UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER OF SCIENCE IN TELECOMMUNICATION ENGINEERING

MASTER THESIS

DESIGN AND IMPLEMENTATION OF A HAND'S POSE ANALYTIC SYSTEM WITH ARTIFICIAL NEURAL NETWORKS AND MACHINE LEARNING ALGORITHMS

ÁLVARO GUTIÉRREZ TENORIO

2021

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER OF SCIENCE IN TELECOMMUNICATION ENGINEERING

MASTER THESIS

DESIGN AND IMPLEMENTATION OF A HAND'S POSE ANALYTIC SYSTEM WITH ARTIFICIAL NEURAL NETWORKS AND MACHINE LEARNING ALGORITHMS

Author
ÁLVARO GUTIÉRREZ TENORIO

Tutor
ÁLVARO GUTIÉRREZ MARTÍN

2021

Resumen

La esclerosis múltiple (EM) es una enfermedad que afecta a más de 47.000 personas en España y a 2,5 millones en todo el mundo. Los pacientes con EM sufren un daño en el recubrimiento de las células nerviosas del cerebro y de la médula espinal, provocando un deterioro del rendimiento motor y cognitivo y, en particular, problemas de coordinación entre la mano y el ojo.

Existen varias pruebas para cuantificar los efectos de la EM en el sistema neurológico del paciente, entre las que se encuentra el Nine-Hole Peg Test (NHPT). El objetivo de este Trabajo de Fin de Máster es extraer datos de la agilidad y destreza de la mano del sujeto mientras realiza el NHPT mediante técnicas automatizadas y objetivas. Con estos datos se implementarán modelos de Machine Learning (ML) para predecir el estado del paciente y su evolución futura. Para ello, se pretende grabar la prueba, extraer la pose de la mano con Redes Neuronales Convolucionales, obtener información cinemática del paciente y, finalmente, utilizar estos datos para clasificar la gravedad de la enfermedad y compararla con otros pacientes. Con esta información se pretende mejorar el tratamiento del paciente, aumentando su calidad de vida.

El desarrollo de este Trabajo de Fin de Máster demostrará la gran importancia del NHPT para predecir la progresión de la esclerosis múltiple. Este proyecto cambiará el proceso de diagnóstico y tratamiento de los pacientes con EM, pudiendo obtener información altamente fiable sobre el estado de la enfermedad. De este modo, no sólo el colectivo de pacientes de EM, sino también el de los clínicos, se beneficiarán de las ventajas que ofrecerán los modelos creados, ahorrando tiempo y recursos.

En primer lugar, se construye una estructura mecánica que sirve de soporte para colocar las cámaras. A continuación, se graban varios vídeos en la Unidad de Neuroinmunología y Esclerosis Múltiple del Hospital La Paz, tanto de pacientes con EM como de sujetos de control. Posteriormente, se extrae la pose de la mano con la ayuda de DeepLabCut, una herramienta que usa el Deep Learning para la estimación de la posición sin marcadores. Finalmente, se obtienen las características cinemáticas de la mano del sujeto. Con esta información se crean modelos de Machine Learning que clasifican el estado de la enfermedad. Para extraer conclusiones preliminares sobre la importancia diagnóstica de la información obtenida, se proponen dos ejercicios de clasificación.

El primer ejercicio es un problema de clasificación binaria, en el que se entrenan los modelos de ML para separar la información entre los sujetos de control y los pacientes reales. Los resultados obtenidos tienen una prometedora tasa de acierto de casi el 90% para diferenciar a los pacientes de EM de las personas sin enfermedad.

El segundo ejercicio es un problema de clasificación multiclase. Las clases se extraerán de los datos proporcionados por el Hospital, separando a los pacientes reales de EM en tres clases: enfermedad leve, media y grave. Esto, sumado a la clase de sujetos de control, compone el problema de clasificación multiclase. Los modelos entrenados tuvieron un 72% de precisión en la predicción del estado de la enfermedad, siendo capaces de diferenciar con gran exactitud entre pacientes leves y graves.

Palabras clave: Esclerosis Múltiple, Nine-Hole Peg Test, Deep Learning, DeepLabCut, Redes Neuronales Convolucionales, Machine Learning.

Abstract

Multiple sclerosis (MS) is a disease that affects over 47,000 people in Spain, and 2.5 million people all over the world. MS patients suffer from a damage in the covers of the brain's and the spinal cord's nerve cells, leading to an impairment in motor and cognitive performance and, in particular, problems with hand-eye coordination.

There are several tests to quantify the effects of MS on the neurological system of the patient, among which is the Nine-Hole Peg Test (NHPT). The aim of this Master Thesis is to extract data of the agility and skill of the subject's hand while performing the NHPT with automated and objective measurements. With this data, Machine Learning models will be implemented to predict the state of the patient and its future development. This is intended to be achieved recording the test, extracting the hand's pose with Convolutional Neural Networks, acquiring the patient's kinematic features, and finally, using this data to classify the severity of the disease and comparing it to other patients. With this information, the patient's treatment is to be improved, increasing his/her quality of life.

The development of this Master Thesis will prove the great importance of the NHPT to predict the progression of Multiple Sclerosis. This project will change the diagnosis process and treatment of MS patients, while being able to obtain highly reliable information about the state of the disease. In this way, not only the group of MS patients, but also the clinicians, will benefit of the advantages that the created models will offer, saving time and resources.

First, a mechanical structure was built to provide a stand where the cameras will be placed. Next, several videos are recorded in the Neuroimmunology and Multiple Sclerosis Unit in La Paz Hospital, both for MS patients and control subjects. Afterwards, the hand's pose is extracted with the help of DeepLabCut, a Deep Learning method for markerless pose estimation. Finally, kinematic features of the subject's hand are obtained. With this information, Machine Learning models that classify the state of the disease are created. To extract preliminary conclusions about the diagnostic importance of the obtained information, two classification exercises are proposed.

The first exercise is a binary classification problem, where the ML models will be trained to separate the information between control subjects and actual patients. The results obtained had a promising accuracy of almost 90% in differentiating MS patients from people without disease.

The second exercise is a multiclass classification problem. The classes will be extracted from the data provided by the Hospital, separating the actual MS patients into three classes: mild, middle and severe disease. This, added to the control subject class, composes the multiclass classification problem. The trained models had a 72% of accuracy predicting the state of the disease, being able to differentiate with high precision between mild and severe patients.

Keywords: Multiple Sclerosis, Nine-Hole Peg Test, Deep Learning, DeepLabCut, Convolutional Neural Networks, Machine Learning.

Agradecimientos

A mi tutor, Álvaro. Gracias por haberme dado de nuevo la oportunidad de trabajar contigo en un TFM tan interesante como este, y por ayudarme a introducirme de lleno en un proyecto real de Data Science. También gracias por haber confiado en mí la responsabilidad de ser tutor de un TFG. Ha resultado toda una aventura.

Gracias a la familia de Robolabo, a Rafa y a Miguel, por haber prestado vuestra ayuda a pesar de mi insistencia y la lata que os di los primeros meses del curso. Gracias a Pablo, por haber confiado en mi como tutor y haber hecho todo el trabajo que yo no podía hacer cuando estaba lejos. Has hecho un trabajazo.

También quiero dar las gracias a la Unidad de Neuroinmunología y Esclerosis Múltiple del Hospital Universitario La Paz por haber hecho este TFM posible.

Gracias a mis compañeros de máster, por haber hecho que estos dos años pasaran volando. A Sergio, Jaime y Pablo, porque aunque pasen los años, cuando nos vemos, seguimos siendo los que éramos todas las mañanas y tardes que pasábamos en la ETSIT. A mis amigos de Múnich, por vivir con vosotros estas experiencias inolvidables. A todos mis amigos que han estado ahí, gracias.

Gracias Isa, por haber soportado mis enfados y agobios, siguiendo a mi lado y haciéndome ver la parte positiva de todo. Te admiro mucho. Aún nos quedan muchas cosas por vivir juntos.

A mis hermanas, Daniela y Valeria, y a David, por ser el mejor hermano y amigo que se puede tener. A mis padres, porque sin vosotros nada de esto habría sido posible. Gracias a mi madre, por inculcarme su perseverancia y paciencia. Gracias a mi padre por enseñarme a trabajar duro para conseguir lo que quiero. Sois mi mayor ejemplo a seguir.

Contents

List of Figures

List of tables

Acronym List

NHPT: Nine-Hole Peg Test.

MS: Multiple Sclerosis.

MRI: Magnetic Resonance Image.

AI: Artificial Intelligence.

CNN: Convolutional Neural Network.

LSTM: Long Short-Term Memory.

RNN: Recurrent Neural Network.

GPU: Graphic Processing Unit.

DL: Deep Learning.

ANN: Artificial Neural Network.

ML: Machine Learning.

KNN: K-Nearest Neighbors.

NN: Nearest Neighbor.

MSE: Mean Square Error.

SVM: Support Vectors Machine.

PVC: Polyvinyl Choride.

RAM: Random Access Memory.

SD: Secure Digital.

USB: Universal Serial Bus.

DLC: DeepLabCut.

YAML: YAML Ain't Markup Language.

CSV: Comma Separated Values.

MAE: Mean Absolute Error.

Chapter 1

Introduction

This Master Final Thesis has been developed in the Robotics and Control Laboratory (Robolabo).

In this Chapter, the context where this Master Thesis is developed is presented. First, the project will be introduced, alongside with its objectives. Later, the State of the Art of Multiple Sclerosis, Machine Learning and Deep Learning will be discused.

1.1 Project overview and motivation

The aim of this Master Thesis is to standardize the whole process of the Nine-Hole Peg Test (NHPT) [10]. This test is performed and analysed in order to extract data of the agility and the skill of a subject's hand. The test consists in taking nine pegs from a container, one at a time, to nine holes that are found next to the container, following a square-like shape (see Figure 1.1).

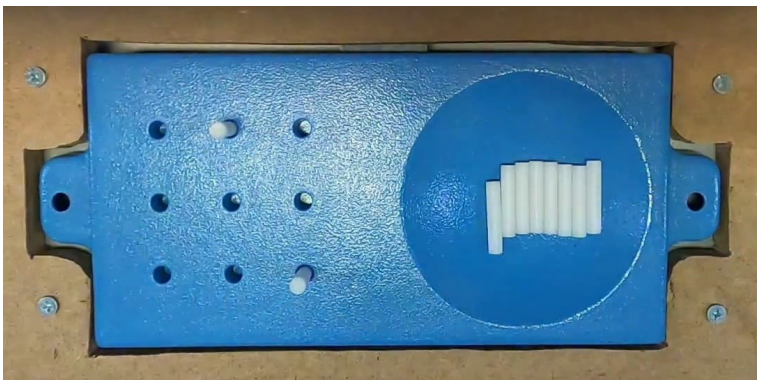


Figure 1.1: Nine-Hole Peg Test Kit.

At the moment, the way in which the test is analysed extracts very few aspects from the exercise; as the total time of performance or the number of times a peg falls from the hand. This way, this project aims to develop a way in which the test is exhaustively analysed and objective data is obtained. For that purpose, a mechanical structure will be designed and built in order to capture the whole movement with the help of three orthogonally placed cameras. Then the recorded videos will be analysed

with Convolutional Neural Networks, whose purpose is to extract the position of the hand for every frame in the video. The extracted data will be processed to obtain relevant indicators of how the test was performed. Finally, this processed data will be used to build Machine Learning models trained to classify the hand's movement between different classes of Multiple Sclerosis.

1.2 Multiple sclerosis

1.2.1 Definition and effects

Multiple sclerosis (MS), also known as encephalomyelitis disseminata, is a disease in which the covers that insulate the brain's and the spinal cord's nerve cells are damaged. Depending on its severity, MS can range from relatively benign to devastating, as the communications between the brain and some parts of the body are ceased [11].

Multiple sclerosis patients are impaired in motor and cognitive performance, and its change with age is still unkown [12]. Some effects examples are memory problems, confusion, emotional or personality changes, vision and hearing loss, problems with speaking, fatigue, muscular weakness and balance issues among others. Many patients experience effects in their limbs, as the damage to the myelin sheath in neurons results in pain, tingling and numbness in arms and legs. In particular, problems with hand-eye coordination may appear [13].

1.2.2 Diagnosis and treatment

Multiple sclerosis can be one of the most difficult diseases to diagnose because of the multiple symptoms it causes and the ways in which they can present [14]. To diagnose MS, several symptoms must be observed; such as evidence of damage in at least two separate areas of the central nervous system, determine that the damaged areas developed at least one month apart, perform a spinal tap and examination for oligoclonal bands and perform an MRI. Lastly, all other possible diagnoses must be discarded [15].

Treatment is focused in four areas: acute attacks, prevention of relapses and progression, management of symptoms, and rehabilitation. Although most of alternative therapies have not been tried yet, this research is not being funded because of the difficulty of the design and lack of interest [16].

There are several tests to quantify the effects of MS on the neurological system of the patient. Symbol Digit Modalities Test (SDMT) and Nine-Hole Peg Test (NHPT) are some of these tests. SDMT consists in linking nine symbols with different geometric shapes, to the numbers from 1 to 9. The score of the test is the number of completed symbols in 90 seconds. NHPT, the tests used in this Master Thesis, will be explained in depth in the following subsection.

1.2.3 Nine-Hole Peg Test

The Nine-Hole Peg Test was developed to measure finger dexterity, also known as fine manual dexterity. It is currently used with a wide range of population as it is relatively inexpensive and can be administered quickly. Additionally, this test should be used in association with other upper extremity performance tests, aiming to get a more accurate estimate of the upper limb function [17].

The test kit is composed by a square board and nine pegs. On one side of the board, there are nine holes placed in a squared form for the pegs to fit in. On the other side, there is a rounded dish where the pegs are stored. The subject that performs the test is asked to take every peg, one by one, from the dish to the holes and then take them back to the dish, as quickly as possible.

The score obtained in a test is based on the time taken to carry out the whole exercise. However, there are alternative ways of scoring as the number of pegs placed in 50 or in 100 seconds. In this case, the score is expressed in number of placed pegs [18][19].

In the past few years, the NHPT has been one of the most used measures of the upper extremity function in MS. Although previously the assessment of gait was more relevant in clinical studies than the assessment of arm and hand function, the latter has increased its recognition, specially in patients with severe disability. The NHPT provides a stardardized approach to assessment and can be administered easily .

In clinical studies of MS, assessment of gait has generally had a more prominent role than assessment of arm and hand function. However, in recent years, there has been increasing recognition of the usefulness of measuring arm and hand function in clinical studies, especially in patients with severe disability. In the last few years, the NHPT has been one of the most frequently used measures of upper extremity function in MS. The 9-HPT provides a brief, standardized approach to assessment and can be administered by a wide variety of trained examiners [20].

1.3 Deep Learning

In this section the State of the Art and the insights of Deep Learning and Neural Networks will be assessed.

1.3.1 Definition and history

Deep Learning is a class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification [21].

Deep Learning uses layers of algorithms to process all kinds of data, such as human speech, object pictures for recognition and videos for pose estimation. The information goes through each layer, being the input of one, the output of the previous one [22]. It is inspired in the workings of the human brain in processing data and creating patterns for use in decision making. It has networks capable of performing unsupervised learning from unstructured or unlabeled data, or supervised learning from labeled data.

Its history can be traced back to 1943, when a computer model based on the human brain was created by Walter Pitts and Warren McCulloch. It used a combination of algortihms called "threshold logic" to mimic the same thought process of the brain. Henry J. Kelley was he first to develop a continuous Back Propagation Model in 1960; which was simplified in 1962 by Stuart Dreyfus while only using the chain rule.

In 1965, Alexey Grigoryevich and Valentin Grigor'evich Lapa invested their efforts in developing deep learning algorithms that used polynomial activation functions, that were then statistically analysed. From each one of the layers the best statistically chosen features were then forwarded to the next layer, on a slow and manual process [22].

It was in the 1970s decade where the first AI winter appeared. This term was first used in 1984 in order to reference the years where the AI's funding was limited and resulted in the extinction of AI companies. This was the repercussion of the hype over expert systems and the disappointment it caused when the sector discovered its limitations [23]. After this lack of money in the industry, deep learning developed steadily until the start of the second AI winter in 1984. Until then, in 1979 Kunihiko Fukushima introduced the first "Convolutional Neural Network". He designed a network with multiple pooling and convolutional layers, which he called Neocognitron, whose design allowed the computer to learn to recognise visual patterns [24]. When the second AI winter came in 1984, although the means were little, some researches kept working on Deep Learning. As a result, Sepp Hochreiter and Juergen Schmidhuber developed the Long Short-Term Memory (LSTM) for Recurrent Neural Networks (RNN) in 1997 [25].

The breakthrough was finally reached in 1999 when computers got faster and GPUs (Graphic Processing Unit) were developed, which increased the computational power by 1000 times. In 2001, a report by Gartner [26] described the increase in the volume and speed of data, calling to prepare for the advance of Big Data, which was just starting. In the 2010s, with the speed of GPUs increasing rapidly, new architectures of networks appeared such as AlexNet or ImageNet [27], giving the evidence that Deep Learning had significant advantages in terms of efficiency and speed.

Currently, both the evolution of Artificial Intelligence and Big Data are dependent on Deep Learning. All things considered, Deep Learning surpassess various Machine Learning approaches in performance nowadays and it is widely used for multiple tasks. However, it is still evolving steadily and in need for new ideas to keep going further [28, 29].

1.3.2 Types of Deep Learning Algorithms

As aforementioned, Deep Learning was firstly inspired by the structure of biological neural networks. Deep Learning algorithms are separated in diverse types of specific models and, because of that, they are currently used to solve all kinds of problems worldwide. There are several types of DL algorithms; however, only the most commonly used ones will be described to give a general idea of how they work and their history.

A. Artificial Neural Networks

Artificial Neural Networks (ANN) are the attempt of emulation of the human brain as a piece of computing and they are the foundation of Deep Learning. They try to resemble the way the human brain analyses and processes information [30].

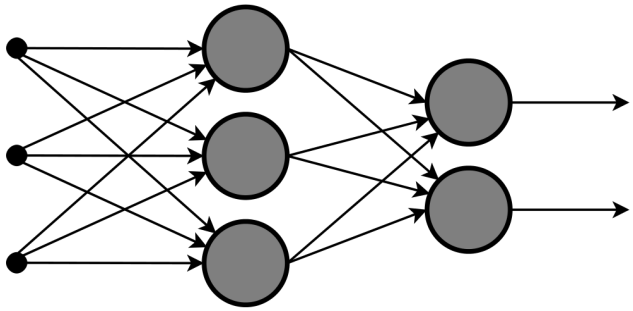


Figure 1.2: Artificial Neural Network architecture [1]

.

Neural Networks consist on thousands of processing nodes that are interconnected. Most of today's ANN are organised into layers and they are 'feed-forward'. This term means that data moves through them in just one direction. One node receives data from the previous layer, process it, and sends it to then next layer (see Figure 1.2).

To each of the incoming connections, the node assigns a number known as "weight", and the data coming into this input, will be multiplied by this weight. It then adds all the results together, getting a single number. If this result is greater than a defined threshold, the node send the information to the next layer through all its output connections [31].

When the training of an ANN starts, its weights are initialised to random values. Then the training data is fed to the input layer and it passes trhough the succeding layers, reaching the output radically transformed. During this process, weights and thresholds are adjusted continuously until the output and the labels of the data are similar [31].

These adjustments depend on the optimization algorithm that is used to minimize the error. There are several optimizers and each one of them are useful in certain situations. One of the most used ones is the Gradient Descent. Gradient Descent is frequently used in linear regression and classification algorithms and it is a first-order algorithm that depends on the first derivative of the loss function. Using this optimizer, the weights are updated subtracting the gradient of the loss function multiplied by a learning rate α. Hence, Gradient Descent follows the fastest way to reach a minimum of the loss function [32].

$$\theta := \theta - \alpha \nabla L(\theta)$$

Through backpropagation, loss is transfered from one layer to the previous one, modifying the weights until the loss is minimized.

The selection of the aforementioned learning rate is also crucial in the training

process. It controls the speed at which the model learns. Having a high learning rate allows the model to learn faster but it may never find the minimum as the weights would change abruptly. On the other side, a very low learning rate would imply that the model learns more slowly and although a minimum is reached, it may not be the global minimum of the loss function.

B. Convolutional Neural Networks

The Convolutional Neural Network (CNN) or ConvNet is not quite the normal Neural Network. It is specialized in processing grid-like data. A CNN has typically three layers: a convolution layer, a pooling layer and a fully connected layer. Just as the neurons in the human brain, neurons in CNN process data only in its receptive way. There are layers that are specialized in detecting simple patterns (like strait lines or curves) and other ones in detecting more complex patterns (like faces or objects) [2].

Just like ANNs, CNNs are formed by layers, but these are called convolutional in the way they process data. These layers perform the dot product of two matrices. One of these matrices is known as kernel, which includes the trainable parameters, and the other matrix is a part of the data to be processed (see Figure 1.3). The kernel is smaller than the image in the first two dimensions but has the same depth than the image, i.e., RGB images with three channels.

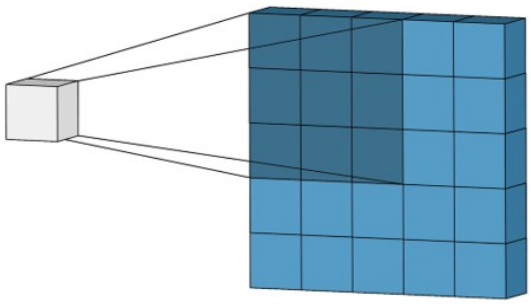


Figure 1.3: Convolutional process of a 3x3 kernel with a 5x5 image [2].

In this process, multiple aspects take part in the output of the layer dimension. First, the dimension of the image (n_h, n_w) and the filter size f. Into these aspects, the stride must be added. The stride is the 'step' that the filter may take between each matrix multiplication, and it influences the step horizontally and vertically. Finally, the padding also affects the output dimension, as it adds zeros at both sides of the image in order to have a greater dimension of the input. In this way, the output dimension is [33]:

$$n_{H\_new} = \left\lfloor \frac{n_H + 2p - f}{s} + 1 \right\rfloor$$

In addition to the convolutional layer, CNNs also use two more types of layers: pooling layers and fully-connected layers.

Pooling layers help reducing the representation of the layer into a fewer number of pixels. It uses nearby outputs to obtain a summary statistic, like the maximum value or the average. This operation is processed on every slice of the representation individually. As for this layer, we do not contemplate the possibility of adding zero padding.

$$n_{H\_new} = \left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor$$

Fully-connected layers are layers that completely connect with the previous and the preceding ones. This means that each neuron in the layer is connected to all the neurons of the next and previous layers. This layer type is the same one as seen in a normal Artificial Fully-Connected Neural Network [33].

C. Long Short-Term Memory Networks

Long Short-Term Memory networks are a type of Recurrent Neural Networks capable of learning order dependence in sequence prediction problems. They are commonly used in speech recognition problems, machine translation and word recommendation systems [34]. Recurrent Neural Networks are different from the traditional Feed-Forward Neural Networks. These networks have an internal state that can represent context information, keeping information about past inputs for an amount of time that is not fixed [35]. The system is able to store information for an arbitrary duration, it is resistant to noise, and parameters are trainable. They are designed to learn sequential or timevarying patterns [36].

The core concept of LSTM is the cell state. This cell state is like the "memory" of the network. The relevant information from previous steps can be carried to and used in the next steps of execution. The cell state keeps changing thanks to the gates, that control which information is allowed into the cell state. These gates learn what information is relevant to keep or forget during training [37].

1.4 Machine Learning Algorithms

This section will tackle the State of the Art of Machine Learning and some of its algorithms for classification.

1.4.1 Definition and history

Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data [38]. ML builds models based on sampled data (training data) in order to predict values or classes. In short, it tries to adapt and model the statistical model of the variable to predict.

There are three types of ML algorithms: Supervised, Unsupervised and Reinforcement Learning. Supervised learning tries to predict a variable from a given set of predictors (independent variables). It is characterised because training data is

given along with the desired output or label. In Unsupervised Learning, there is no target variable to predict. It is normally use to cluster the data into different groups for segmentation.Finally, Reinforcement Learning tries to teach the algorithm to take decisions while receiving a reward depending on these decisions. Based on the learned outcomes on the past, the algorithms learns and gives the result in order to get the highest reward.

The term of Machine Learning brings us back to 1952, when Arthur Samuel of IBM developed a computer program for playing checkers. This program included a scoring function that attempted to measure the probablity of each side winning. The program used its next move using the so called minimax algorithm, which tried to minimize the maximum loss in games with opponents [39]. This program also had some mechanisms that allowed the system to become better, recording all positions it had already seen and combined them with the values of the reward function. This is when Arthur Samuel first used the term "Machine Learning".

Earlier, in 1951 one of the most commonly known supervised ML algorithms was introduced by Evelyn Fix and Joseph Hodges, the K-nearest neighbors algorithm (kNN) [40]. This algorithm was expanded by Thomas Cover in its use for both regression and classification problems [41].

In 1967, Nearest Neighbor (NN) Algorithm was introduced by Marcello Pelillo, although he gives credit to the Cover and Hart paper [42]. This algorithm was used for finding the most efficient route when mapping them. It was used to solve the traveling salesperson's problem [43] and with it, the salesperson would enter a selected city and then the nearest ones until all have been visited. This algorithm minimized the distance traveled by the salesperson.

Alongside NN, an unsupervised algorithm called K-means clustering appeared. This term was first used in 1967 by James MacQueen but the algorithm had appeared in 1957 as a technique for pulse-code modulation. However, this algorithm was not published until 1982, being similar to the Edward W. Forgy publication. As they were both refering to the same method, this algorithm is sometimes called Lloyd-Forgy algorithm [44, 45].

In between the 70s and the beginning of the 80s, Machine Learning and Artificial Intelligence take different paths, as AI had focused on knowledge-based approaches instead of using algorithms. Neural network research was abandoned in these years, due to the aforementioned AI winters [23]. Therefore, Machine Learning shifted from training for AI to solving practical problems.

It was in 1990, when a necessary development for Machine Learning appeared, boosting algorithms [46]. These algorithms are used to reduce bias during supervised learning and they are not algorithmically constraint, but they add several weak classifiers (unaccurate) to build a strong one. This concept, was first introduced by Robert Schapire in a 1990 paper called "The strenght of Weak Learnability" [47].

This concept leads us to another commonly known ML algorithm, Random Forest. This algorithm was proposed by Tin Kam Ho in 1995 when introducing the idea of doing a majority vote on multiple undertrained, overfitted decision trees [48]. These trees would be trained with a reduced sample of the training data and it may not have access to all the features of the dataset. This way, we can get a generalisation

of the data and get an accurate result.

Nowadays, Machine Learning algorithms have become a really powerful tool that keeps learning and increasing its accuracy the more they are used, gaining importance in an increasingly data-driven world.

1.4.2 Types of Machine Learning Algorithms

In the previous definition, we stated that there are three subgroups of Machine Learning algorithms: Supervised, Unsupervised and Reinforcement Learning. However, as in the Deep Learning algorithms section, only the mostly used ones and most relevant for this project will be explained.

A. Regression

Regression is a supervised ML method that tries to estimate the relationships between a dependent variable or target with other independent variables or features. There are multiple regression types, that depend on the type of target variable, the shape of the regression and the number of independent variables to use.

Linear regression is the most common type of regression and it tries to model the relationship between one or more features in a linear way. The case in which it is one feature is modelled with the information of the other is called Simple Linear Regression. If there is more than one feature to estimate the objective feature it is called Multiple Linear Regression [49].

The Linear Regression model is:

$$\widehat{y_i} = \boldsymbol{\beta}^T \boldsymbol{x_i} + \varepsilon$$

Where $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_n]$ is the coefficient vector, n the number of features used for the modelling and ε is the intercept or offset. Vector $\boldsymbol{x_i}$ are the values of the features used for the model and $\widehat{y_i}$ is the prediction of the goal feature, both in the i^{th} observation.

The objective of this modelling is to find a vector $\boldsymbol{\beta}$ and a value of ε that minimizes the Mean Square Error (MSE):

$$\underset{\beta,\varepsilon}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - (\boldsymbol{\beta}^T \boldsymbol{x_i} + \varepsilon))^2$$

We can also describe the model of the Polynomial regression, which is similar to the linear one but increasing the degree of the model function (see Figure 1.4).

$$\widehat{y_i} = \boldsymbol{\beta_1}^T \boldsymbol{x_i} + \boldsymbol{\beta_2}^T \boldsymbol{x_i^2} + ... + \boldsymbol{\beta_n}^T \boldsymbol{x_i^n} + \varepsilon$$

This model also minimizes the MSE and helps us model more complex data distributions.

In addition to Polynomial and Linear regression, we also have Logistic regression, Lasso regresion, Ridge Regression, Polynomial regression and Bayesian regression. All of them are also well-known Machine Learning models and their uses are very diverse.

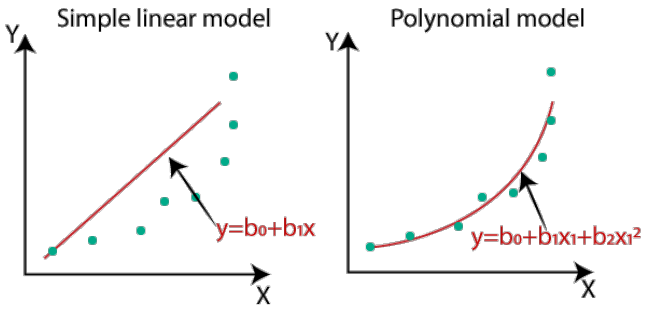


Figure 1.4: Linear and polynomial regression [3].

B. Logistic Classification

Classification has a similar approach to the one of the Regression. However, instead of predicting a continuous value, classifications tries to predict discrete values, classes.

For this matter, Logistic Regression provides a really powerful method for classification to classify observations.

Logistic Regression can be explained with the sigmoid function, which takes an input and outputs a probability between 0 and 1 [50].

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

where t is going to be a linear function dependent on $\boldsymbol{x}$:

$$t = \boldsymbol{\beta}^T \boldsymbol{x} + \varepsilon$$

And then logistic regression tries to find a function $\widehat{y}$ that minimizes the error while predicting the class of the observation (example for two classes) (see Figure 1.5).

$$\widehat{y} = \begin{cases} 1 & if \sigma(t) \geq 0.5 \\ 0 & else \end{cases}$$

C. Support Vector Machines

A Support-Vector Machine (SVM) is a model developed by Corinna Cortes and Vladimir Vapnik in 1995 [51] and builds a hyperplane or set of hyperplanes in a high-dimensional space, which can be used as the estimated separating surface between classes. These hyperplanes are defined by a linear combination of vectors set by some of the data points. These vectors are called Support Vectors.

For the classification problem, it sometimes happens that classes were not linearly separable in the observations sub-space. However, a solution that consisted in mapping this sub-space to a much higher dimensional space, where it could already be separable, with the help of the kernel trick was proposed [52].

The kernel trick provides a solution to the impractical computational costs of mapping every observation into this new sub-space (see Figure 1.6). With the kernel

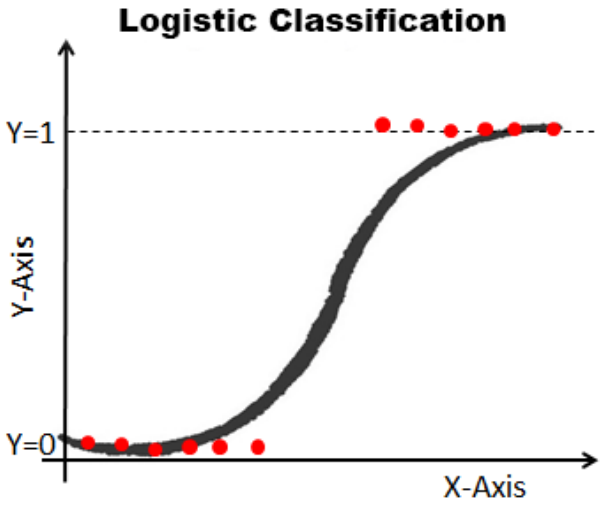


Figure 1.5: Logistic Classification example [4]

trick, the definition of the subspace transformation is not needed to be known, but only with the dot product definition of the subspace is enough. The kernel function is defined as $k(x, z) = \langle \phi(x), \phi(z) \rangle$ where $\phi : X \mapsto \Re^N, x, z \in X$

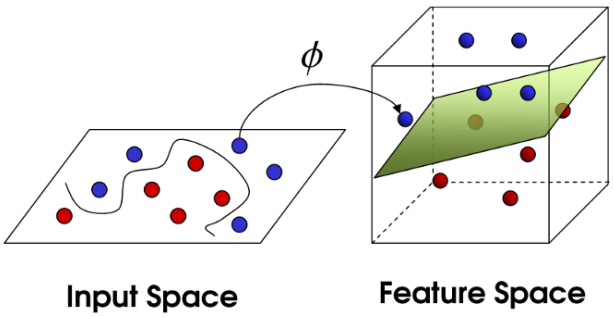


Figure 1.6: Kernel trick explanation. [5]

D. K-Nearest Neighbors

K-Nearest Neighbors (KNN) was developed by Evelyn Fix and Joseph Hodges in 1951 [40]. This is a non-parametric method where the k nearest observations are used to predict the class or value of the new observation. In classification, a majority vote takes place and the most frequent class is assigned to this new observation. In regression, the output is the average of the nearest neighbors.

This algorithm is highly dependent on two measures: the k value and the distance between observations. The most used distance measure is the Euclidean distance for continuous variables. Also, when selecting the k value, classification accuracy must be analised as the output of the model can be highly modified by just changing the

k. Figure 1.7 represents the importance of this problem, as when choosing $k = 3$, the assigned class would be red, but when choosing $k = 5$, it would change to blue.

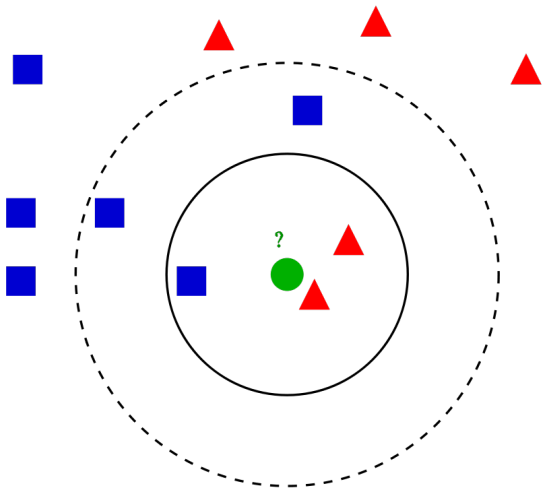


Figure 1.7: K-nearest neighbors example [6].

E. Random Forest

To understand this method, we must first explain the concept of a decision tree. A decision tree is not more than concatenated if-else conditions, which starts in a top node and keeps 'branching out' in tow (right and left) until it gets to the 'leaf' nodes. These leaf nodes contain the mean of the observations that meet the specified conditions that define a region. In classification, the leaf nodes contain the class that the sample would be assigned to.

A Random Forest is made of different decision trees. After this, each tree makes a prediction and the final value for our prediction is the mean of every tree estimation or the result of a majority vote on the predicted classes [53]. To obtain stochastically independent binary trees, they are built randomly through two measures: using different training sample sets, called bootstrapping, or using a reduced index set of features. Although there is still a high probability that these trees are overfitted with their input data, the Random Forest principle relies that the majority vote of all these random trees concludes in a proper generalisation of the true model [54]. An important factor to be mentioned of this algorithm is that, unlike linear regression, it will never predict a value outside of the interval of values of the dataset [55].

F. K-Means Clustering

K-means Clustering is an unsupervised Machine Learning method that separates a given data with n observations into k clusters. The membership of each data point is decided on the distance to each one of the centroids (mean of all cluster member points). This technique tries to minimize the variance within the clusters, the squared Euclidean distances. In other words, it tries to concentrate points that are most

similar together [7].

K-means starts with k randomly selected centroids, then, performs iterative modifications to optimize the variance of the cluster until their points stabilize and they centroids remain constant (see Figure 1.8)).

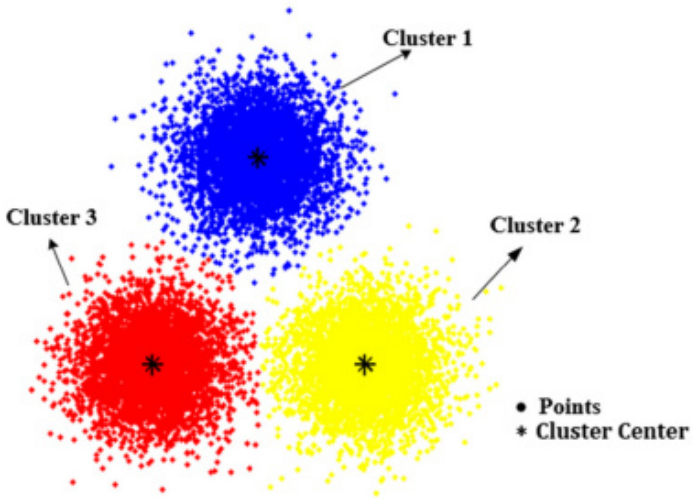


Figure 1.8: K-means example [7].

G. Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality-reduction algorithm, that is used to decrease the number of features of large datasets, by transforming these features into a smaller set that contains most of the information in the large dataset. PCA contributes by simplifying the dataset, sacrificing accuracy, as part of the information is lost.

To identify the principal components, PCA calculates the covariance matrix and then its eigenvectors and eigenvalues. In this way, sorting the eigenvalues from greatest to smallest we obtain the principal components ordered. By doing this, PCA gives more importance to the directions of the data that explain maximal amount of variance [56].

Once the principal components are calculated, PCA projects the dataset into the first k dimensions, being k chosen by the user, giving as result a new dataset that contains the most information in fewer dimensions [57].

H. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) was introduced by Sir Rondal Fisher in 1936 and it is a method used to find a linear combination of features that separates classes [58]. This algorithm is normally used as a dimensionality-reduction method before classification.

Before performing the dimensionality-reduction, LDA assumes that all the features of the dataset are independent and have normal distributions, homogeneity of variance and covariance and multicolinearity [59]. Homogeneity means that LDA makes the assumption that the covariance matrices of all the classes are similar and multicolinearity means that the predictive power can decrease with the increase of correlation between two predictors.

LDA consists in three steps. First, the between-class variance is calculated, also known as separability. This is measured as the distance between the mean of the different classes C.

$$S_B = \sum_{i=1}^{C} N_i(\overline{x_i} - \overline{x})(\overline{x_i} - \overline{x})^T$$

Then, the within-class variance is obtained, defined as the distance between the mean and every sample of the class.

$$S_W = \sum_{i=1}^{C} \sum_{j=1}^{N_i} (x_{i,j} - \overline{x_i})(x_{i,j} - \overline{x_i})^T$$

Finally, a subspace that maximizes the separability and minimizes the within-class variance [60].

$$P = \underset{P}{\operatorname{argmax}} \frac{|P^T S_B P|}{|P^T S_W P|}$$

The difference between LDA and PCA is that PCA tries to find the subspace that preserves the most information possible, ignoring the classes that each observation may belong to, and LDA uses the class information to find the subspace that maximizes class separability, while being limited to only $\#classes - 1$ dimensions.

1.5 Project goals and requirements

In this Section, the Master Thesis goals and its requirements are explained.

1.5.1 Project goals

The goals of this Master Thesis are

- Design and building of a mechanical structure capable of supporting three cameras and purchase said cameras.
- Research and find the way to apply Neural Networks to the pose estimation problem and search for specific libraries on the topic.
- Process the obtained estimations of the hand's pose resulting in relevant indicators of the performance in the test.
- Use the processed data to build Machine Learning models that are able to differentiate between control subjects and patients.

1.5.2 Requirements

The following section will tackle all the issues related to the needs of the full system in order to totally fulfill the defined project goals.

A. Video Capture

In order to expect a good performance of the CNN in the task of estimating the hand's pose, all videos used to train, test and in production must be of similar characteristics. This is that they all should have similar light, the same number of frames per second and, most importantly, they should all be recorded from the same point of view.

With this outline in mind, a mechanical structure with attached cameras is proposed as a solution and its requirements are defined as:

1. The structure must be robust and allow the placement of the NHPT Kit easily.
2. The structure must be easily portable as it is intended to be used in different locations.
3. The system must be able to record the whole NHPT exercise from 3 orthogonal points of view in order to capture all possible information.
4. The cameras placement must not interfere in the performance of the exercise and in the recording of the videos.
5. The cameras must record at least 60 frames per second, although rates higher than 100 are desired.
6. All cameras must start its recording at the same moment and provide the same video characteristics, as time synchronisation is an important factor for the analysis of the video.
7. The cameras must have a common power supply system in the structure itself with the aim of charging them whenever it is needed.
8. All videos must be recorded in an individual external drive or, failing this, they must be able to be copied in one.

B. Hand's pose estimation

The problem of estimating the hand's pose in this project is to be tackled by the use of Deep Learning algorithms, Convolutional Neural Networks in particular.

With this in mind, the requirements of this part of the project are described:

1. The system must be fed with a video and output the hand's position through time, preferable in a .csv file.
2. Training and testing data must be clearly separated so the model does not overfit.
3. The model must be exportable. Open source libraries are preferred so that the model is easily used once trained.

C. Machine Learning algorithms

Finally, with the obtained data from the hand's pose estimating model, we can obtain some relevant parameters regarding the movement of the hand, analyse them and apply machine learning algorithms to it.

The requirements for this part of the project are:

1. Be able to extract relevant values from the hand's pose estimation .csv file, like hand speed, peg frequency, time spent placing a peg, among others.
2. Build multiple Machine Learning models in order to predict on the extracted data if the observation is from a control subject or a patient, i.e., classification.
3. Extract conclusions about the hand's extracted parameters regarding their relevance for this classification problem and discuss if the data is separable attending to their classes.

Chapter 2

Desing and Implementation

In this Chapter, the whole phase of design and implementation is reported.

First, the mechanical structure will be sketched and built, validating afterwards the already established requirements of Chapter 2. Then, some Deep Learning models will be designed, trained and evaluated in order to see which one fits best as a solution to the proposed problem. Finally, with the selected model, data will be extracted from multiple videos obtained from subjects from the Neurology unit of La Paz Hospital, and it will be analysed.

2.1 Mechanical structure

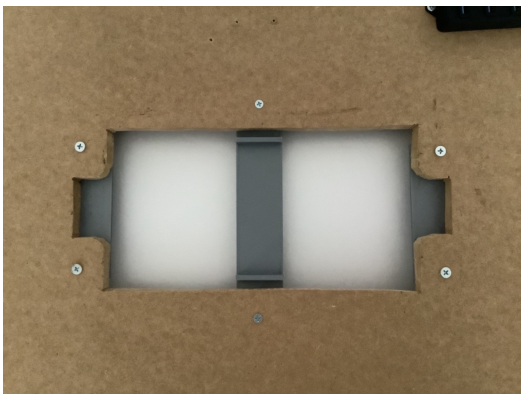
This section addresses the building process of the mechanical structure, from its sketch to its assemble.

To fulfill requirement 1 and 2 of the video capture, a PVC and wood structure is proposed. This structure will consist on a wooden and robust base, with four 3D-printed support legs for stability. The cameras will be attached to a PVC arch with 3D-printed strap-like pieces (Figure 2.1a).

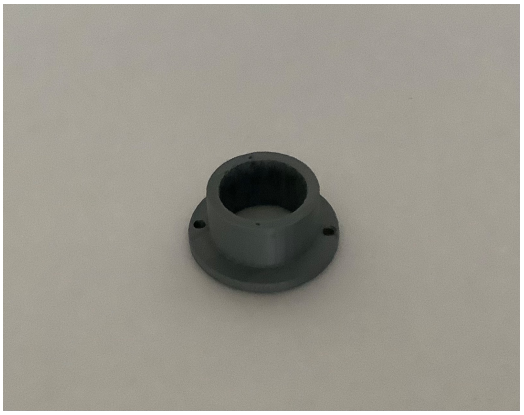
To complete this whole process, the shape of the NHPT will be carved into a wooden board using an electric drill. Once the shape is perfectly carved, three 3D-printed pieces will be attached to the base in order to support the NHPT board and prevent undesired movement (Figure 2.1b).

For the PVC structure, three more plastic 3D-printed pieces will be attached with two wood screws each to the wooden board (Figure 2.1c). These pieces will provide the holder for the PVC tubes. Then, with the help of three PVC elbows and a T, a three-legged arch is built on the wooden base. This arch is the place where our three cameras (Requirement 3) will be attached. Note in Figure 2.1d that there is enough distance between the PVC tubes and the NHPT board, not to interfere with the proper performance of the exercise (Requirement 4).

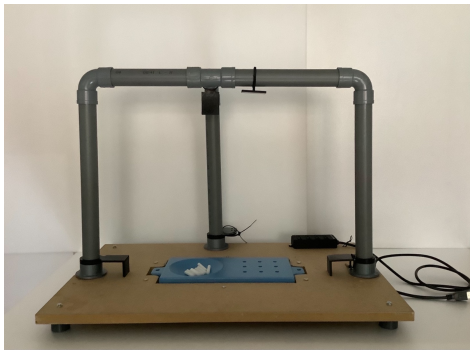
(a) 3D printed camera mount.



(b) Hole for the NHPT board with 3D-printed pieces attached.



(c) PVC attaching piece.



(d) Finished mechanical structure

Figure 2.1: 3-D printed pieces and the built mechanical structure.

2.1.1 Camera selection

For this objective, Requirements 5 and 6 will be taken into account. In a first attempt, webcams are considered as a possible option, as they could provide a centralised recording device and a real-time view of the captured frames. However, they are discarded, for the moment, for their low framerate capacity.

Our second and final option are the high-definition sport cameras. Although the activity that we will be using the cameras for is not a sport-type activity, these cameras will provide us with the resolution, framerate and capture angle that we need.

The chosen camera model for this Master Thesis is the GoPro Hero 7 Black and three of them are used (see Figure 2.2a). However, it is important to remark that one of the cameras was faulty and the provider made up for it replacing it for a GoPro Hero 8 Black (see Figure 2.2b). The specifications for this camera from the video capture point of view are nearly the same as the Hero 7 Black model. However, Hero 8 Black has a bigger RAM and a 'supersmooth' video capture mode.

(a) GoPro Hero 7 Black (b) GoPro Hero 8 Black

Figure 2.2: GoPro Hero 7 and 8 Black cameras [8].

	Hero 7 Black	**Hero 8 Black**
Cost	$\tilde{2}$50€	$\tilde{3}$00€
Weight	116g	126g
MP	12	
Aspect Ratio	16:9, 4:3	
Resolution	4K@60fps, 1080p@240fps, 720p@240fps	4K@60fps, 2.7K@120fps, 1080p@240fps
Battery life	45 min @4K60, 35 min @1080p240, 85 min @720p240	
RAM	78 Mb/s @4K	100 Mb/s @4K

Table 2.1: GoPro Hero 7 Black and Hero 8 Black specifications[8].

Attending to Table 2.1, we comply with Requirement 5 of video capture. Also, Requirement 8 is fulfilled as GoPro cameras store data in a removable SD card, that the user can then plug into other system and extract everything in it.

To satisfy Requirement 6, the GoPro Smart Remote is proposed (see Figure 2.3a). This remote can be connected to and control up to 50 GoPro cameras. Its maximum working distance is 180 meters and it is waterproof to 10 meters. This remote, will allow us to connect the 3 cameras before starting the exercise and to start the cameras all at the same time to guarantee synchrony. Also, once the cameras have been connected, they can all be turned on and off simultaneously.

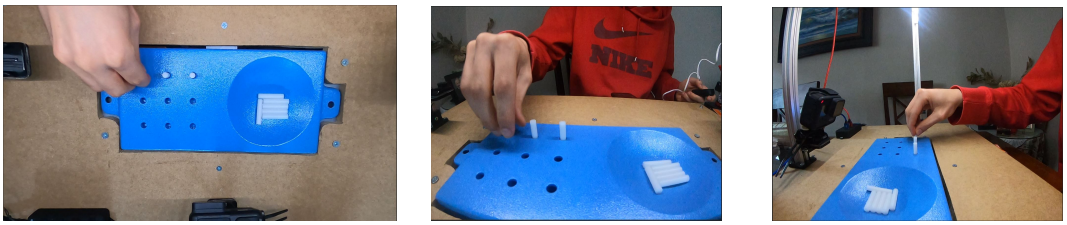
Finally, to meet Requirement 7, an USB hub with external power supply has been

installed. This device will allow us to charge the three cameras and the remote at the same time. It also grants the possibility of transmiting data at a speed of up to 5Gbps. Every port provides a 5V voltage with an up to 2A current. Not only Requirement 7, but this hub also opens a new way of satisfying Requirement 8, as it also provides a possible connection of the 4 ports to a computer with a USB 3.0 output. Additionally, three 1-meter USB 3.0 cables connect the cameras to the USB hub (see Figure 2.3b) [61].

(a) GoPro Smart Remote [8]. (b) VEMONT USB Hub 3.0 with 4 ports.[9].

Figure 2.3: GoPro Hero 7 and 8 Black cameras [8].

With all this, the video capture system is fully complaint with the Video Capture Requirements that were previously set. Therefore, the cameras are placed in their designed mounts resulting in the capture angle shown in Figure 2.4.



(a) Zenital camera frame. (b) Frontal camera frame. (c) Lateral camera frame

Figure 2.4: Camera positioning and their captured frames.

2.2 DeepLabCut and Hand's pose estimation

This section will discuss the software system that will be used to estimate the hand's position in each frame of the recorded videos. As aforementioned, Convolutional Neural Networks (CNNs) will be used for this case.

First, the chosen library to apply these CNNs to the videos will be explained in depth. Secondly, some models will be tested to fully understand the functioning of the software library and its models. Finally, a model will be selected considering the previous learnt information and it will be trained with the final videos of the subjects

taking the NHPT.

DeepLabCut (DLC) is a method for 3D markerless pose and movement estimation based on transfer learning and deep neural networks with minimal training data (50 – 200 frames) [62]. It is an open source package and it was developed by the Mathis Group and Mathis Lab at EPFL. DeepLabCut runs over Python, and it mainly uses Tensorflow, another open source library that provides an easy way to create, train and evaluate Machine Learning and Deep Learning models.

Although its functioning may be difficult to understand at the beginning, DLC is a very user-friendly environment: it has its own Graphical User Interface, which means that there is no need for writing code; and it is so simple that it is not necessary to know about how Deep Learning or CNNs work, although it would really be of use.

2.2.1 Applications and advantages

DLC has multiple applications, always used to track user-defined body part locations in a defined framework. Most of current applications consist in mice behaviour monitoring, such as extracting the movement trajectory of the mouse or a more complicated hand articulation. It is also used for eye tracking of humans or for analysing human babies pose.

Pose-estimation is one of the classical computer vision problems, whose limitations have been greatly reshaped by Deep Learning [63]. Once Deep Learning is the chosen approach, there are multiple factors to consider before selecting an algorithm: the speed, the accuracy and the required amount of data.

DLC was designed to need little training data and be as accurate as human annotators [64]. Unlike the others, DeepLabCut uses an algorithm called DeeperCut, which was addressed as one of the most powerful algorithms for pose estimation in the recent years [65]. It was also design for real-time processing being able to reach inference frequencies of 10-90 frames per second [64].

The main advantages of DLC are the aforementioned non-necessity for a big training dataset, the minimization of the cost of manual behaviour analysis that can reach human-level accuracy and it also eliminates the need for visible markers on the desired points of interest. It can also be easily adapted for any other animal species and it is open source and free of cost [66]. DLC can robustly extract the body parts position despite changes in the background, illumination or camera distortions, which makes the whole process easier for the user as he/she will not need to design their tests attending to framework constraints.

2.2.2 Neural Network Types

Although there are several types of convolutional neural networks types, DLC uses only two of them: ResNets and MobileNets.

MobileNet is a class of CNN open-sourced by Google for mobile devices that uses depthwise separable convolutions. This results in a reduced number of parameters in comparison to nets with the same depth. This depthwise separable convolution consists in two operations. First, a depthwise convolution is performed, which is a map of a single convolution on each input separately. This operation relies on the

depth and spatial dimension separability of the filter, that allows us to reduce the number of parameters. Then, a pointwise convolution (1x1 filters) is performed to change the dimension of the data. MobileNets are designed to effectively maximize accuracy while taking into account the restricted resources of a mobile device.

On the other hand, ResNets or Residual Nets creates connection between non-consecutive layers called "skip connections". An additional weight matrix is used to learn the skip weights. The main reason to add this connections is that when adding more layers to a model, the training error increases (degradation). Another very important reason is to avoid the vanishing gradient problem. The vanishing gradient problem apears in deep neural networks in backpropagation. In this step, each layer receives the update proportional to the gradient of the loss function, which in very deep networks may be not big enough to change the weight value. The addition of these connections does not hurt the performance of the convolutional network because its ability to learn the identity function [67].

The available networks in DLC are: mobilenet_v2_0.35, mobilenet_v2_0.5, mobilenet_v2_0.75, mobilenet_v2_1.0, resnet_50, resnet_101, and resnet_152. All these models are pretrained and available in the TensorFlow Keras library.

2.2.3 Nine Hole Peg Test

DLC runs over the following pipeline, the user starts by creating a blank project and performs an initial selection of videos to be used as training dataset. After that, DLC extracts automatically the frames of the videos and enables the user to select the most relevant ones manually or automatically by an embedded K-means algorithm. Then, the user labels these most relevant frames and they are separated into the training and testing datasets. Eventually, a pre-trained CNN is retrained with the corresponding dataset and its performance is evaluated with the testing dataset. If the performance is satisfactory, the model can be used to analyse new videos with the same characteristics and get the predicted positions of the labeled bodyparts. In the case that the performance doesn't comply with the desired result, a refinement step can be performed to extract the frames with poor results of the dataset.

To explain how the whole process of building and training a model in DeepLabCut is done, an installation and user guide is developed and annexed to this report (see Annex C).

The aim of this Master Thesis is to be able to compare the performance of some control subjects and some patients in the La Paz Hospital. With this goal in mind, the mechanical structure was taken to the hospital and a member of Robolabo controlled the capture of all the videos.

As explained above, three cameras were used to capture the movement of the hand. However, it is observed that most of the information is stored in the zenital camera videos as it shows properly the movement over the horizontal axis. Therefore, this Master Thesis only focuses on the analysis of the zenital videos. Also, two different models will be created, one for each hand. By doing this, the performance of the models is expected to be better than just creating one model for both hands. Both models will be created with the same characteristics and trained with the same

subjects.

First, the projects were created and 10 videos were added to them. These videos capture the performance of the control subjects as they were the only videos available.

The configuration files were modified and five parts of the hand are to be detected: center of the hand, index knuckle, middle finger knuckle, ring finder knuckle and little finger knuckle. For abbreviation, we will refer to them as 'center', 'index', 'middle', 'ring', and 'little'. The skeleton is also defined as the union of every adjacent point and the endpoints with the center. The number of frames to pick was set to 40 and the net type to resnet_101 to add more complexity to the model than the resnet_50.

After the configuration of the projects, data was extracted both manually and automatically, half of the videos each, to see the differences of the results. In the manual process, frames were selected attending to their considered importance. Several frames were the hand is taking or leaving a peg are selected. Frames of the hand in the midle of both actions are also selected. This criterion tries to capture the most important phases of the activity and with them, to be able to model the whole movement. The automatical method was performed with the K-means algorithm, taking a k value of 40 and extracting the cluster centers.

Once the frames are selected, they are labeled using the DLC labeling toolbox (see Figure 2.5). In this case, the most complicated label is the center of the hand, as it may not be a fixed point for every patient's hand. However, an attempt has been made to fix the label always at the same point. Later, the frames are checked for wrong labels, since an incorrect one would trigger a systematic error in the prediction. Finally, the training dataset is created and the training is started. The training process was carried out in one of the Robolabo's machines with a GeForce RTX 2080 Ti with a memory of 11GB GDDR6 [68]. The required 250,000 iterations of training were carried out in no more than 7 hours.

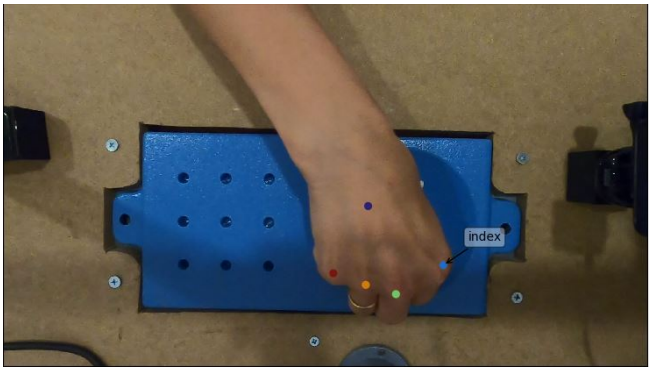


Figure 2.5: Labeled frame of a hand performing the NHPT.

After training, the model was evaluated and the results in Table 2.2 were obtained.

These networks gave considerably good results and was considered to be good for

	Right hand model	Left hand model
Training error (px)	3.38	2.72
Test error (px)	32.4	22.76
p_{cutoff}	0.6	0.6
Test error with p_{cutoff} (px)	19.49	15.25

Table 2.2: Evaluation results for the first training process.

production. Yet, the recording video characteristics changed, as a zenital light was added to the mechanical structure to improve vision quality and avoid shadows. Then, the networks had to be retrained adding new frames with this new light conditions. The expected result of this addition is to get light-unbiased results with the new model. In consequence, 10 videos of each hand of actual MS patients are added to the projects to provide enough new data for the network to recognise the movement in these new conditions. After training for 200,000 iterations more, the obtained evaluation results are the ones showed in Table 2.3. It is clear to see that the training error and test error is almost the same, although both have been reduced by some pixel units. This decrease means not only that the network has learned to predict the positions in these new conditions, but also that it has improved the previous performance of the network. However, paying attention to the filtered test error, the performance on the most-likely predictions ($p > 0.6$) has decreased a little bit, which leads us to infere an increase of the error in the most reliable predictions.

	Right hand model	Left hand model
Training error (px)	2.87	2.72
Test error (px)	28.37	22.76
p_{cutoff}	0.6	0.6
Test error with p_{cutoff} (px)	22.22	15.25

Table 2.3: Evaluation results for the second training process.

These error values consider the five bodyparts that we tried to track to calculate the error. Then, it is important to remark that the center of the hand and the little finger and ring finger knuckles are the greatest contributors to this error. This can be infered looking at one of the CSV output files when analyzing a video, where the likelihood of that bodyparts are significantly lower than the other two (see Table 2.4). The reason of this is that the the ring finger and the little finger aren't always on frame, like the index and middle ones, leading to a false prediction as the bodypart is not visible.

Bodypart	Likelihood
Center	0.79
Index finger	0.97
Middle finger	0.95
Ring finger	0.77
Little finger	0.56

Table 2.4: Mean likelihood for the predictions of Control Subject Number 1.

Taking into account the previous data, the network is considered to be good enough for starting with the video analysis. For this, a small Python script is developed in the Robolabo computer, due to its high computing capacity. This script is in charge of detecting all the videos in a selected folder and analysing them, saving the obtained data in CSV files in the same folder. Once analysed, they are copied into a different computer where the processing of this data will be performed.

2.3 Data preprocessing

In this Section, the preprocessing phase of the data will be explained.

First, we need to explain what kind of data is exported by DeepLabCut. The CSV file that DLC exports contains 3 columns for each bodypart that was tracked, two for the (x, y) coordinates and one for the likelihood of the prediction. Rows contain the values of each video frame. The (x, y) coordinates have their origin on the top left corner of the frame, being the horizontal axis the one referencing the x and the vertical references the y, increasing its value from up to down.

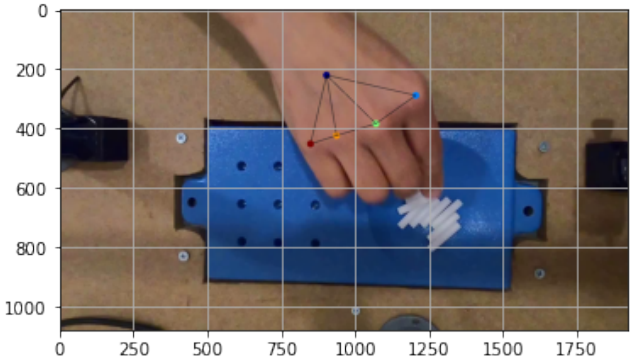


Figure 2.6: DeepLabCut predictions and skeleton for a frame.

Bodypart	x	y
Center	901	223
Index finger	1203	290
Middle finger	1068	385
Ring finger	935	425
Little finger	847	453

Table 2.5: Coordinates of the predicted bodyparts in the right hand of control subject 10.

The likelihood value, will enable us to filter the values and taking them only if the are reliable. The likelihood threshold is chosen empirically to be 0.7.

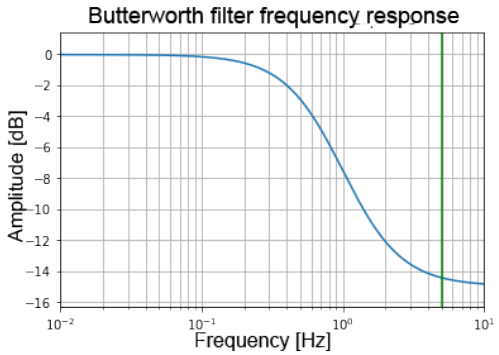
As aforementioned, the most realiable points are the index finger and the middle finger, which will be the ones that we would use for our analysis. However, as they are nearly the same point in the hand, the middle finger is discarded and only the index knuckle will be analysed. In addition to this, the y coordinate is also considered to be irrelevant as the movement appears only in the horizontal axis.

To smoothen the curve of the index's movement, a Butterworth filter is designed. First, the cutoff frequency is to be defined. Analysing the videos, it is observed that the movement of the hand in the horizontal axis is not higher than 2 Hz, i.e., no more than two pegs per second. So for this matter, the cutoff frequency is set in 5 Hz (see Figure 2.7).

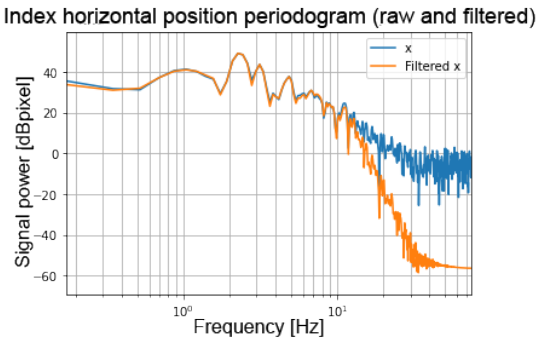
Once the signal is filtered, the middle of the movement in the horizontal axis is calculated. This is done by taking the mean between the maximum and minimum values of the movement, in order to find the point where the hand changes from the peg container side, to the holes side. With this information, the subject is considered to be in the container side if the predicted value is over the middle and in the hole side otherwise. After that, in the first part of the exercise, once the subject is in the container zone and reaches a maximum in the horizontal axis, it is considered that he/she has grabbed a peg. The same deduction is made with the hole zone and a minimum. Once the movement has performed 9 periods, i.e., the 9 pegs have been placed in the holes, the first part of the exercises finishes and the second one starts.

In this way, we can separate four states in each of the two parts of the exercise (see Figure 2.8). However, it is considered that beyond the speed of the hand, we cannot extract relevant information from the second part of the exercise, because most of the subjects throw the peg back into the container instead of leaving them.

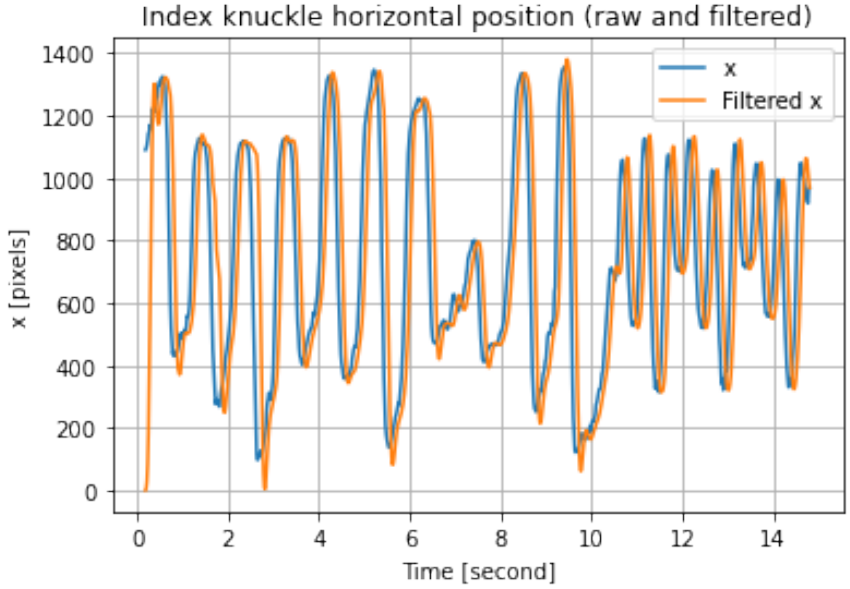
Afterwards, some relevant measures of the movement are calculated, such as the peg mean period, the speed of the hand's movement, the time that the subject takes to finish the exercise, or the time that the subject uses to take or leave the peg once he/she is in the correspondent zone. The latter is calculated by taking the time that the subject is inside 200 pixels (4 cm) radius from the maximum or minimum until it is reached (see Figure 2.9). To extract all this information, a Python script that performs all the calculations for every CSV file is developed.



(a) Butterworth filter $f_{cutoff} = 5Hz$.



(b) Raw and filtered signals in frequency domain.



(c) Raw and filtered signals in time domain.

Figure 2.7: Butterworth filter and filtered signal.

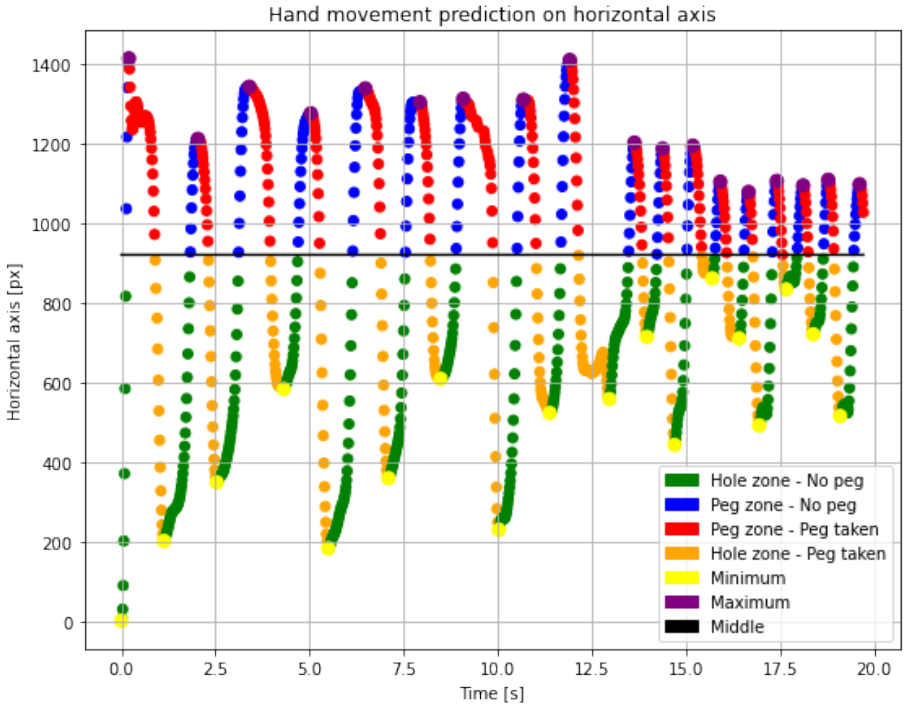


Figure 2.8: Horizontal movement prediction in the right hand of Control Subject 10.

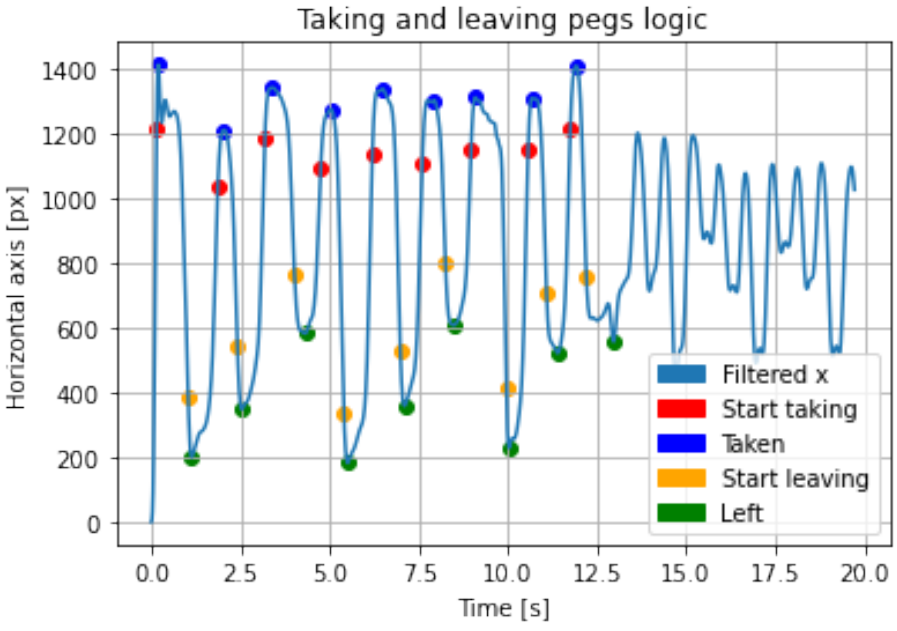


Figure 2.9: Taking and leaving pegs logic.

Finally, with the help of the information provided by the La Paz Hospital, we can add to our dataset the gender, the age and the dominant hand of each patient, which we will use to transform our dataset from a right/left hand separation to a

dominant/non-dominant hand separation.

2.4 Exploratory Data Analysis

In this Section, the Exploratory Data Analysis (EDA), where possible relationships between the features will be explored.

The first step in this phase is to explore the kind of data we are dealing with. In order of not losing any information, some additional values were extracted from the position predictions, such as the standard deviation of every mean value, or maxima and minima of the peg period or the speed.

The dataset is composed by the following features (see Figure 2.10):

- **Subject**: Letter and number of the subject identification, 'C' if it is control subject of 'P' if it is a patient.
- **Peg_period_dominant**: Mean, standard deviation, maximum and minimum values of the peg placement period. It measures in seconds the time between two consecutive peg placements of the dominant hand.
- **Peg_period_nodominant**: Peg period for the non-dominant hand.
- **Velocity_dominant**: Mean, standard deviation, maximum and minimum values of speed of the dominant hand's movement in the horizontal axis.
- **Velocity_nodominant**: velocity of the non-dominant hand.
- **T_total_dominant**: performance's total time measured in seconds.
- **T_total_nodominant**: T_total for the non-dominant hand.
- **T_reaction_dominant**: magnitude that measures the time elapsed between the start of the exercise and the first peg grabbed by the dominant hand.
- **T_reaction_nodominant**: T_reaction for the non-dominant hand.
- **T_take_dominant**: time elapsed between entering a radius of 4cm from the peg container and the capture of the peg measured in seconds for the dominant hand.
- **T_take_nodominant**: T_take for the non-dominant hand.
- **T_leave_dominant**: time elapsed between entering a radius of 4cm from the holes zone and the release of the peg measured in seconds for the dominant hand.
- **T_leave_nodominant**: T_leave for the non-dominant hand.
- **Gender**: subject's gender, 'H' for men and 'M' for women.
- **Age**: subject's age measured in years.

- **EM**: Appearance of Multiple Sclerosis. '1' if the subject is a patient and '0' if it is a control subject.

- **SDMT**: Number of symbols completed by the patient in the SDMT in 90 seconds. Will only be used to separate patients between different classes of MS. Only patients with MS have this score.

Figure 2.10: Complete dataset sorted by type.

Table 2.6 shows the representation of the statistical summary of the data. There are 74 observations, which seems partially constraining for a Data Science project, as data volume is one of the most important requirements. However, a full Machine Learning framework will be designed and carried out in order to extract preliminary conclusions regarding the relevance of the NHPT movement features to classify Multiple Sclerosis.

Regarding the information of Table 2.6, data is biased on the men and control subjects side. We can also see that the maximum peg frequency is never over 1Hz, which confirms our assumption for the filter in Figure 2.7. A notable difference between the total time of the exercise between the dominant hand and the non-dominant one can also be observed.

After analysing the variables, it is decided to keep only the mean value of the features and discard the rest. And to see the relationships between features, a pairplot for each of the hands is arranged. However, for space reasons, only the features that maintain a clear relationship with another will be showed in this document (see Figure 2.11). Beyond the trivial relationships like peg_period and T_total or T_total and T_first, we can extract some conclusions such as there is a linear relationship between the taking time and the leaving time, which would mean that every subject that struggles taking the peg, will also struggle leaving it in the hole and viceversa.

	Peg period D	Peg period non-D	Velocity D	Velocity non-D	T total D	T total non-D	T reaction D
Min	1.08	1.09	16.01	13.43	9.35	6.85	0.09
Q1	1.34	1.45	27.96	26.7	11.46	12.52	0.21
Median	1.46	1.58	31.39	29.43	12.28	13.63	0.28
Mean	1.55	1.85	32.56	29.77	13.47	16.03	0.33
Q3	1.61	1.82	38.23	33.26	13.92	16.1	0.36
Max	4.9	9.28	47.01	46.61	43.48	83.21	1.86

	T reaction non-D	T take D	T take non-D	T leave D	T leave non-D	Age
Min	0.06	0.06	0.07	0.06	0.07	21
Q1	0.14	0.09	0.35	0.09	0.35	28
Median	0.26	0.18	0.43	0.18	0.44	35
Mean	0.36	0.25	0.56	0.25	0.56	38.04
Q3	0.49	0.29	0.65	0.29	0.65	46
Max	1.64	1.87	1.63	1.87	1.63	64

Gender			EM
Men	45	Patient	27
Women	29	Control	47

Table 2.6: Dataset relevant statistical values.

There is also an exponential relationship between the velocity and the Total time of the exercise.

To actually see the relationships between features, a correlation heatmap is plotted (see Figure 2.12). Although it is relevant to search for relationships between features, we want to search for features related to the existence of Multiple Sclerosis, as, after all, we are in a classification problem. In this way, we will only pay attention to its column (see Figure 2.13). Here we can see that the most correlated features to EM are peg period and the total time for both hands and the time spent taking a peg and the reaction time of the dominant hand. To finish the EDA phase, we will represent the values of these features in a boxplot, separating them by the 'EM' class (see Figure 2.14). In these plots, we can see a clear difference between classes, yet the boxes are still overlapping in every case. This overlapping leads us to think that although differences between classes seem likely, they might not exist. We can also see the existence of some points considered outliers for being away 1.5 times the size of the box [69].

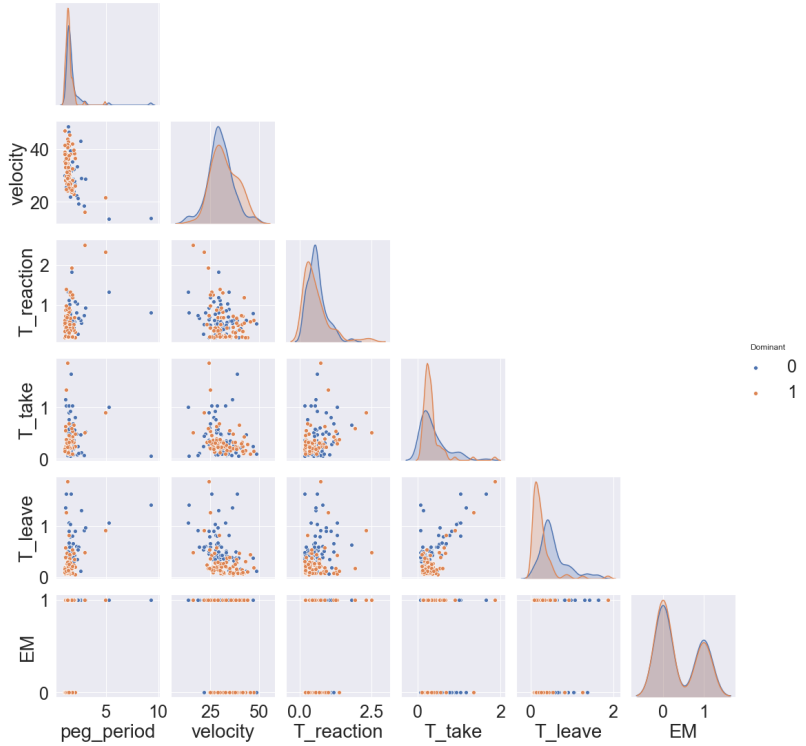


Figure 2.11: Pairplot of both dominant and non-dominant hands.

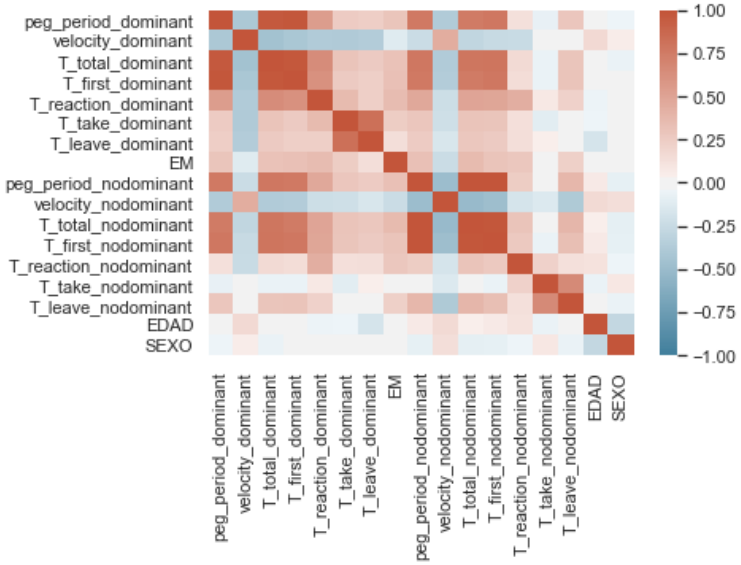


Figure 2.12: Correlation heatmap for all features.

2.5 Machine Learning

In this Subsection, a ML framework to design and create models will be explained. This framework will be divided into two phases. The first one will tackle the

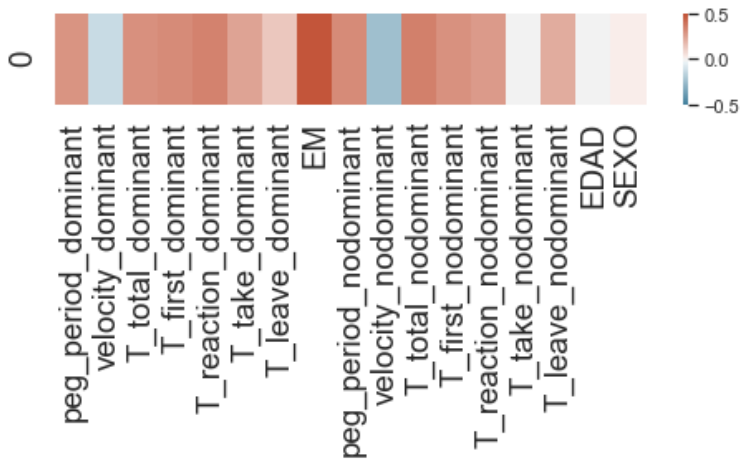


Figure 2.13: Correlation heatmap for EM feature.

dimensionality problem. The second one will consist on creating ML models two solve two different classification problems: a binary classification and a multiclass classification. Additionally, to recreate the results obtained in this Section, the steps explained in Appendix D can be followed.

2.5.1 Binary Classification Problem

The binary classification problem consists in assigning to a subject one of two classes by means of a ML model. The two classes that we will use in this exercise are the Control Subject class and the Patient class.

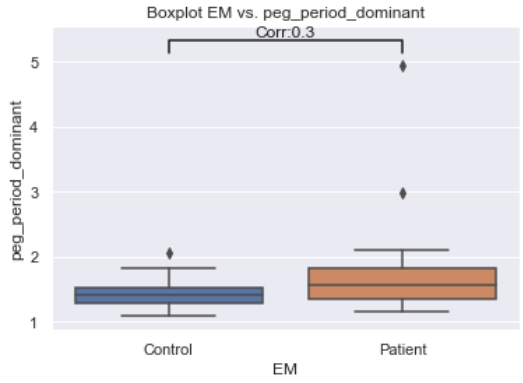
A. Dimensionality Reduction

In Subsection 2.4, the correlation of every feature was calculated and the most correlated ones were selected. However, for this exercise we will use the whole dataset, not to lose any possible information.

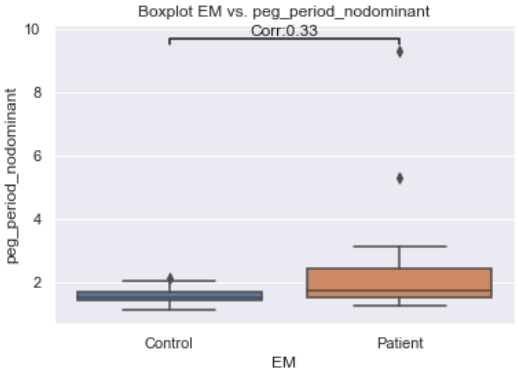
Two Dimensionality Reduction algorithms will be used: Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). As explained in Chapter 1, PCA is an unsupervised algorithm and searches for the Principal Components where the data will keep most of the variance when projected. To be able to compare it with LDA, a supervised method where the search components are the ones that maximize the variance between classes, PCA will be performed with the whole dataset, and both classes separately.

Starting with the PCA, we want to see the differences between the obtained datasets after projecting the original data onto the principal components. First, a PCA model is trained with the whole dataset and it is fully projected to these components, obtaining the observations shown in Figure 2.15a. Then the same process is repeated but the new two models are trained only with the data of the control subjects and the patient subjects separately. After that, the whole dataset is projected into these new subspaces and we obtain Figures 2.15b and 2.15c.

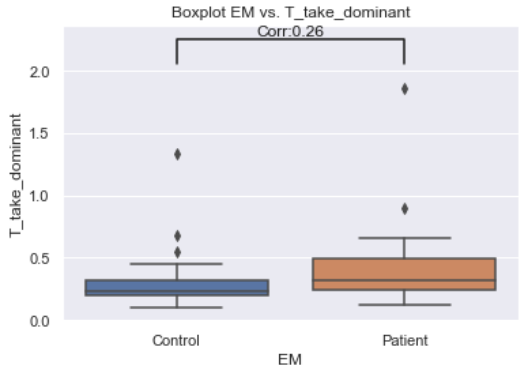
There is a lack of observable differences between all the plots in Figure 2.15.



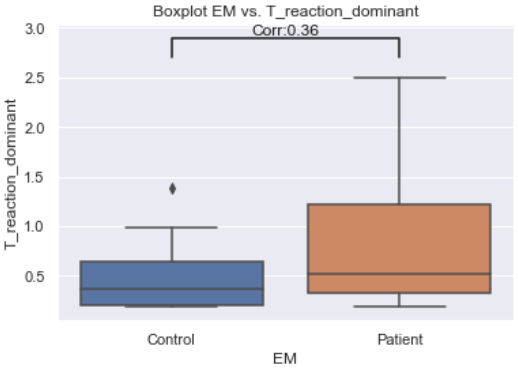
(a) Peg period dominant hand boxplot.



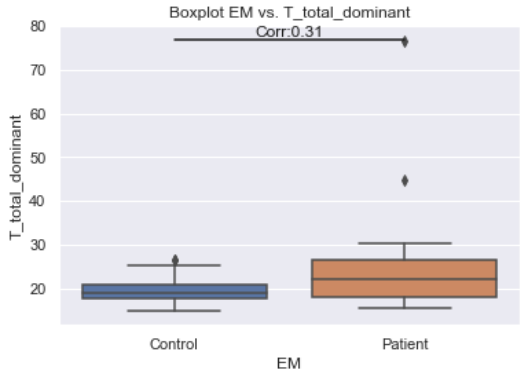
(b) Peg period non-dominant hand boxplot.



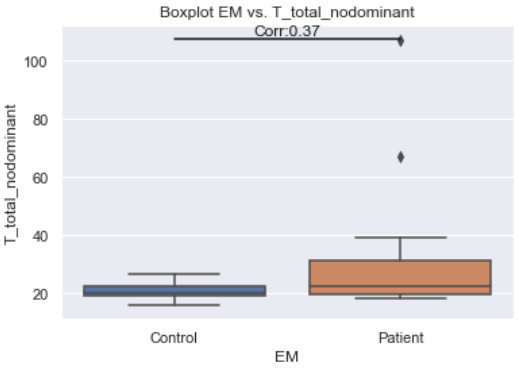
(c) Time of peg taking dominant hand boxplot.



(d) Time of reaction dominant hand boxplot.



(e) Total time dominant hand boxplot.



(f) Total time non-dominant hand boxplot.

Figure 2.14: Boxplots of the most correlated features to the 'EM' class.

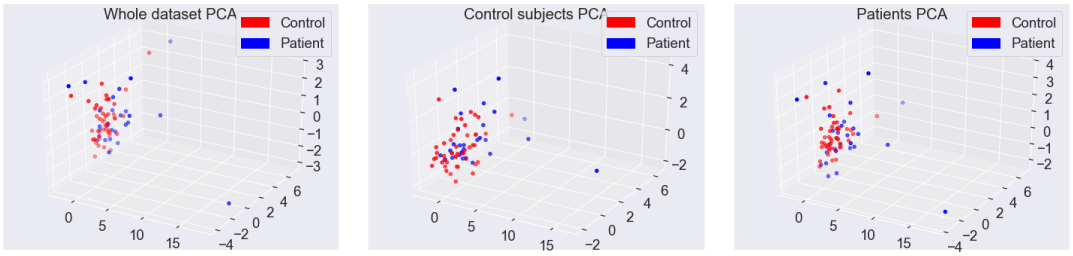
The first plot shows that all the information is equally distributed through the axis. However, in the other two plots, the training data does not has more variance through the components than the other class respectively. This leads us to think that there may not exist subspaces that maximize the variance in just one class, sugesting that both classes may not be different, and therefore, not separable. These lack of differences can be seen clearly in Figure 2.16. It is also important to see the 'Explained Variance Ratio' (EVR) of the three performed PCAs, that shows us the percentage of the original variance that is attributed by each of the selected components [70].

EVR	PC1	PC2	PC3	Total
Whole dataset	41.6%	11.9%	10.4%	63.9%
Control	31.2%	14.8%	13.9%	59.9%
Patient	42.9%	13.0%	10.5%	66.4%

Table 2.7: Explained Variance Ratio for each of the three PCA models.

Looking into Table 2.7, we can see that the first Principal Component of both the whole dataset and the patient dataset keeps around a 42% of the total variance of the dataset, unlike the control dataset's PCA, that keeps only a 31%. Although it is important to try and maximize the total percentage of variance that contains the three dimensions, we will focus in the first component to compare it with the LDA performance.

PCA can also be used to visualize where the information (variance) of the dataset resides. This can be seen by checking the coefficient values of the PCA first components, i.e., the values of the dataset first eigenvectors. In Figure 2.17, the highest coefficients of the first principal component are plotted. Here we can see that the greatest coefficients in absolute value are the peg period and total time of both hands, the time of reaction of the dominant hand and the velocity of the non-dominant hand. However, although in this features reside the most information of the dataset, they do not have to be the best features to use in a classificator, as PCA does not consider the class.



(a) PCA of the whole dataset as training data.

(b) PCA of the control subjects dataset as training data.

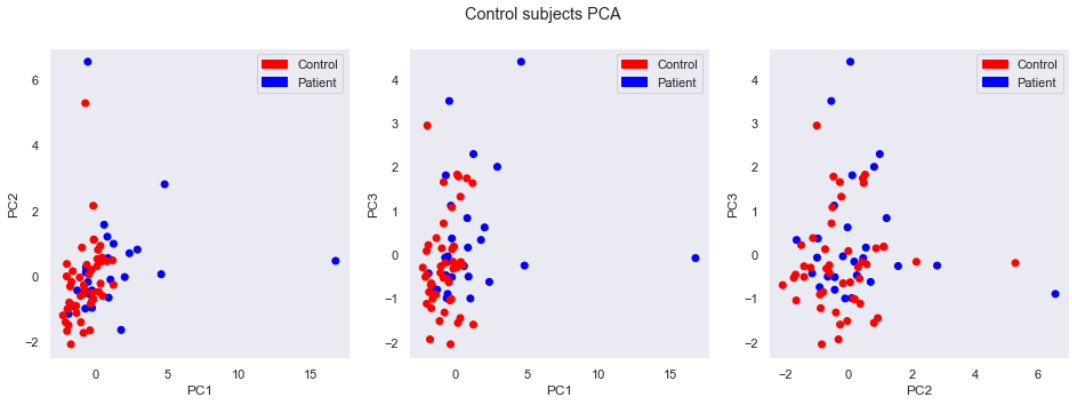
(c) PCA of the patient subjects dataset as training data.

Figure 2.15: 3D representation of the dataset after performing PCA algorithm.

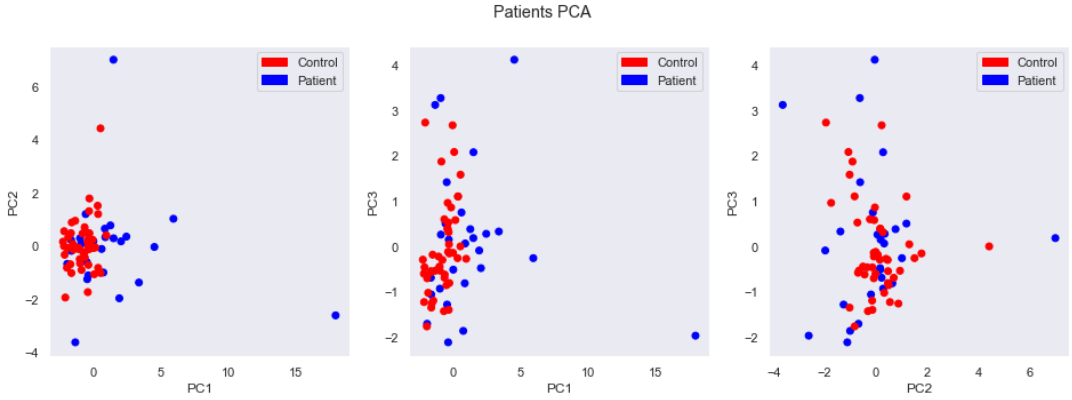
After finishing the whole PCA exercises, we will continue with the LDA. As LDA is a supervised method, we must try to avoid overfitting, by separating our dataset into training and test dataset. To do this, we will use the K-Fold Cross Validation method, that will allow us to separate the dataset, and train and evaluate the model.

K-Fold Cross Validation is a method that estimates the actual performance of a Machine Learning model. It separates the dataset into k different groups. Then, for each group, takes the group as test dataset and trains the model on the other $k-1$ groups, saving the score of the model. Finally, it gives an overall score to the model as the average of all k iterations. Aditionally, a parameter is set to always introduce

(a) PCA of the whole dataset as training data.



(b) PCA of the control subjects dataset as training data.



(c) PCA of the patient subjects dataset as training data.

Figure 2.16: 2D projection of the 3 Principal Components of the dataset after performing PCA algorithm.

the same amount of observations of each class, so that the model is not trained on a biased dataset.

Once the LDA model is properly trained with $k = 10$, data is projected into its unique dimension, as we have only two classes (see Figure 2.18).

Like PCA, LDA can also be used to visualize the most important features in the dataset, with the additional advantage that this features are the ones that maximizes

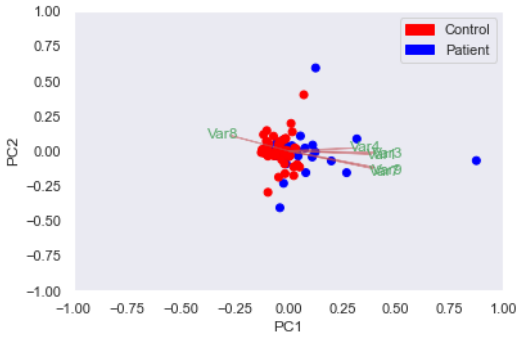


Figure 2.17: PCA most important first component coefficients of the whole dataset.

Feature	Coefficient
Peg_period_D	0.3819
T_total_D	0.3990
T_reaction_D	0.3113
Peg_period_non-D	0.3895
Velocity_non-D	-0.2621
T_total_non-D	0.3984

Table 2.8: Greatest coefficients of the first principal component.

the variance between classes. In Table 2.9 the coefficients of the six most important features are gathered. With this information we can confirm that LDA gave more importance to the features with the highest correlation with the Multiple Sclerosis class, the ones that were selected in Subsection 2.4.

T total non-D	Peg period non-D	Peg period D	T total D	T reaction D	T take D
10.75	-10.27	3.02	-3.00	0.78	0.60

Table 2.9: LDA six greatest coefficients.

To finish the Dimensionality Reduction Phase, we store all the projected datasets into different variables to compare the performance of different models with different datasets.

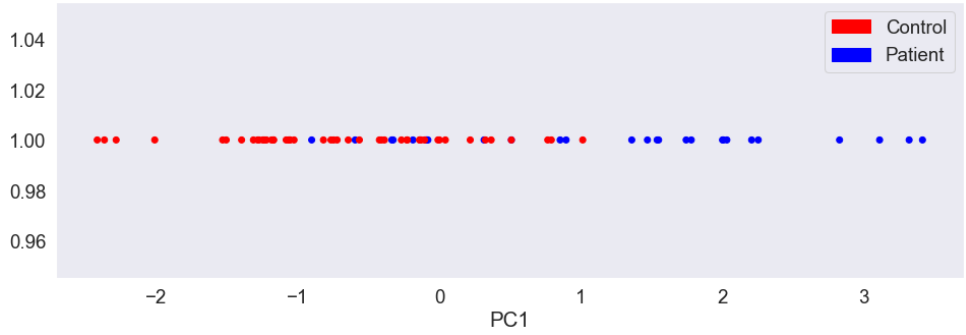


Figure 2.18: 1D projection of the dataset in the LDA component.

B. Model Implementation

We have prepared different datasets to train our models for a classification problem. The implementation of these models will be now explained.

Four different types of models will be built to classify our data between the Control Subject class and the Patient class: Logistic Classification, Support Vector Machines, K-Nearest Neighbors and Random Forest. All these models will be trained using a 10-Fold Cross Validation method.

The datasets used for training will be:

- Original dataset.
- 3-dimensional PCA trained with the whole dataset.
- 3-dimensional PCA trained with the control subject dataset.
- 3-dimensional PCA trained with the patients dataset.
- 1-dimensional PCA trained with the whole dataset.
- 1-dimensional PCA trained with the control subject dataset.
- 1-dimensional PCA trained with the patients dataset.
- 1-dimensional LDA dataset.

To build the models, some parameters will be tuned to be compared and to obtain the best possible accuracy of the model. In the case of the Logistic Classifier, the C penalization parameter will be tuned. This value will increase the regularization strength in training while decreasing and otherwise while increasing. In SVM, C will also be tuned alongside with the kernel type. In KNN, different values of k will be used and we will se the differences between the Manhattan distance and the Euclidean distance performances. Finally, in Random Forest, we will tune the number of estimators and the maximum depth of the decision trees that compose it.

Results will be observed and compared in Chapter 3.

2.5.2 Multiclass Classification Problem

In this case, instead of using two classes, we will try to decide a class among four. To define these four classes, we will draw upon the data provided by the Hospital and use the results of the Symbol Digit Modalities Test (SDMT) to separate the Multiple Sclerosis patients into three degrees of the disease, obtaining a total of four classes. The SDMT results are measured in total number of symbols completed in a fixed amount of time. In order to obtain three classes from this califications, two thresholds have been set: cases with less than 30 completed symbols are considered severe (class 3), cases between 30 and 50 completed symbols are medium cases of MS (class 2) and over 50 symbols will be classified as mild (class 1). These three classes are added to the control subjects (class 0) and the exercise is started.

In this second problem, correlations do change notably. To check this, a fast EDA is done and the most correlated features are observed in a boxplot (see Figure 2.19

and Figure 2.20). This time, the most correlated features to the class are the peg period and total time of both hands, the reaction time of the dominant hand, and the velocity of the non-dominant hand. Additionally, we can see that the values of the correlation increase from a maximum of 0.37 to one of 0.54.

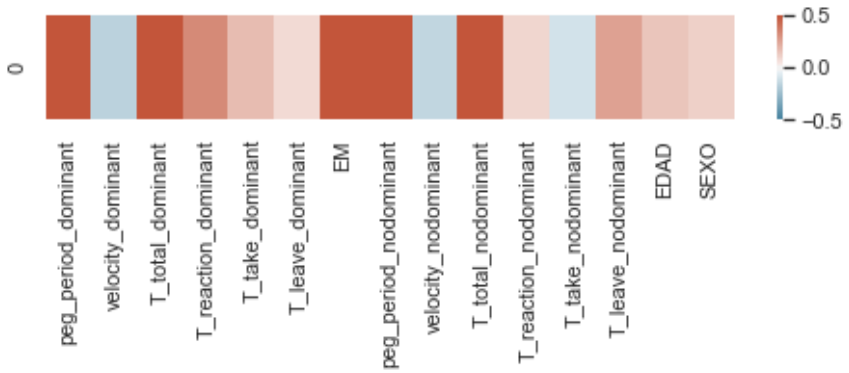


Figure 2.19: Correlation heatmap for EM feature.

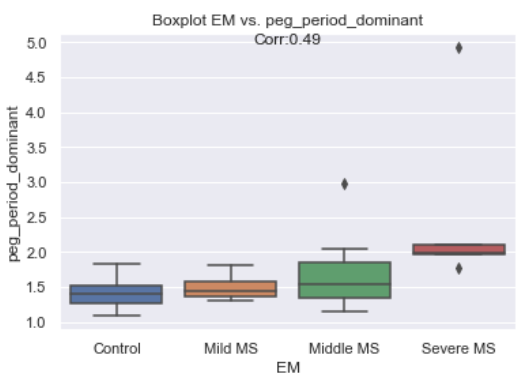
A. Dimensionality Reduction

For this problem, a simpler exercise than in the previous one is proposed: comparing the differences between models trained with the original dataset, a 3-dimensional PCA and a 3-dimensional LDA. Although LDA is expected to give a better performance, it is an interesting comparison of two dimensionality reduction variance-maximizing algorithms.

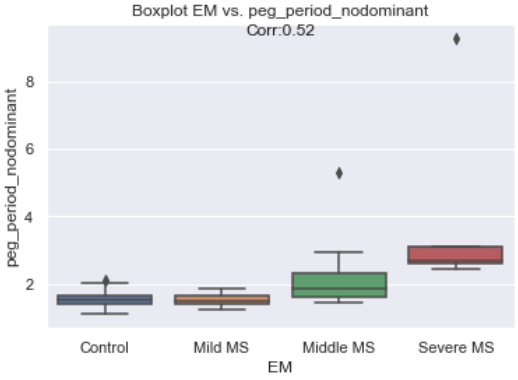
The same steps than in the previous section were followed, obtaining the projected datasets showed in Figure 2.21. Both projections seem very different, although they have something in common: separating the classes from one another will be challenging.

B. Model Implementation

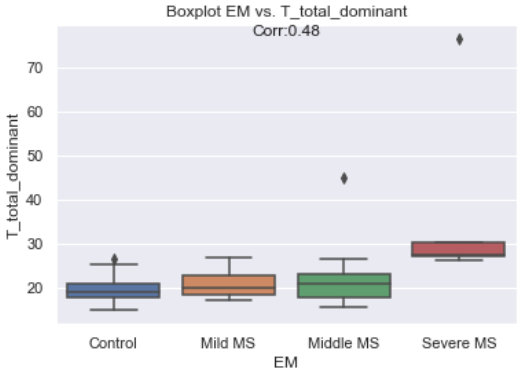
As well as in the binary classification problem, we will compare the performance of the models trained with different datasets, although at this time, we will use only three: the original dataset, the 3-dimensional PCA and the 3-dimensional LDA. The models that will be assessed are the same than in the previous section. Even though conclusions were extracted from the previous exercises regarding regularization, kernels, tree depth and number of neighbors, these exercises will be repeated to see the changes in performance when the training dataset has four classes instead of two.



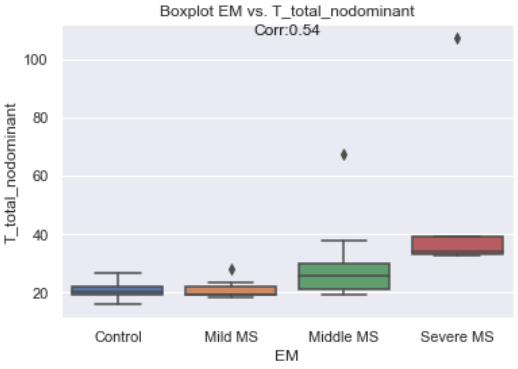
(a) Peg period dominant hand boxplot.



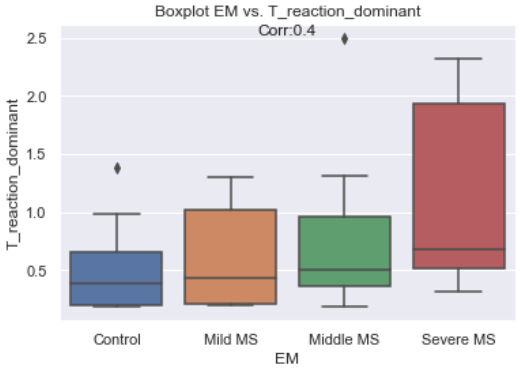
(b) Peg period non-dominant hand boxplot.



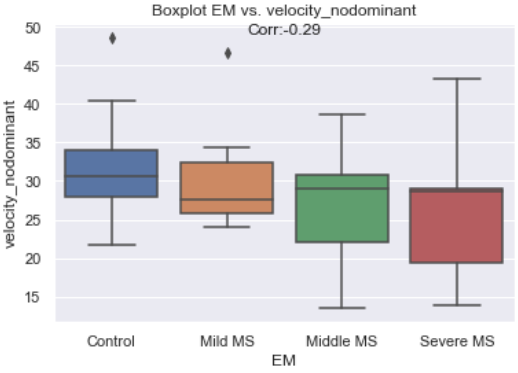
(c) Total time dominant hand boxplot.



(d) Total time non-dominant hand boxplot.

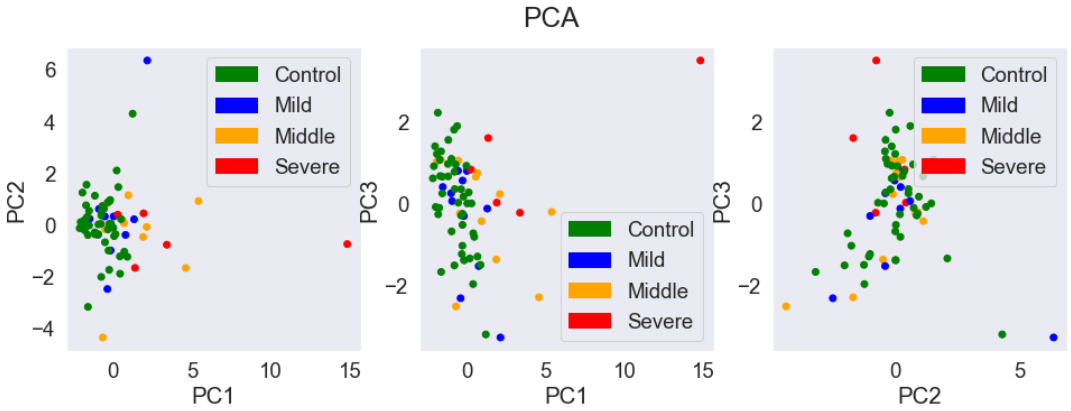


(e) Reaction time dominant hand boxplot.

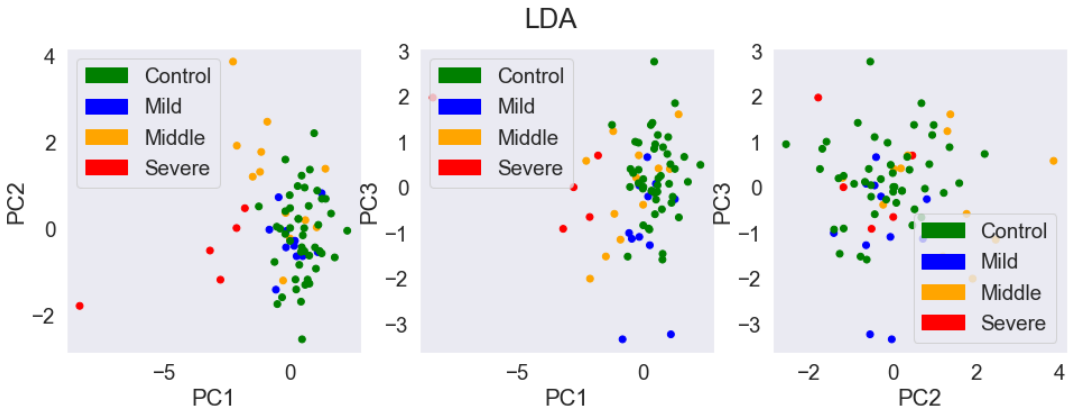


(f) Velocity non-dominant hand boxplot.

Figure 2.20: Boxplots of the most correlated features to the 'EM' class.



(a) PCA 2D projection on its 3 dimensions.



(b) LDA 2D projection on its 3 dimensions.

Figure 2.21: 2D projections of the PCA and LDA datasets with 4 classes.

Chapter 3

Results and model comparison

In this Chapter, the results of the trained Machine Learning models will be analysed, extracting conclusions on what is the model with the best performance and if classification between the Control Subject Class and the Patient Class can be done properly.

3.1 Results

To address which is the most suitable ML model for this classification problem, several metrics will be used in order to study their performances:

- **The ROC curve**: The Receiver Operating Characteristic Curve or ROC Curve is a graph showing the performance of a classification model at all classification thresholds. It has two parameters, True Positive Rate or Sensitivity and True Negative Rate or Specificity [71].
- **AUC**: Area Under the ROC Curve, and also represents the performance across all possible classification thresholds. It could be seen as the probability of the model ranking a random positive example higher than a random negative example.
- **Accuracy**: The ratio between the sum of the right guesses and the total guesses.

$$Accuracy = \frac{Right\ guesses}{Total\ guesses}$$

- **Precision**: The proportion of the right positive guesses and the total positive guesses.

$$Precision = \frac{Right\ positive\ guesses}{Right\ positive\ guesses + Wrongpossitive\ guesses}$$

- **Sensitivity or True Positive Rate**: The ratio between the right positive guesses and the total observations that are actually positive.

$$Sensitivity = \frac{Right\ positive\ guesses}{Total\ actualpositive}$$

- **F1 Score**: Metric that takes into account the Precision and the Sensitivity [72].

$$F1\ Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Sensitivity}}$$

- **Confusion Matrix**: 2x2 Matrix that has the right positive and negative guesses on the main diagonal, and the wrong in the other diagonal.

$$Confusion\ Matrix = \begin{pmatrix} True\ Positive & False\ Positive \\ Flase\ Negative & True\ Negative \end{pmatrix}$$

3.1.1 Binary Classification Problem

In this Subsection, the four proposed models for classification will be tested with the binary classification problem, while tuning their parameters to obtain the best possible model. It is important to remember that as we are classifying observations into two classes, the probability of guessing the class by chance is 50%. Therefore, the performance of our models shall be notable over this value so that we can consider them good enough.

A. Logistic Classification

A Logistic Regression Classifier does not have too many parameters that can be tuned: the solver, the type of norm used for regularization and the C regularization value. The first parameter references the algorithm that will be used to solve the optimization problem. This value will be fixed to 'liblinear' as it is the best one for small datasets [73]. The second value is the regularization method to be used while training. However, as we chose the 'liblinear' solver, only the Ridge regularization method is allowed. In Ridge regularization we add a penalty proportional to the square of the model coefficients. This constraint will lead to low variance, as some coefficients may have no influence in prediction, and low bias, as low coefficients reduce the dependency of the prediction in a particular variable. The value of C is the inverse of the regularization strength, which means that a smaller values specifies a stronger regularization. As the first two parameters were fixed, we only tuned the C regularization value.

Figure 3.1 represents the test accuracy of a Logistic Regression Classifier trained with all the datasets obtained in the previous section, while changing the value of regularization. Looking at the plot, it is easy to extract two main conclusions: in all cases, regularization helps increasing the accuracy and the model trained with LDA has clearly a better performance. We can also see that the next best models are the ones trained with the original dataset and the one-dimensional PCA of the whole dataset. The performance of the latter is surprising as a model trained with a PCA of one dimension seems to have a better performance than with three dimensions.

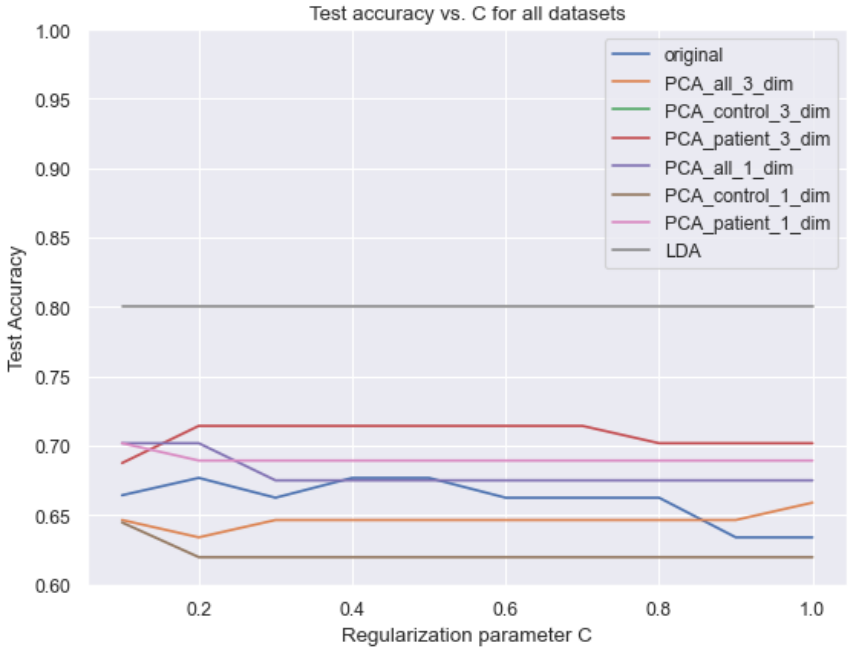


Figure 3.1: Logistic Classificator accuracy for different values of C and different training datasets for the binary classification problem. The models trained with different datasets are plotted in different colors.

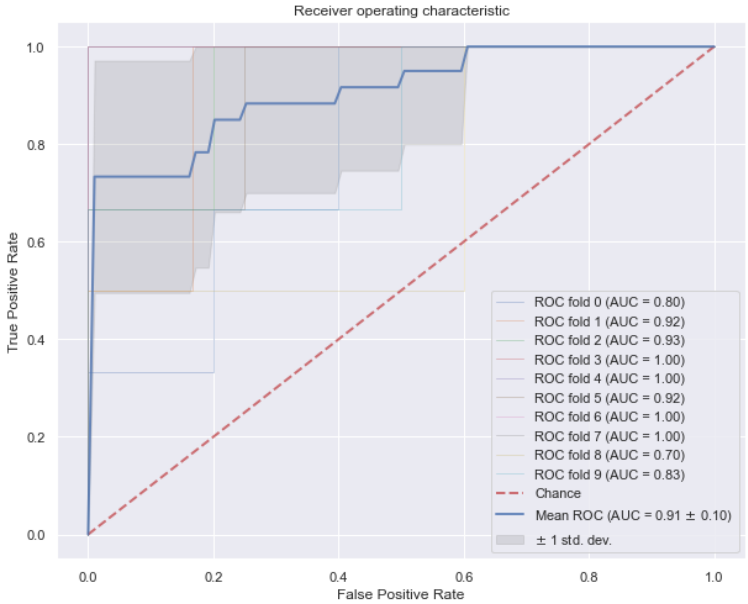


Figure 3.2: Logistic Classificator ROC curve. In blue the mean of the ROC curves obtained in a 10-Fold cross-validation. In red, the ROC curve for a model that classifies randomly.

Looking at Table 3.2 and Figure 3.2 promising results, we can see the performance of the implemented model.

The Logistic Classificator gives a prediction of the class given an input. However, this prediction is extracted from the probabilities of belonging to each class. Then, depending on a threshold τ, the model decides for one class or the other. In Figure 3.2, we can see in blue the mean of the obtained ROC curves while performing the 10-Fold cross-validation. This curve represents the values of $specificity$ and $(1 - sensitivity)$ for all possible thresholds $\tau \in [0, 1]$. The red line represents the ROC curve that would be obtained with model that classifies by chance. Any model whose ROC curve is over this red line, has a better performance than choosing a class randomly.

Additionally, we must give credit to the LDA algorithm, that has increased performance in roughly 15 points over the rest of the models.

B. Support Vector Machine

Additionally to the C regularization parameter, we porform a classification by means of SVM. Remember that SVM has the kernel type as tunable parameter. In this way, Figure 3.3 shows the accuracies for all the datasets for each type of kernel. In this Figure, we can see the accuracy of each kernel, on the horizontal axis, for each training dataset, separated by color.

Once again, LDA has outperformed the rest of the datasets on accuracy terms. However, this time we obtain similar results for both the Radial Basis Function (RBF) and the sigmoid kernels. Thus, we will compare both with the rest of their metrics.

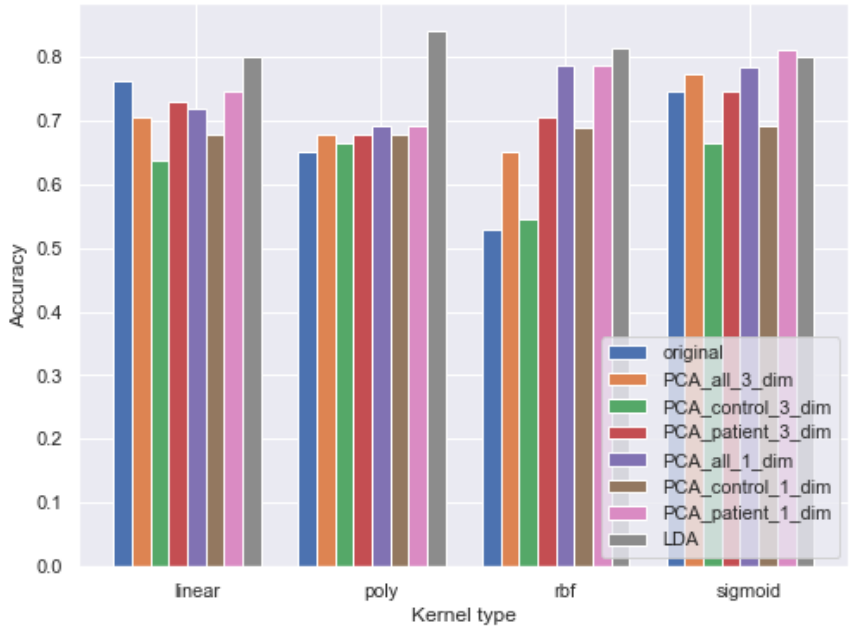


Figure 3.3: Support Vector Machine accuracy for different types of kernel and different training datasets.

Kernel	Accuracy	Precision	Sensitivity	F1-Score	AUC	Confusion Matrix
RBF	0.91	0.96	0.875	0.897	0.925	$\begin{pmatrix} 40 & 2 \\ 4 & 20 \end{pmatrix}$
Sigmoid	0.91	0.96	0.875	0.897	0.961	$\begin{pmatrix} 40 & 2 \\ 4 & 20 \end{pmatrix}$

Table 3.1: Performance metrics of the two best SVM classification models.

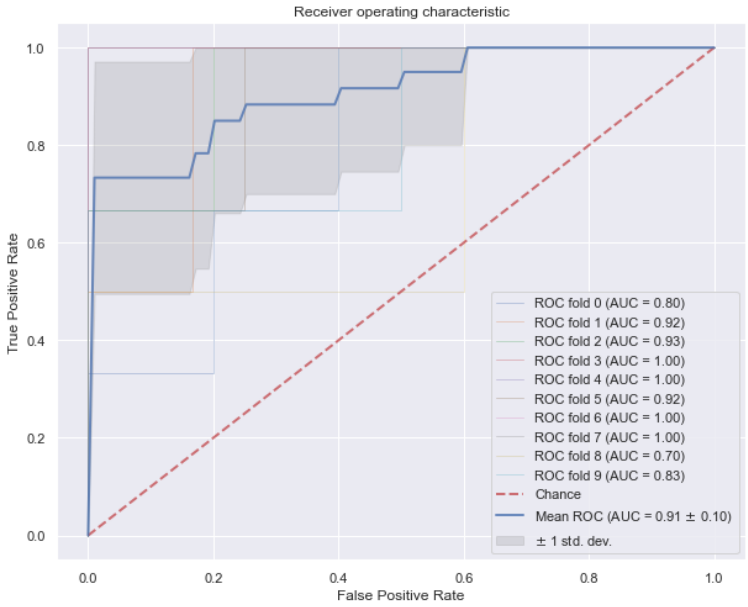


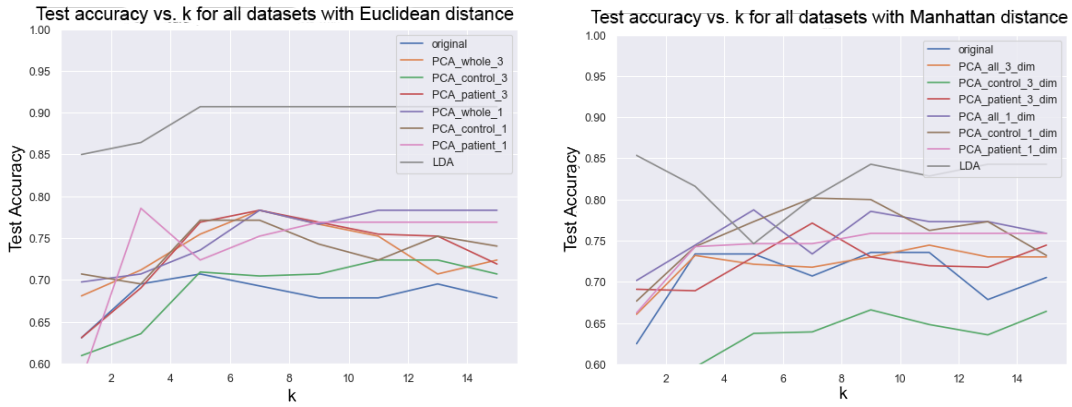
Figure 3.4: SVM Classificator ROC curve.

Table 3.1 compares all the performance metrics of the rbf and sigmoid kernels models. We can see that they are almost similar, but differ in the AUC value, which is a little bit bigger for the sigmoid one. Because of this, we plot the ROC curve only for the sigmoid kernel model to take a glimpse of its shape (see Figure 3.4). Finally, we establish the LDA-trained sigmoid-kernel model as the best SVM we can achieve.

C. K-Nearest Neighbors

Unlike the previous models, KNN does not follow any regularization, as, instead of weights, it just extracts the prediction by looking into the k nearest neighbors. However, two important parameters can be adjusted: the value of k and the used distance type. We will test the models with values of k between 1 and 15 (only odd numbers as we cannot allow a tie in the majority vote) and the Manhattan and Euclidean distances.

Definitely, LDA outperforms again to every other training dataset and its performance is the same on both types of distances as they are both equivalent on a



(a) Accuracy of the KNN model for different values of k and the Manhattan distance.

(b) Accuracy of the KNN model for different values of k and the Manhattan distance.

Figure 3.5: KNN models for both Manhattan and Euclidean distances.

1-dimensional subspace (see Figure 3.5). Furthermore, the maxima of accuracies are concentrated between the values of $k \in [3, 7]$, which seems reasonable as our dataset is moderately small, and a greater value of k would lead to assign the most frequent class of our dataset.

However, the next best model in this range is the one trained with the patient 3-dimensional PCA dataset, which is something to remark in comparison to the first two algorithms.

This time, the best model will be the one trained with the LDA, Euclidean distance, and with a value of $k = 5$, as performance doesn't increase with k and we save computational resources (see Table 3.2 and Figure 3.6).

D. Random Forest

Random Forest can also be tunable. In this case, we can select then number of estimators (decision trees) that the forest is made up of and the depth of said trees. First, one first simulation will show the dependency of the performance on the number of estimators, with a fixed depth of 5, extracting the best and most reliable value. Then, with this value, we will try to extract information regarding how the depth of the trees affects the accuracy.

Looking into Figure 3.7a, The LDA-trained model shows no dependency on the number of estimators, yet it is still the best model of all. However, looking at the remaining ones, it seems that around the value of 100, all the models are settled in their accuracies, and therefore, we will select the value of 100 estimators as best fit.

Then, paying attention to Figure 3.7b, it can be seen that there is an overall tendency of decreasing accuracy while increasing the depth of the trees. The reason for this behaviour could be that, when we increase the depth of the tree, we are also increasing the factor of overfitting in our model, and therefore, decreasing performance. In this way, we select a depth of 2, obtaining the results on Table 3.2 and in Figure 3.8.

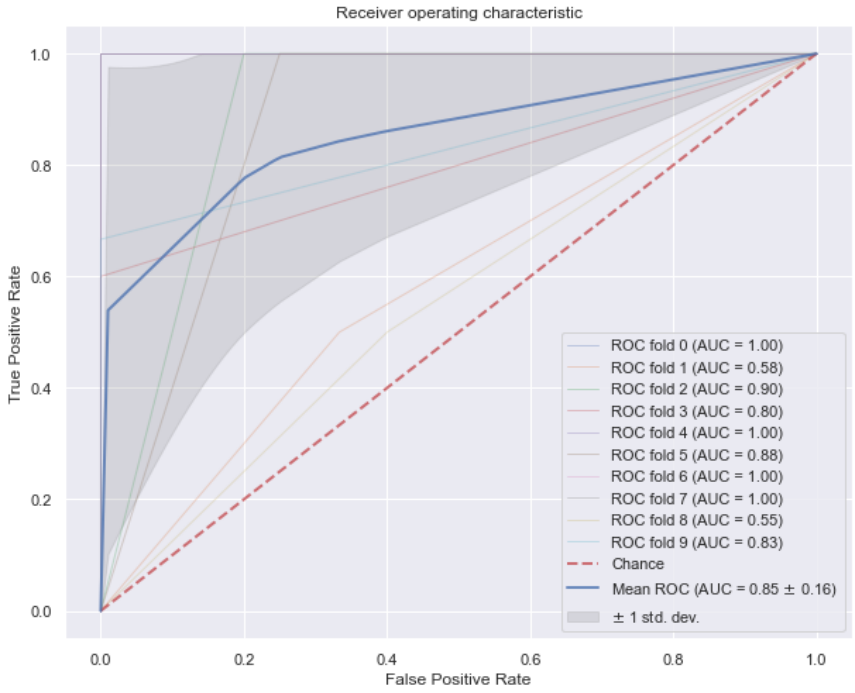
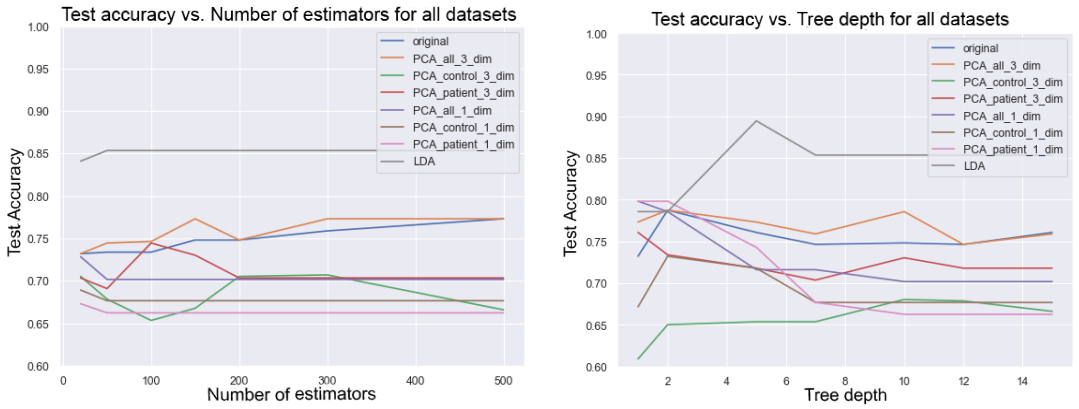


Figure 3.6: KNN Classificator ROC curve.



(a) Accuracy of the RF model for different values of $n\_estimators$ and $depth = 5$.

(b) Accuracy of the RF model for different values of $depth$ and $n\_estimators = 100$.

Figure 3.7: RF models for different values of $depth$ and $n\_estimators$.

3.1.2 Multiclass Classification Problem

In this Subsection, the four models will also be tested with the multiclass classification problem. These models will also be tuned in order to find the best performance. This time, since we will classify between four classes, the probability of guessing right by chance is 25%, and the performance of the models is expected to be better.

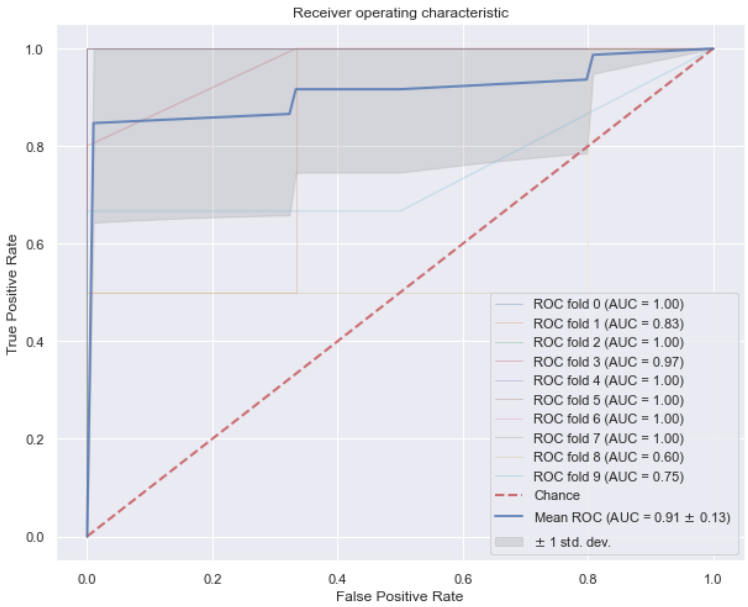


Figure 3.8: RF Classificator ROC curve.

A. Logistic Classification

The parameters that will be tuned in this section are the same as for the binary classification problem. However, looking at Figure 3.9, we can see a different behaviour while changing the Ridge regularization weight. Again, we can see that the best performance is obtained while training with the LDA dataset, reaching accuracies over 60%. Unlike the other two datasets, that provide accuracies under 50%.

In this case, we will select the $C = 0.2$ as best model and then we observe the rest of its performance metrics in Table 3.3. The ROC curve is also obtained (see Figure 3.10).

The curves in Figure 3.10 are the ROC curves for each of the four classes. These curves represent the True Positive Rate and the False Positive Rate for all the possible decision thresholds in each class. The first thing that can be observed is that for every decision threshold, the model is better or equal than deciding by chance. Additionally, the curve of the 'Severe' class is the one of a perfect model. However, as there are very few cases of the 'Severe' class, the algorithm does not choose the treshold that gives a perfect performance for that class, because in terms of accuracy, it changes almost nothing and it is better to choose a worse one for this class that increases remarkably the peformance in the other classes. Looking at Table 3.3, we can see in the decision matrix that for this class we obtain a 100% of True Positive Rate, but a False Negative Rate of 80%, resulting in a sensitivity of 20% in that class.

Finally, looking at the performance of this model in Table 3.3, we can see that this model classifies best the 'Control' class, which may be influenced by the high percentage of healthy cases that we have in our dataset.

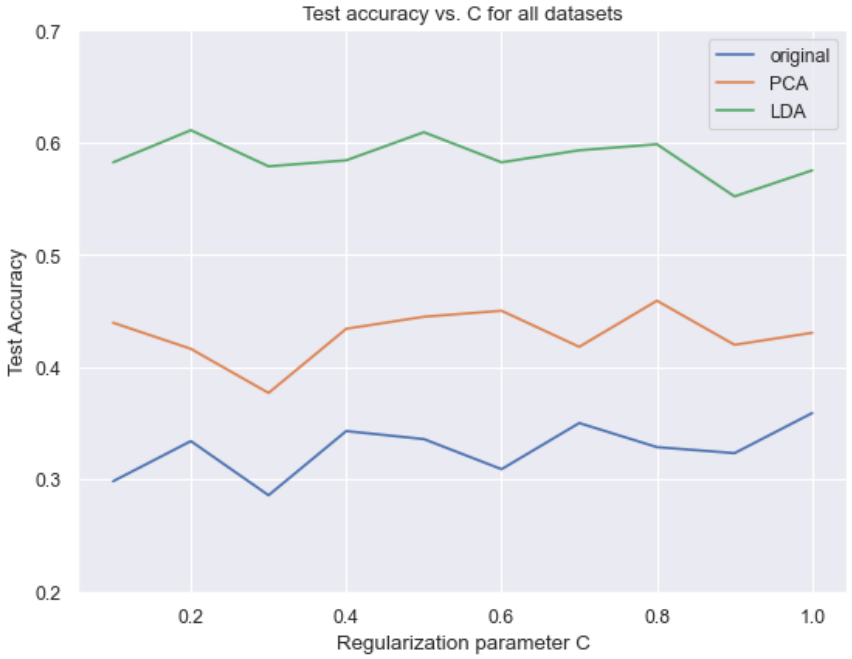


Figure 3.9: Logistic Classificator accuracy for different values of C and different training datasets for the multiclass classification problem. The models trained with different datasets are plotted in different colors.

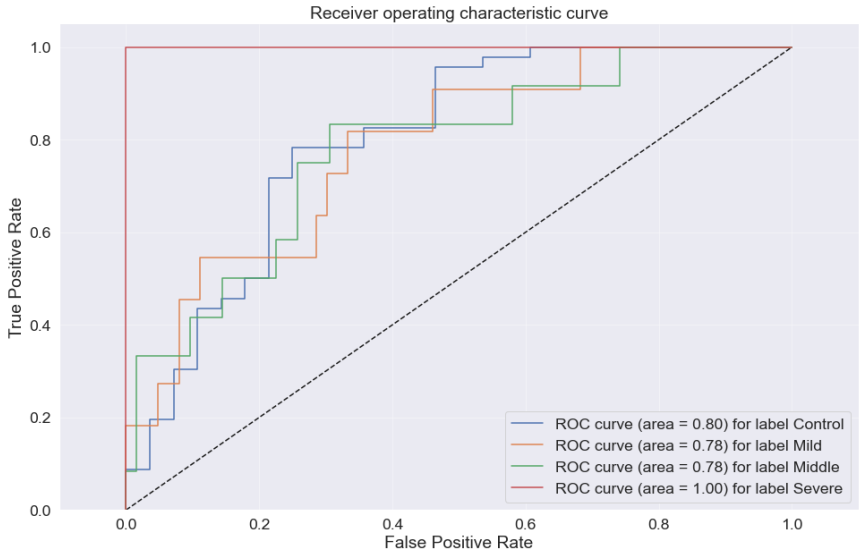


Figure 3.10: Multiclass Logistic Classificator ROC curves separated by class in different colors. In dotted black, the ROC curve for a model that classifies randomly.

B. Support Vector Machine

Support Vector Machine models are trained with the three proposed datasets for each one of the kernels, and its accuracies are obtained (see Figure 3.11). Once again, LDA allows us to obtain a higher performance than in the other models, remarking

the power of this method. The best model is selected as the polynomial kernel SVM trained with the LDA dataset.

The ROC curve and the performance metrics are obtained. In Figure 3.12, we can see the four ROC curves for each one of the classes. Although all the AUC values are over 0.7, we can see a remarkably better performance in terms of the AUC for the 'Control', 'Middle' and 'Severe' classes. Looking at Table 3.3, we can see that the model obtains a great performance in terms of accuracy and in the F1-Score for the 'Control' class. However, in the same way than in the previous case, this could be caused by the unbalanced volume of classes in our training dataset. Additionally, we can also highlight the 100% precision in the 'Severe' class, meaning that when the model classifies an observation to class 3, there is a very high chance that is a correct guess.

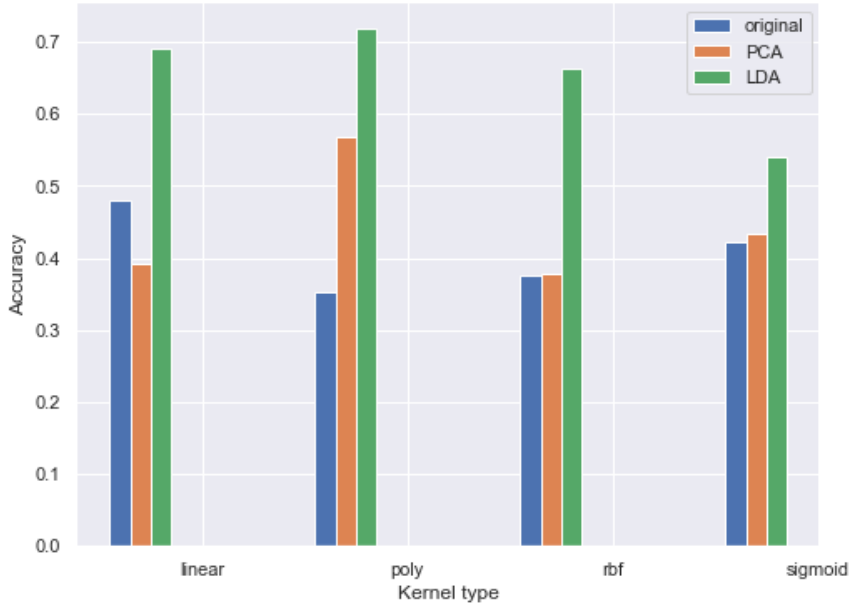


Figure 3.11: SVM accuracy for different types of kernels and different training datasets for the multiclass classification problem. The models trained with different datasets are plotted in different colors.

C. K-Nearest Neighbors

The tune parameters for the multiclass KNN classificator are the same than in the revious section: the type of used distance and the value of k. Attending to Figure 3.13, the highest accuracy is obtained for the values of $k \in [5, 9]$, with a downward trend for higher values. Again, we can conclude that this behaviour may be produced because of the high amount of 'Control' subjects in our dataset, meaning that when

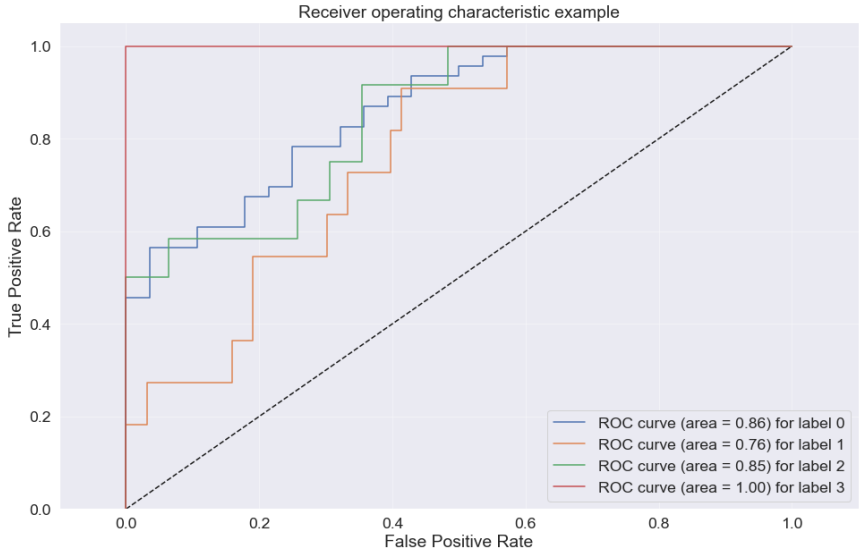
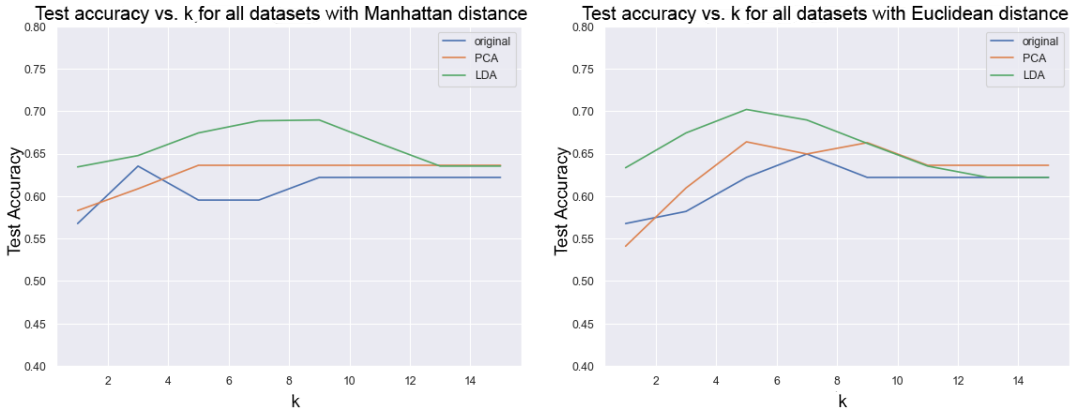


Figure 3.12: Multiclass SVM ROC curves separated by class in different colors. In dotted black, the ROC curve for a model that classifies randomly.

the value of k is increased, there is a higher probability that the 'Control' class wins the majority vote.

In this case, the Euclidean distance model delivers better results for $k = 5$, and therefore, we will select this model as best fit. Then, its performance metrics and ROC are showed in Table 3.3 and Figure 3.14. For this model, we can remark that the most extreme classes, 'Control' and 'Severe' have a great performance in terms of F1-Score. This is to be considered if the aim of our model was to detect severe cases of MS. Additionally, looking at the Confusion Matrix in Table 3.3, we can also conclude that this model struggles to classify the 'Mild' cases of MS and it can be confirmed by its F1-Score for this class.



(a) Accuracy of the KNN model for different values of k and the Manhattan distance.

(b) Accuracy of the KNN model for different values of k and the Manhattan distance.

Figure 3.13: Multiclass classificator KNN models for both Manhattan and Euclidean distances.

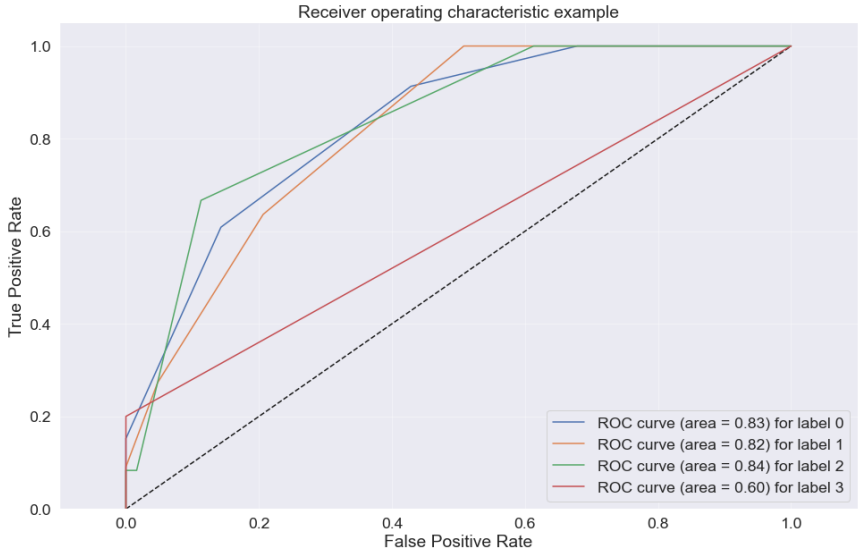


Figure 3.14: Multiclass KNN ROC curves separated by class in different colors. In dotted black, the ROC curve for a model that classifies randomly.

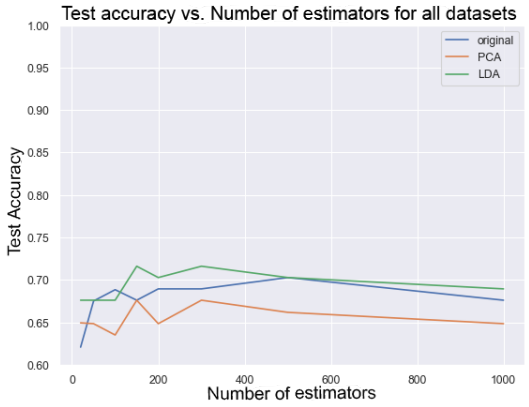
D. Random Forest

Several RF models are trained with different parameters for the depth of each estimator and the numbers of estimators in each forest. The accuracies of each model are shown in Figure 3.15. First, we observed that the best performance is obtained with $n\_estimators = 150$, as there is a tendency of decreasing performance while increasing the number of estimators. Then, we choose a $depth = 10$ to avoid overfitting and computational costs, reaching at the same time the highest possible accuracy.

The best model's performance is represented in Table 3.3 and Figure 3.16. In this case, it is necessary to remark the high values of AUC obtained for every class, except for the 'Severe' one (see Figure 3.16). The ROC curve for this class is almost the one of a random classifier, which leads us to question the results in the Confusion Matrix, where we get a 0.8 value of the F1-Score. This issue can be again related to the lack of data volume.

3.2 Model Comparison

In this Section, the four best models for each one of the classification problems will be compared and one of them will be named the best one for solving each problem.



(a) Accuracy of the Multiclass RF model for different values of *n_estimators* and *depth* = 10.

(b) Accuracy of the Multiclass RF model for different values of *depth* and *n_estimators* = 150.

Figure 3.15: Multiclass RF models accuracies for different values of *depth* and *n_estimators*

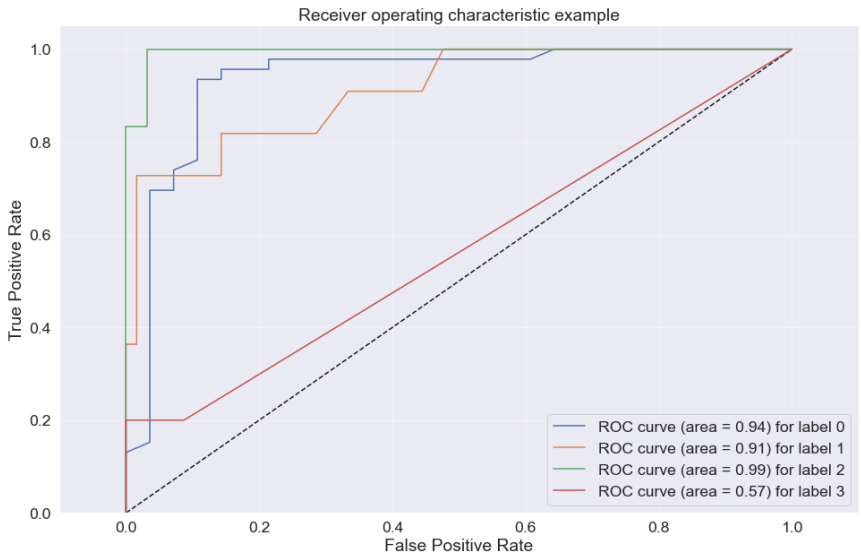


Figure 3.16: Multiclass RF ROC curves separated by class in different colors. In dotted black, the ROC curve for a model that classifies randomly.

3.2.1 Binary Classification Problem

From the previous exercises, we can extract a main conclusion: LDA is a very powerfull Dimensionality Reduction method and it has helped us increasing the accuracy in every case at least 10 points over the next best model.

Model	Test Accuracy	Precision	Sensitivity	F1-Score	AUC	Confusion Matrix
LC	0.829	0.867	0.673	0.710	0.910	$\begin{pmatrix} 44 & 3 \\ 10 & 17 \end{pmatrix}$
SVM	0.841	0.900	0.523	0.647	0.910	$\begin{pmatrix} 47 & 0 \\ 12 & 15 \end{pmatrix}$
KNN	0.854	0.817	0.827	0.806	0.854	$\begin{pmatrix} 41 & 6 \\ 5 & 22 \end{pmatrix}$
RF	0.882	0.883	0.793	0.825	0.904	$\begin{pmatrix} 42 & 5 \\ 5 & 22 \end{pmatrix}$

Table 3.2: Performance metrics of the best models.

Attending to Table 3.2, all models have presented an outstanding performance in the binary classification problem. They all have an over 90% accuracy rate and a sensitivity around 85%. However, we have to highlight Random Forest as the best model for its higher value in the F1-Score, induced by the high values of precision and sensitivity.

3.2.2 Multiclass Classification Problem

As well as in the Binary Classification Problem, the LDA-trained models outperforms every other model. This results confirm that LDA is also powerfull for multiclass datasets. Also, every model has a great performance in terms of accuracy reaching levels between 60% and 70%.

To conclude and select the best model, we should decide the criteria that makes a model the best fit. For applications where the principal aim is to detect severe cases of MS, the KNN and the RF should be strongly considered. If we only had a MS patients dataset and we wanted to classify them along the three degrees of illnes, the SVM model would probably be the best. However, as this is a classification problem, the selected model is the KNN model with euclidean distance and a $k = 5$, for having the best accuracy.

Model	Class	Test Accuracy	Precision	Sensitivity	F1-Score	AUC	Confusion Matrix
LC	0	0.608	0.757	0.587	0.661	0.840	$\begin{pmatrix} 28 & 7 & 11 & 0 \end{pmatrix}$
	1		0.316	0.545	0.40		$\begin{pmatrix} 5 & 6 & 0 & 0 \end{pmatrix}$
	2		0.294	0.417	0.345		$\begin{pmatrix} 4 & 3 & 5 & 0 \end{pmatrix}$
	3		1.0	0.20	0.333		$\begin{pmatrix} 0 & 3 & 1 & 1 \end{pmatrix}$
SVM	0	0.717	0.768	0.935	0.843	0.868	$\begin{pmatrix} 43 & 1 & 2 & 0 \end{pmatrix}$
	1		0.375	0.727	0.495		$\begin{pmatrix} 8 & 3 & 0 & 0 \end{pmatrix}$
	2		0.667	0.417	0.523		$\begin{pmatrix} 5 & 1 & 6 & 0 \end{pmatrix}$
	3		1.0	0.20	0.333		$\begin{pmatrix} 0 & 3 & 1 & 1 \end{pmatrix}$
KNN	0	0.729	0.703	0.978	0.818	0.772	$\begin{pmatrix} 45 & 1 & 0 & 0 \end{pmatrix}$
	1		0.50	0.091	0.154		$\begin{pmatrix} 10 & 1 & 0 & 0 \end{pmatrix}$
	2		1.0	0.333	0.495		$\begin{pmatrix} 8 & 0 & 4 & 0 \end{pmatrix}$
	3		1.0	0.80	0.889		$\begin{pmatrix} 1 & 0 & 0 & 4 \end{pmatrix}$
RF	0	0.702	0.70	0.913	0.792	0.858	$\begin{pmatrix} 42 & 3 & 1 & 0 \end{pmatrix}$
	1		0.25	0.091	0.133		$\begin{pmatrix} 10 & 1 & 0 & 0 \end{pmatrix}$
	2		0.80	0.333	0.470		$\begin{pmatrix} 7 & 0 & 4 & 1 \end{pmatrix}$
	3		0.80	0.80	0.80		$\begin{pmatrix} 1 & 0 & 0 & 4 \end{pmatrix}$

Table 3.3: Performance metrics of the best models.

Chapter 4

Conclusions

In the following Chapter, the obtained conclusions throughout the whole Master Thesis development are presented. Additionally, some future investigation lines will be proposed.

4.1 Conclusions

Throughout the development of the Master Thesis, the following conclusions have been reached:

The use of high quality cameras that capture at least 60 frames per second are suitable to capture the hand movement of a subject that is performing the Nine-Hole Peg Test. Additionally, a resolution of 1080p is desirable to provide a higher precision of the hand's position. The framerate has been confirmed to be correct as we have captured times of at least 50 milliseconds, that, following the Nyquist theorem, at least a frequency of 40 frames per second should be selected.

Moreover, Deep Learning and Machine Learning models, in addition to a standardized camera set, have made it possible to create a full pipeline where a subject performs the Nine-Hole Peg Test and his/her degree of illness is predicted.

DeepLabCut has provided a tool of differential character that enables the tracking of a hand performing all kinds of movements. With a frame dataset of around 300 frames, it has been able to predict the position of multiple parts of the hand in new videos.

On the other hand, Machine Learning algorithms, in particular Linear Discriminant Analysis, have proven to be very powerfull in separating classes given an input data. Although LDA has boosted the results of all models, these could also get promising results even with the original dataset. This leads to the conclusion that the information extracted from the NHPT is highly relevant in differentiating MS patients from healthy subjects.

The experiments have been a success and this project is expected to be continued in order to extract more information related to patients and their future development. With this in mind, several future investigation lines are proposed.

4.2 Future investigation lines

The future investigation lines of this Master Thesis aim at increasing the performance of both Deep Learning and Machine Learning models, as well as the implementation of new models that can predict the patient's development.

The collection of more patient videos for analysis is expected, to increase the training dataset and achieve a better performance in the models. In this way, the DL models could make better predictions of the hand's position in different contexts. For example, changing the light conditions or the type of hand performing the exercise. Additionally, the dataset for the ML models would be increased, leading to more reliable classification predictions.

It is also proposed to collect additional parameters that denote the state of a patient and, with them, find correlations with the parameters extracted from the NHPT. This could be a very important breakthrough, as instead of exposing the patient to multiple types of tests, all this information could be extracted from the NHPT, saving time and resources.

It is also expected the development of a model that can use the NHPT data to predict patient development, reducing the need for unnecessary MRI scans or using them to improve prediction. Having gathered a big patient parameter database along time, observations of new patients could be compared with patients whose disease has evolved. Thus, patients could be treated in the same way as the previous patients in the case of good results, or change the treatment to prevent the problems that may have appeared otherwise.

Finally, it is suggested to develop a phone application that enables the user to record a video of a subject performing the NHPT and provides the clinician with the kinematic parameters. Additionally, the clinician would be able to compare this performance with other patients and thus, diagnose the severity of the illness and predict its future development.

Bibliography

[1] Offnfopt. https://commons.wikimedia.org/wiki/User:Offnfopt.

[2] Mayank Mishra. Convolutional neural networks, explained. 2020.

[3] Machine learning polynomial regression - javatpoint.

[4] Avinash Navlani. (tutorial) understanding logistic regression in python - datacamp, 12 2019.

[5] Pier Paolo Ippolito. Svm: Feature selection and kernels — by pier paolo ippolito — towards data science, 6 2019.

[6] Antti Ajanki. Example of k-nn classification.

[7] Junfeng Zhang, Wei Chen, Mingyi Gao, and Gangxiang Shen. K-means-clustering-based fiber nonlinearity equalization techniques for 64-qam coherent optical communication system. *Optics Express*, 25:27570, 10 2017.

[8] Gopro.

[9] Vemont usb hub 3.0.

[10] Peter Feys, Ilse Lamers, Gordon Francis, Ralph Benedict, Glenn Phillips, Nicholas Larocca, Lynn D. Hudson, and Richard Rudick. The nine-hole peg test as a manual dexterity performance measure for multiple sclerosis, 4 2017.

[11] National institute of neurological disorders and stroke multiple sclerosis information page. https://web.archive.org/web/20160213025406/ http://www.ninds.nih.gov/disorders/multiple$_s$*clerosis*/*multiple*$_s$*clerosis.htm.*[*Accessed* : $2021 - 04 - 07$].

[12] Allison S Drake Lauren Irwin Robert Zivadinov Bianca Weinstock-Guttman Ralph HB Benedict Shumita Roy, Seth Frndak. Differential effects of aging on motor and cognitive functioning in multiple sclerosis. 11 2016.

[13] The effects of multiple sclerosis on your body. https://www.healthline.com/health/multiple-sclerosis/effects-on-the-bodySkeletal-system. [Accessed: 2021-04-08].

[14] Loren A. Rolak. Multiple sclerosis: It's not the disease you thought it was. 1 2003.

[15] Marvin M. Goldenberg. Multiple sclerosis review. 3 2012.

[16] T.J. Murray. Diagnosis and treatment of mutiple slcerosis. pages 525–527, 2006.

[17] Sabrina Figueiredo. Nine-hole peg test (nhpt). 6 2011.

[18] Nicola D Brain Sarah E Lamb Hilary A Jacob-Lloyd, Orla M Dunn. Effective measurement of the functional progress of stroke clients. 6 2005.

[19] Viet Le Ana Mitchell Sonia Muniz Mary Ann Vollmer Kimatha Oxford Grice, Kimberly A Vogel. Adult norms for a commercially available nine hole peg test for finger dexterity. 10 2003.

[20] National multiple sclerosis society - 9-hpt. https://www.nationalmssociety.org/For-Professionals/Researchers/Resources-for-MS-Researchers/Research-Tools/Clinical-Study-Measures/9-Hole-Peg-Test-(9-HPT): :text=The [Accessed: 2021-04-12].

[21] Dong Yu Li Deng. Deep learning: Methods and applications. 2014.

[22] Keith D. Foote. A brief history of deep learning. 2017. [Accessed: 2021-04-14].

[23] Daniel Crevier. *AI: The Tumultuous Search for Artificial Intelligence*. BasicBooks, New York, 1993.

[24] Yoshua; Hinton Geoffrey LeCun, Yann; Bengio. Deep learning. 2015.

[25] Jürgen Schmidhuber Sepp Hochreiter. Long short-term memory. 1997.

[26] Doug Laney. 3d data management: Controlling datavolume, velocity, and variety. 2001.

[27] Ilya; Hinton Geoffrey E. Krizhevsky, Alex; Sutskever. Imagenet classification with deep convolutional neural networks. 2017.

[28] Kamalika Some. The history, evolution and growth of deep learning. 2018.

[29] Varun Bansal. The evolution of deep learning. 2020.

[30] Jake Frankenfield. Artificial neural network (ann). 2020.

[31] Larry Hardesty. Explained: Neural networks. 2017.

[32] Sanket Doshi. Various optimization algorithms for training neural network — by sanket doshi — towards data science.

[33] Andrew Ng. Convolutional neural networks. `deeplearning.ai`.

[34] Jason Brownlee. A gentle introduction to long short-term memory networks by the experts. 2020.

[35] P. Frasconi Y. Bengio, P. Simard. Learning long-term dependencies with gradient descent is difficult. 1994.

[36] L.R.Medsker. *Recurrent Neural Networks: Design and Applications*. New York, 2001.

[37] Michael Phi. Illustrated guide to lstm's and gru's: A step by step explanation. 2018.

[38] Tom Mitchel. *Machine Learning.* McGraw Hill, New York, 1997.

[39] Keith D. Foote. A brief history of machine learning - dataversity, 3 2019.

[40] Joseph L Fix, Evelyn; Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties, 1951.

[41] Naomi S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, pages 175–185, 1992.

[42] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[43] David L. Applegate. *The traveling salesman problem : a computational study.* 2006.

[44] Stuart P Lloyd. Least squares quantization in pcm, 1982.

[45] J Macqueen. Some methods for classification and analysis of multivariate observations, 1 1967.

[46] Sunil Ray. Boosting algorithm — boosting algorithms in machine learning, 2015.

[47] Robert E Schapire. The strength of weak learnability. *Machine Learning*, page 197–227, 1990.

[48] Tin Kam Ho. Random decision forests. *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, pages 278–282, 1995.

[49] A. Geron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques for Building Intelligent Systems.* O'REILLY, 2017.

[50] Sparsh Gupta. What makes logistic regression a classification algorithm? — by sparsh gupta — towards data science.

[51] Vladimir Vapnik Corinna Cortes. Support-vector networks. 1995.

[52] Isabelle M.; Vapnik Vladimir N. Boser, Bernhard E.; Guyon. A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory – COLT '92.*, page 144, 1992.

[53] G. Louppe. *Understanding Random Forests: From Theory to Practice.* Cornell University, 2014.

[54] Wolfgang Utschick. 4.2 random forests.

[55] D. Mwiti. Random forest regression: When does it fail and why?

[56] Zakaria Jaadi. A step-by-step explanation of principal component analysis (pca) — built in, 4 2021.

[57] Steven M Holland. Principal components a n alysi s (pca), 2019.

[58] R. A. FISHER. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 9 1936.

[59] John Hansen. Using spss for windows and macintosh: Analyzing and understanding data. *The American Statistician*, 59:113–113, 2 2005.

[60] Anukrati Mehta. Everything you need to know about linear discriminant analysis, 1 2020.

[61] Aukey cable usb c a usb a 3.0.

[62] Deeplabcut. http://www.mackenziemathislab.org/deeplabcut.

[63] Sudharshan Chandra Babu. A 2019 guide to human pose estimation with deep learning. https://nanonets.com/blog/human-pose-estimation-2d-guide/.

[64] Tanmay Nath, Alexander Mathis, An Chi Chen, Amir Patel, Matthias Bethge, and Mackenzie Weygandt Mathis. Using deeplabcut for 3d markerless pose estimation across species and behaviors, 11 2018.

[65] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. volume 9910 LNCS, pages 34–50. Springer Verlag, 5 2016.

[66] Alexander Mathis, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21:1281–1289, 9 2018.

[67] Andrew Ng. Convolutional neural networks.

[68] The performance of the geforce rtx 2080 ti.

[69] Linh Ngo. How to compare box plots.

[70] Isabella Lindgren. Dealing with highly dimensional data using principal component analysis (pca).

[71] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters 27*, 2005.

[72] Y. Sasaki. The truth of the f-measure. *University of Manchester*, 2007.

[73] Sci kit Learn. sklearn.linear$_m$*odel.logisticregression*.

[74] Yaml ain't markup language (yaml™) version 1.2.

[75] Multiindex / advanced indexing — pandas 1.2.4 documentation.

[76] Matplotlib: Python plotting — matplotlib 3.4.2 documentation.

[77] 7.1.6. what are outliers in the data?

Appendix A

Ethical, economic, social and environmental aspects

After developing this Master Final Thesis, an analysis has been carried out of the impact of the project on the economy, society, ethics and the environment.

A.1 Introduction

The aim of this Final Thesis was to standardize the Nine Hole Peg Test, that is mainly used in Multiple Sclerosis patients diagnosis. At the moment the results obtained from this test is the total time of the exercise and it is stored and compared with the times that other patients used until completion.

There is a need detected in the La Paz Hospital in Madrid, where the Neurology Unit aims to improve the diagnostic capacity in terms of illness severity. This project solves the problem and enables the possibility, not only of obtaining more data from this exercise, but to compare it to a database of all the previous collected data saving lots of time to the hospital staff and to the patients. In this way, the quality of life of patients will be directly improved by providing them with a more specialised diagnosis and by seeking correlations of their data with those previously obtained and predicting future developments.

Additionally, while reducing the diagnostic tests and the time spent on them, we reduce not only on human resources, but in transport expenses that affect the patient directly.

A.2 Ethical impact

This project has no ethical risks as long as the data provided by the models are used as a recommendation and not as the truth, avoiding wrong diagnoses. Additionally, it complies with all data regulations regarding the use of patients as they were properly informed that this was a voluntary activity and their data was to be treated in accordance with the Organic Law 2/2018, of 5 December, on the Protection of Personal Data and Guarantee of Digital Rights (LOPDGDD). Moreover, it was validated by the La Paz Hospital Ethics Comitee.

A.3 Economic impact

The data obtained with the tools developed in this project allow a more exhaustive analysis of the disease allowing the medical staff to reach a diagnosis more easily. This would help reducing the costs of future treatments through early detection of problems in the patient's treatment. Additionally, as this project will ease the diagnosis process and reduce its time, it will also reduce costs in human resources.

A.4 Social impact

This project was born out of respect for patients with multiple sclerosis and the desire to be able to improve their lives as much as possible. By means of the tools developed in this project, it is expected to reduce the time spent on analysis and treatment so that the patient spends less time in hospital. Additionally, as it is mostly a software-based solution, it is easily transferable. And, although at the moment only La Paz Hospital is involved in the project, it is expected that any patient throughout the world could benefit from this tool.

A.5 Environmental impact

The impact of this project on the environment is indirect, since, as aforementioned, treatment periods can be shortened. This reduction in time will result in fewer journeys being made by many of the patients to the hospital, reducing the amount of CO_2 emitted by the cars in which they travel.

Appendix B

Financial Budget

This project was developed during nine months in the Robotics and Control Laboratory (Robolabo) using their resources. Therefore, the project's budget has been calculated considering both material and human resources costs.

- **Personal**: Human resources have been calculated with an approximated value for the hourly cost of a project director, an engineering graduate and an engineering student (see Table B.1).

	Hourly cost (€)	**Hours**	**Total (€)**
Project director	60	50	3,000
Engineering graduate	30	900	27,000
Engineering student	15	450	6,750
TOTAL			**36,750**

Table B.1: Human resources related costs.

- **Material costs**: Regarding the material costs, costs have been calculated considering their cost value and the amortisation of said materials(see Table B.2).

  Materials have been divided into three categories. One category without amortisation, in which the materials are considered to be a punctual expense such as the PVC tubes, the printer's ink and the wooden board. A second category in which the NHPT kit, the cables and the USB hub are included with an amortisation of 1 year. And a last category, where computers, printers, hard drives and GPUs are included.

Considering these two costs, the total cost of this project is 37,336.75 €(see Table B.3).

	Life span (years)	Units	Cost (€)	Amortisation (€/month)	Use (moths)	Total (€)
Wooden board	-	1	15	-	-	15
PVC pieces	-	1	5	-	-	5
3D printer Ink	-	1	20	-	-	20
USB cables	1	1	15	1.25	9	11.25
USB Hub	1	1	20	1.66	9	15
NHPT Kit	1	1	120	10	9	90
GoPro Camera	5	3	250	4.16	9	112.5
GoPro Smart Remote	5	1	100	8.33	9	75
3D Printer	5	1	300	5	9	45
External HDD	5	1	100	1.66	9	15
Laptop	5	1	750	12.5	9	75
NVIDIA GPU	5	1	720	12	9	108
TOTAL						**586.75**

Table B.2: Material costs

	Coste
Personal costs	36,750€
Material costs	586.75€
Subtotal	37,336.75€
IVA	7,840.72€
Total	45,177.47€

Table B.3: Total costs.

Appendix C

How does DeepLabCut work? - Installation and User guide

First, the user must decide wether the GPU or the CPU is going to be used. For this decision, the user must consider that only NVIDIA GPUs are compatible. Then, the proper NVIDIA driver must be installed alongside CUDA Toolkit, version 10.0 or lower. Once CUDA is installed, DLC package must be also installed on the computer that we will use for the video analysis. As the operative system of the chosen machine for this project is Ubuntu, this task is fairly easy typing on a terminal the following command:

```
$ pip install deeplabcut
```

However, previous to this command, we should decide if we want to use the computing power of the CPU or the one of our GPU. The obvious choice is to select the GPU for its greater computing power in comparison with the CPU. However, this selection should be considered and, depending on the decision, we must type on a terminal

```
$ pip install tensorflow==1.13.1
```

for the CPU or

```
$ pip install tensorflow-gpu==1.13.1
```

in the case of the GPU.

Once the package is properly installed we can start creating the project. First, the whole coding process will be explained and then the explanation will be repeated following same road map on the GUI.

- **Creating a project.**

  DLC includes a function called *create_new_project* that creates a new project directory with a basic configuration file. The project is identified by the name of the experiment, the name of the experimenter and the date of its creation. In order to create the project, the aforemention data has to be provided alognside with the set of videos that will be used for training.

```
[1] import deeplabcut as dlc
[2] dlc.create_new_project('Project name','Name of the experimenter',
['Full path of video 1','Full path of video2'],copy_videos=True/False)
```

The user can also get the configuration file path, that will be used in the next steps, just by typing:

```
[2] config_path = dlc.create_new_project('Project name',...)
```

This will result in the creation of four directories in the project folder:

-**dlc-models**: it contains two folders **"train"** and **"test"** that will hold the information of the feature detectors in the configuration file. Also, the **"train"** folder will hold the training checkpoints of the built models.

-**labeled-data**:it contains the extracted frames that will make up the training dataset. The frames of each video will be stored separately in a specific folder, and their names will reference the frame number of the video starting on zero.

-**training-datasets**: in this folder the training datasets and their metadata.

-**videos**: if we specified $copy_videos$ to $False$, this folder will store symbolic links to the videos. It is also important to know that there is a function "add_new_videos" that will allow the user to add new videos whenever he/she wants.

- **Configuring the project.**

  The configuration file is store in the root folder of the project and is a YAML file. YAML is a human-readable data-serialization language commonly used in configuration files [74]. The fields of this file will be explained below:

  - **task**: name of the project.
  - **scorer**: name of the experimenter.
  - **date**: date of the project's creation.
  - **project_path**: Full path of the project.
  - **video_sets**: dictionary of the full paths of the videos as keys and their cropping coordinates as value.
  - **bodyparts**: the name of the labeled bodyparts that the experimente wants to track.
  - **start**: starting point of the interval to sample frames when extraicting them.
  - **stop**: finishing point of the interval.
  - **numframes2pick**: maximum number of frames to extract from each one of the videos.

- **skeleton, skeleton_color**: combination of bodyparts to build the skeleton and its color.
- **pcutoff**: threshold of likelihood to define if the prediction is certain or not. Default on 0.6.
- **dotsize**: diameter of the labeling points in the frame measured in pixels.
- **alphavalue**: transparency of the plotted labels from 0 to 1.
- **colormap**: it defines the colors used for the label points.
- **TrainingFraction**: percentage of the frames that will be used as training dataset. Its a two digit float with range 0-1.
- **iteration**: Keeps track of the number of iterations performed to create the training dataset.
- **resnet**: the pre-trained model that will be used.
- **snapshotindex**: specifies the index of the checkpoint to use when analysing. Default to -1, the last checkpoint.
- **batch_size**: number of frames used in each training iteration.
- **cropping**: boolean that specifies if the videos are to be cropped and its cropping coordinates.
- **corner2move2, move2corner**: In some cases, DLC predictions may be outside of the frame. This configuration variables will specify if these points are to be moved inside the frame and where.
- **default_net_type**: specifies the default type of model to use. Default to resnet_50.

Once the configuration file is edited to suit the necessities of the experimenter is saved and we can begin with the next step.

- **Selecting the data**

  With the help of DLC, we will be able to extract the frames from the video and select some to include them on the dataset. This can be performed manually or automatically with DLC.

  The manual process will open a frame extraction GUI, that will allow the user to navigate through all the frames and decide which one to select. On the other side, the automatic process will not need the GUI and its criterion of frame selection can be "uniform" or "kmeans". The uniform criterion will select a frame every fixed interval of time in the video until it reaches the maximum number of frames. The kmeans criterion will extract all the frames of the video and will perform a kmeans clustering with k equal to the maximum number of frames and then select the cluster centers.

  Although the automatic way of selecting the frames is really easy for the user, it may be counterproductive. This behaviour will be cause by frames in the video that don't represent a relevant part of the activity, i.e., the start and end of the video where the points of interest may not be visible. A good dataset should

consist in a sufficient number of frames of the relevant positions of the exercise and, thus, be able to capture all the characteristics of the behaviour.

To execute this part of the process, type:

```
[3] dlc.extract_frames(config_path,'automatic/manual','uniform/kmeans')
```

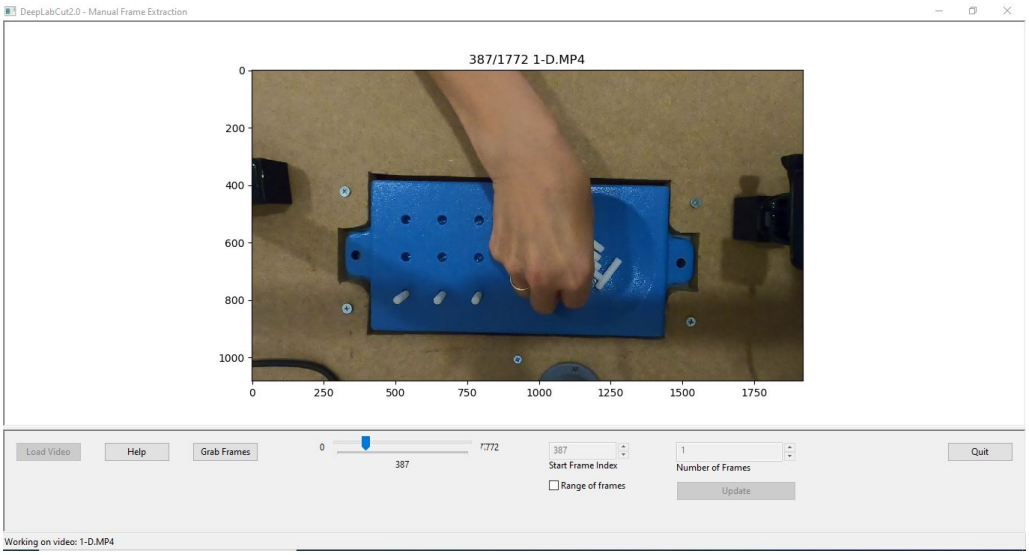


Figure C.1: DLC Manual Frame Extraction.

- **Labeling the data.**

  Once the dataset is gathered, we can proceed with the labeling of the frames by means of the function 'label_frames' that DLC provides. Before going ahead with the labeling, the user should have introduced the points of interest to be labeled in the configuration filed. Then, with the help of the DLC GUI, we will load the frames and start labeling.

  ```
  [4] dlc.label_frames(config_path)
  ```

  First, the user must click on the 'Load frames' button and select one of the folders that were automatically created when extracting the frames. Then, by right-clicking in the frame, the labels are placed with the form of a point with a unique color representing the label. If a point of interest is not on the frame, there is no need to place the label. The user can also adjust the marker size in the range of 1 to 36 pixels. It is also strongly recommended that the user tries to place the label always in the same location; for example, if the center of the hand is really large, try to place the label on the same spot.

  Once the labels are placed, they can be moved through the frame by left-clicking on it and drag it to the new position if they were misplaced. If a label is placed

on the frame by mistake, it also can be removed by clicking with the scrolling wheel on it.

Finally, after clicking on the 'Save' button, the labels are stored in a CSV file with their x and y coordinates.

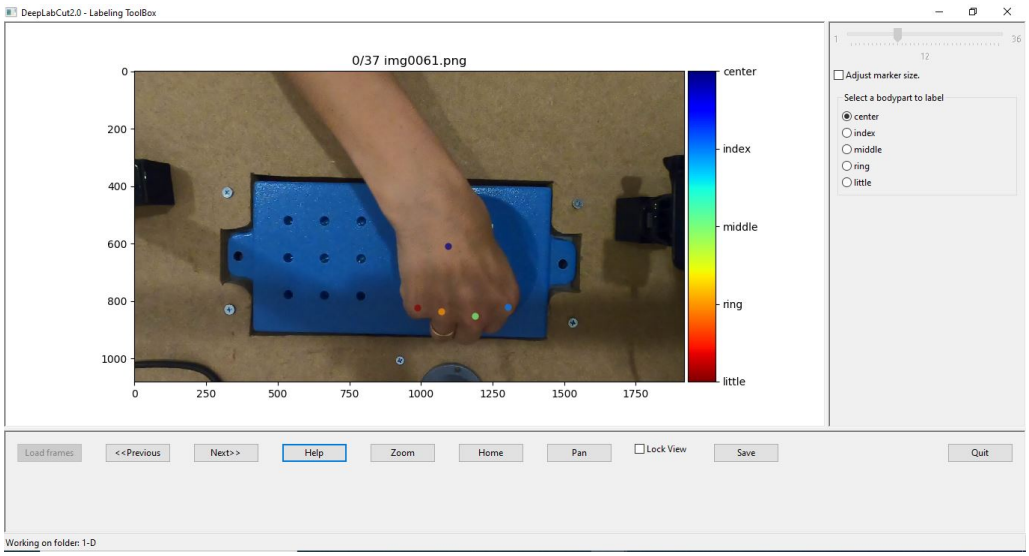


Figure C.2: DLC Labeling Toolbox.

- **Checking the labeled frames.**

  This step may be one of the most important ones in the process, as an incorrectly labeled dataset may be counterproductive for training and may produce a systematic error in the prediction model.

  Executing the 'check_labels' function, DLC will replicate the folders of the extracted frames and will plot in them the coloured points of the labels. This will allowe the user to check the frames and confirm that all the boyparts are labeled correctly. If they are not correct, the user can user the refinement GUI and modify the original labels.

```
[5] dlc.check_labels(config_path)
```

- **Creating the training dataset.**

  Once the frames are checked to be correctly labeled, these frames must be put together and be separated in test and training datasets. This can be performed by the 'create_training_dataset' function of DLC, that will use the 'TrainingFraction' parameter from the config file in order to separate the desired percentage of data for training and testing. These datasets information will be stored under the 'training-datasets' directory, with a '.mat' file and a '.pickle' file. The '.mat' file is a MATLAB file and the '.pickle' file is a Python object serialization file, and both contain the metadata of the training dataset.

```
[6] dlc.create_training_dataset(config_path)
```

Two more folders are created under the 'dlc-models' directory, 'test' and 'train'. These two have a 'pose_cfg.yaml' files that can be edited by the user before stating the training process. These fields are explained hereunder (note that there are some that are not included as they are copied directly from the configuration file):

- **display_iters**: specifies the period in which the iteration loss is displayed on the console.
- **init_weights**: path of the weights to be used from the beginning of the training. Normally the pre-trained DLC network.
- **global_scale**: All data will be scaled before being introduced in the CNN with this value, Default 0.8.
- **pos_dist_thresh**: any predicted point withing this range from the labeled point will be considered positive for training. Default 17.
- **save_iters**: specifies the period in which a checkpoint of the model is stored.
- **scale_jitter_up, scale_jitter_lo**: during training, data augmentation will be carried out and the frames will be rescaled between these two values. Default 1.5 and 0.5.
- **mirror**: if the training dataset was vertically symmetrical, this value could be set to True for data augmentation.
- **cropping**: boolean that indicates if the frames are to be cropped.

Once the training configuration file is all set, training can start.

- **Training the model.**

The step where the network is trained arrives and it is done by executing:

```
[6] dlc.train_network(config_path, trainingsetindex=0, gputouse=None,
max_snapshots_to_keep=5, autotune=False, displayiters=None, saveiters=None)
```

Trainingsetindex specifies the index of the training set that the user wants to use. *Gputouse* is a natural number that indicates the GPU that the user wants to use, if the user wants to use the CPU, 'None' is written. *Autotune* is a Tensorflow variable and if set to False, training is faster.*Displayiters* overrides the value in the 'pose_cfg.yaml' file. And finally, *max_snapshots_to_keep* and *saveiters* are the maximum number of checkpoints to keep and the iteration period of saving.

If there was no specified network in the *init_weights* field, the pre-trained selected network will be downloaded from TensorFlow servers and stored in 'pre-trained' in the folder where DeepLabCut is installed.

After loading the network, DLC will start with the training. It is important to say that the training process should be done until the error stabilizes, which is empirically proven to be around 200,000 and 300,000 iterations.

- **Evaluating the model.**

  Once the model is trained, we can evaluate its performance using the previously defined test dataset. These evaluation results are obtained by typing:

  ```
  [7]  dlc.evaluate_network(config_path, plotting=True)
  ```

  The *plotting* parameter specifies if the user wants to plot the predictions both of the training and test dataset.

  The performance of the model is assessed by obtaining the Mean Absolute Error (MAE) between the manual labels and the ones predicted by the model. This measure is given as the euclidean distance of the predicted position to the labeled one in pixels.

  $$MAE = \frac{1}{M}\sum_{i=0}^{M} |\sqrt{(x_{labeled,i} - x_{predicted,i})^2 + (y_{labeled,i} - y_{predicted,i})^2}|$$

  Evaluating the model is also a very important part of the process as it helps us see the performance that the model has with unseen images (test dataset), that is, after all, the final application of our system. This way, we can check that the loss has actually converged and decide whether to keep training or increase the size of the training dataset.

- **Analysing videos.**

  If the model is properly trained and the user is happy with the results given on the validation frames, the model is ready to be deployed into production.

  This means that the model that we trained can be used to predict the position of the points of interest inside the frame of any similar video that the ones we used for training. Needless to say, that if the video is different enough, the model may not recognise any bodypart in the video and get useless results. Thata is why it is important to consider different types of lights and possitions of the camera for the network to have enough variance in its training dataset.

  ```
  [8]  dlc.analyze_videos(config_path,['video1.mp4'],
  save_as_csv=True, videotype=.mp4)
  ```

  Executing the last Python line, DLC will extract all the frames from the introduced video and analyse everyone of them. Once it analyses the video, if the *save_as_csv* parameter is set to True, all the predictions will be saved to a CSV file with the x and y coordinates of every bodypart, alongside with the likelihood value of these values. However, if its value is set to False, the labels

will be stored in a MultiIndex Pandas Array [75]. These arrays are stored in Hierarchical Data Format (HDF) with the .h5 extension.

After analyzing the video, the user can plot the predictions in the video and obtain a new labeled video with the predicted positions of the bodyparts by executing:

```
[9]  dlc.create_labeled_video(config_path,['video1.mp4'])
```

This way, the new video is stored in the same directory as the original one and the user will be able to see the results in the context of the video.

The user can also plot the trajectories with the help of Matplotlib [76] whose plots can be easily customized as this library is very well known and intuitive. This can be done by typing:

```
[10]  dlc.plot_trajectories(config_path,['video1.mp4'])
```

- **Refinement.**

  Although the model may give satisfying results, there may exist cases in which the model gives a wrong prediction. This is when considering refinement seems relevant, to optimize the perfomance in some specific situations.

  DLC allows the user to carry out this refinement with two possible activities: extract outlier frames and refine labels.

  Outliers are defined as observations that lie an abnormal distance away from the other values of the population [77]. In a 2-dimensional space, the definition of outlier is trivial. For example, if our dataset is composed of the points in Figure C.3, we can clearly see that all points lie in between the (-2,2) intervals for both edges. It is easy to see that in this plot there is an outlier that seems not to represent relevant information as it differs from the rest. In this case, we would discard the red point.

  However, the definition of outlier for a DLC dataset is not that easy. That is why DLC offers some criteria to find these outliers [64]. If giving a prediction, all bodyparts have a likelihood under a threshold, this prediction would be considered outlier. Also, a frame would be considered outlier if the predictions of the points of interest have moved over a threshold in comparison to the previous frame. This can be done by executing:

```
[11]  dlc.extract_outlier_frames(config_path,['video1.mp4'])
```

  DLC also offers the posibility of refining the outlier labels and use them to augment the training dataset in order to improve the performance in these situations.

  We can have two possibilities: the bodypart is visible but the prediction is wrong or the bodypart is hidden and a prediction in the frame is given. In the

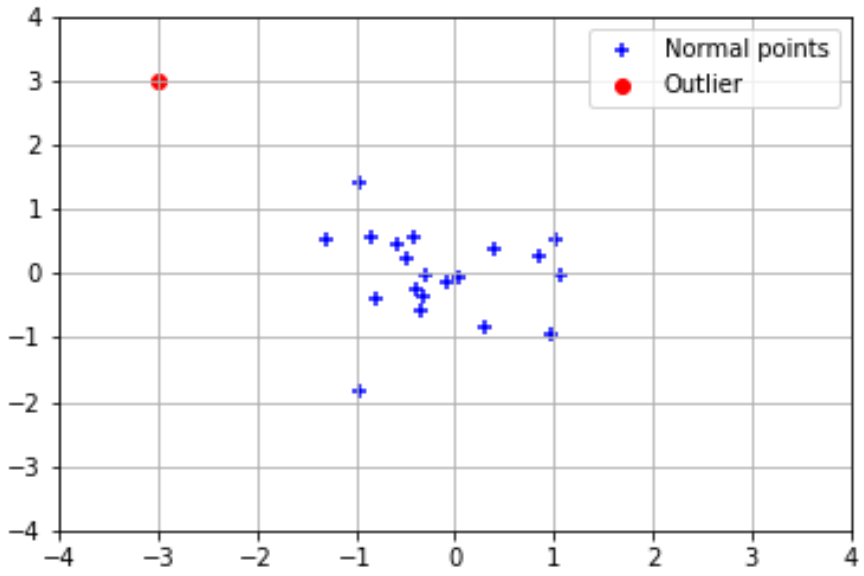


Figure C.3: Outlier example in a 2-dimensional space.

first case, the label should be moved to the actual position of the bodypart and, in the second case, the label should be removed.

Outlier frames labels can be refined by typing:

```
[12]  dlc.refine_labels(config_path)
```

This function will open the GUI for label refinement. Then, the user can specify the probability threshold to identify the labels with low certainty. This process is similar to the labeling one but instead of placing a new label, the label is to be moved by draging it with the left mouse button. If the user desires to delete a label, the right mouse button is used.

Once the labels are corrected by the user, a new training process can be performed. But before starting the training, the new frames have to be merged with the previous ones.

```
[13]  dlc.merge_datasets(config_path)
```

Finally, the user can use the 'check_labels' function to check that the labels are placed correctly and start the process again from the training phase.

This whole process can also be performed following the same steps but in the DLC GUI. For this matter, the user can just execute:

```
$ python -m deeplabcut
```

or

```
[2]  dlc.launch_dlc()
```

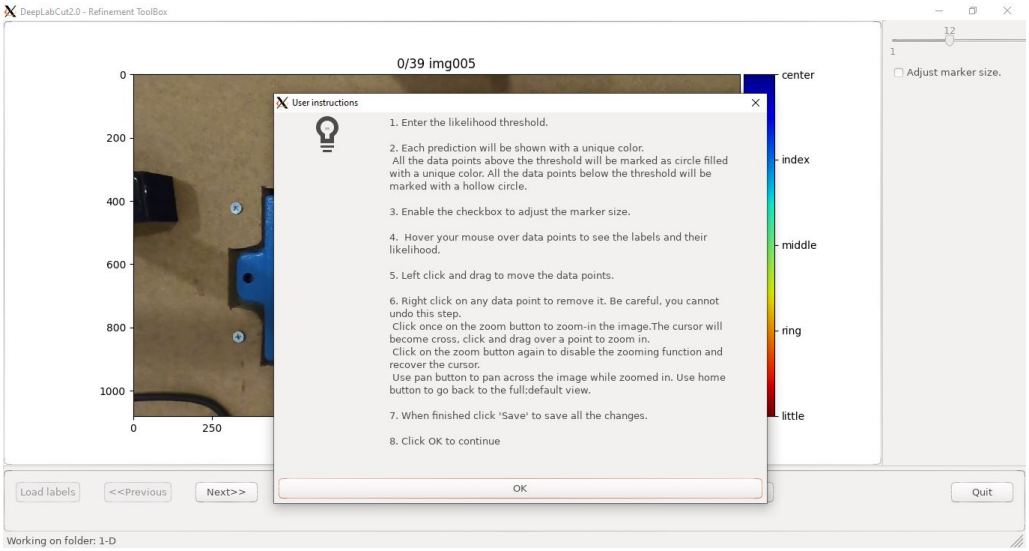


Figure C.4: DeepLabCut Refinement Toolbox.

Appendix D

Kinematic Parameters and Multiple Sclerosis Classification - User's guide

D.1 Introduction

In this document, the steps that should be followed to extract the kinematic information from the NHPT are explained. These steps are: video capture, video analysis and pose estimation, data preprocessing and kinematic features extraction and, finally, classification. Alongside this manual, some additional files are to be found:

- **dlc-right (Directory)**
- **dlc-left (Directory)**
- **analyse_video.py**
- **dataset_right.csv**
- **dataset_left.csv**
- **NHPT_datos.xlsx**
- **extract_data_from_csv.py**
- **ML_binary.ipynb**
- **ML_multiclass.ipynb**

D.2 Video Capture

It is recommended to capture the videos with the provided structure of the project (see Figure D.1). The Neural Networks have not been trained in a different framework, so performance may decrease significantly.

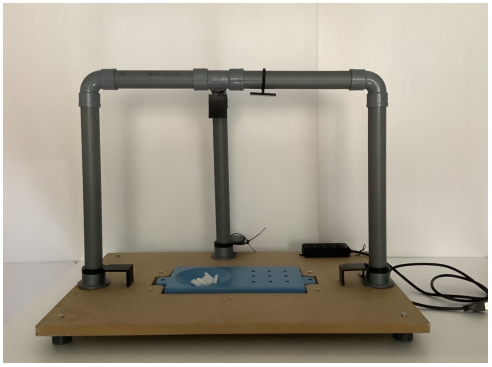


Figure D.1: Mechanical structure for video capturing.

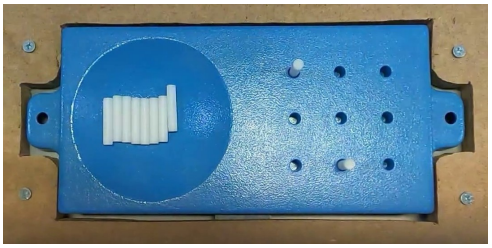


Figure D.2: Positionning of the NHPT kit from the patient's view.

The structure is placed in front of the patient, with the NHPT kit in the center of the board. The peg container should be on the left side of the patient, leaving the holes to the right (see Figure D.2).

With the help of the GoPro Smart Remote, the three cameras are turned on. When the remote is not connected to all cameras, it should be reset. This can be done by pressing and holding the red button, then pressing the power-mode button and finally, releasing both. The remote should go into pairing mode and the cameras can be connected again by selecting in the menu `Settings > Connections > Connect Device > Smart Remote`.

Once the cameras are turned on, select the video mode on the remote. Additionally, it should be checked that the capture mode of the zenital camera is at least of 60 frames per second.

Finally, the capture button can be pressed in the remote control and start recording the performance of the NHPT. Once the test is finished, press the capture button again to conclude the recording.

The recorded videos are stored in the SD card of the camera whose free space will have to be checked ocasionally. These videos can be copied by extracting the SD card and plugging it to a camputer or by connecting the camera directly to the computer using a USB-C cable.

Then, the subject's data should be added to the `NHPT_datos.xlsx` Excel file, in the control sheet if he is a control subject, or in the patients one otherwise.

D.3 Video Analysis

Once the videos are copied into the computer where the analysis will be performed, the DeepLabCut framework should be installed. For this, follow the introduction of the Installation Guide that can be found in Appendix C.

It is important that the videos to be analysed are named with the format `TXXXH.mp4`, where `T` specifies if the type of the subject and can take values of `C` for control subjects and `P` for patients, `XXX` is the ID that corresponds to the subject in the `NHPT_datos.xlsx` Excel sheet, and `H` references the hand that the subject is using, `D` for right, and `I` for left (see Figure D.3). Then, the videos should be separated by hand in two directories, one for the right hand and other one for the left one.

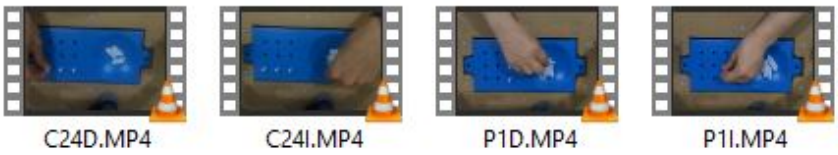


Figure D.3: Example of video naming format for the 24^{th} control subject and for the 1^{st} patient.

Once the videos are properly named and stored all together in two different folders, the file `analyse_video.py` is to be opened and modified. Two variables should be changed. First, the variable `path`, that specifies the absolute route to the config file of the DeepLabCut project to use. If the videos to be analysed are the ones of the right hand, use the config file of the `dlc-right` project, and the `dlc-left` otherwise. Then the `video_path` variable, that must be set to the absolute path of the folder where the videos are stored, considering also the separation between hands. When these two variables are set, save the file and execute it with the command:

```
$ python analyse_video.py
```

This script will detect all the videos in the folder, it will transform their format into a compatible one with DeepLabCut and will analyse the hand's pose in the video. The script will output a '.csv' file for each video that contains the coordinates and likelihood of each of the tracked bodyparts for each frame.

These data files can be copied into another folder or be left where they are, but the right-left separation must remain between files.

D.4 Data Preprocessing

For this step, the Python file `extract_data_from_csv.py` and the two '.csv' files, `dataset_right.csv` and `dataset_left.csv` are needed. In these files, the kinematic features of the subjects will be stored.

To extract these features, the Python script should be modified, changing the `data_path` and `csv_path`. The former specifies de path where the files output by DeepLabCut are stored. The latter is the full path to the `dataset_xxx.csv` file, that should be selected depending on the hand files to be analysed.

Once the path variables are set, execute the Python script with the command

```
$ python extract_data_from_csv.py
```

The script will then detect all the DeepLabCut '.csv' files and start analysing them.

As the data provided by DeepLabCut may not be fully correct, a visualizing and correcting framework has been built in this script. First, a plot of the hand's movement through the horizontal axis and time is showed to the user. This plot will also show the maxima and minima of the exercise alongside with the defined 'middle' of the movement. In this plot, the user must check that for the first nine oscillations (the first part of the NHPT), the maxima and minima are in their corresponding peaks (see Figure D.4). If this is not the case, and in one of the peaks there is no singular point mark, two methods can be used to correct it: adding new points or changing the middle value.

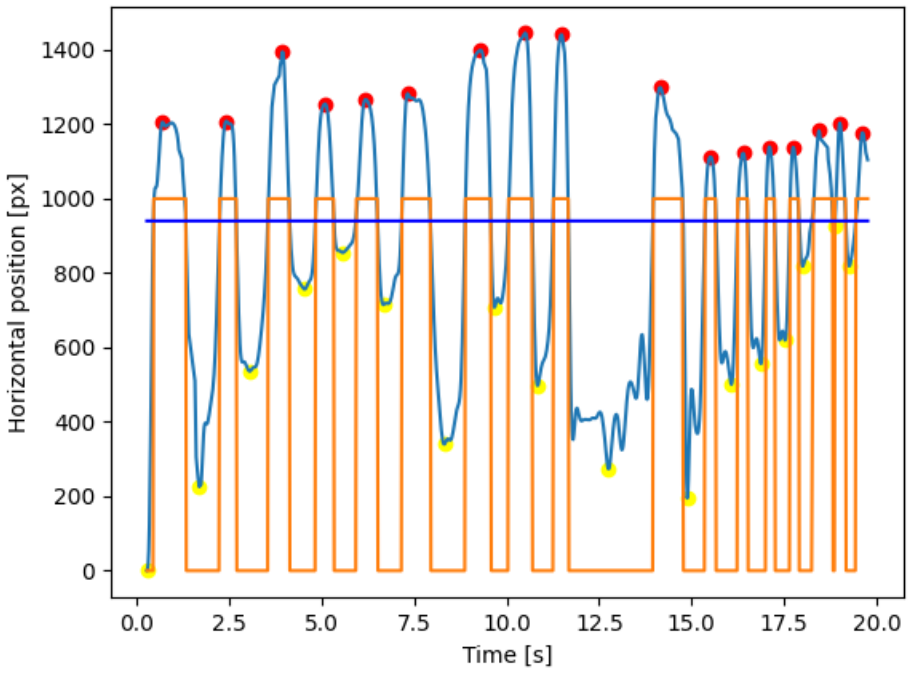


Figure D.4: Example of a correct plot.

When the user closes the plot window, s/he will be asked if new points are to be added. With this option we can add a dummy point that extends the movement of the hand to a point over or under the middle value as needed. This can be easily

understood with an example:

The plot that is showed in Figure D.5 is the one that the script outputs. We can see that around the second 5.5 there is a minimum that does not cross the middle line and thus, it makes the script not to detect the previous maximum and that minimum. Then, once the plot is closed, when the user will be asked the following promt: `New points?[y/n]`, to which s/he has to respond with a `y`. Then, the script will ask how many points are to be added and in which coordinates. In this case, we will add one point in the second 5.5 and with position 780, to cross the middle line, obtaining the plot in Figure D.6. Now the maximum and minimum are detected and the plot is corrected.

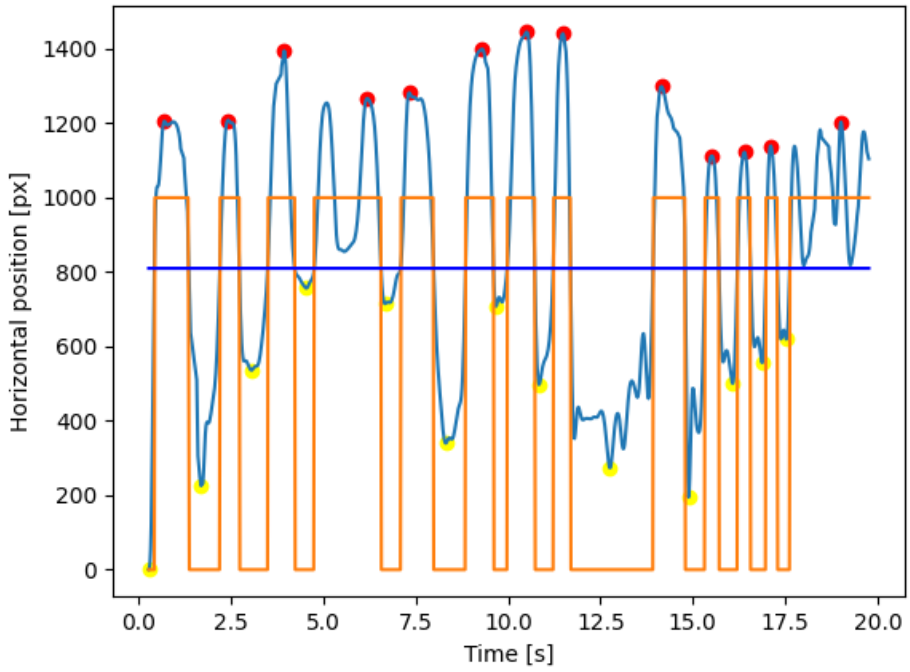


Figure D.5: Incorrect plot of the movement.

Additionally, this can be done in an easier way changing the middle value of the movement. If we take the plot in Figure D.5, and we change the middle value to a value near 900, instead of being near 800, we could correct the data without adding new points, obtaining the plot in Figure D.4. To do this, the user will be asked if s/he wants to change the middle value with the promt `Change middle?[y/n]`, to which s/he has to answer with a `y` and then with the new middle value.

Once the values are corrected, the user will be asked if the extracted values are to be saved with the promt `Valid results?[y/n]`. Answering with a `y` will save the calculated kinematic features and follow with the next data file. Once the '.csv' files are finished, the script will end.

To finish this step, the script should be executed for both hands, resulting in

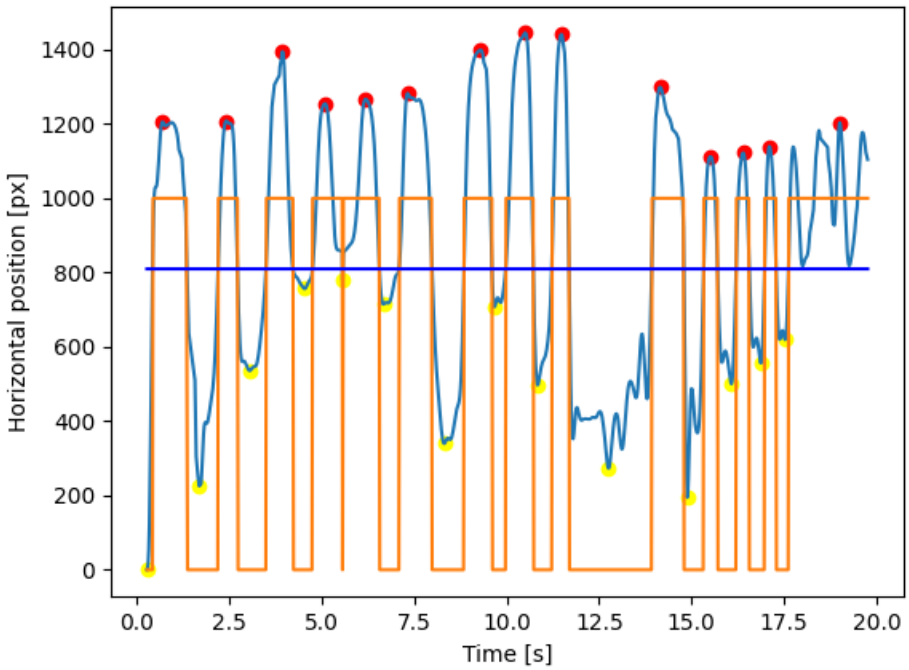


Figure D.6: Incorrect plot of the movement.

the two '.csv' files containing all the kinematic data of both hands for all analysed patients. This data can be used to to be analysed separately or the Machine Learning Classification step can be followed.

D.5 Classification

For this final step, the '.csv' files containing the hand kinematic features and the excel file with the subject information are needed. Additionally, we will use the two Jupyter Notebooks `ML_binary.ipynb` and `ML_multiclass.ipynb`. Although these notebooks are self-explanatory, a fast summary will be followed in this guide.

First, the 3 files are imported and merged into one DataFrame. Taking into account the dominant hand information on the excel file, the information regarding the right and left hands is transformed into dominant and non-dominant hands. Then, information related to standard deviations, maximum and minimum values is discarded.

Once the DataFrame is set, correlations are calculated and the boxplots of the most correlated features are plotted. Additionally, PCA and LDA projections are performed obtaining numerous datasets that will be tested for the best performance.

The aforementioned datasets, are used to train four types of models, tuning their parameters to see changes in performance. These models are Logistic Regression Classifier, Support Vector Machine, K-Nearest Neighbors Classifier and Random Forest. The accuracy for each of the models is plotted and when the best parameters

are selected, the ROC curve and some additional performance metrics are calculated.

Both Jupyter Notebooks follow the same pipeline and calculations.