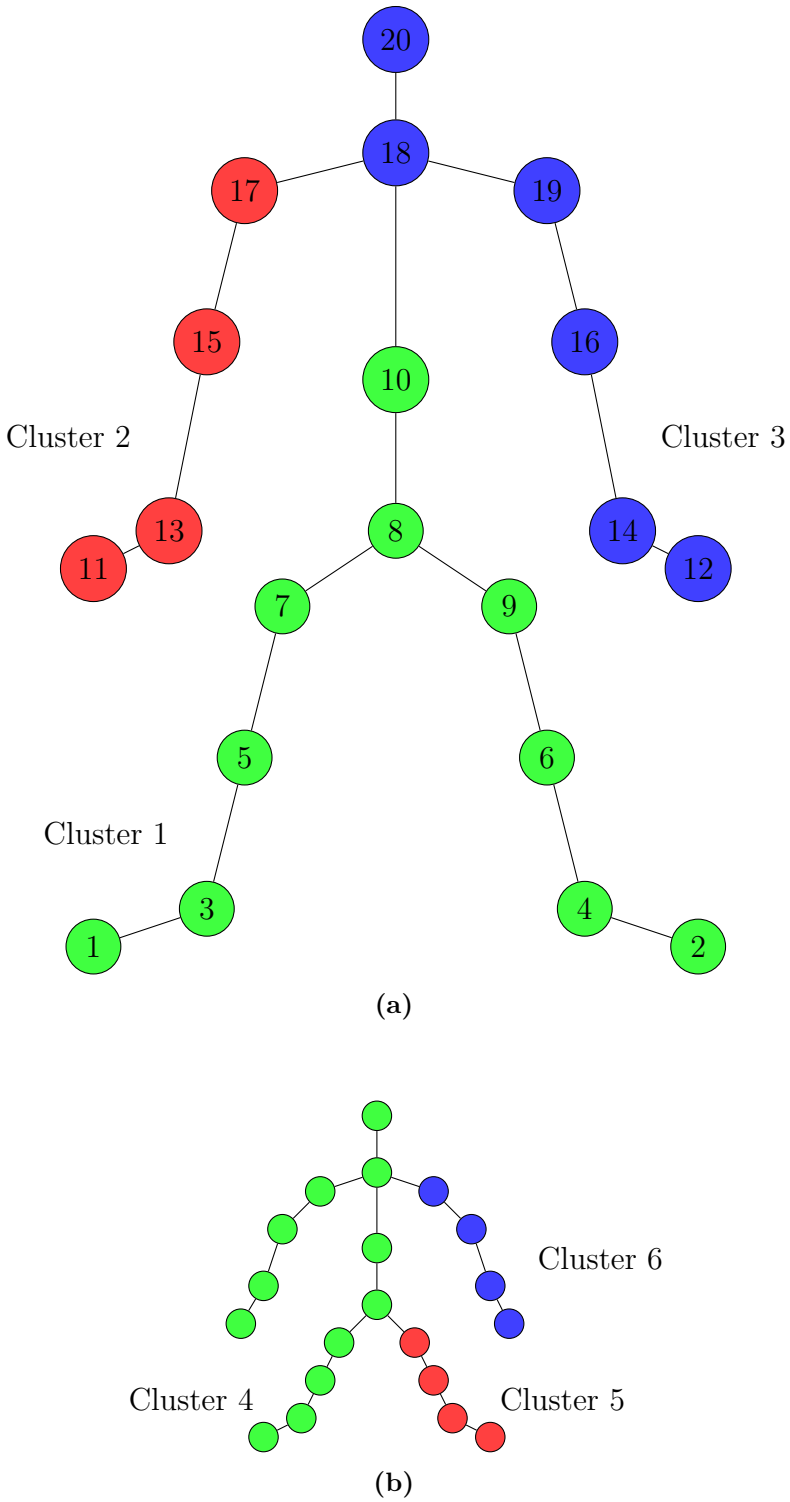


Figure 5.2.1: Color assignment at time t (a) and $t+1$ (b)

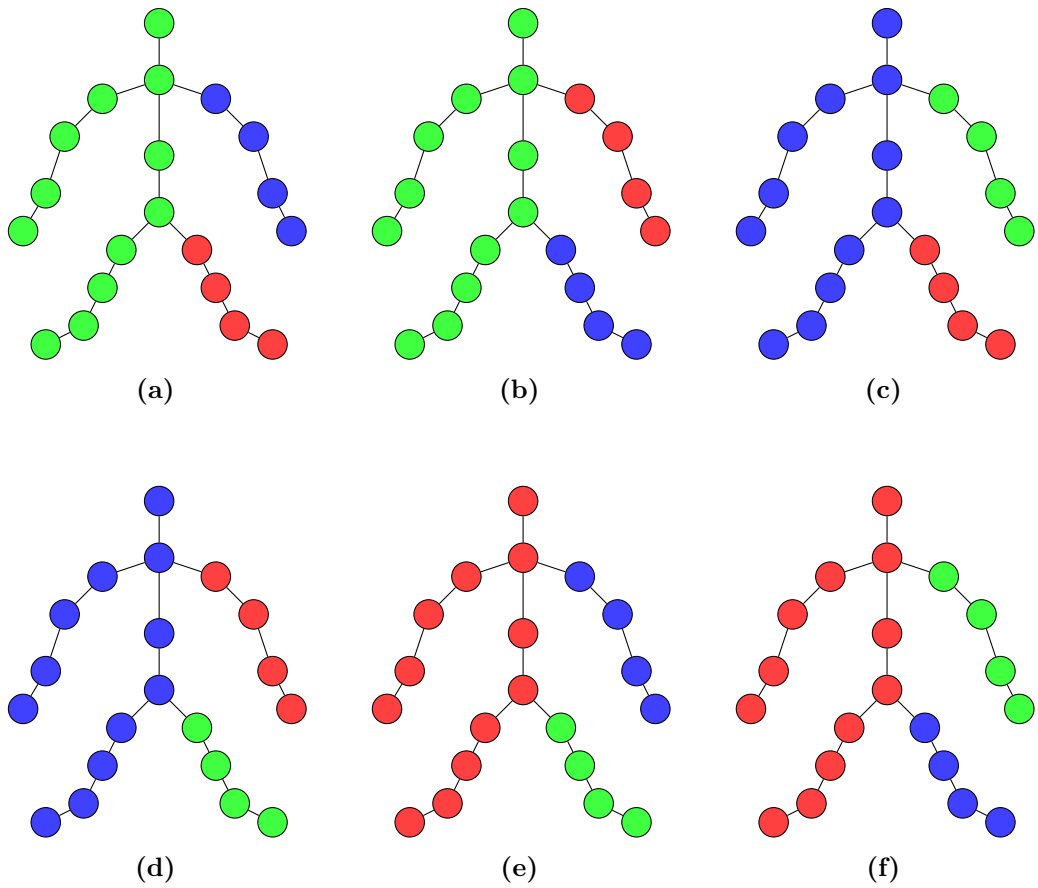


Figure 5.2.2: Possible color assignments at time *t+1*

Assignment	Red match	Blue match	Green match	Utility Value
(a)	0	4	6	10
(b)	0	0	6	6
(c)	0	2	0	2
(d)	0	2	4	6
(e)	4	4	4	12
(f)	4	0	0	4

Table 5.2.1: Matching weights for the 6 assignments

From Table 5.2.1, it can be observed that assignment (e) has the highest utility value. Additionally, even visually, it can be noticed that it is the most consistent assignment compared to the one at time t.

From the perspective of Bipartite Matching, we can consider the clusters from Figure 5.2.1a in the first partition, while in the second partition, we can consider the clusters from Figure 5.2.1b. As mentioned earlier, the weight of an edge is equivalent to the number of nodes that the two clusters being connected by that edge have in common. The following figure shows the matching with the utility value for every possible assignment and the colored ones for the optimal one.

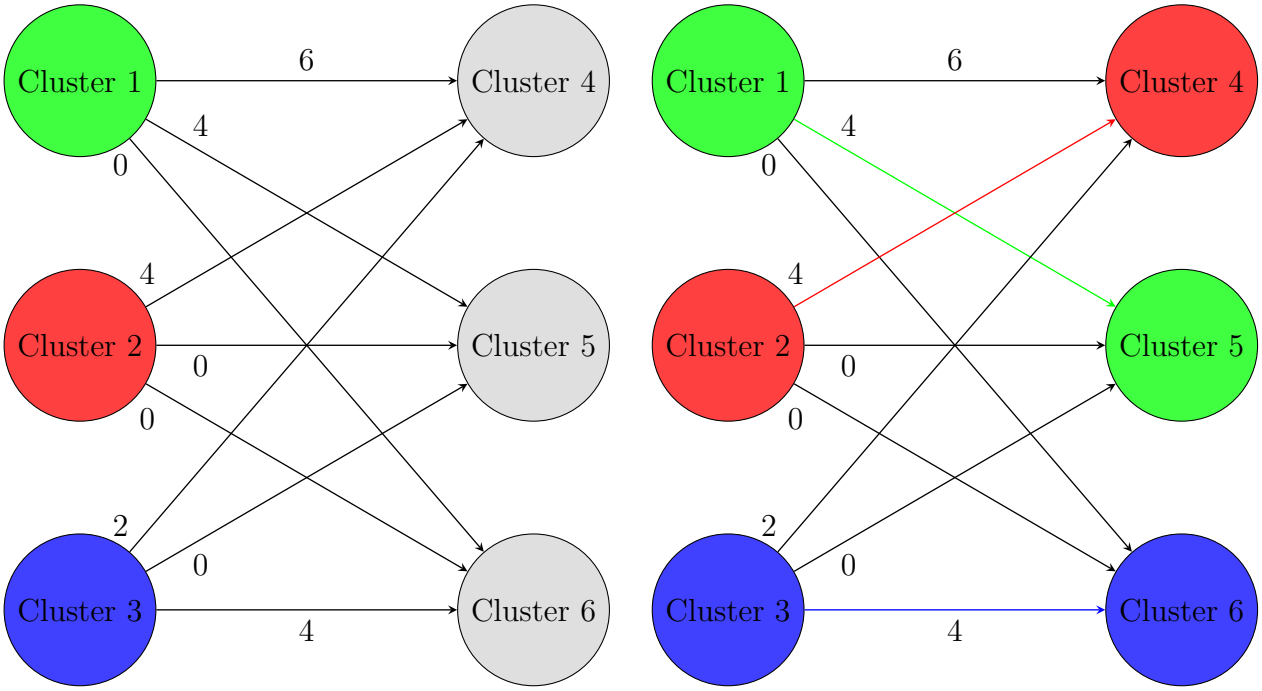


Figure 5.2.3: Bipartite Matching from instant t to instant $t+1$

Now we will discuss two algorithmns for computing the MaxWPM.

Hungarian Matching One approach consists in applying the Hungarian Algorithm which is a combinatorial optimization algorithm that also solves the assignment problem. Now let's see the previous example solved with this algorithm in the Table 5.2.2: The initial step (a) involves working with a table describing the utility values associated with each assignment. Each cell in this table represents the cost or utility of assigning a task to a resource.

In (b) we inverted each element in the table to obtain the cost value associated with each assignment. This step transforms the utility table into a cost table, turning the problem into a minimization one.

For each row in the cost table, subtract the smallest value in that row from all the elements (c).

Similarly, for each column in the table, subtract the smallest value in that column from all the elements (d).

Draw the minimum number of lines required to cover rows and columns that have zero entries (e). Since the number of lines equals the number of rows, we can already reach an optimal solution.

Starting from the top-left corner, select a zero that doesn't cover any other zeros (f). This selection initiates the assignment process.

Continue by selecting cells that cover the minimum number of zeros for each row and are not in columns that have already been covered (g).

Proceed by selecting the remaining cell that hasn't been covered by any column (h). We reached the end of the assignment process.

Finally, revert back to the original cost matrix (i).

6	4	0
4	0	0
2	0	4

(a)

-6	-4	0
-4	0	0
-2	0	-4

(b)

0	2	6
0	4	4
2	4	0

(c)

0	0	6
0	2	4
2	2	0

(d)

0	0	6
0	2	4
2	2	0

(e)

0	0	6
0	2	4
2	2	0

(f)

0	0	6
0	2	4
2	2	0

(g)

0	0	6
0	2	4
2	2	0

(h)

6	4	0
4	0	0
2	0	4

(i)

Table 5.2.2: Application of the Hungarian Algorithm to clusters stabilization

Brute Force Another approach consists in a brute force method for choosing an option based on the minimum sum of utility values of a graph involves exhaustively evaluating every possible combination of options to find the one with the highest total weight. This method consists of enumerating all possible subsets or combinations of options within the graph, computing theier sum of weights, ranking results and choosing the best one. It can be also used to validate the first one. The Table 5.2.3 is a clear example of this approach.

-6	-4	0
-4	0	0
-2	0	-4

(a)

-6	-4	0
-4	0	0
-2	0	-4

(b)

-6	-4	0
-4	0	0
-2	0	-4

(c)

-6	-4	0
-4	0	0
-2	0	-4

(d)

-6	-4	0
-4	0	0
-2	0	-4

(e)

-6	-4	0
-4	0	0
-2	0	-4

(f)

Table 5.2.3: Real application of the BF Algorithm

BF approach guarantees finding the best option, it can be computationally expensive, especially for large graphs. As we can see, here we have the computation of all the *n!* permutations and the minimum one is (e).

5.2.2 Clustering Transition Smoothing

After achieving cluster stabilization at the interframe level, a persistent issue remains. Despite the smoothness of actual movements, the transition of a cluster from a part of the body is still too fast. This is due to the fact that is a greedy approach and is not considering the temporal progression of clusters. To address this issue a delay and smoothing procedure has been used. It consists in computing the difference between the sets representing clusters at time t+1 and clusters at time t. Apply the weighted degree centrality to the nodes within each cluster in the resulting set, and select the maximum value. For each cluster, update the label of the node with the highest weighted degree centrality. This algorithm is applied only every $k = \frac{1}{4} framerate$ time instant, effectively updating the clusters, in the other time instants these are not updated.

By parameterizing k this process appropriately to ensure that clusters do not undergo

excessive changes during this time interval, an oscillating effect is achieved. This effect propagates the cluster across the connected nodes that belong to it, thereby making any potential origin of the movement more conspicuous.

5.3 Machine Learning Framework

In this section, we describe the process of extracting and engineering features from the raw data, the process to refine the model in order to obtain highly accurate model, and the different formulations of binary questions posed to it.

5.3.1 Features engineering

The initial dataset we started from, as described in Section 4.4, consists of 60 time series of the 20 joints trajectories. In order to avoid noise or presence of more origin of movements, we trimmed the videos finely, resulting in clips of variable lengths ($\mu = 1.96s, \sigma^2 = 1.49s$).

Frames sampling To address the variable length of segments within the dataset, in instances where we aim to attain a uniform number of frames than the actual length of the segments, we have utilized the *uniform spacing sampling* technique to address the variability in segment lengths within the dataset. This method entails selecting frames from a list or sequence at regular, even intervals, by fixing a number of frames that we wanted, in our case 200, and then keeping only those frames equally spaced between eachother.

Conversely, when dealing with segments that are shorter than the required number of frames, a different technique is implemented. In such cases, for each frame, a variable and uniform number of interpolation frames are inserted to reach the desired frame count. This approach compensates for the shorter segment length by adding approximately the same amount of interpolated frames across the whole segment, maintaining the consistency required for the following statistical analysis.

Dataset Normalization From the timeseries produced by the sampling process we calculate the x, y, and z coordinates of the skeleton's barycenter for each time step. We define the barycenter as the point where all the mass of an object or a system of objects is concentrated.

In this context, each joint is considered to have unit mass, meaning that all joints contribute equally to the barycenter.

If we represent n as the number of 20 joints, the coordinates of the barycenter are obtained by taking the average of the coordinates of these 20 joints, as we can see in the formula:

$$Barycenter(x, y, z) = \left(\frac{1}{n} \sum_{i=1}^{n} x_i, \frac{1}{n} \sum_{i=1}^{n} y_i, \frac{1}{n} \sum_{i=1}^{n} z_i \right) \tag{5.8}$$

where x_i, y_i, and z_i represent the x, y, and z coordinates of the i-th joint, respectively.

Subsequently, we calculate the distance between each joint and the barycenter, at each time step, using the Euclidean distance in Formula 3.4.

These distances from the barycenter will be normalized for each joint, in order to obtain a time series whose values space between 0 and 1.

$$x_{norm} = \frac{x - min(x)}{max(x) - min(x)} \tag{5.9}$$

The choice to calculate the distances between joints and the barycenter for each sample plays a crucial role in ensuring the robustness of our ML approach to variations in scale. In scenarios where dancers or subjects may have different heights or body proportions, relying solely on absolute joint coordinates could introduce bias into the model. However, by computing these distances and further normalizing them within the range of 0 to 1, we effectively eliminate the influence of scale variations.

Features extraction This process involves transforming raw data into a more suitable format for analysis and model building. Here are all the functions calculated for every sample of the dataset (note that n is the number of frames of the sample).

Mean is the expected value of the distribution:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5.10}$$

Variance is the second momentum of the distribution (or the squared deviation of the expected value from the mean):

$$\sigma^2 = \mu_2 \qquad \mu_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^k \tag{5.11}$$

Kurtosis is the fourth standardized moment:

$$Kurt[X] = \frac{\mu_4}{\sigma^4} - 3 \tag{5.12}$$

Skewness is the third standardized moment:

$$\gamma_1 = \frac{\mu_3}{\sigma^3} \tag{5.13}$$

Mean Standard Error:

$$\sigma_{\bar{x}} = \sqrt{\frac{\sigma^2}{n}} \tag{5.14}$$

Correlation between two joints (x,y):

$$\rho_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \mu_X)(y_i - \mu_Y)}{\sigma_X \cdot \sigma_Y} \tag{5.15}$$

Mean Absolute Deviation:

$$mad(\bar{x}) = \frac{1}{n}\sum_{i=1}^{n}|x_i - \mu| \tag{5.16}$$

Average Sum of Squares:

$$energy(\bar{x}) = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{5.17}$$

Interquantile Range measures statistical dispersion in the range between the 25th (Q1) and the 75th percentile (Q3) of the data distribution ($iqr = Q3 - Q1$):

$$iqr(\bar{x}) = \begin{cases} \bar{x}_{\frac{3}{4}(n+1)} - \bar{x}_{(\frac{n+1}{4})} & \text{if n is odd} \\ \frac{(\bar{x}_{(\frac{3}{4}n)} + \bar{x}_{(\frac{3}{4}n+1)}) - (\bar{x}_{(\frac{n}{4})} + \bar{x}_{(\frac{n}{4}+1)})}{2} & \text{otherwise} \end{cases} \tag{5.18}$$

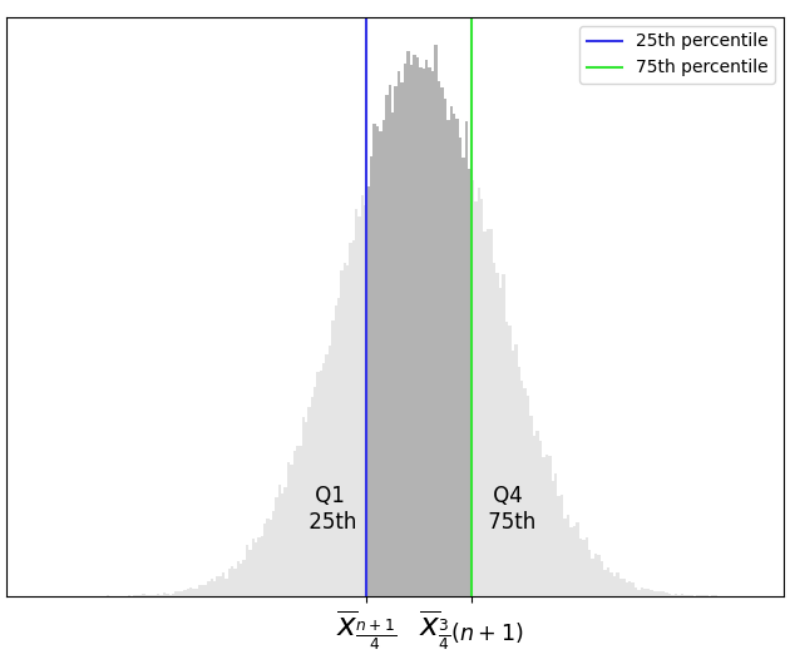


Figure 5.3.1: Interquantile range of a normal distribution

Fundamental Frequency Signal from the Discrete Fourier Transform (DFT):

$$firstFreq(\bar{x}) = \angle DFT(\bar{x})[1] \tag{5.19}$$

Real part's Average of Frequency Signal:

$$meanFreq(\bar{x}) = \sum_{i=1}^{n} Re(DFT(x_i)) \tag{5.20}$$

Real part's Max of Frequency Signal:

$$maxFreq(\bar{x}) = Max(Re(DFT(x))) \tag{5.21}$$

Index of the Max of the Real part of Frequency Signal:

$$maxFreq(\bar{x}) = Argmax(Re(DFT(x))) \tag{5.22}$$

The last four values correspond to the minimum value and its index, skewness, and kurtosis of the Fourier transform. To choose the features to extract, we referred to [2] and [24]. In Table 5.3.1 we can see all the features and their relative function.

Function	Description
mean	Mean
var	Variance
kurt	Kurtosis
skew	Skewness
corr	Correlation Coefficient
mad	Mean Absolute Deviation
sem	Mean Standard Error
energy	Average Sum of Squares
iqr	Interquantile Range
min	Smallest Value in Array
max	Largest Value in Array
firstFreq	Fundamental Frequency Signal
meanFreq	Real part's Average of Frequency Signal
maxFreq	Real part's Max of Frequency Signal
maxFreqIndx	Index of the Max of the Real part of Frequency Signal
minFreq	Real part's Min of Frequency Signal
minFreqIndx	Index of the Min of the Real part of Frequency Signal
skewFreq	Frequency Signal Skewness
kurtFreq	Frequency Signal Kurtosis

Table 5.3.1: List of measures for computing feature vectors

Here is the operative Python code for the extraction of the features from the raw data after the normalization.

```
import numpy
import scipy.stats

def extractFeatures(raw_feat):
  mean = numpy.mean(raw_feat, axis=0)
  var = numpy.var(raw_feat, axis=0)
  kurt = scipy.stats.kurtosis(raw_feat, axis=0, bias=True)
  skew = scipy.statsskew(raw_feat, axis=0, bias=True)
  corr = numpy.corrcoef(raw_feat, rowvar=False)[numpy.triu_indices(raw_feat,k=1)]
  mad = numpy.mean(numpy.abs(raw_feat - mean), axis=0)
  sem = numpy.std(raw_feat, axis=0) / numpy.sqrt(raw_feat.shape[0])
  energy = numpy.sqrt(numpy.sum(raw_feat ** 2, axis=0))
  iqr = scipy.stats.iqr(raw_feat,axis=0)
  mi = numpy.min(raw_feat, axis=0)
  ma = numpy.max(raw_feat, axis=0)

  fft = numpy.fft.fft(raw_feat, axis=0)
  amplitude_spectrum = numpy.abs(fft)
  frq_info = [numpy.angle(fft)[0, :], # First freq
              numpy.mean(fft.real, axis=0),
              numpy.max(fft.real, axis=0),
              numpy.argmax(fft.real, axis=0),
              numpy.min(fft.real, axis=0),
              numpy.argmin(fft.real, axis=0),
              scipy.stats.skew(amplitude_spectrum, axis=0, bias=True),
              scipy.stats.kurtosis(amplitude_spectrum, axis=0, bias=True) ]

  refined_features = numpy.hstack([mean,var,kurt,skew,corr,mad,sem,energy,iqr,mi,ma,
      numpy.hstack(frq_info)])

  return refined_features
```

5.3.2 Classification

The initial approach involves a simple binary classification of the origin of the movement in a specific edge or not. If the results had been satisfactory, we would then have opted to evaluate progressively more complex models, eventually leading to the multiclass classification. We started by addressing the issue of imbalanced data in classification.
The choice of LOOCV was motivated by the presence of a very small dataset, allowing us to compute the maximum number of training cycles to have an almost unbiased model.
To mitigate this imbalance, a resampling technique known as B-SMOTE is employed (*BS1*). This technique equalizes the number of training samples for the minority class with that of the majority class at each iteration.
B-SMOTE generates additional samples for the minority class to balance the number of samples of the majority class. For instance, if there are 50 samples of one class and 9 of the other during a training step, B-SMOTE will artificially create 41 samples for the second class that are more at risk of being misclassified because they are isolated from the rest of the samples of the same class.
The operational concept is detailed in Section 3.4.3.

Following the resampling of the training data, a RF classifier (*RF1*) is trained on this newly balanced dataset. To further optimize the classifier's performance, a feature selection process is employed with the goal of identifying the most influential features that contribute to accurate classification. The most important features are determined through the training of *RF1* on the resampled data. These features are selected based on the amount of information that they provide in the classification task weighting Gini importance of the feature across all the trees.
Subsequently, a new RF classifier (*RF2*) is trained using only the 50 most important features of the previous model, with the *max_features* hyperparameter tuned to choose among all the available features in each split.

To evaluate the performance of the classifier, a test set is chosen using LOOCV, as detailed in Section 3.4.5.

Model	**Hyperparameter**	**Value**
BS1	*k-neighbors*	$0.4 \cdot$ count(Less Frequent Class)
	m-neighbors	$0.4 \cdot$ count(Most Frequent Class)
RF1	*n_estimators*	500
	max_features	Default
RF2	*n_estimators*	200
	max_features	All

Table 5.3.2: Hyperparameters tuned for our application

Below is the python code used to train and test the RF with the aforementioned parameters. For the BSMOTE implementation we used the `imblearn` library.

The `imbalanced-learn` library, commonly abbreviated as `imblearn`, is a powerful Python library designed for handling imbalanced datasets commonly encountered in machine learning problems. It provides all the main techniques to address the issue of imbalanced class distributions, such as ADASYN, SMOTE, SMOTE-ENN and more.

Instead, for the RF implementation, we used the classical `sklearn`, which is an abbreviation for the `scikit-learn` library.

It is an open-source Python library that provides efficient tools for machine learning and data analysis. It includes various models for classification, regression, clustering, and feature extraction.

Some of the most common models within scikit-learn are Support Vector Machines, decision trees, RF, linear and logistic regression, k-Nearest Neighbors, among others. The library also offers functionalities for model validation, dimensionality reduction, and feature selection.

```python
from imblearn.over_sampling import BorderlineSMOTE
from sklearn.ensemble import RandomForestClassifier

# Define a function to perform the model training and prediction for one iteration
def train_predict(X, y, i):

    # Remove i-th sample from training
    X_tr = numpy.delete(X, i, axis=0)
    y_tr = numpy.delete(y, i, axis=0)

    # Count labels occurencies
    labels_count = numpy.bincount(y_tr)
    ratio = 0.4

    # Defines the neighborhood of samples to use to generate the synthetic samples.
    k_neigh = numpy.clip(int(labels_count[1] * ratio), 1, len(y))

    # Determine if a minority sample is in "danger"
    m_neigh = numpy.clip(int(labels_count[0] * ratio), 1, len(y))

    # Fit the model with different hyperparameters using BorderlineSMOTE
    bsmote = BorderlineSMOTE(k_neigh, m_neigh)
    X_tr_resampled, y_tr_resampled = bsmote.fit_resample(X_tr, y_tr)

    # Train a first forest
    rf = RandomForestClassifier(n_estimators=500)
    rf.fit(X_tr_resampled,y_tr_resampled)

    # Sort importances in descending order and select the top 50 features
    top_features = numpy.argsort(numpy.array(rf.feature_importances_))[::-1][:50]

    # Select the most important features on train and test set
    X_tr_star_resampled = X_tr_resampled[:, top_features]
    X_t_star_resampled = X[i, top_features].reshape(1, -1)

    # Train a new forest
    rf = RandomForestClassifier(n_estimators=200, max_features=None)
    rf.fit(X_tr_star_resampled, y_tr_resampled)

    # Return prediction on the test instance
    return rf.predict(X_t_star_resampled)[0]

# LOOCV on a dataset
X = ... ; y = ...
predicted_labels = [train_predict(X, y, i) for i in range(len(y))]
```

In our ML model, we posed three different questions regarding the OoM. Since our dataset is inherently a multi-class dataset, we performed binary classification by reclassifying the dataset based on our necessities. These binary questions are posed either to edges or to specific parts of the skeleton, which are themselves composed of edges.

In Figures 5.3.3a and 5.3.3b, you can see the edges composing different segments of the skeleton.

Note that not every edge of the reduced marker set (5.1.1) is included because some edges have not been the ground truth for any sample in our dataset. For ML we have in total 15 classes, that is 15 edges. After each reclassification the same pipeline of training and testing is used.

Body Part	Edges
Head	shoulder_center - head
Right Arm	right_hand - right_wrist right_wrist - right_elbow right_elbow - right_shoulder right_shoulder - shoulder_center
Left Arm	left_hand - left_wrist left_elbow - left_shoulder left_shoulder - shoulder_center
Right Leg	right_foot - right_ankle right_ankle - right_knee right_knee - right_hip right_hip - hip_center
Left Leg	left_foot - left_ankle left_knee - left_hip left_hip - hip_center

(a)

Body Part	Edges
Upper	right_hand - right_wrist right_wrist - right_elbow right_elbow - right_shoulder right_shoulder - shoulder_center left_hand - left_wrist left_elbow - left_shoulder left_shoulder - shoulder_center shoulder_center - head
Lower	right_foot - right_ankle right_ankle - right_knee right_knee - right_hip right_hip - hip_center left_foot - left_ankle left_knee - left_hip left_hip - hip_center

(b)

Table 5.3.3: Skeleton Division in 5 parts (a) and in 2 parts (b)

Q1 will be posed considering the 15 edges, Q2 will be posed considering the division of the body into 5 parts (Figure 5.3.3a), while Q3 will be based on the division of the body into 2 parts (Figure 5.3.3b).

Q1: Is or is not a specific edge In our dataset, the most frequent classification is the edge that links *left_hand* and *left_wrist*. Therefore, the dataset will be relabelled in a way that where the classification is *left_hand-left_wrist*, that sample will belong to class 1, otherwise to class 0.

Q2: Is or is not a specific body part The most frequent class to compare against all the others is *Right Leg*. Therefore, the relabelling process will rename *Right Leg* as 1, and all other samples as 0.

Q3: Is or is not a specific body part The most frequent class between the two is *Upper*, so we labeled all the *Upper* with 1 and all the *Lower* with 0.

For each question, the model will be iteratively asked whether the test sample should be predicted as that edge/that part of the body or not.
After cycling through all the samples and comparing the predictions with the true labels, a confusion matrix will be constructed to encapsulate all possible outcomes.
Finally, metrics will be calculated based on this matrix.
This process will be repeated multiple times to ensure result consistency.

Chapter 6

Results and Discussion

In this section, we present the attempts made to achieve the best outcome along with the respective intermediate results obtained at various steps. We discuss the outcomes of the best approach, as described in Chapter 5, and the enhanced visualization of skeletal clusters. The originally planned work underwent modifications due to constraints posed by a limited and imbalanced dataset. Additionally, challenges were encountered in sourcing additional complete and consistent samples to augment the dataset. Despite achieving excellent results, the major limitation stemmed from the considerably small dataset. Its limited size constrained the analysis to a binary classification, preventing the exploration of a multiclass classification approach.
Consequently, the outcome was the ability to predict whether an origin was at a certain point or not within the realm of machine learning. Due to the unfeasibility in machine learning, a decision was made not to pursue this approach in the algorithmic realm to maintain comparability in results.

Subsequent sections offer a summary of the accomplished results and potential implications. The focus lies on performance analysis concerning operational conditions and proposes future developments aimed at enhancing the obtained results.

6.1 Graph-based Framework

The OoM was classified using three distinct features: velocity, acceleration and angular momentum, as described in Section 5.1.1. For each feature, we applied two different similarity functions, as part of the pipeline explained in Chapter 5.1. The two similarity functions are the *Cosine Similarity* (concept explained in Section 3.1) and a custom function described in Section 1.2.

In Table 6.1.1 we can see that the Cosine similarity outperforms the custom-made function. Therefore, it was chosen as the similarity function for this specific dataset.

Similarity Function	**Velocity**	**Acceleration**	**Angular Momentum**
Cosine	28.3%	26.7%	36.6%
Custom	18%	21.1%	34.2%

Table 6.1.1: Classification accuracy over the whole dataset obtained by using the two different similarity functions

From here on, the graphs will reflect the results obtained using cosine similarity as the measurement method. While these results may not be exceptional, they still outperform a random approach, in which two random nodes are selected and compared to the ground truth. Infact such an approach, achieves an accuracy of 19.4%,

The formula to calculate the probability of picking at least one joint of the two by extracting two joints between the 20 available is shown in Equation 6.1, where X is the random variable representing the number of joints in the OoM, so X takes values in {0,1,2}.

$$\mathbb{P}(X \geq 1) = \mathbb{P}(X = 1) + \mathbb{P}(X = 2) = \frac{\binom{2}{1}\binom{18}{1}}{\binom{20}{2}} + \frac{\binom{2}{2}\binom{18}{0}}{\binom{20}{2}} = \frac{37}{190} \approx 19.4\% \tag{6.1}$$

In Figure 6.1.1 we can see the distribution of the results for each edge of the dataset

with respecto to their ground truth. The WDC method seems to perform reasonably well in classifying sources of movement in the upper part of the body, where there are generally a greater number of edges per node. But performs generally bad on lower body and the most borderline nodes (those connected to the rest of the graph by only one arc).

This might suggest that by varying the number of clusters to allow for the creation of smaller clusters, perhaps using a hierarchical clustering approach, it could be possible to isolate multiple splits of the same body part.

However, it is essential to take these results with caution because the dataset is very small and certainly does not have a statistically significant sample of each class. Nevertheless, it remains a good starting point.

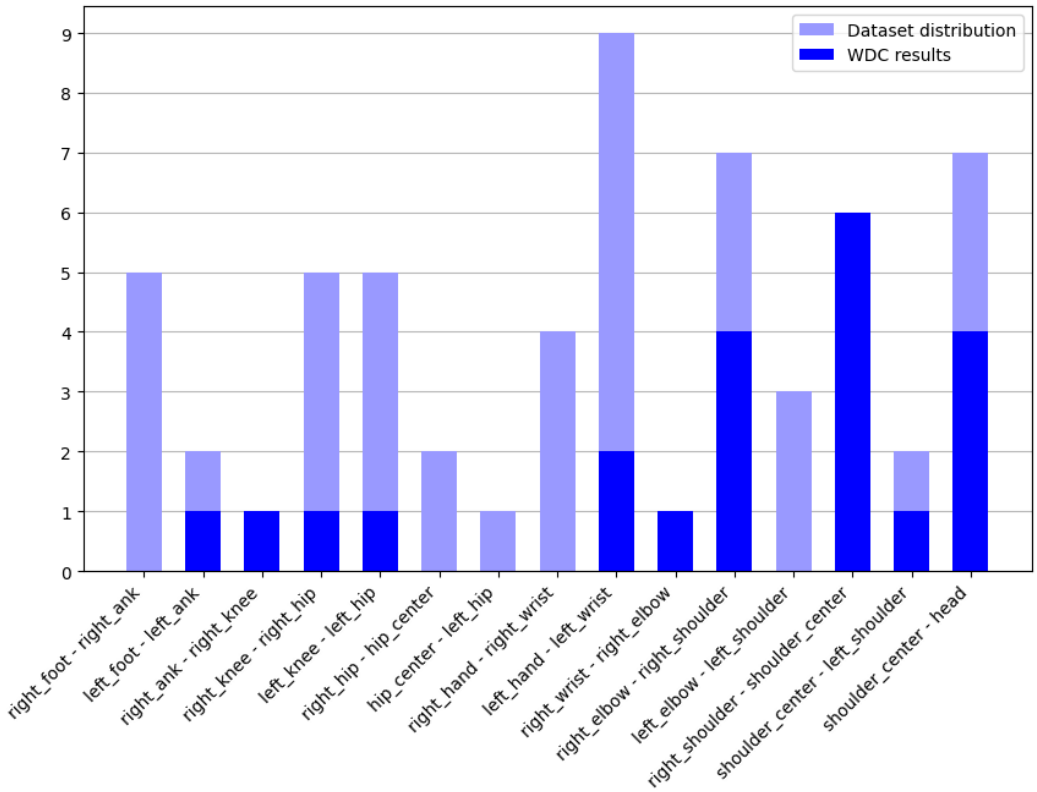


Figure 6.1.1: Joints and frequency of the dataset and the correctly classified graph-based algorithm applied on the angular momentum feature

In the first Figure 6.1.2, by computing the WDC for each sample in the dataset, it is evident that the joints with the highest WDC values are predominantly situated in the upper part of the body. Particularly noteworthy is the observation that the right shoulder appears to be the most frequent. This anomaly might serve as an indicator of bias within the algorithm.

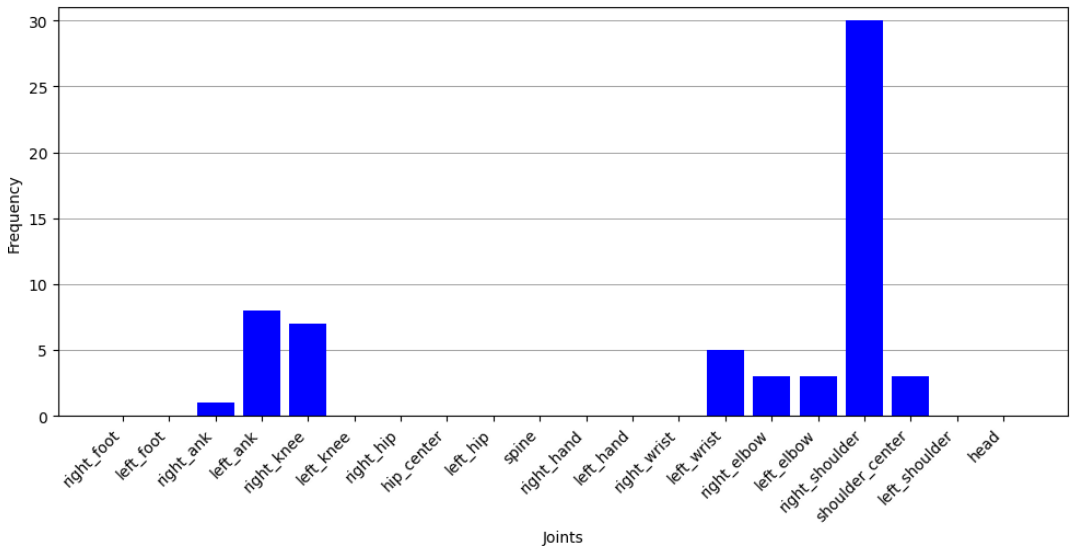


Figure 6.1.2: Distribution of the nodes for first max of the WDC

In the second figure, the frequency distribution of joints with the second-highest WDC value is presented. The joints appear to be more evenly distributed compared to the previous case. In both scenarios, it is evident that the extremities, such as hands, feet, and head, never emerge as the joints with the highest WDC. This observation is likely attributable to certain limitations in the cluster creation process, resulting in clusters with a node count that is certainly greater than 1.

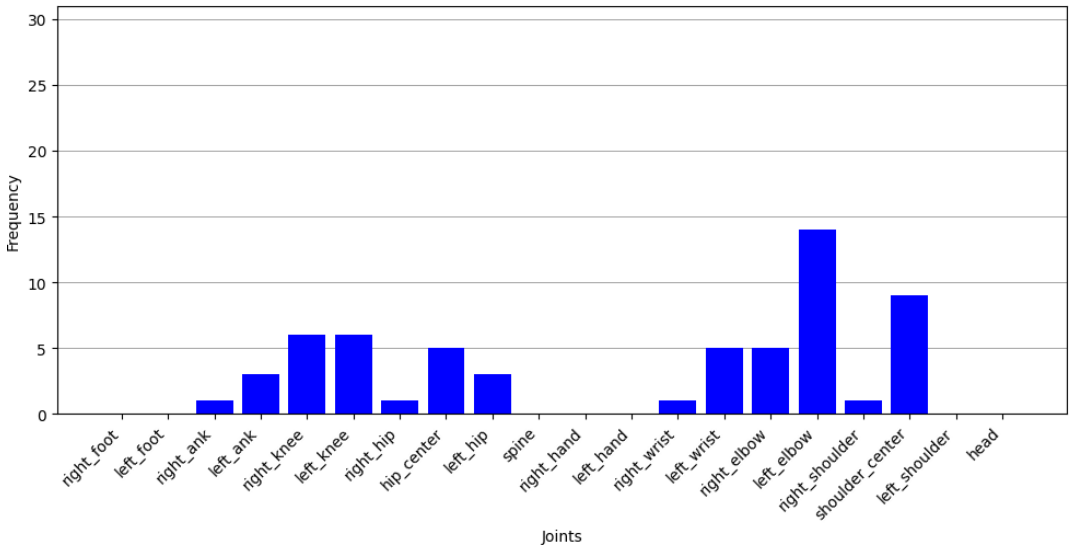


Figure 6.1.3: Distribution of the nodes for the second max WDC

In the last image, the pairs of adjacent nodes for each sample related to the edge classified as the OoM are depicted. Overall, it appears that the combination of the first and second maximums better fits the dataset distribution, even though it struggles to identify peripheral nodes as OoM. It's worth noting that the spine, as structured in the skeleton, would hardly be the origin of movement, and in our experiments, we never managed to recognize it.

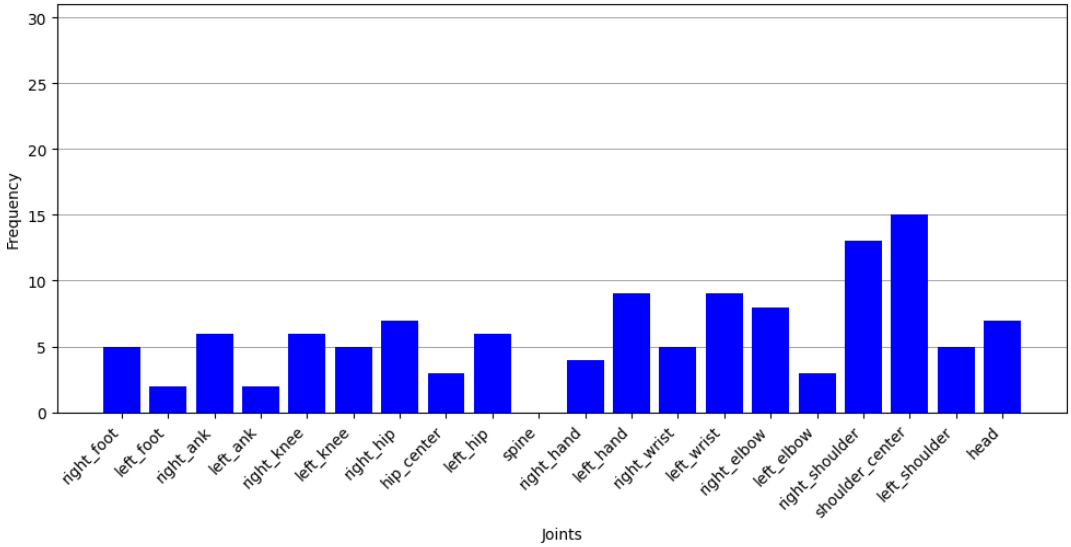


Figure 6.1.4: Distribution of the adjacent nodes of the edges' OoM in the dataset

6.2 Clusters Stabilization

In this first example, in Figure 6.2.1, we can observe the results of stabilization of three clusters with the same color scheme applied on two different time instants. In (a), the preceding time instant is visible, while (b) showcases the subsequent time instant in an unstabilized state. In (c) instead, the subsequent time instant with clusters stabilized.

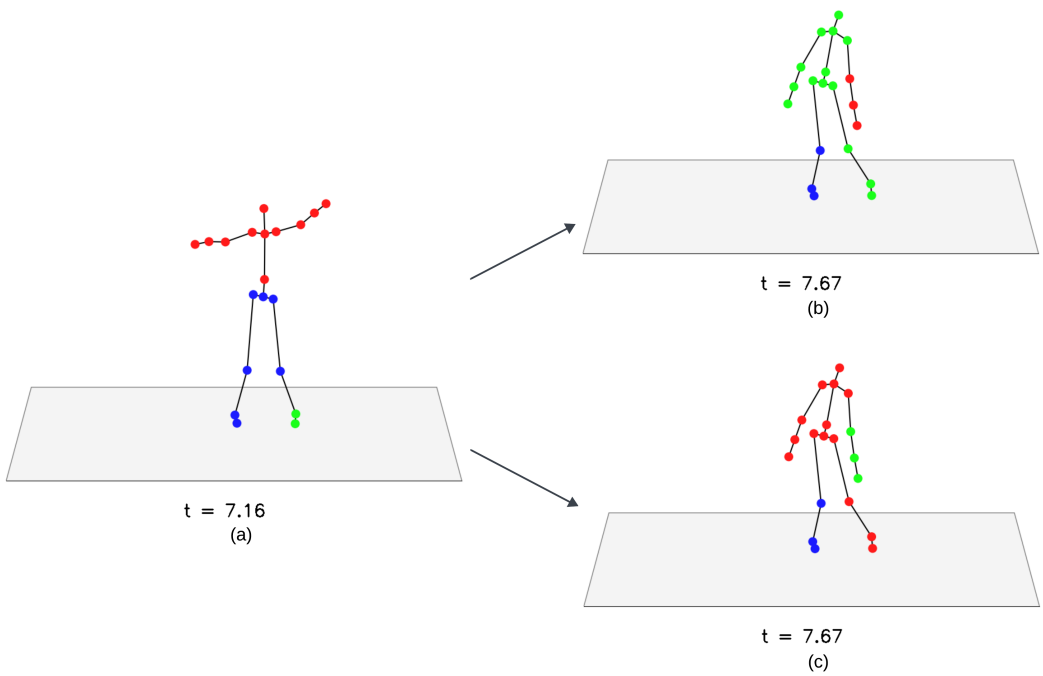


Figure 6.2.1: Clustering at $t = 7.40$ (a), $t = 7.67$ unstabilized (b), and $t = 7.67$ stabilized (c).

A further extension of this concept for visualization purposes can be applied by analyzing a movement evolving in realtime, by recording a small segment of video, which is able to provide a better view of the stabilization outcomes. The transition smoothing algorithm explained in Section 5.2.2 is here visualized in the videos whose link has been embedded in the QR code in Figure 6.2.2.

These 3 QR codes lead to a GIF that represents the same movement in the 20 joints

skeleton, where the dancer begins from a stance with slightly spread legs and the left arm raised upwards. Throughout the movement, the left arm descends while moving backward, the torso leans forward, taking the right arm along with it, and the left foot lifts off the ground, walking back to follow the left arm.
The movement is also rolled back for better view of changes.

The first QR (Figure 6.2.2a) leads to the original clustering method without the application of any stabilization method, visually it can be clearly seen how the instability of the simple clustering creates a flickering effect, resulting in a poor visualization capability. Instead the second (Figure 6.2.2b), which is the stabilized version the flickering has been removed by swapping the clusters colors on subsequent frames based on the proposed approach mentioned in Section 5.2.1. Moreover by applying this method the recognition of eventual origins of movement where they can be clearly seen in a video segment is facilitated.

In the last one (Figure 6.2.2c), clusters' evolution is further smoothed by picking the node with highest degree centrality of the cluster every 5 frames in a 100 fps video. By further smoothing the clusters transition one can also visually inspect the appearing of multiple origins at the expense of the continuity of each cluster. Infact with this method the clusters will have transitory frames in which are inconsistent.

In conclusion, the MaxWPM algorithm is able to provide a clearer and more consistent visualization of clusters, allowing us to understand which joints are most similar during motion, without abrupt color changes.
Furthermore, the additional smoothing technique allows to better see the evolution of clusters across a longer timespan also in those cases where in a short amount of time clusters still tend to change too fast.

(a) (b)

(c)

Figure 6.2.2: QR codes referencing to the pure clustering method (a), the stabilized version (b) and a combination of stabilization and smoothing (c)

6.3 Machine Learning

Prior Attempts The refinement process of the results went through various attempts, as illustrated here. We began with the RF as the initial model, suggested as the best option in previous works in the field of motion analysis [2], with a hyperparameter $n\_estimators = 400$, a generally sufficient value.

The tests to identify the best model were performed on the most frequent edge of the dataset, *left_hand* - *left_wrist*, aiming to balance TPR and TNR.

We directly started with LOOCV as the validation strategy, attempting to rebalance the dataset by applying various balancing strategies. Initially, we attempted a strategy using the *Random Undersampler*, a classic sampling method where the majority class is reduced by randomly extracting the same number of samples as the minority class. These samples were then used for balanced training.

The initial results were quite promising, as depicted in the confusion matrix shown in Table 6.3.1. However, we decided to discard the undersampling approach, considering that with an already limited number of diverse movements, further reducing the number of samples would lead to significant information loss. This conclusion is also supported by the relatively high FP rates, indicating that the model tends to classify a sample as positive with too much confidence.

41	10
2	7

Table 6.3.1: Confusion matrix for Random Undersampler and LOOCV strategy

Even applying a *Random Oversampling* strategy to the minority class (while still excluding the testing sample for each iteration of LOOCV) yields poor results, as seen in Table 6.3.2.

51	0
8	1

Table 6.3.2: Confusion matrix for Random Oversampler and LOOCV strategy

We realized during the various trials that to achieve better results, we needed to carefully choose the resampling strategy. Therefore, we explored alternatives capable of artificially creating samples to better represent the boundaries in the multidimensional space of our features, rather than emphasizing individual points.

Among these alternatives, we also experimented with the SMOTE algorithm, which generates synthetic data by uniformly interpolating points in the multidimensional feature space, and ADASYN, which serves the same purpose as SMOTE but estimates data distribution for resampling.

Below, in Table 6.3.1a and 6.3.1b, the confusion matrices for the two algorithms are presented.

50	1
6	3

(a)

51	0
5	4

(b)

Figure 6.3.1: Confusion matrices of the SMOTE and ADASYN algorithms

In general, these methods exhibit promising results with almost no parameterization required. However, a notable observation is that while the *Random Undersampler* demonstrates similar and high TPR and TNR, it might be challenging to improve significantly due to its simplicity. The same notion applies to the *Random Oversampler*.

Regarding SMOTE and ADASYN, they compete with each other, but ADASYN remains challenging to apply because of its reduced number of samples, making it difficult to infer from such a limited data distribution. In Table 6.3.3

Method	TPR	TSS	TNR	Accuracy
Random Undersampler	78%	79%	80%	82%
Random Oversampler	11%	56%	100%	87%
SMOTE	33%	66%	98%	88%
ADASYN	44%	72%	100%	92%

Table 6.3.3: Metrics of different methods on the most frequent class of the dataset

Finally, considered the potential presence of zones with single points that might have been marginalized during the minimization phase of the Gini impurity loss function and others that could be more dense, which means that are been sufficiently represented, we opted for a variation of the SMOTE.
This assumption lead us to the B-SMOTE to have a better representativeness of those marginalized samples. By balancing the two characteristic hyperparameters of B-SMOTE (*k_neighbors* and *m_neighbors*) in proportion to the ratio between the sizes of the two classes, we aimed to refine the model.

The final result was based on expanding the minority class, training a RF with numerous trees to thoroughly comprehend the best features to select, and subsequently reducing the learning of a second forest optimized on a smaller subset of features.

Best Results In the following Tables (from 6.3.4 to 6.3.6), there are the confusion matrices, the TPR, TSS, TNR and the Accuracy score for the final model defined in Section 5.3 in our three formulations:

- The most frequent 6 classes of our dataset (Table 6.3.4 and 6.3.5) in the by-joint-subdivision.
- The 5 body parts in which have been grouped the joints (Table 6.3.6 and 6.3.7)
- The division into the upper and lower body part, which is in the same tables.

To interpret the meaning of the confusion matrices, please refer to Section 3.4.6. This is because the dataset is unbalanced, and therefore, better prediction results are expected from this class. As observed, accuracy consistently remains very high, almost always exceeding 70%. However, beyond this, the model demonstrates high *specificity*, meaning it excels at maximizing TN but struggles to identify TP effectively. This behavior is evident in the TPR, which varies significantly across different classes. The model requires a high level of certainty in predictions before classifying a sample as positive. You can also observe the model's high specificity from the TNR, which is consistently quite high in all cases, typically greater than 80%.

Furthermore, for a more comprehensive assessment of the results, we included the TSS, normalized between 0 and 1 and presented as a percentage. This metric is considered effective in demonstrating the model's potential to balance predictions and overall incorrect predictions.

As the TSS ranges from -1 (performing poorly in both cases), 0 (performing well in one case but poorly in the other), to 1 (performing well in both cases), in our case, after shifting and scaling, we observed the TSS consistently exceeding halfway, signifying that the model is generally successful in minimizing errors across all predictions.

48	3
3	6

(a)

51	2
6	1

(b)

44	9
7	0

(c)

51	3
4	2

(d)

52	3
4	1

(e)

53	2
2	3

(f)

Table 6.3.4: Confusion matrices of the 6 most frequent classes in the dataset

Label	Edge	TPR	TSS	TNR	Accuracy
(a)	left_hand - left_wrist	66%	80%	94%	90%
(b)	shoulder_center - head	14%	55%	96%	87%
(c)	right_elbow - right_shoulder	0%	42%	83%	73%
(d)	right_shoulder - shoulder_center	33%	64%	94%	88%
(e)	right_knee - right_hip	20%	57%	95%	88%
(f)	left_knee - left_hip	60%	78%	96%	93%

Table 6.3.5: Metrics of the 6 classes most frequent of the dataset

37	5
7	11

(g)

37	9
10	4

(h)

43	4
6	7

(i)

47	5
7	1

(l)

51	2
6	1

(m)

16	5
8	31

(n)

Table 6.3.6: Confusion matrices of the 5 Body Parts and of the Upper/Lower part

Label	Body Part	TPR	TSS	TNR	Accuracy
(g)	Right Arm	61%	75%	88%	80%
(h)	Left Arm	29%	55%	80%	68%
(i)	Right Leg	54%	73%	91%	83%
(l)	Left Leg	12%	51%	90%	80%
(m)	Head	14%	55%	96%	86%
Label	**Body Part**	**TPR**	**TSS**	**TNR**	**Accuracy**
(n)	Upper	79%	78%	76%	78%

Table 6.3.7: Metrics of the 5 Body Parts and the Upper

From the results obtained in Table 6.3.5, a fluctuating trend is evident, probably due to the limited and imbalanced data. Despite efforts to balance the dataset, as they were tested solely on the majority class among the available ones, whose results can be seen in the confusion matrix in Table 6.3.4a, they were not sufficient to compensate for the lack of data in the other classes.

What has been achieved, therefore, is an effectiveness in predicting the majority class effectively and a tendency to balance FN and FP. By merging the classes, as done in Q2 and Q3, the fluctuations in the results appear to be somewhat mitigated at the expense of a slight deterioration in overall performance (Table 6.3.7). With an extension of the dataset, better and more robust accuracies can be achieved across all classes. Furthermore, it would also be possible to perform multiclass classification for recognition of multiple origin of movements from the same movement.

Further insights of the model can be infered by looking at the ROC curve and it's corresponding AUC, in particular let's see what the one associated to the most frequent joint tells us (Figure 6.3.2): By following the thresholds we can see that $Thresh = 0.28$ provides the best results. This means that by thresholding in that way the output of the forest (which is a score) we can achieve a 90% of TPR with just a 20% of FPR. By default the RF will apply a threshold of 0.5 (in other words classify as positive a sample whose score is greater than 0.5).

Moreover, with an AUC of 0.83, it's evident that the model has almost reached curve saturation. This signifies the model's high performance level.

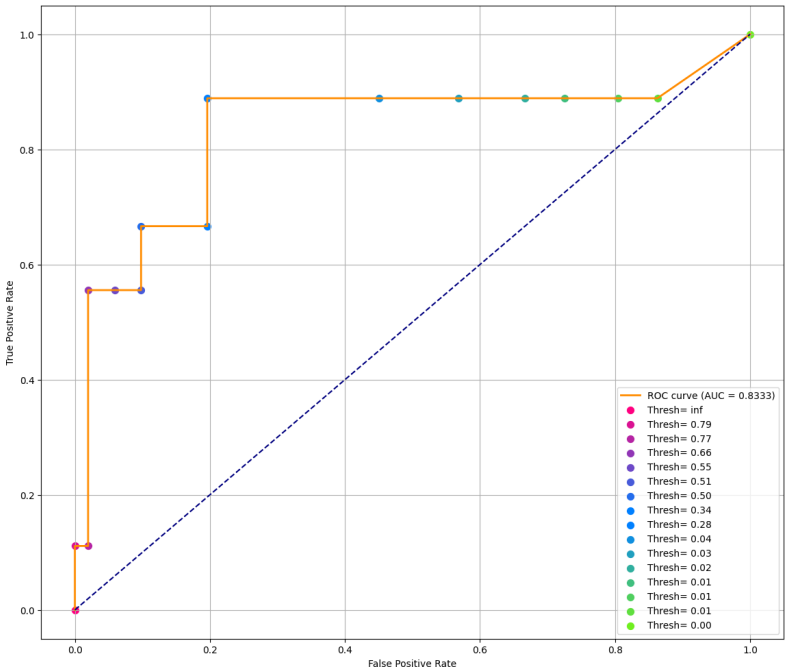


Figure 6.3.2: ROC curve of the most frequent joint of the dataset

Moreover, it is essential to note that the results remained unstable, and the previous ones were the best outcomes across multiple runs of the algorithm, as explained in Section 3.4.5 of the LOOCV.

To compensate for this, 30 runs were executed, and the mean and variance of the main metrics were calculated. In Table 6.3.8, it can be observed that the metrics on the majority class remain generally high, but with a variability that exceeds 10% in terms of standard deviation. Meanwhile, the minority class presents intermediate scores, as expected from the model.

Finally, concerning the overall accuracy, it also averages at 0.868 with a standard deviation of 0.023.

Class	**precision mean**	**precision std**	**recall mean**	**recall std**	**f1-score mean**	**f1-score std**
Negative	93.4%	1%	90.9%	2.5%	92.1%	1.5%
Positive	56%	7.7%	63.7%	5.8%	59.4%	5.7%

Table 6.3.8: Statistics of the metrics for the most frequent joint binary classification

6.4 Future researches

There are several possible future extensions of this work, that could improve on the method and yield better results.

To improve prediction accuracy the dataset should be expanded and ensure balance by incorporating new, intentionally designed movement segments. This will be done without introducing overly intricate motions that could potentially complicate the movement classification.

In future revisions, with a more extensive dataset, additive methods could be employed to measure the amount of useful information added by the WDC approach in a classification problem compared to the other high-level features used. To move towards real-time automatic classification, markerless MoCap can be useful.

Achieving real-time classification with a marker-based system is highly improbable. Although markerless systems may exhibit lower precision in comparison to marker-based systems, they represent a viable alternative for real-world applications.

Another potential future expansion of this concept could involve a study on movement fragments characterized by multiple origins. This would be valuable in examining, for instance, during the act of catching a ball, whether the movement origin is symmetrically distributed throughout the body or if there is a dominant component compared to the others.

The final extension of our thesis involves the examination of small groups of individuals

and the exploration of how coordinated movement patterns emerge within these groups. Instead of treating individual joints as players, it can be adopted a similar approach within the framework of graph and game theory, but at a higher anatomical level. Here, the group of individuals operates as a cohesive unit, where each person in the group takes on the role of a player. This approach enables to examine leading behaviors, including those exhibited by individual initiators whose movements influence the entire group. In scenarios with a limited number of potential permutations for clustering, the advantages between favoring the Hungarian algorithm or the BF approach are not substantial. However, in more intricate situations involving numerous nodes, such as those with groups of people, selecting the most efficient algorithm for graph manipulation becomes crucial to achieve real-time feasibility.

Bibliography

[1] Chosa E Arakawa H. *Development of a video camera type kayak motion capture system to measure water kayaking.* PeerJ, 2023.

[2] Vincenzo D'Amato, Erica Volta, Luca Oneto, Gualtiero Volpe, Antonio Camurri, and Davide Anguita. Understanding violin players' skill level based on motion capture: a data-driven perspective. *Cogn. Comput.*, 12(6):1356–1369, 2020.

[3] Graham Welch. Variability of practice and knowledge of results as factors in learning to sing in tune. *Bulletin of the Council for Research in Music Education*, 85:238–247, 11 1985.

[4] Davidson JW. Visual perception of performance manner in the movements of solo musicians. *Psychology of music*, pages 103–113, 1993.

[5] J.L. Aróstegui. *Educating Music Teachers for the 21st Century.* Educational Futures. SensePublishers, 2011.

[6] John Dewey. *How We Think: A Restatement of the Relation of Reflective Thinking to the Educative Process Vol. 8.* Southern Illinois Up, 1986/2008, 1933.

[7] Siw Graabræk Nielsen. Self-regulating learning strategies in instrumental music practice. *Music Education Research - MUSIC EDUC RES*, 3:155–167, 09 2001.

[8] Stefano Piana, Alessandra Staglianò, Francesca Odone, and Antonio Camurri. Adaptive body gesture representation for automatic emotion recognition. *ACM*

Trans. Interact. Intell. Syst., 6(1), mar 2016.

[9] PNUD. Pnud objetivos de desarrollo sostenible. https://www.undp.org/content/undp/es/home/sustainable-development-goals.html.

[10] World Health Organization. *World Report on Disability*. WHO Press, Geneva, Switzerland, 2011.

[11] International Disability Alliance. International disability alliance. http://www.internationaldisabilityalliance.org/.

[12] Andrea Catherine Alarcón-Aldana, Mauro Callejas-Cuervo, and Antonio Padilha Lanari Bo. Upper limb physical rehabilitation using serious videogames and motion capture systems: A systematic review. *Sensors*, 20(21), 2020.

[13] Marianna Liparoti, Emahnuel Troisi Lopez, and Valeria Agosti. Motion capture system: A useful tool to study cycling posture. *Journal of Physical Education and Sport*, 20(4):2364–2367, 2020.

[14] Ksenia Kolykhalova, Giorgio Gnecco, Marcello Sanguineti, Gualtiero Volpe, and Antonio Camurri. Automated analysis of the origin of movement: An approach based on cooperative games on graphs. *IEEE Transactions on Human-Machine Systems*, 50(6):550–560, 2020.

[15] Olga Matthiopoulou, Benoît Bardy, Giorgio Gnecco, Denis Mottet, Marcello Sanguineti, and Antonio Camurri. A computational method to automatically detect the perceived origin of full-body human movement and its propagation. pages 449–453, 10 2020.

[16] Casa Paganini Infomus. http://www.casapaganini.org/.

[17] Robin J. Wilson. Introduction to graph theory. 1:1–17, 05 2010.

[18] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive

synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, 2008.

[19] Hui Han, Wenyuan Wang, and Binghuan Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *International Conference on Intelligent Computing*, 2005.

[20] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[21] Y. Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *CIRANO, CIRANO Working Papers*, 5, 11 2003.

[22] Yongli Zhang and Yuhong Yang. Cross-validation for selecting a model selection procedure. *Journal of Econometrics*, 187(1):95–112, 2015.

[23] Huazhen Wang, Fan Yang, and Zhiyuan Luo. An experimental study of the intrinsic stability of random forest variable importance measures. *BMC Bioinformatics*, 17, 02 2016.

[24] Albert Sama, Diego E. Pardo-Ayala, Joan Cabestany, and Alejandro Rodríguez-Molinero. Time series analysis of inertial-body signals for the extraction of dynamic properties from human gait. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–5, 2010.