TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Diseño e integración de un sistema de clasificación de alteraciones del movimiento de pacientes con rodillas lesionadas

Autor

Juan Manuel López Castro

Directores

Oresti Baños Legrán

Miguel Damas Hermoso

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Agosto de 2020

Diseño e integración de un sistema de clasificación de alteraciones del movimiento de pacientes con rodillas lesionadas

Juan Manuel López Castro

Palabras clave: EMGs, fatiga, clasificación, lesión, diagnóstico, LCA, sistema nervioso central, aprendizaje automático, selección de características

Resumen

En la actualidad la electromiografía de superficie está jugando un papel muy importante apoyando a los fisioterapeutas y médicos para analizar el rendimiento de atletas y pacientes, así como para detectar posibles problemas musculares a la hora de realizar tanto una actividad física como una actividad cotidiana.

Existen evidencias de que personas con lesiones en la rodilla, como por ejemplo, la rotura total o parcial del ligamento anterior cruzado, sufren cierta compensación muscular entre el cuadriceps y el isquiotibial. Basándonos en esto se plantea realizar un sistema automático para la detección de la fatiga muscular. Con dicho sistema un fisioterapeuta podría hallar que músculos entran en fatiga antes y determinar si hay o no cierta compensación entre músculos antagonistas, y por lo tanto probabilidad de mal funcionamiento de la rodilla debido a una lesión.

Para ello se ha realizado un estudio sobre que tipos de clasificadores obtienen un mejor rendimiento a la hora de clasificar las señales de electromiografía entre fatiga y no fatiga. Se han hecho uso de datos de electromiografía tomados a un conjunto de bomberos que han realizado sentadillas, todos con un peso cargado del 70 por ciento de su RM. En cada repetición tomada también se ha registrado la velocidad de ejecución, para poder estimar la fatiga muscular.

Design and integration of a classification system for movement disorders in patients with injured knees

Juan Manuel López Castro

Key words: EMGs, fatigue, detection, injury, medical diagnostic, Central Nervous System, machine learning, feature selection

Abstract

Nowadays surface electromyography is playing a important role supporting to physiotherapist and medics to analyse athlete's performance as well as to detect possible muscular problems when they are performing some activity.

There is evidence that people with knee injuries, such as a total or partial tear of the anterior cruciate ligament, suffer some muscle compensation between the quadriceps and the hamstring. Based on this, the goal is create an automatic system for the detection of muscle fatigue. With this system, a physiotherapist could find that muscles go into fatigue earlier and determine whether or not there is some compensation between antagonist muscles and therefore the probability of knee malfunction due to injury.

To solve the problem, a study about which types of classifiers have better performance when classifying electromyography signals between fatigue and non-fatigue. Electromyography data was taken from a set of firefighters who have performed squats, all with a loaded weight of 70 percent of their MR. In each repetition taken, the execution speed has also been recorded, to be able to estimate muscle fatigue.

Yo, **Juan Manuel López Castro**, alumno de la titulación INGENIERÍA INFORMÁTICA de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI , autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Juan Manuel López Castro

Granada a 2 de Septiembre de 2020

D. **Oresti Baños Legrán** , Profesor del Área de Ingeniería de Sistemas y Automática del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

D. **Miguel Damas Hermoso**, Profesor del Área de Arquitectura y Tecnología de Computadores del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Diseño e integración de un sistema de clasificación de alteraciones del movimiento de pacientes con rodillas lesionadas***, ha sido realizado bajo su supervisión por **Juan Manuel López Castro**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 2 de Septiembre de 2020

Los directores:

Oresti Baños Legrán **Miguel Damas Hermoso**

Agradecimientos

Gracias por el apoyo a todos mis familiares y amigos.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Breve introducción donde se explica la motivación de dicho trabajo, como ha sido planteado en el tiempo, sus objetivos principales y secundarios y la estructura que tiene cada uno de los capítulos que forman la memoria.

1.1. Motivación

Cierto es que el uso de la informática y la tecnología está en auge en nuestra sociedad. Esto está llegando no solo a facilitar nuestra vida cotidiana, sino también a apoyar y beneficiar la aplicación de la medicina y la salud. Por ejemplo, la electromiografía (EMG) es de gran utilidad para analizar el comportamiento y la activación de las unidades motoras (Figura 1.1) a la hora de realizar cierta actividad física. Esto, junto con la multitud de características que se pueden extraer de las señales EMG adquiridas, hace que se puedan realizar multitud de estudios para analizar el rendimiento y patologías de pacientes. Haciendo uso de un buen diseño e implementación se puede llegar a obtener un sistema de tiempo real donde se pueda realizar el diagnostico en el mismo momento de la actividad física y poder analizar los diferentes datos tomados [30].

Existe un gran número de deportistas que cuando entrenan hacen todo lo posible para evitar la fatiga muscular. Por ejemplo, sería el caso de atletas que practican powerlifting o halterofilia, donde el objetivo es levantar el mayor peso posible en una sola repetición. Estos atletas evitan los entrenamientos bajo fatiga debido a que cuando entrenan bajo fatiga no consiguen un rendimiento óptimo. Esto se debe a que cuando se entrena bajo fatiga, el esfuerzo que se tiene que realizar es mucho mayor, así como la probabilidad de lesión. Además aunque el esfuerzo realizado sea mayor, no implica que la fuerza ejercida sobre la carga levantada también lo sea, ya que no se están reclutando tantas fibras musculares.

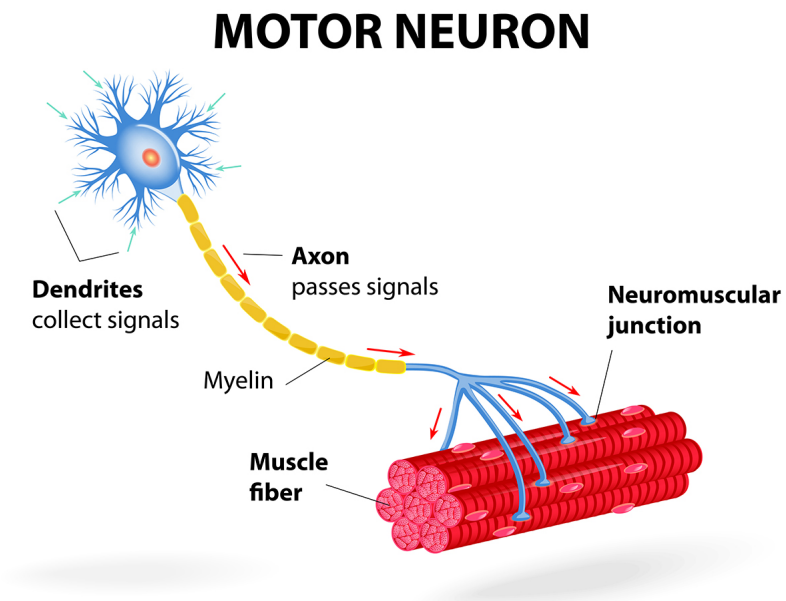

Figura 1.1: Ejemplo de motoneurona [19]

El aumento del esfuerzo realizado y la mayor probabilidad de lesión se deben a que cuando el sistema nervioso central entra en fatiga no recluta tantas unidades motoras y por ende, no existe tanta activación muscular. Como sabemos la musculatura complementa al sistema óseo y por lo tanto un déficit en la implicación de algún músculo debido a la fatiga hará que se incremente la posibilidad de lesión en articulaciones y ligamentos. Por ejemplo, en algunas lesiones de rodilla como la rotura del ligamento cruzado anterior, se produce una compensación entre cuádriceps e isquiotibial. Una entrada en fatiga de ambos músculos, quitaría mucha estabilidad a la rodilla y aumentaría el daño en el ligamento. Si el ligamento, en lugar de sufrir una rotura total, solo estuviese roto parcialmente, es decir, una lesión más leve, lo que se conseguiría entrenando bajo fatiga sería incrementar la lesión. Todo esto se explica con mayor detalle en el capítulo 4 .

En este proyecto se ha tratado de desarrollar una herramienta que ayuda a los fisioterapeutas a la hora de detectar la fatiga muscular. Con ello se pretende dotar al fisioterapeuta de la suficiente información como para determinar que músculo ha experimentado dicha fatiga y así poder estudiar si se ha producido una compensación muscular a la hora de la ejecución de una sentadilla y determinar si hay una lesión en dicha rodilla. Además, como se ha comentado en párrafos anteriores, en multitud de sistemas de entrenamientos basados en fuerza, está desaconsejado el entreno bajo fatiga, ya que se considera contraproducente. Esto implica otro posible uso de dicho sistema, y es que podría ser utilizado por atletas expertos, en entrenamientos de fuerza como powerlifting o halterofilia para detectar la fatiga en el

momento del entreno y cortar la serie para descansar y no tener un entreno contraproducente.

Para determinar la fatiga en este proyecto, se ha puesto la condición de que cuando un atleta está realizando una serie de sentadillas y su velocidad de ejecución decae un 20 por ciento, es un indicativo de fatiga. Esta condición se debe de cumplir en dos repeticiones seguidas, para así determinar una fatiga más representativa. La condición de la caída de velocidad fue propuesta por el fisioterapeuta de la empresa encargada de la toma de datos, mDurance [26], además de ser bastante utilizada en el mundo del deporte [28].

Cabe destacar que este proyecto ha sido llevado a cabo con la colaboración de mDurance [26], una empresa especializada en electromiografía y que ha sido la encargada de aportar los datos para realizar el estudio de los diferentes clasificadores. Han aportado datos de electromiografía sobre los músculos vasto medial y recto femoral de ambas extremidades inferiores a la hora de realizar sentadillas.

1.2. Planteamiento del trabajo

Para solucionar el problema de la detección de la fatiga muscular y también poder determinar si existe o no compensación muscular y por ende una lesión en la rodilla se pretende desarrollar un modelo que clasifique las señales de EMG recibidas entre fatiga y no fatiga.

Para ello, el desarrollo del modelo consta de tres fases principales. Una primera fase de extracción de características de la señal EMG, tales como RMS, MAV, MDF, etc. En el capítulo 5 se comentan todas las características estudiadas junto con su definición. Una segunda fase de selección de las características más representativas de la fatiga, para quedarnos con la menor cantidad posible y así obtener un clasificador más eficiente y rápido. Finalmente una fase de clasificación donde se probarán diferentes clasificadores para finalmente quedarnos con el que mejor prestaciones tenga para dicho problema. Con dicho clasificador se podría clasificar las señales EMG recibidas en tiempo real para determinar si la persona que está realizando las repeticiones de sentadillas ha sufrido fatiga y en el momento en el que la ha sufrido.

Con esta herramienta los fisioterapeutas y médicos podrían realizar pruebas a sus pacientes para luego llegar a ciertas conclusiones. Todo esto sin la necesidad de tener una maquinaria muy compleja, ya que una vez implementado y entrenado el clasificador, no sería necesario medir la velocidad de ejecución de las series de los pacientes a los que se va a realizar la prueba.

Por último comentar que para la realización de este proyecto han sido

necesarios datos reales, tomados y filtrados por la empresa especializada en electromiografía mDurance. Además de los datos de las diferentes señales de electromiografía, fue necesario una medición de la velocidad de cada repetición para así, al tener la velocidad de cada repetición poder estimar que sujetos experimentaron fatiga y utilizar dichos datos para entrenar y analizar el rendimiento de los diferentes clasificadores.

1.3. Objetivos

Como consecuencia de lo planteado anteriormente el objetivo general sería obtener un clasificador que reciba los datos de la señal EMG y clasifique en tiempo real entre fatiga y no fatiga. Esto servirá para detectar la fatiga en tiempo de ejecución con una herramienta muy sencilla y una vez entrenado el modelo, no sería necesario la utilización de un velocímetro para medir la velocidad de ejecución de cada repetición y estimar la fatiga. Esto se debe a que nuestro modelo, una vez entrenado, solo por el análisis de las señales será capaz de catalogar las señales de EMG entre fatiga y no fatiga. Además una vez clasificadas las señales EMG, el personal especializado para ello podrá determinar si existe compensación entre músculos implicados en el movimiento normal de la rodilla y determinar si existe algún tipo de lesión.

Con respecto a los objetivos específicos, están muy relacionados con las tres fases comentadas anteriormente.

- Analizar y procesar las señales brutas de EMG.
- Analizar la velocidad registrada de cada sujeto.
- Estimar la fatiga de cada uno de los sujetos.
- Extraer características de las señales de EMG.
- Crear los archivos de entrenamiento-testeo.
- Seleccionar las características más útiles para el modelo final.
- Clasificar las señales de EMG.
- Validar el rendimiento de los modelos.

1.4. Estructura del trabajo

En este capítulo 1, todo lo comentado es una breve introducción al problema a resolver y a la realización del proyecto. Más adelante se explicarán con más detalle las pautas que se han seguido para el desarrollo del proyecto.

En el capítulo 2 se comenta el estado del arte donde se hará un análisis en profundidad del problema en cuestión, de toda la tecnología que abarca y su posible aplicación en la actualidad.

En el capítulo 3 se comenta la planificación llevada a cabo para la realización del proyecto, así como el coste económico final y parcial.

En el capítulo 4 se hace un análisis del problema tratado. La fatiga muscular, como afecta al sistema nervioso y los problemas musculares asociados a las lesiones de rodilla.

En el capítulo 5 se realiza un diseño de todas las fases realizadas para el proyecto. Consta de un gran número de esquemas donde se explica el funcionamiento de los algoritmos utilizados y los tipos de archivos de entrenamiento-testeo implementados.

El capítulo 6 es el dedicado a la implementación. En él se puede encontrar el código asociado al desarrollo del proyecto.

El capitulo 7 está dedicado a la sección de pruebas. En él se pueden encontrar todos los resultados de los clasificadores diseñados y una opinión personal sobre sus rendimientos.

Por último, el capítulo 8 es el dedicado a las conclusiones y trabajos futuros. En él se comenta si se han cumplido los objetivos de este proyecto y consejos para una futura aplicación junto a una herramienta de electromiografía.

Capítulo 2

Estado del arte

A continuación se explica a fondo la temática de este proyecto, así como sus posibles aplicaciones en la actualidad.

2.1. EMG

La electromiografía se ha convertido en una de las técnicas más utilizadas en la actualidad para estudiar el potencial de acción de los músculos. Hacen uso de ella tanto neurofisiólogos, neurólogos y rehabilitadores. Se basa en la captación de las señales electromiográficas, las cuáles, son señales producidas por la musculatura a la hora de realizar una fase de contracción o relajación. Esta fase de contracción o relajación es producida por el estímulo de las motoneuronas. Existen dos tipos de motoneuronas. En este ámbito nos referimos a las motoneuronas somáticas que son las encargadas de las contracciones y relajaciones de la musculatura cuya acción es voluntaria.

Las motoneuronas forman parte de las unidades motoras (Figura 2.1). Una unidad motora es realmente la encargada de las contracciones y relajaciones musculares y está formada por el conjunto de fibras musculares que afecta y la motoneurona encargada de enviar el estímulo. Cuando el impulso eléctrico de la motoneurona llega a las fibras musculares que afecta se obtiene un potencial de activación. El conjunto de todos los potenciales de activación de todas las fibras musculares que son activadas por una motoneurona se le conoce como el potencial de acción de la unidad motora. Lo que se analiza en la electromiografía es la actividad de las múltiples unidades motoras encargadas de un músculo [30, 37].

Estos impulsos eléctricos generados por las motoneuronas somáticas y que contraen o relajan el músculo son registrados en forma de señales electromiográficas y son utilizadas para ayudar a diagnosticar problemas de salud, que a simple vista, no pueden ser identificados.

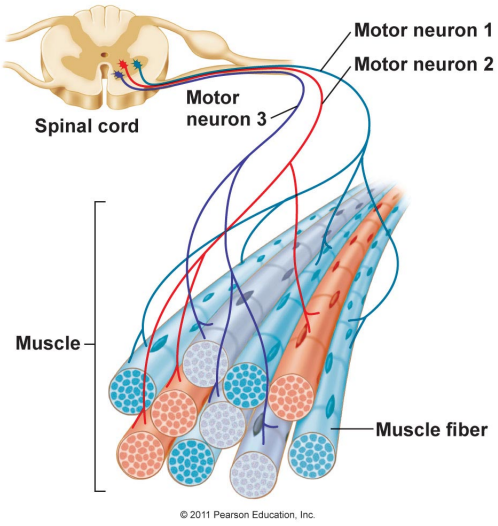

Figura 2.1: Ejemplo de unidad motora [20]

Para medir estas señales electromiográficas se pueden seguir dos métodos, el superficial y el intramuscular [30].

En la electromiografía intramuscular (Figura 2.2), se utiliza una aguja electrodo para medir la actividad muscular. Esta aguja es introducida por la piel hasta llegar al músculo que queremos estudiar. Es mucho más precisa que la electromiografía de superficie, ya que al ser una aguja introducida directamente en el músculo, no recibe ruido externo de músculos adyacentes al estudiado. Por el contrario es más incomoda, ya que cada vez que se quiera hacer dicho análisis el paciente debe de ser pinchado, y puede que no esté dispuesto a dicho tratamiento. Además si se quiere analizar un actividad física compleja, como salto vertical, horizontal, carrera o sentadilla, como en el caso de este proyecto, el tener unas agujas introducidas en la musculatura puede ser molesto y hacer que la actividad física no se desarrolle a plenitud. Por último, para la utilización de esta técnica, se deberían de tener ciertos conocimientos médicos, ya que una persona sin dichos conocimientos no puede ser capaz de realizar el análisis de la forma correcta, al no conocer por ejemplo, el lugar óptimo para introducir la aguja. Otro agravante es que sin estos conocimientos, se podría llegar hasta la lesión del paciente, debido a introducir la aguja en el lugar no indicado.

En la electromiografía de superficie (Figura 2.3) se utilizan electrodos situados en la piel, que captan dichas señales electromiográficas. Es un método no invasivo y fácil de utilizar, donde solo es necesario tener el conocimiento de cual es el lugar óptimo para la colocación de los electrodos. Existen varios tipos de electrodos que serán explicados en el próximo apartado. En

este proyecto se ha hecho uso de la electromiografía de superficie.

Figura 2.2: Electromiografía intramuscular [16]

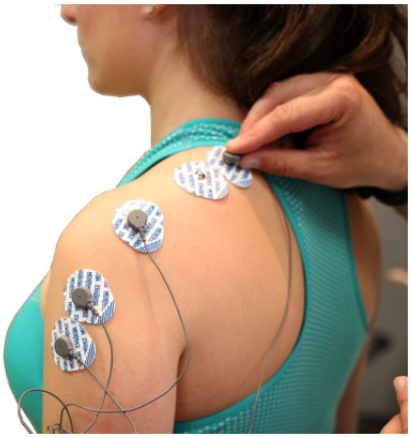

Figura 2.3: Electromiografía de superficie [17]

EMG de superficie

Para la realización de este proyecto se ha hecho uso de la electromiografía de superficie (EMGs). Esta técnica implica la utilización de electrodos situados en la piel, para la captación de las señales eléctricas. Por esto mismo, es necesario que en la mayoría de las ocasiones, para una mayor precisión en los datos tomados se realice una preparación de la piel en donde se van a situar los electrodos.

Por ejemplo, para que los electrodos queden mejor adheridos a la piel, a veces es necesario rasurar la piel para eliminar los pelos y que haya más superficie de contacto. También suele ser necesario limpiar la piel para eliminar partículas que dificulten la adherencia de los electrodos. Existen también electrodos que necesitan de la aplicación de un gel para su colocación, pa-

ra así aumentar la precisión del electrodo. La funcionalidad de este gel es aumentar la conductividad entre la piel y el electrodo.

Otro factor importante es el tamaño del electrodo. Cuanto más pequeño sea, más precisión muscular nos dará, ya que, no cubrirá tanta superficie y no captará así actividad de músculos en los que no estamos interesados. El tamaño de los electrodos debería ser de 1 cm o menor [30].

2.2. Aplicaciones médicas

Como se ha comentado en secciones anteriores, el estudio de las señales de electromiografía es de gran utilidad para diagnosticar problemas motores y de salud. Algunas de estas enfermedades son: [31]

- Enfermedad de la neurona motora, ELA.
- Radioculopatías, afectan a la columna vertebral, como por ejemplo, la ciática o la hernia de disco.
- Plexopatías. Es difícil de conocer su origen. Produce un dolor y falta de movilidad en el brazo u hombro. Con el uso de la EMG intramuscular se puede llegar a conocer su origen.
- Mononeuropatías agudas. Implican un daño en la medula espinal y produce hormigueo en la zona afectada.
- Miopatía. Anomalía o mal funcionamiento en un grupo muscular, debido a un mal impulso eléctrico de las neuronas o a un mal funcionamiento del grupo muscular.

En este último punto va a ir centrado el proyecto, en facilitar el diagnostico de rodillas lesionadas a los fisioterapeutas que hagan uso de dicha herramienta. Para ello hace falta indagar en la base de la fatiga muscular y en que se ha basado el proyecto.

2.3. Entrenamientos basados en velocidad y fatiga

En la actualidad existe gran interés entre los atletas tanto profesionales como no profesionales, de mejorar su rendimiento. Para ello es indispensable pensar en un buen entrenamiento, ya que es la base de un buen rendimiento en la competición final. Para ello diversos estudios se han centrado en detectar la fatiga en los entrenamientos, ya que se considera, que entrenar bajo fatiga lo único que nos aporta es un lastre en el entrenamiento, ya que

no reclutamos todas la unidades motoras necesarias y no se estimula lo suficiente al músculo. Además de que esto también supone un daño muy alto al sistema nervioso central.

Uno de los factores que indica la fatiga es la perdida de velocidad de ejecución en las actividades físicas, como por ejemplo, en las sentadillas. Se considera que una perdida del 20 por ciento en la velocidad de ejecución es indicativo de que existe una fatiga y por lo tanto se debe proceder a cortar la serie de entrenamiento y dar unos minutos de descanso para proseguir cuando el sujeto ya no esté sufriendo dicha fatiga [28].

Esta técnica dota al atleta de una gran capacidad de entrenamiento, ya que de dicha manera entrenará sin llegar a su limite y realizando siempre series efectivas, las cuáles, harán que se reclute el mayor número de unidades motoras y así obtener una mayor fuerza y desarrollo muscular.

2.4. Aplicación a este proyecto

Como se ha explicado la electromiografía ha permitido grandes avances en la medicina y en el mundo del deporte. Este proyecto está más encaminado hacia los fisioterapeutas. Pretende desarrollar un sistema de clasificación, para que, sin la necesidad de medir la velocidad de ejecución de las series, se pueda determinar en el momento exacto de la prueba si el atleta está experimentando o no fatiga. Esto será de gran utilidad a atletas que quieran mejorar sus entrenamientos, así como, para que los fisioterapeutas obtengan más información, complementaria a los valores utilizados para la determinación de la actividad muscular como RMS, MNF, MDF, MAV, etc.

Con toda esta información podrían llegar a conclusiones muy interesantes, tales como, en la ejecución de una sentadilla ver que músculos son los que entran en fatiga y determinar si se está produciendo una compensación muscular debido a una lesión de rodilla o por ejemplo, si dicho músculo está sufriendo algún tipo de daño y por ende su bajo rendimiento y fatiga tan temprana. También puede ser utilizada para mejorar el rendimiento a la hora de entrenar, ya que detectaría la fatiga en tiempo real, y dotaría al atleta de una información real sobre su fatiga y así podría determinar si se necesita un descanso o por el contrario seguir realizando la actividad física.

Capítulo 3

Planificación y coste

Este capítulo trata sobre la planificación llevada a cabo y el coste total de la implementación. Con respecto a la planificación se explica tanto la planificación llevada con los tutores responsables de dicho proyecto como la planificación personal a la hora de desarrollar el proyecto.

3.1. Planificación

La planificación de este proyecto comenzó con una reunión a finales de noviembre de 2019, para tratar el problema a resolver. La reunión fue llevada a cabo por Miguel Damas Hermoso, Oresti Baños Legrán, ambos tutores de este proyecto e Ignacio Díaz Reyes, CEO de mDurance, empresa encargada de aportar todas las herramientas necesarias para la toma de datos, así como, los datos en cuestión para la realización de este proyecto. En esta reunión se debatió sobre la temática del proyecto y sobre lo que se intentaba aportar con dicha solución. También se comentó el "dataset"que iba a ser utilizado para el estudio, el cuál se iba a encargar de tomar un centro de Barcelona que trabajaba junto a mDurance.

Para finalizar se me realizó una prueba con la herramienta que iba a ser utilizada para medir las señales de electromiografía para ver como funcionaba y aumentar mi conocimiento sobre el tema.

A continuación se explica el restante de las reuniones llevadas a cabo y la planificación personal a la hora de desarrollar el proyecto.

3.1.1. Sesiones formativas y reuniones

Con respecto a la adquisición de conocimientos sobre electromiografía, toma de datos, filtración y amplificación de las señales tomadas, recibí dos

sesiones formativas en el centro de mDurance de aproximadamente dos horas. En estas reuniones Ignacio Díaz Reyes realizó una introducción hacia la electromiografía y de sus aplicaciones en el mundo actual. A parte de esto, también nos habló sobre las diferentes características de las señales, sobre todo de las más conocidas y utilizadas en la actualidad. Esto fue complementado mostrando análisis realizados sobre pacientes para ver como se puede diagnosticar ciertos déficits en la musculatura solo con extraer características de las señales EMG. Por último comentar que me facilitó documentación sobre la EMG y sobre sus características, tanto en el dominio del tiempo como en el dominio de la frecuencia.

Después de estas dos sesiones, que se realizaron el 21 de enero y el 11 de febrero de 2020, ya estaba preparado para empezar el proyecto en la fase de extracción de características de la señal EMG.

La próxima reunión que realizamos fue por medio de GoogleMet, ya que había un problema con los datos que supuestamente iban a ser utilizados para el proyecto. Debido al COVID-19 y a la declaración del estado de alarma, dichos datos no pudieron ser tomados, por lo que nos encontrábamos a mitad de marzo y sin datos para poder realizar el proyecto. Esta reunión fue clave para poder realizar el proyecto. En ella, ambos tutores, junto con Ignacio y Carlos, el fisioterapeuta de mDurance, hicimos una puesta en común de ideas para poder obtener otro "dataset"que supliera al original. Se acordó realizar una búsqueda sobre posibles ideas y zanjar el problema en una próxima reunión.

En la siguiente reunión, unos tres días posterior a la anterior, Ignacio aportó una solución, ya que había hablado con un colaborador que se iba a encargar de tomar un nuevo "dataset". Esta nueva orientación era muy parecida a la original, solo que añadía una nueva idea, la fatiga. Con esto además de hallar que pacientes sufrían de rodilla lesionada, también podía ser utilizado para detectar la fatiga muscular. Este "dataset"trataba sobre un ejercicio multiarticular, la sentadilla. Se acordó que dispondría de los datos en la mayor brevedad posible. Finalmente tuve acceso a parte de los datos el 24 de mayo y el 20 de junio a la totalidad de ellos. Por lo que esto implicó la imposibilidad de poder presentar el proyecto en la primera convocatoria.

Una vez que ya disponía de los datos, a partir del 13 de julio, tuve reuniones consecutivas con ambos tutores, para ir mostrándoles como iba el desarrollo del proyecto e ir acordando el formato de la memoria y todo lo que debía de añadir.

El 27 de julio tuvimos una ultima reunión donde les mostré un prototipo bastante avanzado del desarrollo, el cual, vieron bastante bien, y se acordó que ya solo tenía que desarrollar el proyecto final y complejo, así como el desarrollo de la memoria.

Para la memoria se acordó enviar un prototipo para mediados de agosto, y la memoria final para el 3 de septiembre, para así tener siempre un margen de error y poder corregir posibles errores antes de la entrega final de los proyectos.

Con respecto a mi propio trabajo, se comenta en la siguiente sección.

3.1.2. Implementación y memoria

Una vez realizadas las dos sesiones formativas de EMG comentadas anteriormente, ya podía empezar a indagar en el tema de la electromiografía y su análisis de datos. Durante los meses de enero, febrero y marzo, debido a la incertidumbre de los datos explicada, tuve acceso a unas señales EMG de prueba facilitadas por mDurance para poder hacer el desarrollo de la extracción de las características. Durante estos meses fui obteniendo más conocimientos sobre las señales de EMG, sus características y utilidad, así como, sobre la clasificación binaria de dichas señales. Además con los datos de prueba, pude realizar la implementación de los algoritmos de extracción de características que yo consideré más relevantes para la realización de este proyecto.

A partir del 24 de mayo, cuando recibí la primera parte del dataset final, debido a que la fase de extracción de características ya la tenía realizada, me dediqué a indagar sobre el tema de clasificación de señales usando librerías de Python, y a investigar que algoritmo de selección de características era el más indicado para obtener un mejor rendimiento en este proyecto.

Durante las reuniones de julio, mis tutores fueron orientándome sobre el trabajo que iba realizando, y así el 27 de julio pude tener un prototipo, al que sólo sería necesario añadirle más complementos para que fuese el resultado final. Con todo esto para el 3 de agosto, ya tuve una versión final del proyecto y empecé el desarrollo de la memoria del proyecto.

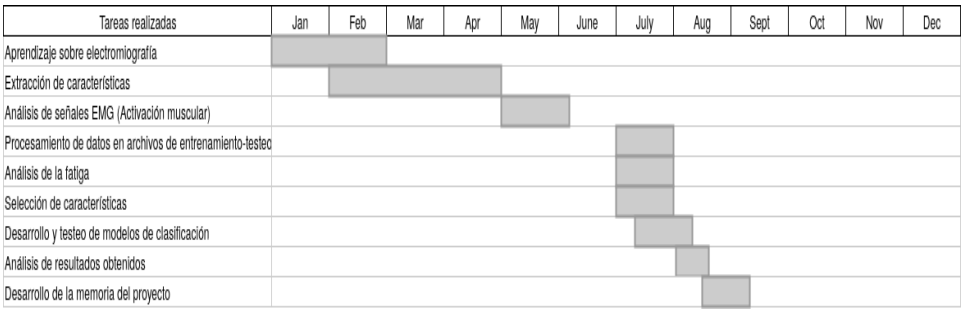

Figura 3.1: Diagrama de Gantt sobre el trabajo personal

3.2. Coste

Una vez realizado el estudio sobre la clasificación y viendo cual sería el mejor clasificador que resolviese el problema, el siguiente paso sería llevarlo al mercado. Habría que realizar una aplicación móvil, conectada a dicho sensor para que mientras se tome los datos, un programa paralelo los analice y nos de los resultados de la fatiga en el tiempo de ejecución, intentando una eficiencia máxima y el mínimo tiempo de procesamiento de datos. Para realizar dicho trabajo los costes se podrían dividir entre el coste de la tecnología utilizada para la obtención de los datos y el coste de los recursos humanos encargados de desarrollar el software.

3.2.1. Tecnología utilizada

Con respecto al precio de la tecnología utilizada, para la medición de los datos se ha utilizado el sensor de mDurance. Se han evaluado ambas rodillas a la hora de realizar una sentadilla por lo que se ha hecho uso del mDurance Premium de 4 canales (Ver Imagen 3.2), que además también incluye un sensor para medir el rango de movimiento de cada repetición. Esta herramienta tiene un coste actual de 5.445 € (Enlace a tienda [23]).

Figura 3.2: Electromiógrafo mDurance [22]

3.2.2. Recursos humanos

Para el desarrollo software de la aplicación un ingeniero informático especializado en el desarrollo y la implementación de software cobra unos 2000 €/mes [25]. Estimo que el desarrollo de dicha aplicación se podría llevar a cabo en 2 meses de trabajo, teniendo en cuenta un testeo de funcionamiento para que no requiera un gran mantenimiento al tener la mayoría de errores depurados.

A este precio habría que añadirle lo que cobra un fisioterapeuta/entrenador que será el encargado de tomar los datos a sus clientes. El sueldo medio en España de un fisioterapeuta es de 1217 €/mes [24].

El precio total de los recursos humanos seria de unos 5217 €, debido a dos meses de trabajo del ingeniero informático y a un mes de trabajo del fisioterapeuta encargado de la toma de datos para haber podido realizar el estudio sobre que clasificador es el que obtiene mejores prestaciones.

3.2.3. Precio del servicio final

Todo esto haría un presupuesto final de unos 10662 €, un precio bastante elevado, pero si lo analizamos muy rentable, ya que se trata de una herramienta pionera en el sector, que si se desarrolla de la forma adecuada de seguro que sería utilizada por una gran cantidad de atletas y dejaría grandes beneficios.

Este software podría ser implementado dentro de la herramienta base de mDurance, aportando así una funcionalidad extra.

Servicio	Precio / Mes	Duración	Precio total
electromiografo MDurance Premium de 4 canales	-	-	5.445 €
Ingeniero de software	2.000 € / mes	2 meses	4.000 €
Fisioterapeuta	1.217 € / mes	1 mes	1.217 €
			10.662 €

Figura 3.3: Desglose de los precios del proyecto

Capítulo 4

Análisis del problema

Este capítulo está dedicado a una explicación a fondo sobre el análisis del problema. Para ello se comentan los factores más relevantes sobre la fatiga muscular y su detección. Además se comenta un posible caso de lesión en rodillas a la hora de analizar la fatiga.

4.1. Fatiga y sistema nervioso central

La aparición de la fatiga durante la actividad física implica grandes cambios en la realización y percepción del esfuerzo. Esto hace que al realizar varias repeticiones, de por ejemplo en este caso, sentadillas, la acumulación de fatiga haga que cada vez realicemos una peor técnica y debido a esto aumenten las posibilidades de sufrir una lesión [32]. Es decir, después de una sesión de entrenamiento, debido a la fatiga, no podemos realizar el mismo esfuerzo muscular y esto aumenta las deficiencias en la musculatura y puede llegar a provocar lesiones si no se trabaja con cuidado.

Otro factor a tener en cuenta, es que ciertos grupos musculares antagonistas pueden entrar en fatiga antes que otros debido a una mayor activación. Esto será tratado más a fondo en la sección de compensación muscular (Ver 4.3).

La fatiga está muy relacionada con el sistema nervioso central, ya que el sistema nervioso central (Ver 4.1) es el encargado de generar los impulsos eléctricos para que se produzcan las contracciones musculares. Cuanto mayor sean estos impulsos mayor grupo de motoneuronas serán implicadas y un mayor grupo de fibras musculares reclutadas, por lo que se tendrá una mayor fuerza física, así como un mayor control del peso cargado y una mayor técnica. Por ende si el sistema nervioso central está fatigado se tendrá un menor control del peso cargado, menor fuerza y peor técnica. Eso sería la llamada fatiga central, es decir, la fatiga del sistema nerviosos central.

Como bien se comenta en [32] esta fatiga todavía es difícil de detectar, pero puede provocar una alteración en la transmisión del sistema nervioso central y en el reclutamiento de los axones motores. Debido a esto, el sujeto que está experimentando dicha fatiga no puede realizar toda la fuerza que sería capaz de realizar en condiciones normales, ya que como se ha explicado antes, no se reciben tantos impulsos para reclutar las fibras musculares. Esto hace que se vea reflejada una caída en la velocidad de ejecución de las sentadillas. Esto es debido a que la velocidad de ejecución y la fuerza realizada están muy relacionadas debido a que cuanto mayor es el número de fibras reclutadas mayor es la fuerza realizada y por consiguiente una mayor velocidad a la hora de realizar las sentadillas. Debido a esto, se ve bastante claro que fatiga y velocidad de ejecución en los entrenamientos son dos términos que están estrechamente relacionados.

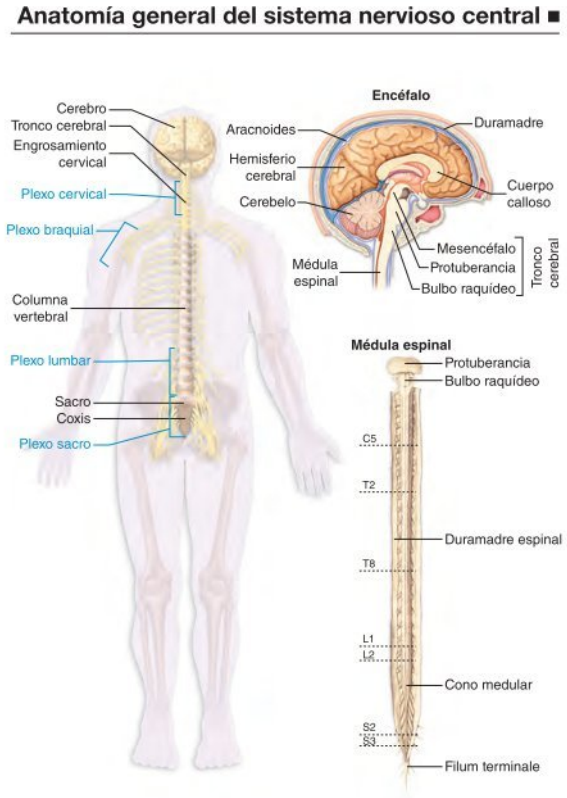

Figura 4.1: Sistema nervioso central [21]

4.2. Velocidad de ejecución

Como bien se ha explicado anteriormente, la fatiga y la velocidad de ejecución son dos términos muy relacionados. Debido a ello un gran número de atletas actuales han basado sus entrenamientos en la velocidad de ejecución. Cuando detectan una caída de velocidad realizan un descanso de la actividad física, para no llegar a niveles muy extremos de fatiga y así poder sacar el máximo partido a sus sesiones de entrenamiento.

Para la detección de esta fatiga se ha estimado que la caída de velocidad de ejecución debe de ser de un 20 por ciento [28]. Para ello se tendrá en cuenta dicho porcentaje con respecto a la primera repetición realizada.

A continuación se muestra una tabla-resumen, extraída de los datos utilizados para este proyecto, donde se pueden ver los valores de velocidad de ejecución de las 12 repeticiones registradas junto con el umbral de fatiga resultante. Finalmente una determinación de la fatiga muscular y una conclusión donde el entrenamiento debería de haber sido parado. Para la realización de este proyecto además de la caída del 20 por ciento en la velocidad, se ha estimado que dicha condición se debe de cumplir dos veces seguidas, es decir, en dos repeticiones realizadas consecutivamente, para así eliminar errores y obtener datos más precisos, ya que hay veces que existe caída de velocidad, pero no por fatiga, sino por factores externos, como la falta de concentración del atleta.

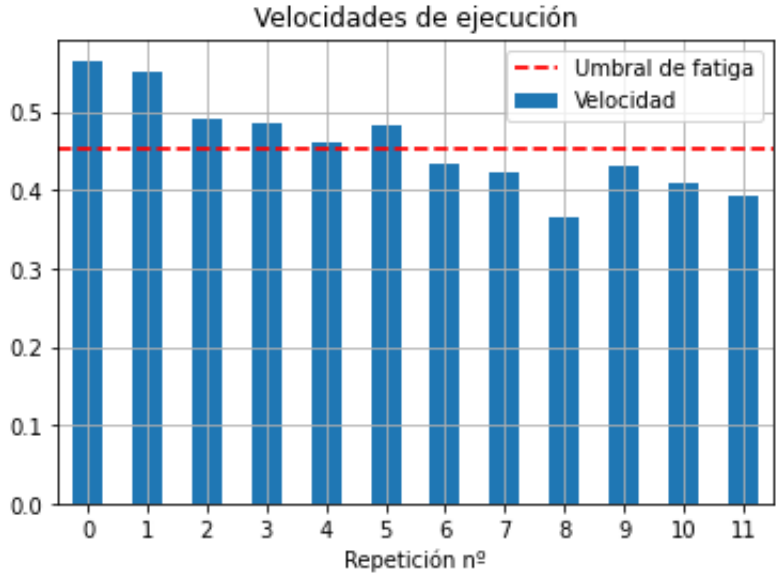

Figura 4.2: Velocidad y umbral de fatiga

4.3. Lesiones y compensación muscular

Anteriormente se ha comentado que el hecho de realizar esfuerzo físico bajo una gran fatiga puede hacer que nuestros déficits, musculares o articulares aumenten, debido a la perdida de control del sistema nervioso central, y se produzca una lesión.

A parte de esto, en sujetos que ya tienen una lesión en la rodilla, esto se ve más incrementado. Por ejemplo en la lesiones de ligamento cruzado anterior (Ver 4.3), se ha detectado que las personas que la sufren experimentan una serie de cambios en la musculatura de su pierna. En este estudio realizado sobre personas con dicha lesión [29] se demuestra que existe cierta compensación muscular entre los cuádriceps y los isquiotibiales cuando se sufre dicha lesión. Se demuestra que al haber sufrido una rotura del ligamento cruzado anterior, se produce un perdida de fuerza en la musculatura implicada en la rodilla, isquitibiales y cuadriceps. Sin embargo, esta perdida de fuerza es mucho mayor en los cuadriceps que en los isquiotibiales. Esto es debido a que los pacientes con el ligamento cruzado anterior lesionado, producen una compensación muscular. Se debe a que los cuadriceps son los encargados de la extensión de la rodilla y los isquiotibilaes de la contracción de esta. Por ello, cuando existe dicha lesión, el isquiotibial no deja que el cuadriceps funcione a plenitud, para así, evitar una fuerte extensión de rodilla y evitar producir una gran tensión en el ligamento anterior cruzado, el cual, está dañado.

Figura 4.3: Ligamento cruzado anterior: lesión parcial y completa [18]

Para ello la detección de la fatiga mediante diferentes clasificadores que reciban la información de los diferentes músculos examinados, puede ser clave para ver que músculos entran antes en fatiga, y si se produce en el

orden correcto. Si no es así, sería un gran indicativo de que existe cierta anomalía en la rodilla.

Por ejemplo, en la ejecución de una sentadilla, la activación de los cruadriceps debería de ser siempre mucho mayor que la de los isquiotibiales.

Capítulo 5

Metodología

Este capitulo está dedicado a la explicación sobre la metodología del proyecto. Se explica el diseño llevado a cabo a la hora de la extracción y clasificación de las características de la señal EMG. Además se explica el diseño de los 3 tipos de clasificadores que finalmente se han implementado, junto con sus diferentes archivos de entrenamiento-testeo.

5.1. Pipeline del proyecto

A continuación se muestra un breve resumen junto a un diagrama para mostrar el pipeline del proyecto y facilitar la compresión de la parte restante del diseño:

- Obtención de las señales EMG.
- Una vez obtenidas las señales EMG, pasamos a una fase de extracción de características, donde se extrae información relevante de las señales.
- Después de esto se procede con la creación de los diferentes archivos de entrenamiento-testeo que van a ser utilizados y con la estimación de la correspondiente fatiga de cada repetición registrada.
- El siguiente paso consta de una fase de selección de características de los archivos implementados para obtener así modelos sencillos y eficientes.
- Finalmente una fase de clasificación de las señales.

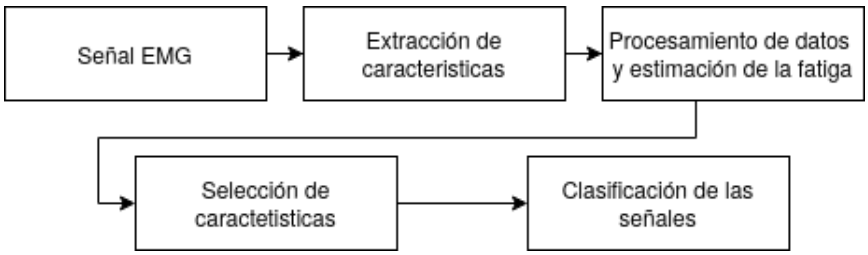

Figura 5.1: Pipeline del proyecto

5.2. Obtención de las señales EMG

Los datos utilizados para la realización del proyecto han sido tomados y filtrados con el electromiógrafo de mDurance ya comentado en secciones anteriores. Este electromiógrafo consta de 4 canales, por lo que se han podido analizar el rendimiento de 4 músculos, 2 de ellos de la extremidad inferior derecha y otros 2 de la extremidad inferior izquierda. Con esto lo que se ha obtenido son los datos de las señales de electromiografía asociados a unas pruebas. Para la obtención de los datos se tomó un conjunto de personas, que realizaron 6 series de 12 repeticiones de sentadillas con un peso relativo al 70 por ciento del RM (repetición máxima) de cada persona. Las 6 series fueron repartidas en dos días diferentes, es decir, 3 series por día, siendo estos días no consecutivos.

También se registraron las velocidades de cada repetición. Con las velocidades de cada repetición y partiendo de la condición explicada en el análisis del problema (ver 4.2), se pueden estimar las repeticiones en donde los sujetos experimentaron fatiga y utilizar todos estos datos para el desarrollo de nuestro modelo de clasificación.

5.3. Estimación del tamaño de ventana

Los datos obtenidos comentados anteriormente y facilitados por mDurance, constan de señales de electromiografía brutas. Para obtener información relevante de estas señales EMG primeramente hace falta estimar el tamaño de ventana de cada una de las repeticiones registradas en las señales.

Para ello se ha hecho uso de un proceso de detección de la actividad muscular a partir de una señal EMG, para así poder detectar fácilmente los tramos de la señal representativos de cada repetición y poder asociar a cada repetición su correspondiente estado de fatiga, analizando su velocidad registrada. Además con el diseño de dividir una señal EMG en sus diferentes activaciones (ventana de activación), nos aseguramos que los datos obtenidos

son reales y más precisos, ya que solo tiene el valor de cada repetición y eliminamos el ruido que ha podido ser producido a la hora de situar o quitar el electromiógrafo.

Además se llevó a cabo una selección de los archivos de EMG más útiles, ya que ciertos archivos, vienen con mucho ruido externo y no se llega a detectar las 12 repeticiones claramente.

5.4. Extracción de características

Una vez se ha estimado el tamaño de ventana de cada una de las repeticiones, el siguiente paso es la extracción de las características de la señal de cada una de las repeticiones registradas. Para ello con la estimación del tamaño de ventana podemos obtener unas características con unos valores muy fiables.

Existen un gran número de características, tanto en el dominio del tiempo, como en el dominio de la frecuencia. En este caso se han estudiado 15 características diferentes. (Para la obtención de conocimientos sobre estas características así como sus funciones se ha hecho uso de [36, 34]).

Con respecto a las características del dominio del tiempo he investigado sobre:

- Root mean square (RMS) [34]:

$$\mathrm{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}.$$

Figura 5.2: Definición de RMS [34]

- Mean absolute value (MAV) [34]:

$$\mathrm{MAV} = \frac{1}{N}\sum_{i=1}^{N} |x_i|.$$

Figura 5.3: Definición de MAV [34]

- Zero crossing (ZC) con umbral de 0,01 [34]:

$$ZC = \sum_{i=1}^{N-1} [\mathrm{sgn}(x_i \times x_{i+1}) \cap |x_i - x_{i+1}| \geqslant \mathrm{threshold}];$$
$$\mathrm{sgn}(x) = \begin{cases} 1, & \textit{if } x \geqslant \mathrm{threshold} \\ 0, & \mathrm{otherwise} \end{cases}$$

Figura 5.4: Definición de ZC [34]

- Waveform length (WL) [34]:

$$WL = \sum_{i=1}^{N-1} |x_{i+1} - x_i|.$$

Figura 5.5: Definición de WL [34]

- Slope sign change (SSC) con umbral de 0,01 [34]:

$$SSC = \sum_{i=2}^{N-1} [f[(x_i - x_{i-1}) \times (x_i - x_{i+1})]];$$
$$f(x) = \begin{cases} 1, & \mathrm{if} \quad x \geqslant \mathrm{threshold} \\ 0, & \mathrm{otherwise}. \end{cases}$$

Figura 5.6: Definición de SSC [34]

- Integrated EMG (IEMG) [34]:

$$IEMG = \sum_{i=1}^{N} |x_i|,$$

Figura 5.7: Definición de IEMG [34]

- Simple square integral (SSI) [34]:

$$\mathrm{SSI} = \sum_{i=1}^{N} x_i^2.$$

Figura 5.8: Definición de SSI [34]

- Variance of EMG (VAR) [34]:

$$\mathrm{VAR} = \frac{1}{N-1} \sum_{i=1}^{N} x_i^2.$$

Figura 5.9: Definición de VAR [34]

- Absolute value 3rd (TM3) [34]:

$$\mathrm{TM3} = \left| \frac{1}{N} \sum_{i=1}^{N} x_i^3 \right|;$$

Figura 5.10: Definición de TM3 [34]

- Absolute value 4th (TM4) [34]:

$$\mathrm{TM4} = \frac{1}{N} \sum_{i=1}^{N} x_i^4;$$

Figura 5.11: Definición de TM4 [34]

- Absolute value 5th (TM5) [34]:

$$\mathrm{TM5} = \left| \frac{1}{N} \sum_{i=1}^{N} x_i^5 \right|.$$

Figura 5.12: Definición de TM5 [34]

- Log detector (LOG) [34]:

$$\mathrm{LOG} = e^{\frac{1}{N}\sum_{i=1}^{N}\log(|x_i|)}.$$

Figura 5.13: Definición de LOG [34]

- Average amplitude change (ACC) [34]:

$$\mathrm{AAC} = \frac{1}{N}\sum_{i=1}^{N-1}|x_{i+1} - x_i|.$$

Figura 5.14: Definición de ACC [34]

Con respecto al dominio de la frecuencia he investigado sobre:

- Median frecuency (MDF) [34]:

$$\sum_{j=1}^{\mathrm{MDF}} P_j = \sum_{j=\mathrm{MDF}}^{M} P_j = \frac{1}{2}\sum_{j=1}^{M} P_j.$$

Figura 5.15: Definición de MDF [34]

- Mean frecuency (MNF) [34]:

$$\mathrm{MNF} = \sum_{j=1}^{M} f_j P_j \Big/ \sum_{j=1}^{M} P_j,$$

Figura 5.16: Definición de MNF [34]

5.5. Estimación de fatiga

Para estimar la fatiga, como ya se ha comentado en capítulos anteriores, el proyecto se ha basado en que una vez que se registra una velocidad que decae un 20 por ciento, con respecto a la velocidad de la primera repetición y esto se cumple en dos repeticiones seguidas, se considera que el atleta ha entrado en fatiga y la repetición en cuestión analizada y todas las siguientes se etiquetarán como fatiga.

5.6. Selección de características: RFE

Para dotar al clasificador de una mayor simpleza y un mejor rendimiento tanto a la hora de clasificar como en tiempo de ejecución, se ha utilizado un algoritmo para estimar que características de todas las estudiadas anteriormente son las más representativas de la fatiga. Se ha hecho uso del algoritmo Recursive Feature Elimination (RFE) [35]. Este algoritmo parte del dataset completo y en cada iteración elimina la característica menos representativa, hasta que se alcanza el número de características deseado. Con el uso de este algoritmo se han obtenido tres tipos de dataset con diferente número de variables para realizar el estudio. En este caso fueron estudiados con 1 característica, con 2 características y con 5 características. Por lo que finalmente obtendremos un gran número de resultados para realizar las conclusiones oportunas. El esquema final de todos los dataset que serán analizados es el siguiente (Ver 5.26).

5.7. Clasificación binaria de señales

Una vez realizada toda la fase de extracción y selección de características, debemos de realizar el diseño de los clasificadores. Para nuestro problema de clasificación binaria, donde 1 es identificador de fatiga y 0 es identificador de no fatiga, hemos estudiado el rendimiento de tres clasificadores. Se trata de 3 conjuntos de algoritmos de aprendizaje supervisado donde es necesario un conjunto de datos para el entrenamiento del modelo y otro conjunto de datos no conocido por el modelo, para estudiar su rendimiento a la hora de clasificar. A continuación se explica el funcionamiento y diseño de cada uno de ellos.

5.7.1. Support vector machines (SVMs): máquinas de vector de soporte

Su clasificación reside en que se construye un modelo que es capaz de predecir a que subgrupo de los dos, pertenece un nuevo punto del cual no conocemos su categoría. Para ello lo que se realiza es la obtención de un hiperplano que divida el conjunto de datos iniciales en dos grupos bien diferenciados. Los vectores de soporte son los encargados de definir el espacio de separación entre el hiperplano y las dos clases en cuestión (1,0). Cuanto mayor sea el margen entre los vectores de soporte de cada clase y el hiperplano, mejor será el modelo, ya que tendrá más claro donde clasificar datos futuros. Para el diseño del modelo se ha utilizado el kernel tipo sigmoide.

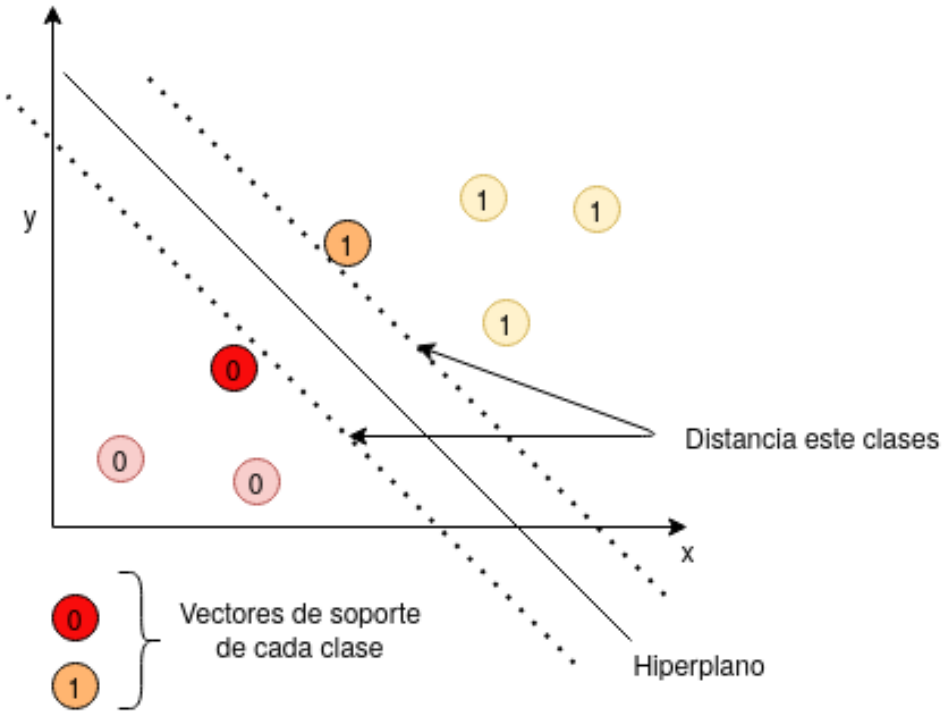

Figura 5.17: Esquema sobre funcionamiento del clasificador SVM

5.7.2. Decision trees (Trees): árboles de decisión

Los árboles de decisión son unos de los métodos más populares en machine learning debido a su gran eficiencia y sencillez. Se trata de un algoritmo de aprendizaje supervisado. Se puede entender como un árbol general formado por subárboles, donde cada nodo raíz de cada subárbol es una condición y las hojas de dichos subárboles, las etiquetas. El algoritmo selecciona las características del dataset más influyentes y va dividiendo el dataset completo en subconjuntos de datos, por lo que, dichas características formarán parte de los nodos de decisión, hasta que todos los nodos del árbol sean nodos hoja.

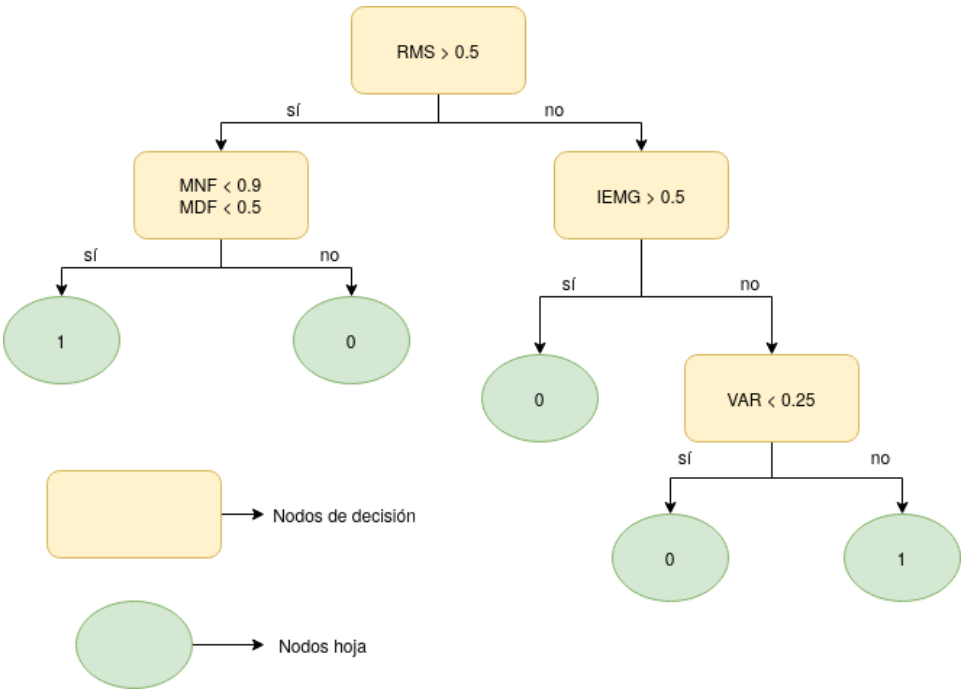

Figura 5.18: Esquema sobre funcionamiento del árbol de decisión

5.7.3. K nearest neighbors (KNN): K vecinos más cercanos

Este algoritmo de clasificación opera de una forma muy sencilla. Se basa en estimar la categoría de un dato nuevo, en base a la categoría de sus K vecinos más cercanos. De estos K vecinos más cercanos sí conoce su categoría debido a que forman parte del conjunto de datos de entrenamiento. Debido a esto el valor de K que sea utilizado influirá mucho en el rendimiento de nuestro clasificador. El valor de K que se ha utilizado para este proyecto es de 6 vecinos.

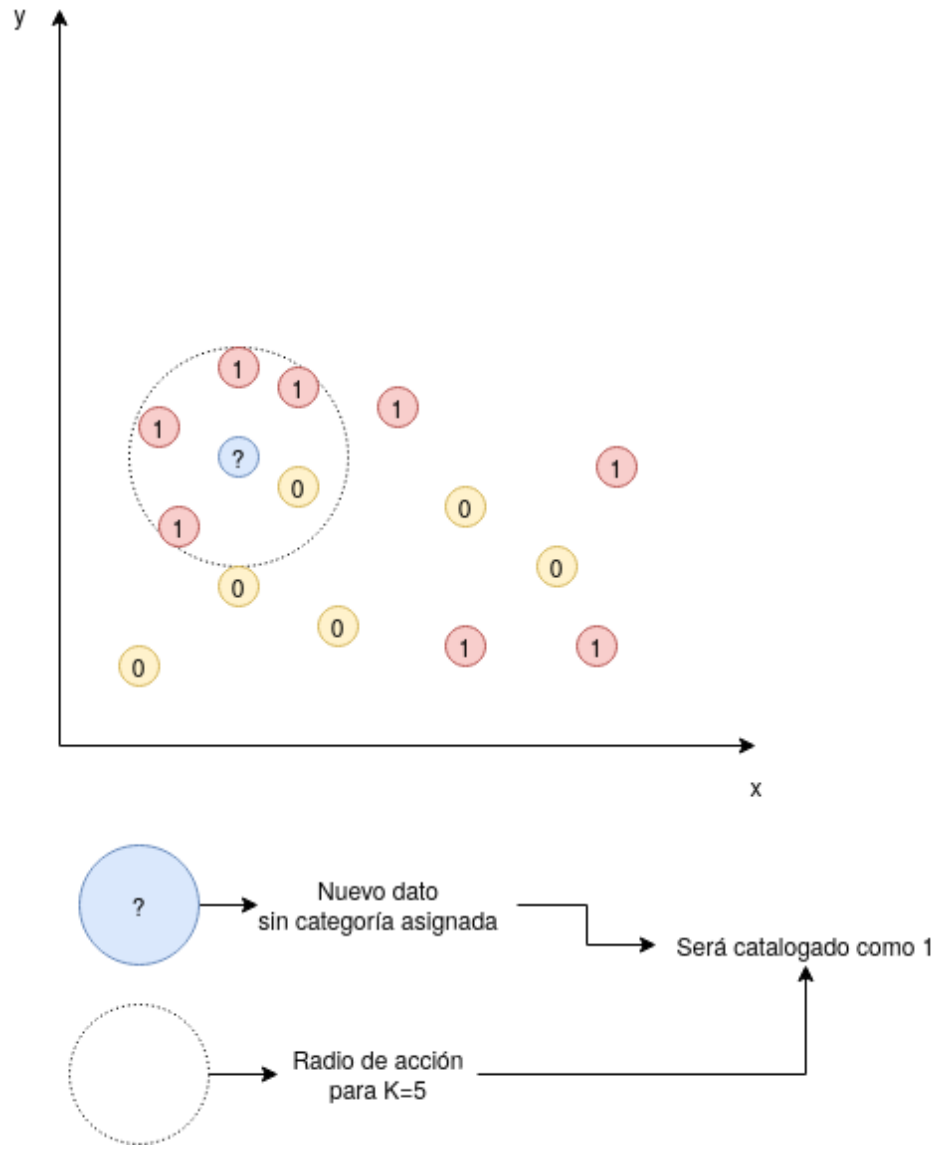

Figura 5.19: Esquema sobre funcionamiento del KNN para K = 5

5.8. Estudio del rendimiento obtenido: validación cruzada

Para validar el rendimiento de los tres clasificadores expuestos anteriormente se ha hecho uso de la validación cruzada. Con esto, sobre un mismo archivo de entrenamiento-testeo obtenemos diferentes conjuntos de datos para evaluar y testear el clasificador. En cada iteración el modelo se entrena con una parte diferente del conjunto de datos del archivo entrenamiento-testeo

y con la parte restante se realiza el testeo. (Ver 5.20).

Se obtendrá una serie de precisiones, una por cada iteración de la validación cruzada, y podremos obtener una precisión media del clasificador y una desviación estándar, que nos dará una idea de cuanto de bueno es nuestro clasificador. Para el diseño de nuestra validación cruzada se ha utilizado un valor de k = 15.

Figura 5.20: Validación cruzada para k = 5

5.9. Procesamiento de datos: diferentes modelos de estudio

El electromiógrafo utilizado para la medición de la actividad muscular a la hora de realizar las sentadillas consta de 4 canales diferentes. (Ver Imagen 5.21).

- Canal 1: registra la actividad muscular del vasto medial de la pierna izquierda.
- Canal 2: registra la actividad muscular del recto femoral de la pierna izquierda.

- Canal 3: registra la actividad muscular del vasto medial de la pierna derecha.
- Canal 4: registra la actividad muscular del recto femoral de la pierna derecha.

Debido a la gran ventaja que nos aporta la herramienta utilizada se ha optado por realizar diferentes tipos de archivos de entrenamiento-testeo y ver cual de ellos sería el más indicado a la hora de resolver nuestro problema. Cada uno de los archivos de entrenamiento-testeo que se van a describir a continuación están formados mediante la agrupación de los diferentes canales. Con esto se pretende simular diferentes situaciones y ver cual de ellas sería la más indicada para resolver el proyecto.

A continuación se muestra un esquema resumen sobre los archivos EMG utilizados y sus características extraídas. Con estos datos se formarán los archivos de entrenamiento-testeo.

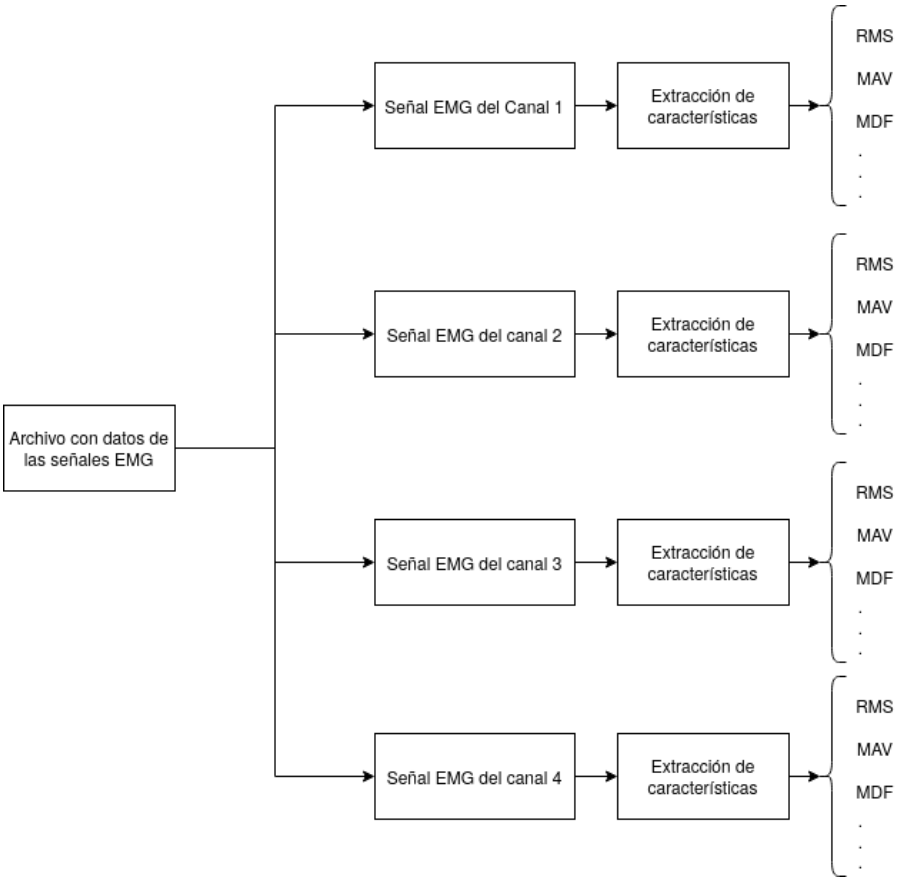

Figura 5.21: Esquema de archivo EMG bruto: 1 serie de 12 repeticiones de sentadillas

5.9.1. Canal 1, canal 2, canal 3, canal 4

La primera idea llevada a cabo es implementar un clasificador para cada canal. Con esto lo que se obtendría es una precisión muscular muy alta a la hora de detectar la fatiga muscular. Es decir, a la hora de realizar una serie de sentadillas podríamos averiguar que músculo de los 4 registrados es el que entra en fatiga. Para ello se han diseñado 4 tipos de archivos de entrenamiento-testeo. Los 4 tipos de archivos tienen la misma organización, lo único que cambia entre ellos es el valor que contienen. Por ejemplo, el archivo del canal 1, solo contiene los valores de las características extraídas de las señales EMG del vasto medial izquierdo. Para ello cada columna del dataset es identificada por cada una de las 15 características estudiadas, siendo la última columna, la representación de la fatiga asociada a cada repetición. Con respecto a las filas, las 12 primeras filas corresponden a las 12 repeticiones del primer archivo EMG analizado, las 12 siguientes a las 12 repeticiones del segundo archivo, etc.

Esto se puede aplicar también a los canales 2, 3 y 4. El esquema para el archivo de entrenamiento-testeo del canal N siendo N=1, 2, 3, 4 y de cada archivo K siendo K=1, 2, ..., 19 (número total de archivos analizados) es:

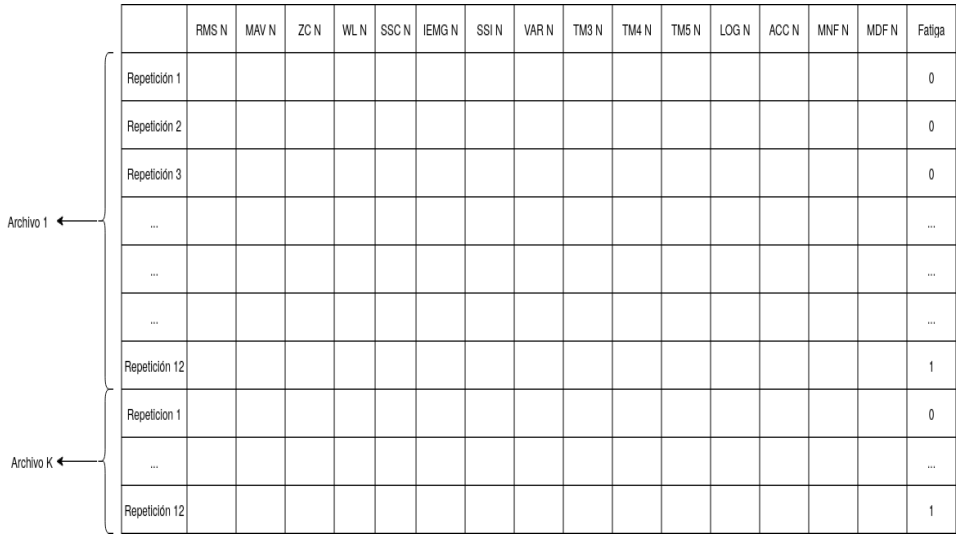

Figura 5.22: Esquema de archivo de entrenamiento-testeo para cada canal 1, 2, 3, 4

5.9.2. Canal 1,2 y canal 3,4

Debido a que la sentadilla es un ejercicio multiarticular e implica a las dos extremidades inferiores, la segunda idea consta de diseñar un clasificador para cada extremidad. Con esto tendremos un nivel de especificidad a nivel de extremidad y podremos detectar que extremidad inferior se fatiga antes, si la izquierda o la derecha. Para ello en esta ocasión tenemos dos archivos de entrenamiento-testeo, uno de cada extremidad. Lo que se ha llevado a cabo es la agrupación de los datos recolectados por el canal 1 y canal 2 (extremidad izquierda) y por el canal 3 y canal 4 (extremidad derecha). A continuación se muestra una breve explicación de cada archivo, así como su correspondiente esquema.

Canal 1,2: extremidad izquierda

Como se ha comentado anteriormente, este archivo de entrenamiento-testeo consta de los datos agrupados del canal 1 y del canal 2. En este caso el número de filas aumenta, ya que tratamos las repeticiones del canal 1 y las repeticiones del canal 2. Su esquema es el siguiente (Ver 5.23).

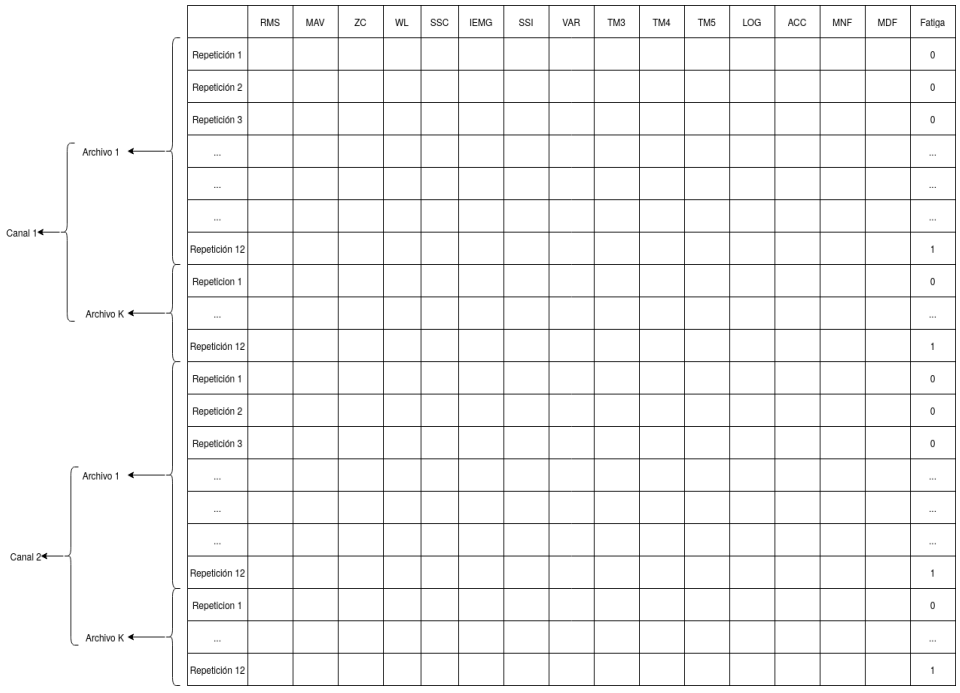

Figura 5.23: Esquema de archivo de entrenamiento-testeo para el canal 1,2

Canal 3,4: extremidad derecha

El archivo de entrenamiento-testeo del clasificador de la extremidad derecha, es similar al de la extremidad izquierda. La única diferencia es que en este caso se agrupan los datos del canal 3 y del canal 4. Su esquema es el siguiente (Ver 5.24).

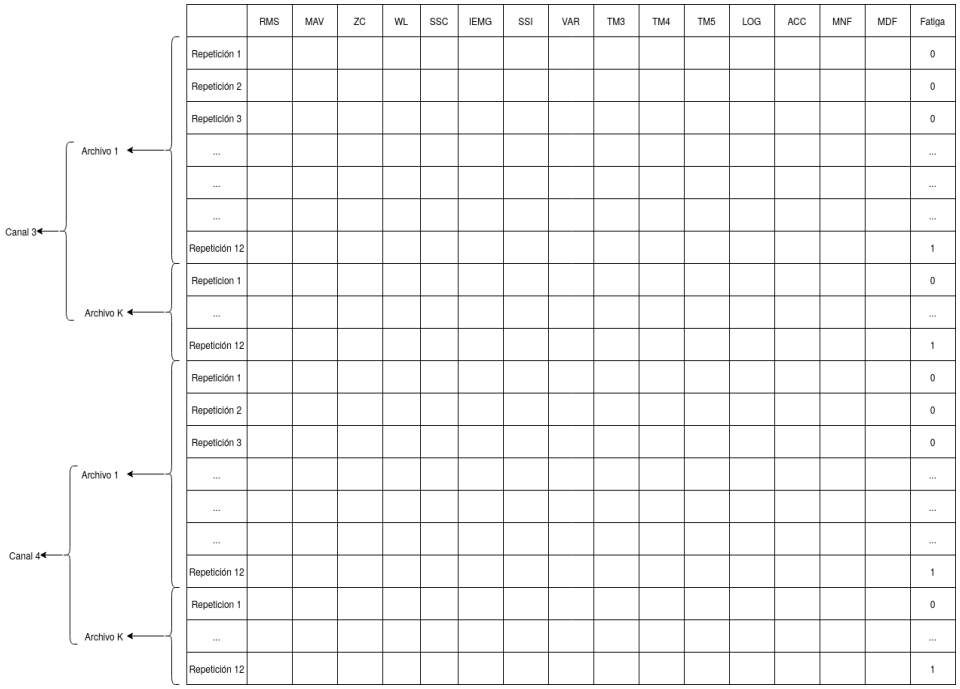

Figura 5.24: Esquema de archivo de entrenamiento-testeo para el canal 3,4

5.9.3. Canal total

Por último, se ha pensado en el diseño de un clasificador que ha sido denominado total, donde se hallará la fatiga global del individuo en cuestión. Para ello se agruparán los datos de los 4 canales utilizados en un mismo archivo de entrenamiento-testeo. En este caso el número de filas aumenta, debido a que tenemos en conjunto la información de todos los canales. El esquema de dicho archivo es el siguiente (Ver 5.25).

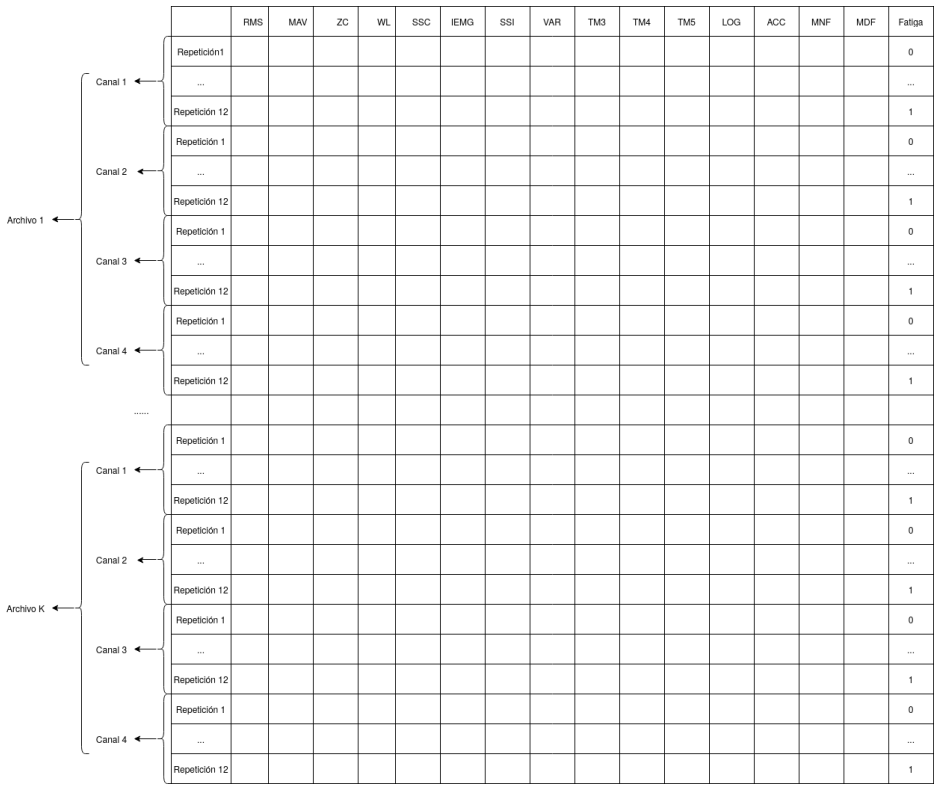

Figura 5.25: Esquema de archivo de entrenamiento-testeo para el canal total

5.10. Esquema final del estudio

Para finalizar la fase de metodología se pretende mostrar un esquema final sobre los diferentes tipos de clasificadores finales. El objetivo es analizar cuál sería el número óptimo de características (N=1,2,5) y el tipo de clasificador (SVM, Tree, KNN) para cada uno de los canales que son estudiados (canal 1, canal 2, canal 3, canal 4, canal (1,2), canal (3,4), canal total). Ver sección 5.9 para mayor detalle. (Ver esquema 5.26).

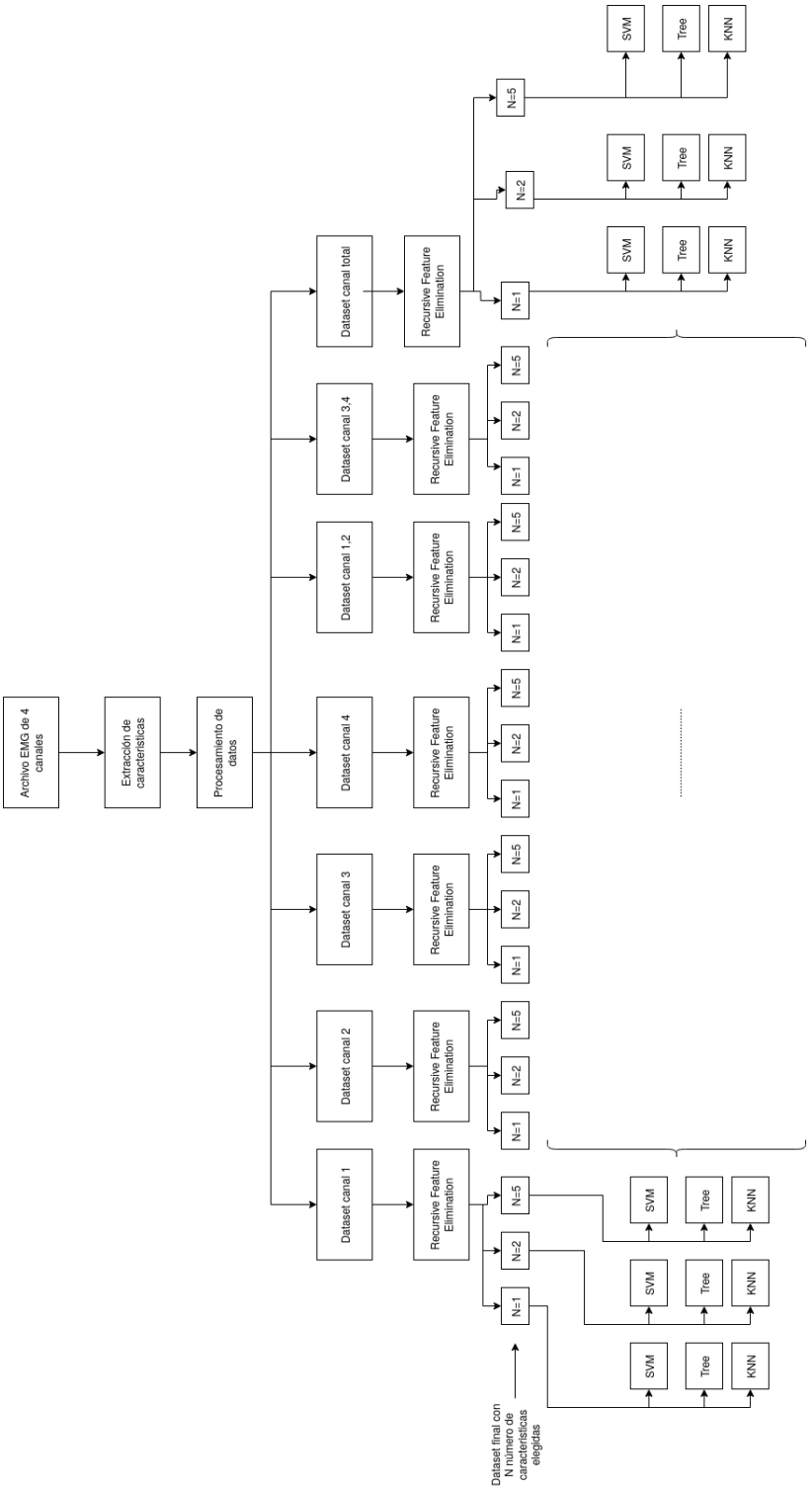

Figura 5.26: Esquema general del sistema

Capítulo 6

Implementación

Este capítulo está dedicado a la implementación realizada del proyecto. Tiene una estructura muy similar a la del capítulo 5, solo que en dicho capítulo se habla sobre el diseño y la metodología del proyecto y este está dedicado a su implementación en Python 3 [9]. Para su desarrollo se han utilizado los notebooks de Jupyter [4]. Jupyter es una aplicación web de código abierto, que permite crear documentos donde se incluyen ecuaciones, texto, código, gráficas, etc. Además es compatible con una gran cantidad de bibliotecas de machine learning y procesamiento de datos.

Las bibliotecas de ciencia de datos utilizadas para el desarrollo de este proyecto han sido:

- pandas: utilizada para el tratamiento y el análisis de los datos tratados. De gran utilidad con sus dataFrames para organizar los datos y exportarlos e importarlos en sus correspondientes .csv [8].

```
1 import pandas as pd
```

- numpy: librería que añade funcionalidad a Python con un gran aporte sobre matrices y vectores. Es complementaria a pandas y de gran utilidad en conjunto [7].

```
1 import numpy as np
```

Para el procesamiento de señales se ha utilizado:

- scipy: librería especializada en matemáticas, ciencia e ingeniería. Aporta diversos módulos especializados en diversos temas. El más utilizado

en este proyecto fue signal, utilizada para el procesamiento de señales (FFT, PSD, welch) [12, 13].

```
import scipy as sp
from scipy import signal
from scipy.fftpack import rfft,rfftfreq
```

Para dibujar las diferentes gráficas obtenidas se ha hecho uso de:

- matplotlib: aporta la funcionalidad para la visualización de los datos procesados con Python [6].

```
import matplotlib.pyplot as plt
```

Para el procesamiento de archivos:

- glob: aporta la funcionalidad para abrir la base de datos utilizada y examinar cada uno de los archivos que la forman.

```
import glob
```

Para la parte de machine learning del proyecto se ha utilizado la biblioteca de pyhton:

- scikit-learn: aporta una gran funcionalidad con una gran cantidad de algoritmos relacionados con el machine learning [11].

El resto de módulos relacionados con scikit-learn, como por ejemplo, para los clasificadores, selección de características y rendimiento, serán explicados en sus correspondientes secciones. Esto fue una breve introducción con las bibliotecas generales.

6.1. Estimación del tamaño de ventana

El proceso dedicado a la estimación del tamaño de ventana se ha realizado con el uso de la biblioteca de biosignalsnotebooks [1].

```
import biosignalsnotebooks as bsnb
```

Para dicho proceso, con el uso de la función:

```
activacion1=bsnb.detect_emg_activations(vastoMedialIzquierdo,fs,
    smooth_level=smooth,threshold_level=threshold,time_units=False,
    volts=False, resolution=None,plot_result=plot)
```

se obtiene un array con todos los intervalos donde se ha detectado actividad muscular en el archivo de EMG bruto. Con esto ya tenemos el tamaño de ventana de cada repetición y podemos continuar con la extracción de características con dicho tamaño de ventana. Cabe destacar que cada repetición de sentadilla puede tener un tamaño de ventana diferente.

Por último antes de seguir con la fase de extracción de características, habría que señalar el proceso de selección de los archivos brutos de EMG mejor definidos. Esto es debido a que había ciertos archivos donde claramente no había 12 repeticiones registradas o donde había demasiado ruido y el algoritmo utilizado no detectaba claramente las 12 repeticiones, por lo que opté, por implementar un código donde se examinan todos los archivos definidos con diferentes valores de threshold y de smooth, para así hallar cual sería el par de valores más adecuado a cada archivo de EMG y utilizar dichos archivos junto con su par de valores "threshold, smooth"para la extracción de características. A continuación se muestra la función que realiza dicho procedimiento:

```
def hallarArchivosEMG():

    fs=1024
    dataset=glob.glob("../Estudio tDCS/*.csv")
    dataset.sort()

    archivosFinales=[]
    archivos=[]
    valoresSM=[22,21,20,19,18,17,16,15,14]
    valoresTH=[60,57,55,52,50,47,45,42,40,
    37,35,32,30,27,25,22,20,19,18,17,16,15,14,13,12,11,10]

    for sm in valoresSM:

        for th in valoresTH:

            for archivo in dataset:

                if archivo not in archivos:

                    df = pd.read_csv(archivo,delimiter=";",decimal="."
                        )
                    df=df.fillna(0)

                    #Pierna izquieda
                    vastoMedialIzquierdo= df['emg_muscle_1_( V )'].
                        to_numpy()
                    rectoFemoralIzquierdo=df['emg_muscle_2_( V )'].
                        to_numpy()

                    #Pierna derecha
```

```
29                      vastoMedialDerecho=df['emg_muscle_3_( V )'].
                            to_numpy()
30                      rectoFemoralDerecho=df['emg_muscle_4_( V )'].
                            to_numpy()
31
32
33                      #Atributos para detectar EMG
34                      plot=False
35                      smooth=sm
36                      threshold=th
37
38                      activacion1=bsnb.detect_emg_activations(
                            vastoMedialIzquierdo,fs,smooth_level=smooth,
                            threshold_level=threshold, time_units=False,
                            volts=False, resolution=None,plot_result=plot)
39
40                      if (len(activacion1[0]) != 12):
41                          continue
42
43                      activacion2=bsnb.detect_emg_activations(
                            rectoFemoralIzquierdo,fs,smooth_level=smooth,
                            threshold_level=threshold,
44  time_units=False,volts=False, resolution=None,plot_result=plot)
45
46                      if (len(activacion2[0]) != 12):
47                          continue
48
49                      activacion3=bsnb.detect_emg_activations(
                            vastoMedialDerecho,fs,smooth_level=smooth,
                            threshold_level=threshold,
50  time_units=False,volts=False, resolution=None,plot_result=plot)
51
52                      if (len(activacion3[0]) != 12):
53                          continue
54                      activacion4=bsnb.detect_emg_activations(
                            rectoFemoralDerecho,fs,smooth_level=smooth,
                            threshold_level=threshold,
55  time_units=False,volts=False, resolution=None,plot_result=plot)
56                      if (len(activacion4[0]) != 12):
57                          continue
58
59                      datosImportantes=[]
60                      datosImportantes.append(archivo)
61                      datosImportantes.append(threshold)
62                      datosImportantes.append(smooth)
63                      archivos.append(archivo)
64
65                      archivosFinales.append(datosImportantes)
66
67      return archivosFinales
```

6.2. Extracción de características

Después de analizar todos los archivos disponibles y realizar su correspondiente estimación de los valores de ventana, podemos proceder a comentar la siguiente fase, la fase de extracción de características. Para la implementación de cada característica me he basado en la función que la

define y la he implementado sobre Python. Se aconseja visitar el anexo para ver el código implementado asociado a cada característica. (Ver A.4.1).

6.3. Estimación de fatiga

Para la estimación de la fatiga comentada, se obtienen las etiquetas de fatiga experimentada, representadas por 1, y fatiga no experimentada, representadas por 0, con el siguiente código. Su funcionamiento es recibir las velocidades de cada una de de las repeticiones registradas y hallar el valor de la etiqueta de la fatiga, basándose en un umbral de caída de velocidad, el cuál viene dado por una caída del 20 por ciento en la velocidad con respecto a la primera repetición.

```
def calculoDeFatiga(velocidad):
    fatiga=[]

    umbral=0.8*velocidad[0]
    c=0
    for i in range(0,10):
        c=c+1
        if ((velocidad[i] <= umbral) and (velocidad[i+1] <= umbral) ):
            for j in range(i,12):
                fatiga.append(1)

            return np.asarray(fatiga)

        else:
            fatiga.append(0)


    if (len(fatiga) != 12):
        if ((velocidad[c] <= umbral) and (velocidad[c+1] <= umbral)):
            for i in range(len(fatiga),12):
                fatiga.append(1)
        else:
            for i in range(len(fatiga),12):
                fatiga.append(0)

    return np.asarray(fatiga)
```

6.4. Selección de características: RFE

Como se ha explicado en la fase de diseño, el algoritmo utilizado para la selección de las características más representativas de la fatiga y así obtener unos modelos más sencillos ha sido el Recursive Feature Elimination (RFE). Para su desarrollo se ha hecho uso de la biblioteca scikit-learn de Python con su módulo de selección de características [10].

A cada uno de los archivos de entrenamiento-testeo de cada canal se le

aplica dicho algoritmo, y se extraen 3 archivos de entrenamiento-testeo para cada canal, cada uno con un número diferente de características.

6.4.1. Canal 1, canal 2, canal 3, canal 4

Aplicación del RFE a los archivos de entrenamiento-testeo de los canales 1, 2, 3 y 4 con valores de N = 1, 2, 5, donde N es el número de características finales del modelo.

```
caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','TM3
    ','TM4','TM5','LOG','ACC','MNF','MDF']
#[0] -> N=1 canal 1
#[1] -> N=1 canal 2
#[2] -> N=1 canal 3
#[3] -> N=1 canal 4

#[4] -> N=2 canal 1
#[5] -> N=2 canal 2
#[6] -> N=2 canal 3
#[7] -> N=2 canal 4

listaDeAtributos=[]
numeroCaracteristicas=[1,2,5]


for i in numeroCaracteristicas:
    canal=1
    #Recorro la lista con los valores de los 4 canales
    for dato in datos1234:

        x=dato.to_numpy()
        y=valores1234['fatiga'].to_numpy()


        estimator = SVC(kernel="linear")

        print("N = " + str(i) + "
            ------------------------------------------------")
        print("estimacion canal : " + str(canal))

        selector= RFECV(estimator,step=1,cv=15,min_features_to_select=
            i)
        selector=selector.fit(x,y)

        ni=np.array(selector.support_)
        pi=pd.DataFrame(data=ni,columns=['Validez'])

        pi['Caracteristica']=caracteristicas
        pi['canal']=canal
        pi = pi.drop(pi[pi.Validez == False].index)

        print(pi)
        print("\n\n")

        listaDeAtributos.append(pi['Caracteristica'])
        #listaDeAtributos.append(pi)

        canal=canal+1
```

```
47
48 #obtengo el dataset para cada clasificador
49 datosFinales1234=[]
50
51 i=0
52 for caracteristicas in listaDeAtributos:
53     if (i > 3):
54         i=0
55
56     p=pd.DataFrame()
57     for caracteristica in caracteristicas:
58         p[caracteristica + str(i+1)]=valores1234[caracteristica + str(
               i+1)]
59     i=i+1
60
61     p=(p-p.min())/(p.max()-p.min())
62     datosFinales1234.append(p)
```

6.4.2. Canal 1,2 y canal 3,4

Canal 1,2

Aplicación del RFE al archivo de entrenamiento-testeo del canal 1, 2 con valores de N = 1, 2, 5, donde N es el número de características finales del modelo.

```
1  caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','TM3
       ','TM4','TM5','LOG','ACC','MNF','MDF']
2
3  numeroCaracteristicas=[1,2,5]
4
5  RFEcanal12=[]
6
7
8
9  for i in numeroCaracteristicas:
10
11
12     x = valores12.drop(['fatiga'],axis=1).to_numpy()
13     y = valores12['fatiga'].to_numpy()
14
15     estimator= SVC(kernel='linear')
16
17     selector= RFECV(estimator,step=1,cv=15,min_features_to_select=i)
18     selector=selector.fit(x,y)
19     ni=np.array(selector.support_)
20
21     pi=pd.DataFrame(data=ni,columns=['Validez'])
22
23     pi['Caracteristica']=caracteristicas
24
25     pi = pi.drop(pi[pi.Validez == False].index)
26
27
28     print("N= " + str(i))
29     print(pi)
30     print("\n\n")
```

```
31
32
33      datosFinalesTotales=pd.DataFrame()
34      for caracteristica in pi['Caracteristica']:
35          datosFinalesTotales[caracteristica]=valores12[caracteristica]
36
37
38      RFEcanal12.append(datosFinalesTotales)
39
40
41  RFEcanal12
```

Canal 3,4

Aplicación del RFE al archivo de entrenamiento-testeo del canal 3, 4 con valores de N = 1, 2, 5, donde N es el número de características finales del modelo.

```
1   caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','TM3
        ','TM4','TM5','LOG','ACC','MNF','MDF']
2
3   numeroCaracteristicas=[1,2,5]
4
5   RFEcanal34=[]
6
7   for i in numeroCaracteristicas:
8
9       x = valores34.drop(['fatiga'],axis=1).to_numpy()
10      y = valores34['fatiga'].to_numpy()
11
12      estimator= SVC(kernel='linear')
13
14      selector= RFECV(estimator,step=1,cv=15,min_features_to_select=i)
15      selector=selector.fit(x,y)
16      ni=np.array(selector.support_)
17
18      pi=pd.DataFrame(data=ni,columns=['Validez'])
19
20      pi['Caracteristica']=caracteristicas
21
22      pi = pi.drop(pi[pi.Validez == False].index)
23
24
25      print("N= " + str(i))
26      print(pi)
27      print("\n\n")
28
29
30      datosFinalesTotales=pd.DataFrame()
31      for caracteristica in pi['Caracteristica']:
32          datosFinalesTotales[caracteristica]=valores34[caracteristica]
33
34
35      RFEcanal34.append(datosFinalesTotales)
36
37
38  RFEcanal34
```

6.4.3. Canal total

Aplicación del RFE al archivo de entrenamiento-testeo del canal total con valores de N = 1, 2, 5, donde N es el número de características finales del modelo.

```
caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','TM3
    ','TM4','TM5','LOG','ACC','MNF','MDF']
numeroCaracteristicas=[1,2,5]

RFEcompleto=[]

for i in numeroCaracteristicas:



    x = datosTotales.to_numpy()
    y = valoresTotales['fatiga'].to_numpy()

    estimator= SVC(kernel='linear')

    selector= RFECV(estimator,step=1,cv=15,min_features_to_select=i)
    selector=selector.fit(x,y)
    ni=np.array(selector.support_)

    pi=pd.DataFrame(data=ni,columns=['Validez'])

    pi['Caracteristica']=caracteristicas

    pi = pi.drop(pi[pi.Validez == False].index)


    print("N= " + str(i))
    print(pi)
    print("\n\n")


    datosFinalesTotales=pd.DataFrame()
    for caracteristica in pi['Caracteristica']:
        datosFinalesTotales[caracteristica]=datosTotales[
            caracteristica]


    RFEcompleto.append(datosFinalesTotales)
```

6.5. Clasificación binaria de señales

Como se explicó en el capítulo de metodología, se analizará el rendimiento de tres tipos de clasificadores en diferentes situaciones (Ver 5.9).

6.5.1. Support vector machines (SVMs): máquinas de vector de soporte

Para su desarrollo se ha hecho uso del módulo [14]:

```
from sklearn.svm import SVC
```

Canal 1, canal 2, canal 3, canal 4

Código encargado de recorrer la lista de python que contiene los archivos de entrenamiento-testeo del canal 1, canal 2, canal 3 y del canal 4 y proceder a realizar la clasificación con validación cruzada de cada uno de los archivos, para analizar el rendimiento del SVM en dichos canales.

```
canal=1
for dato in datosFinales1234:

    if (canal>4):
        canal=1

    print("N = " + str(len(dato.columns)) + "
        ------------------------->")
    print("Valores de canal " + str(canal) + " ------>")

    print(dato)

    x=dato.to_numpy()
    y=valores1234['fatiga'].to_numpy()

    X_train, X_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.20)

    #Creo clasificador
    clasificadorSVC = SVC(kernel='sigmoid')


    #Entreno
    clasificadorSVC.fit(X_train, y_train)

    #Testeo
    y_pred = clasificadorSVC.predict(X_test)


    #Resultados
    print(confusion_matrix(y_test,y_pred))
    print(classification_report(y_test,y_pred))

```

```
32
33      #Cross validation k-fold (precision total)
34      resultados = cross_validate(clasificadorSVC, x, y, cv=15)
35
36
37      precision=resultados['test_score'].mean()
38      desviacion=resultados['test_score'].std()
39
40      print("Precision : " + str(precision) + " +/- " + str(desviacion))
41
42
43      indice='N='+str(len(dato.columns))
44
45      graficaSVM[indice]['canal '+str(canal)]=precision
46
47
48      if (canal ==1):
49          graficaCanal1[indice]['SVM']=precision
50          graficaCanal1Desviacion[indice]['SVM']=desviacion
51
52      elif (canal == 2):
53          graficaCanal2[indice]['SVM']=precision
54          graficaCanal2Desviacion[indice]['SVM']=desviacion
55
56      elif (canal ==3):
57          graficaCanal3[indice]['SVM']=precision
58          graficaCanal3Desviacion[indice]['SVM']=desviacion
59
60      elif (canal ==4):
61          graficaCanal4[indice]['SVM']=precision
62          graficaCanal4Desviacion[indice]['SVM']=desviacion
63
64      canal=canal+1
65
66      print("\n")
```

Canal 1,2 y canal 3,4

Códigos encargados de clasificar los diferentes archivos de entrenamiento-testeo para el canal 1,2 y el canal 3,4. Se realiza la validación cruzada de cada canal para analizar el rendimiento del SVM en dichos canales.

Canal 1,2

```
1       canal='canal 1,2'
2
3   for dato in RFEcanal12:
4
5       print("N = " + str(len(dato.columns)) + "
            -------------------------->")
6       print(dato)
7
8
9       x=dato.to_numpy()
10      y=valores12['fatiga'].to_numpy()
```

```
11
12
13      #y=np.random.randint(2,size=1056)
14
15      X_train, X_test, y_train, y_test = train_test_split(x, y,
            test_size = 0.20)
16
17      #Creo clasificador
18      clasificadorSVC = SVC(kernel='sigmoid')
19
20
21      #Entreno
22      clasificadorSVC.fit(X_train, y_train)
23
24      #Testeo
25      y_pred = clasificadorSVC.predict(X_test)
26
27
28      #Resultados
29      print(confusion_matrix(y_test,y_pred))
30      print(classification_report(y_test,y_pred))
31
32
33
34      #Cross validation k-fold (precision total)
35      resultados = cross_validate(clasificadorSVC, x, y, cv=15)
36
37      precision=resultados['test_score'].mean()
38      desviacion=resultados['test_score'].std()
39
40
41      print("Precision : " + str(precision) + " +/- " + str(desviacion))
42
43      indice='N='+str(len(dato.columns))
44
45      graficaSVM[indice][canal]=precision
46      graficaCanal12[indice]['SVM']=precision
47      graficaCanal12Desviacion[indice]['SVM']=desviacion
```

Canal 3,4

```
1       canal='canal 3,4'
2
3   for dato in RFEcanal34:
4
5       print("N = " + str(len(dato.columns)) + "
            -------------------------->")
6       print(dato)
7
8
9       x=dato.to_numpy()
10      y=valores34['fatiga'].to_numpy()
11
12
13      #y=np.random.randint(2,size=1056)
14
15      X_train, X_test, y_train, y_test = train_test_split(x, y,
            test_size = 0.20)
```

```
16
17      #Creo clasificador
18      clasificadorSVC = SVC(kernel='sigmoid')
19
20
21      #Entreno
22      clasificadorSVC.fit(X_train, y_train)
23
24      #Testeo
25      y_pred = clasificadorSVC.predict(X_test)
26
27
28      #Resultados
29      print(confusion_matrix(y_test,y_pred))
30      print(classification_report(y_test,y_pred))
31
32
33
34      #Cross validation k-fold (precision total)
35      resultados = cross_validate(clasificadorSVC, x, y, cv=15)
36
37      precision=resultados['test_score'].mean()
38      desviacion=resultados['test_score'].std()
39      print("Precision : " + str(precision) + " +/- " + str(desviacion))
40
41
42      indice='N='+str(len(dato.columns))
43
44      graficaSVM[indice][canal]=precision
45      graficaCanal34[indice]['SVM']=precision
46      graficaCanal34Desviacion[indice]['SVM']=desviacion
```

Canal total

Código encargado de clasificar por medio de la validación cruzada el archivo de entrenamiento-testeo correspondiente al canal total para analizar el rendimiento del SVM en dicho canal.

```
1  canal='canal Total'
2
3  for dato in RFEcompleto:
4
5      print("N = " + str(len(dato.columns)) + "
           ------------------------->")
6      print(dato)
7
8
9      x=dato.to_numpy()
10     y=valoresTotales['fatiga'].to_numpy()
11
12
13     #y=np.random.randint(2,size=1056)
14
15     X_train, X_test, y_train, y_test = train_test_split(x, y,
           test_size = 0.20)
16
17     #Creo clasificador
```

```
18      clasificadorSVC = SVC(kernel='sigmoid')
19
20
21      #Entreno
22      clasificadorSVC.fit(X_train, y_train)
23
24      #Testeo
25      y_pred = clasificadorSVC.predict(X_test)
26
27
28      #Resultados
29      print(confusion_matrix(y_test,y_pred))
30      print(classification_report(y_test,y_pred))
31
32
33
34      #Cross validation k-fold (precision total)
35      resultados = cross_validate(clasificadorSVC, x, y, cv=15)
36
37      precision=resultados['test_score'].mean()
38      desviacion=resultados['test_score'].std()
39
40
41      print("Precision : " + str(precision) + " +/- " + str(desviacion))
42
43      indice='N='+str(len(dato.columns))
44
45
46      graficaSVM[indice][canal]=precision
47      graficaCanalTotal[indice]['SVM']=precision
48      graficaCanalTotalDesviacion[indice]['SVM']=resultados['test_score
            '].std()
```

6.5.2. Decision trees (Trees): árboles de decisión

Para su desarrollo se ha hecho uso del módulo [15]:

```
1 from sklearn.tree import DecisionTreeClassifier
```

Canal 1, canal 2, canal 3, canal 4

Código encargado de recorrer la lista de python que contiene los archivos de entrenamiento-testeo del canal 1, canal 2, canal 3 y del canal 4 y proceder a realizar la clasificación con validación cruzada de cada uno de los archivos, para analizar el rendimiento del Tree en dichos canales.

```
1 canal=1
2 for dato in datosFinales1234:
3
4     if (canal>4):
5         canal=1
6
```

```
7       print("N = " + str(len(dato.columns)) + "
            ------------------------->")
8       print("Valores de canal " + str(canal) + " ------>")
9
10      print(dato)
11
12      x=dato.to_numpy()
13      y=valores1234['fatiga'].to_numpy()
14
15
16      X_train, X_test, y_train, y_test = train_test_split(x, y,
            test_size = 0.20)
17
18      #Creo clasificador
19      clasificadorTree = DecisionTreeClassifier(random_state=0)
20
21      #Entreno
22      clasificadorTree.fit(X_train, y_train)
23
24      #Testeo
25      y_pred = clasificadorTree.predict(X_test)
26
27
28      #Resultados
29      print(confusion_matrix(y_test,y_pred))
30      print(classification_report(y_test,y_pred))
31
32
33
34      #Cross validation k-fold (precision total)
35      resultados = cross_validate(clasificadorTree, x, y, cv=15)
36
37
38      precision=resultados['test_score'].mean()
39      desviacion=resultados['test_score'].std()
40
41      print("Precision : " + str(precision) + " +/- " + str(desviacion))
42
43
44      indice='N='+str(len(dato.columns))
45      graficaTree[indice]['canal '+str(canal)]=precision
46
47      if (canal ==1):
48          graficaCanal1[indice]['TREE']=precision
49          graficaCanal1Desviacion[indice]['TREE']=desviacion
50
51      elif (canal == 2):
52          graficaCanal2[indice]['TREE']=precision
53          graficaCanal2Desviacion[indice]['TREE']=desviacion
54
55      elif (canal ==3):
56          graficaCanal3[indice]['TREE']=precision
57          graficaCanal3Desviacion[indice]['TREE']=desviacion
58
59      elif (canal ==4):
60          graficaCanal4[indice]['TREE']=precision
61          graficaCanal4Desviacion[indice]['TREE']=desviacion
62
63      canal=canal+1
64
65      print("\n")
```

Canal 1,2 y canal 3,4

Códigos encargados de clasificar los diferentes archivos de entrenamiento-testeo para el canal 1,2 y el canal 3,4. Se realiza la validación cruzada de cada canal para analizar el rendimiento del Tree en dichos canales.

Canal 1,2

```
canal='canal 1,2'

for dato in RFEcanal12:

    print("N = " + str(len(dato.columns)) + "
        -------------------------->")
    print(dato)


    x=dato.to_numpy()
    y=valores12['fatiga'].to_numpy()


    #y=np.random.randint(2,size=1056)

    X_train, X_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.20)

    #Creo clasificador
    clasificadorTree = DecisionTreeClassifier(random_state=0)


    #Entreno
    clasificadorTree.fit(X_train, y_train)

    #Testeo
    y_pred = clasificadorTree.predict(X_test)


    #Resultados
    print(confusion_matrix(y_test,y_pred))
    print(classification_report(y_test,y_pred))



    #Cross validation k-fold (precision total)
    resultados = cross_validate(clasificadorTree, x, y, cv=15)

    precision=resultados['test_score'].mean()
    desviacion=resultados['test_score'].std()
    print("Precision : " + str(precision) + " +/- " + str(desviacion))


    indice='N='+str(len(dato.columns))

    graficaTree[indice][canal]=precision
    graficaCanal12[indice]['TREE']=precision
    graficaCanal12Desviacion[indice]['TREE']=desviacion
```

Canal 3,4

```
canal='canal 3,4'

for dato in RFEcanal34:

    print("N = " + str(len(dato.columns)) + "
        ------------------------->")
    print(dato)


    x=dato.to_numpy()
    y=valores34['fatiga'].to_numpy()


    #y=np.random.randint(2,size=1056)

    X_train, X_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.20)

    #Creo clasificador
    clasificadorTree = DecisionTreeClassifier(random_state=0)


    #Entreno
    clasificadorTree.fit(X_train, y_train)

    #Testeo
    y_pred = clasificadorTree.predict(X_test)


    #Resultados
    print(confusion_matrix(y_test,y_pred))
    print(classification_report(y_test,y_pred))



    #Cross validation k-fold (precision total)
    resultados = cross_validate(clasificadorTree, x, y, cv=15)

    precision=resultados['test_score'].mean()
    desviacion=resultados['test_score'].std()

    print("Precision : " + str(precision) + " +/- " + str(desviacion))

    indice='N='+str(len(dato.columns))

    graficaTree[indice][canal]=precision
    graficaCanal34[indice]['TREE']=precision
    graficaCanal34Desviacion[indice]['TREE']=desviacion
```

Canal total

Código encargado de clasificar por medio de la validación cruzada el archivo de entrenamiento-testeo correspondiente al canal total para analizar el rendimiento del Tree en dicho canal.

```
for dato in RFEcompleto:

    print("N = " + str(len(dato.columns)) + "
        -------------------------->")
    print(dato)

    x=dato.to_numpy()
    y=valoresTotales['fatiga'].to_numpy()

    X_train, X_test, y_train, y_test = train_test_split(x, y,
        test_size = 0.20)

    #Creo clasificador
    clasificadorTree = DecisionTreeClassifier(random_state=0)


    #Entreno
    clasificadorTree.fit(X_train,y_train)

    #Testeo
    y_pred = clasificadorTree.predict(X_test)


    #Resultados
    print(confusion_matrix(y_test,y_pred))
    print(classification_report(y_test,y_pred))



    #Cross validation k-fold (precision total)
    resultados = cross_validate(clasificadorTree, x, y, cv=15)

    precision=resultados['test_score'].mean()
    desviacion=resultados['test_score'].std()
    print("Precision : " + str(precision) + " +/- " + str(desviacion))


    indice='N='+str(len(dato.columns))

    graficaTree[indice]['canal Total']=precision
    graficaCanalTotal[indice]['TREE']=precision
    graficaCanalTotalDesviacion[indice]['TREE']=desviacion
```

6.5.3. K nearest neighbors (KNN): K vecinos más cercanos

Para su desarrollo se ha hecho uso del módulo [5]:

```
from sklearn.neighbors import KNeighborsClassifier
```

Canal 1, canal 2, canal 3, canal 4

Código encargado de recorrer la lista de python que contiene los archivos de entrenamiento-testeo del canal 1, canal 2, canal 3 y del canal 4 y proceder

a realizar la clasificación con validación cruzada de cada uno de los archivos, para analizar el rendimiento del KNN en dichos canales.

```
canal=1
for dato in datosFinales1234:
    if (canal > 4):
        canal=1

    print("N = " + str(len(dato.columns)) + "
        ------------------------->")
    print("Valores de canal " + str(canal) + " ------>")

    print(dato)

    x=dato.to_numpy()
    y=valores1234['fatiga'].to_numpy()


    X_train, X_test, y_train, y_test = train_test_split(x, y,
        test_size=0.20)

    #K=6
    KNN = KNeighborsClassifier(n_neighbors=6)

    KNN.fit(X_train, y_train)


    y_pred = KNN.predict(X_test)

    #Resultados
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))


    #Cross validation k-fold (precision total)
    resultados = cross_validate(KNN, x, y, cv=15)

    precision=resultados['test_score'].mean()
    desviacion=resultados['test_score'].std()

    print("Precision : " + str(precision) + " +/- " + str(desviacion))

    indice='N='+str(len(dato.columns))


    graficaKNN[indice]['canal '+str(canal)]=precision
    if (canal ==1):
        graficaCanal1[indice]['KNN']=precision
        graficaCanal1Desviacion[indice]['KNN']=desviacion

    elif (canal == 2):
        graficaCanal2[indice]['KNN']=precision
        graficaCanal2Desviacion[indice]['KNN']=desviacion

    elif (canal ==3):
        graficaCanal3[indice]['KNN']=precision
        graficaCanal3Desviacion[indice]['KNN']=desviacion

    elif (canal ==4):
        graficaCanal4[indice]['KNN']=precision
        graficaCanal4Desviacion[indice]['KNN']=desviacion
```

```
57
58      canal=canal+1
59
60      print("\n")
```

Canal 1,2 y canal 3,4

Códigos encargados de clasificar los diferentes archivos de entrenamiento-testeo para el canal 1,2 y el canal 3,4. Se realiza la validación cruzada de cada canal para analizar el rendimiento del KNN en dichos canales.

Canal 1,2

```
1  canal='canal 1,2'
2
3  for dato in RFEcanal12:
4
5      print("N = " + str(len(dato.columns)) + "
           ------------------------->")
6      print(dato)
7
8
9      x=dato.to_numpy()
10     y=valores12['fatiga'].to_numpy()
11
12
13     X_train, X_test, y_train, y_test = train_test_split(x, y,
           test_size=0.20)
14
15     #K=6
16     KNN = KNeighborsClassifier(n_neighbors=6)
17
18     KNN.fit(X_train, y_train)
19
20
21     y_pred = KNN.predict(X_test)
22
23     #Resultados
24     print(confusion_matrix(y_test, y_pred))
25     print(classification_report(y_test, y_pred))
26
27
28     #Cross validation k-fold (precision total)
29     resultados = cross_validate(KNN, x, y, cv=15)
30
31     precision=resultados['test_score'].mean()
32     desviacion=resultados['test_score'].std()
33
34     print("Precision : " + str(precision) + " +/- " + str(desviacion))
35
36
37     indice='N='+str(len(dato.columns))
38
39     graficaKNN[indice][canal]=precision
40     graficaCanal12[indice]['KNN']=precision
```

```
41      graficaCanal12Desviacion[indice]['KNN']=desviacion
```

Canal 3,4

```
1  canal='canal 3,4'
2
3  for dato in RFEcanal34:
4
5      print("N = " + str(len(dato.columns)) + "
           ------------------------->")
6      print(dato)
7
8
9      x=dato.to_numpy()
10     y=valores34['fatiga'].to_numpy()
11
12
13     X_train, X_test, y_train, y_test = train_test_split(x, y,
           test_size=0.20)
14
15     #K=6
16     KNN = KNeighborsClassifier(n_neighbors=6)
17
18     KNN.fit(X_train, y_train)
19
20
21     y_pred = KNN.predict(X_test)
22
23     #Resultados
24     print(confusion_matrix(y_test, y_pred))
25     print(classification_report(y_test, y_pred))
26
27
28     #Cross validation k-fold (precision total)
29     resultados = cross_validate(KNN, x, y, cv=15)
30     precision=resultados['test_score'].mean()
31     desviacion=resultados['test_score'].std()
32
33     print("Precision : " + str(precision) + " +/- " + str(desviacion))
34
35     indice='N='+str(len(dato.columns))
36
37     graficaKNN[indice][str(canal)]=precision
38     graficaCanal34[indice]['KNN']=precision
39     graficaCanal34Desviacion[indice]['KNN']=desviacion
```

Canal total

Código encargado de clasificar por medio de la validación cruzada el archivo de entrenamiento-testeo correspondiente al canal total para analizar el rendimiento del KNN en dicho canal.

```
1  canal='canal Total'
```

```
2
3
4  for dato in RFEcompleto:
5
6      print("N = " + str(len(dato.columns)) + "
           ------------------------->")
7      print(dato)
8
9      x=dato.to_numpy()
10     y=valoresTotales['fatiga'].to_numpy()
11
12
13
14     X_train, X_test, y_train, y_test = train_test_split(x, y,
           test_size=0.20)
15
16     #K=6
17     KNN = KNeighborsClassifier(n_neighbors=6)
18
19     KNN.fit(X_train, y_train)
20
21
22     y_pred = KNN.predict(X_test)
23
24     #Resultados
25     print(confusion_matrix(y_test, y_pred))
26     print(classification_report(y_test, y_pred))
27
28
29     #Cross validation k-fold (precision total)
30     resultados = cross_validate(KNN, x, y, cv=15)
31
32     precision=resultados['test_score'].mean()
33     desviacion=resultados['test_score'].std()
34
35     print("Precision : " + str(precision) + " +/- " + str(desviacion))
36
37     indice='N='+str(len(dato.columns))
38
39     graficaKNN[indice][canal]=precision
40     graficaCanalTotal[indice]['KNN']=precision
41     graficaCanalTotalDesviacion[indice]['KNN']=desviacion
```

6.6. Estudio del rendimiento obtenido: validación cruzada

Para analizar el rendimiento de cada clasificador en cada uno de los canales estudiados, se ha hecho uso de la validación cruzada. Se ha usado un valor de 15. Cada uno de los valores hallados (media de la precisión y desviación típica), son almacenados en un dataframe para el posterior análisis del rendimiento. Para hacer esto posible se ha hecho uso de del módulo [2]:

```
1  from sklearn.model_selection import cross_validate
```

Código extraído de los anteriores, donde se muestra el uso de la validación cruzada para el KNN:

```

#K=6
KNN = KNeighborsClassifier(n_neighbors=6)

x=dato.to_numpy()
y=valores34['fatiga'].to_numpy()

#Cross validation k-fold (precision total)
resultados = cross_validate(KNN, x, y, cv=10)
precision=resultados['test_score'].mean()
desviacion=resultados['test_score'].std()

print("Precision : " + str(precision) + " +/- " + str(desviacion))
```

6.7. Procesamiento de datos: diferentes modelos de estudio

Para la obtención de los archivos de entrenamiento-testeo correspondientes a cada uno de los canales estudiados, se ha implementado un código, donde se aplican las funciones de extracción de características anteriormente explicadas y se guardan los correspondientes datos en un archivo .csv, para así no perder tiempo de ejecución en próximas ejecuciones.

Para la extracción de las características se analiza la señal captada por cada canal y se le aplica las funciones de las características implementadas, con el tamaño de ventana oportuno para cada repetición detectada en la señal EMG. Todos los datos finales una vez son cargados desde su correspondiente archivo .csv son normalizados y sus valores entran en el rango [0,1].

6.7.1. Canal 1, canal 2, canal 3, canal 4

Para el procesamiento de datos de los canales 1, 2, 3 y 4, una misma función los realiza todos a la vez. Finalmente los guarda todos en un archivo en conjunto, y a la hora de cargar los datos desde el archivo creado, los divide en 4 dataset, cada uno, asociado a su correspondiente canal.

```

    def getFeaturesToFileCanal1234():

    fs=1024.0

```

```
6       caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','
            TM3','TM4','TM5','LOG','ACC','MNF','MDF']
7       funciones={'RMS':RMS,'MAV':MAV,'ZC':ZC,'WL':WL,'SSC':SSC,'IEMG':
            IEMG,'SSI':SSI,'VAR':VAR,'TM3':TM3,
8                  'TM4':TM4,'TM5':TM5,'LOG':LOG,'ACC':ACC,'MNF':MNF,'MDF
                       ':MDF}
9       index=[]
10
11      for c in caracteristicas:
12          for i in range(0,4):
13              index.append(c+str(i+1))
14
15      #Cargo las velocidades y creo el vector donde guardo si hay o no
            fatiga
16      velocidades=pd.read_excel("../Estudio tDCS/velocidades.xlsx")
17      fatigas=np.array([])
18
19      dfC=pd.DataFrame(columns=index)
20
21      #Recorro todos los archivos anteriormente elegidos con las 12
            repeticiones claras
22      indice=0
23      for dato in archivosElegidos:
24
25          archivo=dato[0]
26          threshold=dato[1]
27          smooth=dato[2]
28
29          archivoSinRuta=archivo.replace("../Estudio tDCS/","")
30          archivoSinRuta=archivoSinRuta.replace(".csv","")
31
32
33          velocidad=velocidades[archivoSinRuta]
34          valoresDeFatiga=calculoDeFatiga(velocidad)
35          print(archivoSinRuta)
36          print(valoresDeFatiga)
37
38          fatigas = np.concatenate((fatigas, valoresDeFatiga))
39
40          df = pd.read_csv(archivo,delimiter=";",decimal=",")
41          df=df.fillna(0)
42
43          #1-Obtengo EMG puro
44
45          #Pierna izquieda
46          vastoMedialIzquierdo= df['emg_muscle_1_( V )'].to_numpy()
47          rectoFemoralIzquierdo=df['emg_muscle_2_( V )'].to_numpy()
48
49          #Pierna derecha
50          vastoMedialDerecho=df['emg_muscle_3_( V )'].to_numpy()
51          rectoFemoralDerecho=df['emg_muscle_4_( V )'].to_numpy()
52
53
54
55          #2-Calculo sus activaciones
56          activacion1=bsnb.detect_emg_activations(vastoMedialIzquierdo,
                fs,smooth_level=smooth, threshold_level=threshold,
57                                                  time_units=False,
                                                        volts=False,
                                                        resolution=
                                                        None,
                                                        plot_result=
```

```
                                                    False)
58
59          activacion2=bsnb.detect_emg_activations(rectoFemoralIzquierdo,
                fs,smooth_level=smooth, threshold_level=threshold,
60                                                  time_units=False,
                                                        volts=False,
                                                        resolution=
                                                        None,
                                                        plot_result=
                                                        False)
61
62          activacion3=bsnb.detect_emg_activations(vastoMedialDerecho,fs,
                smooth_level=smooth, threshold_level=threshold,
63                                                  time_units=False,
                                                        volts=False,
                                                        resolution=
                                                        None,
                                                        plot_result=
                                                        False)
64
65          activacion4=bsnb.detect_emg_activations(rectoFemoralDerecho,fs
                ,smooth_level=smooth, threshold_level=threshold,
66                                                  time_units=False,
                                                        volts=False,
                                                        resolution=
                                                        None,
                                                        plot_result=
                                                        False)
67
68
69          vmI=[]
70          rfI=[]
71          vmD=[]
72          rfD=[]
73
74          #3- Valores de emg acotados dependiendo de sus activaciones
75          #ej: vmI -> lista con los valores EMG de cada repeticion
76          for i,j in zip(activacion1[0],activacion1[1]):
77              vmI.append(vastoMedialIzquierdo[i:j])
78
79          for i,j in zip(activacion2[0],activacion2[1]):
80              rfI.append(rectoFemoralIzquierdo[i:j])
81
82          for i,j in zip(activacion3[0],activacion3[1]):
83              vmD.append(vastoMedialDerecho[i:j])
84
85          for i,j in zip(activacion4[0],activacion4[1]):
86              rfD.append(rectoFemoralDerecho[i:j])
87
88
89          for i,j,k,l in zip(vmI,rfI,vmD,rfD):
90
91              # A cada archivo EMG (4 se ales EMG diferentes = 4
                    canales) le extraigo una caracteristica
92              datos=np.array([])
93              for caracteristica in caracteristicas:
94
95                  x1,y1=funciones[caracteristica](datos=i,ratio=i.size)
96                  x2,y2=funciones[caracteristica](datos=j,ratio=j.size)
97                  x3,y3=funciones[caracteristica](datos=k,ratio=k.size)
98                  x4,y4=funciones[caracteristica](datos=l,ratio=l.size)
99
```

```
100                 #Guardo todos los datos obtenido en un mismo array
101                 datos=np.append(datos,y1)
102                 datos=np.append(datos,y2)
103                 datos=np.append(datos,y3)
104                 datos=np.append(datos,y4)
105
106             #A ado al final del dataFrame los datos de un archivo (
                    RMS,ARV,.....,ZCC)
107             dfC.loc[indice]=datos
108             indice=indice+1
109
110         #print("Log --> Archivo EMG: " + str(archivo) + " \n Archivo
                velocidad : "+ str(archivoSinRuta) )
111
112     dfC['fatiga']=fatigas
113
114
115     dfC.to_csv('caracteristicasCanales1234.csv', index = False)
116
117     return dfC
```

Código correspondiente a cargar el archivo de datos anteriormente creado:

```
1
2 valores1234= pd.read_csv("caracteristicasCanales1234.csv",delimiter
      =",",decimal=".")
3 caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','TM3
      ','TM4','TM5','LOG','ACC','MNF','MDF']
4
5 #Lista de dataframe con los datos de cada canal
6 datos1234=[]
7
8
9 #4 canales -> 4 clasificadores
10 for i in range(1,5):
11     datosProcesados=pd.DataFrame()
12     for caracteristica in caracteristicas:
13         datosProcesados[caracteristica+str(i)]=valores1234[
               caracteristica+str(i)]
14
15
16
17
18     datosProcesados=(datosProcesados-datosProcesados.min())/(
           datosProcesados.max()-datosProcesados.min())
19     datos1234.append(datosProcesados)
```

6.7.2. Canal 1,2 y canal 3,4

Con respecto a los archivos de entrenamiento-testeo de los canales 1,2 y 3,4, se forman a partir del archivo .csv creado anteriormente. Su procesamiento solo trata de agrupar los datos del canal 1,2 y los datos del canal 3,4.

Canal 1,2

```
        valores1234= pd.read_csv("caracteristicasCanales1234.csv",
            delimiter=",",decimal=".")
caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','TM3
    ','TM4','TM5','LOG','ACC','MNF','MDF']

valores12=pd.DataFrame()
for c in caracteristicas:
    g=np.array([])
    for i in range(1,3):
        g=np.append(g,valores1234[c+str(i)].values)

    valores12[c]=g


valores12=(valores12-valores12.min())/(valores12.max()-valores12.min()
    )

fatiga=np.array(valores1234['fatiga'])

fatiga=np.append(fatiga,valores1234['fatiga'])

valores12['fatiga']=fatiga

valores12
```

Canal 3,4

```
        valores1234= pd.read_csv("caracteristicasCanales1234.csv",
            delimiter=",",decimal=".")
caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','TM3
    ','TM4','TM5','LOG','ACC','MNF','MDF']

valores34=pd.DataFrame()
for c in caracteristicas:
    g=np.array([])
    for i in range(3,5):
        g=np.append(g,valores1234[c+str(i)].values)

    valores34[c]=g



valores34=(valores34-valores34.min())/(valores34.max()-valores34.min()
    )

fatiga=np.array(valores1234['fatiga'])

fatiga=np.append(fatiga,valores1234['fatiga'])

valores34['fatiga']=fatiga

valores34
```

6.7.3. Canal total

Con respecto al archivo de entrenamiento-testeo del canal total, se vuelven a analizar todos los archivos EMG del dataset, y se genera un nuevo archivo .csv, que contiene los valores de los 4 canales agrupados.

```
1     def getFeaturesToFileTotales():
2
3     fs=1024.0
4
5     caracteristicas=['RMS','MAV','ZC','WL','SSC','IEMG','SSI','VAR','
          TM3','TM4','TM5','LOG','ACC','MNF','MDF']
6     funciones={'RMS':RMS,'MAV':MAV,'ZC':ZC,'WL':WL,'SSC':SSC,'IEMG':
          IEMG,'SSI':SSI,'VAR':VAR,'TM3':TM3,
7                'TM4':TM4,'TM5':TM5,'LOG':LOG,'ACC':ACC,'MNF':MNF,'MDF
                     ':MDF}
8
9     #Cargo las velocidades y creo el vector donde guardo si hay o no
          fatiga
10    velocidades=pd.read_excel("../Estudio tDCS/velocidades.xlsx")
11
12
13    dfC=pd.DataFrame()
14
15    #Recorro todos los archivos anteriormente elegidos con las 12
          repeticiones claras
16    indice=0
17    for dato in archivosElegidos:
18
19        archivo=dato[0]
20        threshold=dato[1]
21        smooth=dato[2]
22
23        archivoSinRuta=archivo.replace("../Estudio tDCS/","")
24        archivoSinRuta=archivoSinRuta.replace(".csv","")
25
26
27        velocidad=velocidades[archivoSinRuta]
28        valoresDeFatiga=calculoDeFatiga(velocidad)
29
30
31        fatigas=np.array([])
32        fatigas = np.concatenate((fatigas, valoresDeFatiga))
33        fatigas = np.concatenate((fatigas, valoresDeFatiga))
34        fatigas = np.concatenate((fatigas, valoresDeFatiga))
35        fatigas = np.concatenate((fatigas, valoresDeFatiga))
36
37        print(archivoSinRuta)
38        print(valoresDeFatiga)
39
40        df = pd.read_csv(archivo,delimiter=";",decimal=",")
41        df=df.fillna(0)
42
43        #1-Obtengo EMG puro
44
45        #Pierna izquieda
46        vastoMedialIzquierdo= df['emg_muscle_1_( V )'].to_numpy()
47        rectoFemoralIzquierdo=df['emg_muscle_2_( V )'].to_numpy()
48
49        #Pierna derecha
```

```
50          vastoMedialDerecho=df['emg_muscle_3_( V )'].to_numpy()
51          rectoFemoralDerecho=df['emg_muscle_4_( V )'].to_numpy()
52
53
54
55          #2-Calculo sus activaciones
56          activacion1=bsnb.detect_emg_activations(vastoMedialIzquierdo,
                fs,smooth_level=smooth, threshold_level=threshold,
57                                                  time_units=False,
                                                        volts=False,
                                                        resolution=
                                                        None,
                                                        plot_result=
                                                        False)
58
59          activacion2=bsnb.detect_emg_activations(rectoFemoralIzquierdo,
                fs,smooth_level=smooth, threshold_level=threshold,
60                                                  time_units=False,
                                                        volts=False,
                                                        resolution=
                                                        None,
                                                        plot_result=
                                                        False)
61
62          activacion3=bsnb.detect_emg_activations(vastoMedialDerecho,fs,
                smooth_level=smooth, threshold_level=threshold,
63                                                  time_units=False,
                                                        volts=False,
                                                        resolution=
                                                        None,
                                                        plot_result=
                                                        False)
64
65          activacion4=bsnb.detect_emg_activations(rectoFemoralDerecho,fs
                ,smooth_level=smooth, threshold_level=threshold,
66                                                  time_units=False,
                                                        volts=False,
                                                        resolution=
                                                        None,
                                                        plot_result=
                                                        False)
67
68
69          vmI=[]
70          rfI=[]
71          vmD=[]
72          rfD=[]
73
74          #3- Valores de emg acotados dependiendo de sus activaciones
75          #ej: vmI -> lista con los valores EMG de cada repeticion
76          for i,j in zip(activacion1[0],activacion1[1]):
77              vmI.append(vastoMedialIzquierdo[i:j])
78
79          for i,j in zip(activacion2[0],activacion2[1]):
80              rfI.append(rectoFemoralIzquierdo[i:j])
81
82          for i,j in zip(activacion3[0],activacion3[1]):
83              vmD.append(vastoMedialDerecho[i:j])
84
85          for i,j in zip(activacion4[0],activacion4[1]):
86              rfD.append(rectoFemoralDerecho[i:j])
87
```

```
88
89
90          dfCcaracteristica=pd.DataFrame()
91          for caracteristica in caracteristicas:
92
93
94              datos=np.array([])
95              for i in vmI:
96                  x1,y1=funciones[caracteristica](datos=i,ratio=i.size)
97                  datos=np.append(datos,y1)
98
99              for j in rfI:
100                 x2,y2=funciones[caracteristica](datos=j,ratio=j.size)
101                 datos=np.append(datos,y2)
102
103             for k in vmD:
104                 x3,y3=funciones[caracteristica](datos=k,ratio=k.size)
105                 datos=np.append(datos,y3)
106
107             for l in rfD:
108                 x4,y4=funciones[caracteristica](datos=l,ratio=l.size)
109                 datos=np.append(datos,y4)
110
111
112             dfCcaracteristica[caracteristica]=datos
113
114
115
116         dfCcaracteristica['fatiga']=fatigas
117         dfC=pd.concat([dfC, dfCcaracteristica], ignore_index=False)
118
119
120
121     dfC.to_csv('caracteristicasTotales.csv', index = False)
122
123     return dfC
```

Finalmente bastaría con cargar dicho archivo:

```
1       valoresTotales= pd.read_csv("caracteristicasTotales.csv",delimiter
            =",",decimal=".")
2
3
4   datosTotales=valoresTotales.drop(['fatiga'],axis=1)
5
6   datosTotales=(datosTotales-datosTotales.min())/(datosTotales.max()-
        datosTotales.min())
7
8   datosTotales
```

Capítulo 7

Evaluación

Este capítulo está dedicado a la evaluación de los resultados obtenidos por cada uno de los clasificadores. Consta de dos secciones, una dedicada a mostrar los resultados obtenidos con tablas y gráficas, y otra dedicada a la discusión de los resultados obtenidos, donde se da una perspectiva propia sobre las limitaciones y resultados obtenidos.

7.1. Resultados

7.1.1. Estimación del tamaño de ventana

Con respecto a los resultados de la estimación del tamaño de ventana de cada archivo se aconseja visitar el Anexo para ver un ejemplo de dos archivos EMG con sus respectivas ventanas. (Ver A.1).

7.1.2. Extracción de características

Todas las gráficas asociadas a cada una de las características pueden ser consultadas en el Anexo. (Ver A.4.2).

7.1.3. Procesamiento de datos: diferentes modelos de estudio

Esta sección va dedicada a mostrar cada uno de los archivos de entrenamiento-testeo utilizados para el estudio. Hay que recordar que de cada uno de ellos, se sacarán 3 con el RFE, cada uno con un número diferente de características. Pueden ser consultados en el Anexo. (Ver A.2).

7.1.4. Selección de características: RFE

Para ver que características de todas las desarrolladas fueron elegidas en cada canal se aconseja visitar el Anexo. (Ver A.3).

7.1.5. Clasificación binaria de señales y validación cruzada

Esta sección está dedicada a mostrar el rendimiento de los diferentes clasificadores en los diferentes canales estudiados. Para ello se ha hecho uso de una tabla y una gráfica, obtenidas durante la ejecución del código fuente dedicado a los clasificadores. La tabla muestra los valores en bruto de la precisión media de cada clasificador, junto con su desviación típica entre paréntesis. Finalmente se muestra la gráfica asociada a dicha tabla.

Cuadro 7.1: Precisión y desviación media de los clasificadores en cada canal

		N = 1	N = 2	N = 5
canal 1	**SVM**	0.829167 (0.0482327)	0.829167 (0.0482327)	0.8375 (0.0306186)
	TREE	0.754167 (0.0980717)	0.729167 (0.108653)	0.729167 (0.132419)
	KNN	0.825 (0.0520416)	0.829167 (0.0482327)	0.833333 (0.0709558)
canal 2	**SVM**	0.733333 (0.083749)	0.745833 (0.108173)	0.7375 (0.107529)
	TREE	0.783333 (0.111492)	0.770833 (0.101208)	0.766667 (0.119606)
	KNN	0.845833 (0.0386401)	0.833333 (0.0372678)	0.841667 (0.0448764)
canal 3	**SVM**	0.8375 (0.0306186)	0.833333 (0.0372678)	0.766667 (0.0980717)
	TREE	0.758333 (0.0991281)	0.6875 (0.0853913)	0.795833 (0.0953794)
	KNN	0.825 (0.0408248)	0.820833 (0.0386401)	0.820833 (0.0752311)
canal 4	**SVM**	0.795833 (0.125968)	0.791667 (0.130437)	0.829167 (0.0276385)
	TREE	0.708333 (0.0903312)	0.733333 (0.0953794)	0.7 (0.129502)
	KNN	0.833333 (0.0372678)	0.841667 (0.0311805)	0.8125 (0.0603807)
canal total	**SVM**	0.819792 (0.0660124)	0.823958 (0.0661602)	0.827083 (0.0304409)
	TREE	0.7375 (0.0593476)	0.740625 (0.0721237)	0.751042 (0.053966)
	KNN	0.841667 (0.0169891)	0.845833 (0.0295731)	0.835417 (0.0364732)
canal 1,2	**SVM**	0.8375 (0.0125)	0.81875 (0.05)	0.775 (0.128695)
	TREE	0.733333 (0.052125)	0.733333 (0.0557462)	0.747917 (0.077784)
	KNN	0.822917 (0.0186339)	0.822917 (0.0186339)	0.833333 (0.0315953)
canal 3,4	**SVM**	0.8375 (0.0125)	0.8375 (0.0125)	0.8375 (0.0125)
	TREE	0.73125 (0.0682749)	0.735417 (0.052125)	0.735417 (0.0764331)
	KNN	0.827083 (0.01932)	0.825 (0.0222439)	0.827083 (0.0320048)

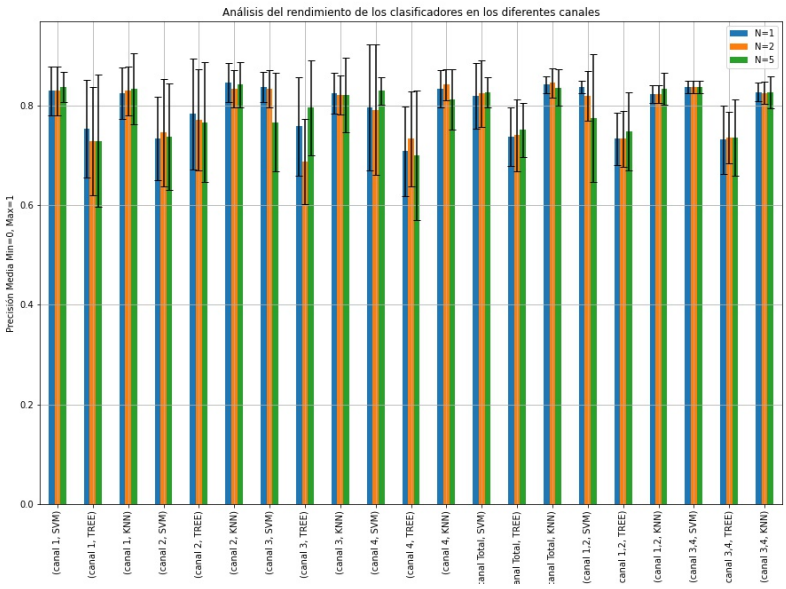

Figura 7.1: Diagrama resumen de la precisión y desviación de cada clasificador

Una vez vistas ambas tablas y la gráfica resumen de las precisiones podemos llegar a indagar en que clasificador es el más conveniente para nuestro problema. A continuación se muestra un breve resumen sobre los resultados obtenidos.

Canal 1: vasto medial izquierdo

La precisión más alta en este canal nos la da el SVM usando un archivo de entrenamiento-testeo con 5 características.

Canal 2: recto femoral izquierdo

En este canal la mejor precisión se obtiene con un KNN utilizando un número de vecinos de 6 y un archivo de entrenamiento-testeo con 1 característica. Su desviación típica es de las menores.

Canal 3: vasto medial derecho

En este canal la mejor precisión viene dada por un SVM con un archivo de entrenamiento-testeo con solo 1 característica. Además su desviación típica es bastante pequeña.

Canal 4: recto femoral derecho

En el canal 4 la mejor precisión nos la da un KNN utilizando un número de vecinos de 6 y un archivo de entrenamiento-testeo con 2 características.

Canal 1,2: extremidad inferior izquierda

Para el canal 1,2 el clasificador que nos da la mejor precisión es un SVM utilizando un archivo de entrenamiento-testeo con 1 característica.

Canal 3,4: extremidad inferior derecha

Para el canal 3,4 el clasificador que nos da la mejor precisión es un SVM utilizando tanto un archivo de entrenamiento-testeo con 1, 2 ó 5 características.

Canal total: agrupación de todos los canales

Para el canal total la mejor precisión viene dada por un KNN utilizando un número de vecinos de 6 y un archivo de entrenamiento-testeo con 2 características.

7.2. Discusión

Como se ha ido explicando a lo largo del proyecto, la idea era desarrollar un clasificador para estimar la fatiga durante los entrenamientos y poder obtener conclusiones de anomalías en las rodillas de los pacientes y atletas. Además de esto, como ya se explicó en el capitulo 5.9, se pretendía analizar diferentes situaciones debido a la ventaja de que la herramienta utilizada para tomar los datos llegaba a registrar los datos de 4 músculos diferentes. Por ello se optó por aumentar el estudio a diferentes niveles musculares.

7.2.1. Modelos utilizados

Los diferentes modelos que fueron utilizados para el proyecto han sido:

- Support vector machines (SVMs): máquinas de vector de soporte.
- Decision trees (Trees): árboles de decisión.
- K nearest neighbors (KNN): K vecinos más cercanos.

Viendo los resultados obtenidos (precisión y desviación) mediante la validación cruzada por cada uno de ellos, considero que el que mejor prestaciones ha dado ha sido el KNN con k = 6. Como se puede ver en la gráfica 7.1, el clasificador KNN ha sido el que más estable se ha mantenido, dando en todas las situaciones estudiadas un rendimiento muy bueno y sin gran variación.

El hecho de que KNN obtenga una precisión mucho mayor a la de los árboles de decisión, pienso que se debe a que la fatiga muscular estudiada puede tener diferentes situaciones. Por ejemplo, que la fatiga esté representada en tres grupos diferentes. Estos tres grupos supongamos que tienen dichos valores:

- Grupo 1: RMS con valores altos, IEMG con valores bajos.
- Grupo 2: RMS con valores bajos pero con un MNF con valores muy altos (cerca del 1).
- Grupo 3: MAV con valores bajos y ZC con valores muy bajos (cerca del 0).

Esta situación daría 3 subgrupos representativos de la fatiga, por ello pienso que como el KNN divide todo el conjunto de datos en subgrupos, en este caso de k = 6 para determinar la nueva categoría, tiene mayor capacidad a la hora de determinar una categoría para una señal no conocida.

En segunda posición dejaría al SVM. Este clasificador también ha obtenido un buen rendimiento, pero en ciertas ocasiones una precisión muy baja y una gran variación indicada por los valores de su desviación, por lo que esto nos indica que hay veces en donde ha dado un rendimiento muy alto y otras ocasiones donde el rendimiento fue bajo. Se puede deber a que en ocasiones la fatiga no estaba realmente clara y su separación entre los vectores de soporte era muy pequeña y debido a esto el clasificador no era capaz de clasificar algunos datos correctamente.

En tercera y última posición situaría al Desion Tree. Pienso que ha sido el que peores resultados ha obtenido. En todas las situaciones estudiadas ha dado la precisión mas baja y con una gran variación.

Los árboles de decisión van realizando ramificaciones para determinar la categoría final de los datos de testeo y esto puede hacer que como no hay un patrón realmente claro en la fatiga con las características utilizadas, que no se obtenga un rendimiento del todo estable. Otra posibilidad de este rendimiento, se debe a que el conjunto de datos utilizados consta de un número mucho mayor de no fatiga = 0, que de fatiga = 1. Por ello, el árbol de decisión puede sesgar los datos al haber una clase dominante.

En conclusión, yo personalmente me quedaría con un clasificador SVM o KNN. A continuación se hace una valoración del rendimiento de cada uno

de estos en los diferentes canales estudiados.

7.2.2. Canal 1, canal 2, canal 3, canal 4

Los datos obtenidos son bastantes curiosos. Podemos ver que para tanto el canal 1 como para el canal 3 (ambos registraban información del vasto medial, solo que uno de la pierna derecha y otro de la pierna izquierda) la mejor opción es el SVM.

Sin embargo, tanto para el canal 2 como para el canal 4 (ambos registraban información del recto femoral , uno en la pierna derecha y otro en la izquierda) la mejor opción es un KNN.

Esto podría justificarse, ya que la sentadilla es un ejercicio muy demandante de vasto medial. Como se puede ver en este estudio [33], donde se analiza que músculos del cuadriceps realizan una mayor activación a la hora de realizar diferentes sentadillas. Es decir, aunque el recto femoral también forma parte del cuadricpes y apoya el movimiento de la rodilla junto con el vasto medial, es el vasto medial el músculo más implicado.

Debido a esto, la señal de los canales del vasto medial puede llegar a ser más intensa que la de los canales del recto femoral. Esto puede hacer que la fatiga quede mejor definida en estos casos, ya que al tener una mayor activación, es lógico que dicho músculo se fatigue antes. Por ello pienso que los dos grupos definidos en el SVM pueden quedar definidos con una mayor separación entre los dos vectores de soporte de cada grupo y por ello clasificar mejor los datos no conocidos.

Otro factor a tener en cuenta es que la desviación típica del clasificador SVM en los respectivos canales del recto femoral, es muy alta. Puede ser debido a que como la señal no tiene valores tan característicos de la fatiga, haya veces donde los dos grupos definidos por los vectores de soporte correspondientes queden mejor definidos y otras veces peor definidos.

Por otro lado el clasificador KNN, al determinar la nueva categoría basándose en sus k vecinos más cercanos, puede llegar a clasificar mejor los datos que estén menos "definidos", es decir, menos representativos de la fatiga en ese caso.

En resumen, pienso que el clasificador SVM puede llegar a tener mejores prestaciones con datos musculares de músculos más demandantes, mientras que el KNN puede ser mejor para analizar la señal muscular en músculos menos demandantes.

7.2.3. Canal 1,2 y canal 3,4

Otra de la observación que se puede hacer, y relacionada con la conclusión anterior, es que una vez que se utiliza un clasificador para detectar la fatiga global de la extremidad inferior, ya sea derecha o izquierda, el clasificador que mejor prestaciones nos da es el SVM.

Esto se puede dar debido a que el canal 1,2 trata de la agrupación de los datos captados por el electromiógrafo en el canal 1 y en el canal 2. Como individualmente para estos dos canales separados, el SVM obtenía mejores prestaciones en el canal del vasto medial y este era el más demandante, para la agrupación de los datos en el canal 1,2 también lo será. Cabe destacar que esto también se cumple en el canal 3,4.

Como se explicó en la sección anterior, como el vasto medial tiene mayor activación, esto puede hacer que a la hora de clasificar los datos agrupados en canales de una misma extremidad, sea el vasto medial el más representativo de la fatiga y por ende el clasificador SVM el que mejor prestaciones nos da, ya que conseguiría dos grupos mejor definidos.

7.2.4. Canal total: agrupación de todos los canales

Sin embargo, para el clasificador de la fatiga global, el mejor rendimiento viene dado por un KNN utilizando un número de vecinos de 6 y dos características en el archivo de entrenamiento-testeo.

Mi opinión es que a la hora de juntar todos los canales en un canal total, es decir, agrupar los datos recibidos de los 4 canales, el conjunto de datos se hace muy grande. Esto hace que el SVM decaiga en prestaciones, ya que no podrá definir un modelo con dos grupos claramente diferenciados.

Debido a esto el KNN obtiene mejores prestaciones, ya que como se explicó anteriormente, basa su predicción en sus k vecinos y esto dota al modelo de una mayor flexibilidad a la hora de predecir la fatiga.

7.2.5. Número de características en los archivos de entrenamiento-testeo

Aunque hay veces que la mayor precisión se obtiene solo con una característica en el archivo de entrenamiento-testeo, existe un gran número de ocasiones donde el mejor rendimiento viene dado por 2 o incluso 5 características. Esto se puede dar debido a que la fatiga es compleja de clasificar solo teniendo en cuenta las características de la señal.

Pueden existir ocasiones donde con sola una característica no sea capaz de clasificar bien la fatiga. Este aumento de las características utilizadas

aunque nos daría un modelo mas complejo, sería complementado con un aumento de la precisión de nuestro modelo.

7.2.6. Mejor opción para el proyecto

Después de realizar el análisis de los resultados obtenidos, me decantaría por la opción de utilizar un clasificador por grupo muscular estudiado. Esto es, la opción de los clasificadores del canal 1, canal 2, canal 3 y canal 4, donde para los canales 1 y 3, utilizaría un SVM y para los canales 2 y 4 un KNN.

Pienso que es la opción que mejor rendimiento aporta a la resolución de nuestro problema, ya que nos da un nivel de exactitud muy alto a la hora de analizar 4 músculos diferentes y con ello, se podrá analizar cuál de los 4 entra antes en fatiga y llegar a conclusiones más especificas, como compensación muscular, anomalías en la rodilla o falta de capacidad muscular.

7.2.7. Limitaciones

Sobre las limitaciones del proyecto, me gustaría destacar que hay ocasiones donde la mejor opción es un clasificador KNN. Como se vio en la fase de diseño, este clasificador utiliza un K = 6, es decir, el número de vecinos para determinar la nueva etiqueta es 6.

Debido a esto, si finalmente a la hora de llevar el proyecto al mercado o simplemente como complemento a la aplicación de mDurance, se elige la opción del KNN, sería conveniente realizar un estudio para estimar cual sería el K, que mejor prestaciones nos da.

También pienso que sería correcto probar los clasificadores estudiados con un conjunto de datos más grande. Para este proyecto se han utilizado 19 muestras y cada muestra contenía 4 señales de EMG. Con un conjunto de datos más grandes, quizá se podrían obtener unas conclusiones con una mayor validez.

Capítulo 8

Conclusiones y Trabajos Futuros

Este capítulo está dedicado a realizar una conclusión final sobre el proyecto y a analizar si realmente se han cumplido los objetivos que se tenían antes de su comienzo. (Objetivos 1.3).

8.1. Conclusiones

El objetivo final de este proyecto era el de diseñar un clasificador para detectar la fatiga muscular y así poder detectar anomalías en la rodilla. Pienso que este objetivo se ha cumplido satisfactoriamente ya que se ha realizado un análisis sobre el rendimiento de diversos clasificadores que resolvían el problema. Con ello se ha obtenido el clasificador con las mejores prestaciones.

Con respecto a los objetivos específicos, se ha conseguido analizar y procesar las señales brutas de EMG. Esto se ha resuelto con el hecho de detectar la activación muscular de cada repetición registrada para así estimar el tamaño de ventana correcto.

Otro de los objetivos específicos fue analizar las velocidades registradas para estimar la fatiga de cada repetición. Para ello se acordó la estimación de un umbral de fatiga y la condición de que dicho umbral tenia que registrase dos veces seguidas.

El siguiente objetivo específico y uno de los más importantes fue la extracción de características. Este objetivo es primordial ya que nos dota de la información necesaria para poder clasificar en un futuro las señales. Opino que se ha cumplido correctamente, ya que se han estudiado 15 características diferentes de la señal EMG.

El objetivo de la creación de los archivos de entrenamiento-testeo también creo que se ha cumplido correctamente. Se han analizado los clasificadores con diferentes archivos de entrenamiento-testeo obtenidos gracias al procesamiento y extracción de las características de la señal EMG, tanto en el dominio del tiempo como en el dominio de la frecuencia.

Otro de los objetivos importantes, era el diseño de un algoritmo de selección de características, para obtener unos modelos sencillos y eficaces. Para ello se ha indagado en el tema de la selección de características y llegado a la conclusión de que uno de los mejores algoritmos para ello era el RFE (Recursive Feautre Elimination).

Los dos últimos objetivos, también se han realizado correctamente. Uno de ellos era investigar sobre que tipo de clasificadores de aprendizaje supervisado era el más adecuado para la resolución del proyecto.

El último objetivo, que es complementario al anterior, ha sido la validación de los resultados obtenidos por los clasificadores. Para ello se ha hecho uso de la validación cruzada, para estimar así una precisión y desviación a cada clasificador en cada una de las situaciones estudiadas.

Como conclusión final, lo que se ha conseguido es realizar un estudio sobre que clasificador sería el mejor a la hora de resolver el problema de la detección de la fatiga. Además se ha indagado en el problema, y también se aportan diferentes soluciones, en caso de que la herramienta futura que se utilice no disponga de 4 canales de registro de la actividad muscular.

Mi opinión es que se ha realizado un proyecto muy interesante, donde era necesario indagar e investigar en cada uno de los objetivos específicos para llegar a obtener una buena solución para el objetivo general del proyecto.

8.2. Trabajos Futuros

Con respecto a los trabajos futuros creo que es un proyecto muy interesante que aporta una gran cantidad de información sobre que tipos de clasificadores y condiciones son las óptimas si se quiere llevar al mercado esta idea.

Debido a todo lo comentado anteriormente, pienso que se ha realizado un buen proyecto, que puede ser utilizado por algún emprendedor, como recurso para ver que sistema de clasificación sería mas conveniente implementar y desarrollar en un sistema de EMG final, para llevarlo al mercado y que sea utilizado por los médicos y fisioterapeutas. Por ejemplo, sería de gran utilidad que si un paciente siente molestia en su rodilla durante la actividad física o su vida cotidiana, realizar un análisis con una herramienta como la explicada en este proyecto y analizar el rendimiento de la musculatura de

su pierna.

También podría ser un complemento a la herramienta base de mDurance. Dicha herramienta solo capta, analiza la señal EMG y extrae las correspondientes características. Con este nuevo complemento, también se dotaría al fisioterapeuta de una opción para analizar la fatiga muscular. Se podría hallar que músculos se han fatigado antes, y si se ha producido dicha fatiga, el orden en que se han fatigado, ya que una alteración en la fatiga puede ser indicativo de una compensación muscular debido a una lesión. (Para mayor detalle ver 4.3).

Para finalizar cabe aclarar que esta herramienta, es claramente orientativa para médicos y fisioterapeutas. Serviría de apoyo a la hora de realizar sus diagnósticos. Ellos son los que realmente tienen la última palabra sobre el diagnostico.

Bibliografía

[1] Enlace a biosignal. https://github.com/biosignalsplux/biosignalsnotebooks.

[2] Enlace a cv de scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.htmll.

[3] Enlace a glob. https://docs.python.org/3/library/glob.html.

[4] Enlace a jupyter. https://jupyter.org/.

[5] Enlace a knn de scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[6] Enlace a matplotlib. https://matplotlib.org/.

[7] Enlace a numpy. https://numpy.org/.

[8] Enlace a pandas. https://pandas.pydata.org/.

[9] Enlace a python 3. https://www.python.org/download/releases/3.0/.

[10] Enlace a rfecv de sklearn. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html.

[11] Enlace a scikit-learn. https://scikit-learn.org/stable/modules/classes.html.

[12] Enlace a scipy. https://www.scipy.org/.

[13] Enlace a scipy.signal. https://docs.scipy.org/doc/scipy/reference/signal.html.

[14] Enlace a svm de scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[15] Enlace a trees de scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[16] Imagen de electrodos intramusculares. https://medlineplus.gov/spanish/ency/images/ency/fullsize/9741.jpg.

[17] Imagen de electrodos superficiales. https://tienda.fisaude.com/images/fotos/mdurance-descripcion-5.jpg.

[18] Imagen de ligamento anterior cruzado. https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcQ9E0yoAgcnwi0wulfmvmFZbJsONQVWeRpvbw&usqp=CAU.

[19] Imagen de motoneurona. https://elaticodejulie.files.wordpress.com/2016/07/mn_dendrites_junction_fiber.jpg.

[20] Imagen de unidad motora. https://image-api.onlineeducation.center/v2/image/max-width/800/blog/c664ddbe01ccb4fe53209850bb8a3ca5ace253f2.jpg.

[21] Imagen del sistema nervioso central. https://img.yumpu.com/2981354/1/500x640/anatoma-a-general-del-sistema-nervioso-central.jpg.

[22] Imagen electromiógrafo mdurance. https://www.fisiomarket.com/24001-thickbox_default/electromiografo-de-superficie-de-dos-canales-mdurance-pro.jpg.

[23] Precio de electromiógrafo. https://www.herycor.com/articulos/000065-DIAGNOSTICO/0000003401-MDurance-Premium---EMG-Superficie-4-canales/.

[24] Salario de un fisioterapeuta. https://es.indeed.com/salaries/fisioterapeuta-Salaries.

[25] Salario de un ingeniero informático. https://www.glassdoor.es/Sueldos/ingeniero-inform%C3%A1tico-sueldo-SRCH_KO0,21.htm.

[26] Web de mdurance. https://www.mdurance.eu/.

[27] Roger Álvarez Fiallo, Carlos Santos Anzorandia, and Esther Medina Herrera. Desarrollo histórico y fundamentos teóricos de la electromiografía como medio diagnóstico. *Revista Cubana de Medicina Militar*, 35(4):0–0, 2006.

[28] D Víctor Amador Fernández. Propuesta de un valor de acumulación de fatiga para el entrenamiento de fuerza basado en la velocidad de ejecución.

[29] Hyun-Jung Kim, Jin-Hyuck Lee, Sung-Eun Ahn, Min-Ji Park, and Dae-Hee Lee. Influencia de la rotura del ligamento cruzado anterior en la fuerza muscular del muslo y la relación isquiotibiales-cuádriceps: Un meta-análisis-ciencias del ejercicio. *PubliCE*, 2018.

[30] Peter Konrad. The abc of emg. 2005.

[31] JI Ibarra Lúzar, E Pérez Zorrilla, and C Fernández García. Electromiografía clínica. *Rehabilitación*, 39(6):265–276, 2005.

[32] Jorge Enrique Moreno Quinchanegua et al. La fatiga, tipos causas y efectos. 2017.

[33] Jose M Muyor, Isabel Martın-Fuentes, David Rodrıguez-Ridao, and Jose A Antequera-Vique. Actividad electromiográfica en el glúteo medio, glúteo mayor, bíceps femoral, vasto externo, vasto interno y recto femoral durante los ejercicios de sentadilla monopodal, estocada frontal y subida lateral-ciencias del ejercicio. *PubliCE*, 2020.

[34] Angkoon Phinyomark, Pornchai Phukpattaranont, and Chusak Limsakul. Feature reduction and selection for emg signal classification. *Expert systems with applications*, 39(8):7420–7431, 2012.

[35] Guilherme Alexandre dos Santos Espadanal Ramos. *EMG Signal Processing in Amateur and Professional Sports with Performance Evaluation and Injury Prevention*. PhD thesis, 2018.

[36] Harold A Romo, Judy C Realpe, and Pablo E Jojoa. Análisis de señales emg superficiales y su aplicación en control de prótesis de mano. *Revista avances en sistemas e informática*, 4(1):127–136, 2007.

[37] Gustavo Ramón Suárez. Bases fisiológicas del reclutamiento de motoneuronas. *VIREF Revista de Educación Física*, 2(1):85–102, 2013.

Apéndice A

Anexo

A.1. Estimación de ventana

Con respecto a la estimación del tamaño de ventana de cada repetición, a continuación se muestra una captura de pantalla sobre dicha resolución al problema:

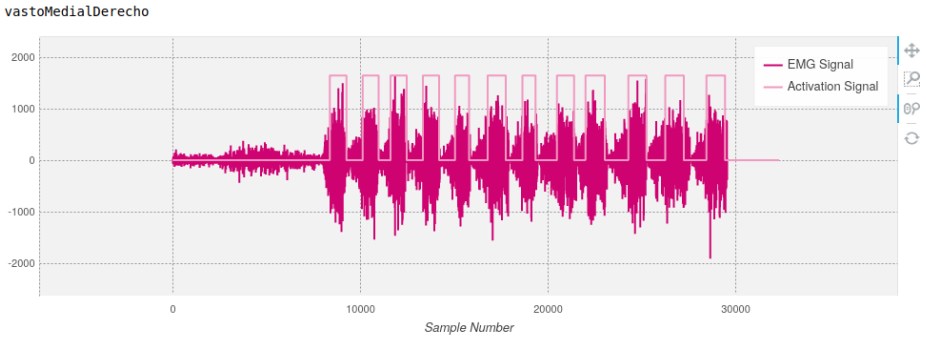

Figura A.1: s1 serie 1 día 1: ejemplo de activación muscular

Otra funcionalidad que debe ser mostrada fue el análisis de todos los archivos de EMG disponibles, para descartar aquellos con mucho ruido o poca información. Podemos ver dicho ejemplo. (Ver imagen A.2).

Como se puede apreciar, en este archivo de EMG, solo había registradas 11 repeticiones, por lo que no es válido para nuestro estudio.

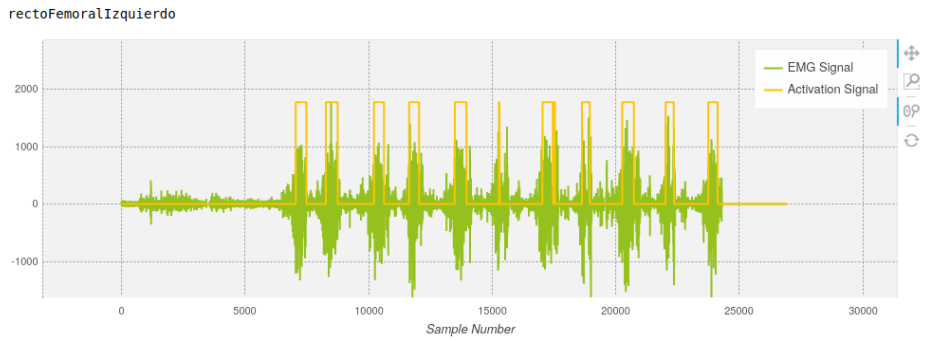

Figura A.2: s3 serie 2 día 2: ejemplo de activación muscular

A.2. Procesamiento de datos: diferentes modelos de estudio

Canal 1, canal 2, canal 3, canal 4

Los archivos de entrenamiento-testeo obtenidos de cada canal tienen el mismo formato, solo cambia el valor de de las características obtenidas. Debido a ello solo se muestra el archivo del canal 1.

Out[39]:

	RMS1	MAV1	ZC1	WL1	SSC1	IEMG1	SSI1	VAR1	TM31	TM41	TM51	LOG1	ACC1	MNF1	MDF1
0	0.112904	0.119980	0.247642	0.121713	0.207703	0.104327	0.044137	0.054549	0.018561	0.010294	0.004709	0.143275	0.185761	0.493776	0.053499
1	0.079236	0.104212	0.056604	0.024804	0.027510	0.018199	0.007668	0.036963	0.006146	0.003252	0.000325	0.144535	0.208452	0.725863	0.011582
2	0.073628	0.080973	0.290094	0.127078	0.246217	0.094894	0.036376	0.033778	0.024842	0.005307	0.003234	0.093598	0.203427	0.722072	0.044682
3	0.088639	0.109129	0.167453	0.085743	0.177442	0.072766	0.028379	0.041559	0.018394	0.004415	0.001493	0.146326	0.174129	0.543155	0.028772
4	0.099598	0.104005	0.094340	0.046944	0.078404	0.039997	0.017703	0.047487	0.024457	0.006891	0.002948	0.069104	0.162880	0.474334	0.031135
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
235	0.854534	0.854469	0.158019	0.250640	0.138927	0.305169	0.332004	0.782663	0.040341	0.568737	0.040763	0.858933	0.799262	0.184372	0.388845
236	0.788662	0.777505	0.117925	0.177800	0.103164	0.218261	0.231434	0.692490	0.300248	0.451391	0.212561	0.651600	0.718958	0.262807	0.223235
237	0.898756	0.908304	0.160377	0.269131	0.144429	0.356641	0.396028	0.846101	0.011845	0.656083	0.015448	0.856327	0.750852	0.157761	0.362445
238	0.869743	0.778255	0.141509	0.212002	0.122421	0.262660	0.316159	0.804372	0.207728	0.760044	0.259495	0.525305	0.701828	0.105706	0.392913
239	1.000000	1.000000	0.169811	0.310141	0.147180	0.437335	0.524393	1.000000	0.021927	0.919953	0.134296	0.932815	0.760332	0.057114	0.571890

240 rows × 15 columns

Figura A.3: Archivo entrenamiento-testeo del canal 1

Canal 1,2

Out[11]:

	RMS	MAV	ZC	WL	SSC	IEMG	SSI	VAR	TM3	TM4	TM5	LOG	ACC	MNF	MDF	fat
0	0.112904	0.119980	0.295806	0.141514	0.259640	0.121682	0.048013	0.054549	0.016006	0.010294	0.004932	0.308502	0.185761	0.493776	0.056086	
1	0.079236	0.104212	0.116998	0.046790	0.091260	0.037224	0.011692	0.036963	0.005408	0.003252	0.000549	0.309520	0.208452	0.725863	0.014283	
2	0.073628	0.080973	0.335541	0.146758	0.295630	0.112433	0.040283	0.033778	0.021368	0.005307	0.003457	0.268406	0.203427	0.722072	0.047292	
3	0.088639	0.109129	0.220751	0.106355	0.231362	0.090733	0.032319	0.041559	0.015864	0.004415	0.001717	0.310965	0.174129	0.543155	0.031426	
4	0.099598	0.104005	0.152318	0.068430	0.138817	0.058599	0.021687	0.047487	0.021040	0.006891	0.003172	0.248636	0.162880	0.474334	0.033782	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
475	0.124753	0.143458	0.011038	0.002248	0.021851	0.004071	0.000000	0.064083	0.057323	0.008325	0.005821	0.336262	0.026267	0.345451	0.000000	
476	0.513304	0.499520	0.205298	0.199163	0.164524	0.170523	0.131001	0.368607	0.244827	0.171450	0.133236	0.552985	0.766456	0.544460	0.155227	
477	0.429387	0.392172	0.026490	0.020952	0.034704	0.021705	0.015476	0.290841	0.126997	0.124772	0.031138	0.523513	0.434119	0.552650	0.009572	
478	0.419884	0.423284	0.090508	0.067569	0.077121	0.063333	0.042269	0.279649	0.121981	0.101222	0.052792	0.541689	0.553101	0.358414	0.048824	
479	0.604293	0.623040	0.161148	0.178169	0.170951	0.172918	0.141591	0.466047	0.175519	0.221381	0.131432	0.746553	0.784452	0.485917	0.146301	

480 rows × 16 columns

Figura A.4: Archivo entrenamiento-testeo del canal 1,2

Canal 3,4

Out[12]:

	RMS	MAV	ZC	WL	SSC	IEMG	SSI	VAR	TM3	TM4	TM5	LOG	ACC	MNF	MDF	fat
0	0.331273	0.344806	0.207977	0.161583	0.270677	0.175932	0.114412	0.190592	0.023537	0.046408	0.009183	0.454697	0.276464	0.262604	0.093717	
1	0.241065	0.261196	0.222222	0.141657	0.268797	0.130809	0.068317	0.124890	0.017248	0.020797	0.004694	0.378979	0.272934	0.434877	0.055972	
2	0.270855	0.279847	0.245014	0.161888	0.321429	0.155328	0.090005	0.145416	0.022373	0.031113	0.001773	0.399695	0.273911	0.232794	0.087140	
3	0.276080	0.300448	0.242165	0.164623	0.304511	0.168654	0.095887	0.149127	0.023577	0.024510	0.005880	0.415551	0.262009	0.282946	0.082755	
4	0.261665	0.275218	0.207977	0.138212	0.246241	0.129222	0.071947	0.139008	0.003083	0.027801	0.001329	0.390622	0.283110	0.361391	0.077937	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
475	0.671051	0.628867	0.219373	0.232775	0.178571	0.198336	0.224703	0.531451	0.357004	0.400919	0.581064	0.607766	0.783246	0.472221	0.216261	
476	0.550686	0.518235	0.162393	0.168298	0.152256	0.134332	0.132853	0.394202	0.034637	0.215837	0.092755	0.575491	0.697124	0.639918	0.138670	
477	0.543528	0.537362	0.222222	0.206691	0.208647	0.158028	0.148193	0.386458	0.001096	0.179664	0.032974	0.563097	0.765956	0.644383	0.136085	
478	0.610175	0.573554	0.165242	0.165289	0.154135	0.140451	0.149999	0.459941	0.067152	0.265266	0.072210	0.599573	0.713842	0.443529	0.155575	
479	0.769511	0.742815	0.150997	0.235096	0.146617	0.244261	0.297741	0.657521	0.232336	0.475471	0.384439	0.631202	0.720911	0.365085	0.294815	

480 rows × 16 columns

Figura A.5: Archivo entrenamiento-testeo del canal 3,4

Canal Total

Out[14]:

	RMS	MAV	ZC	WL	SSC	IEMG	SSI	VAR	TM3	TM4	TM5	LOG	
0	202.308487	156.700907	135.0	66138.359	204.0	103109.197	2.693110e+07	40991.020428	1.782625e+06	6.211790e+09	1.273148e+12	101.339219	100.51
1	187.983354	151.458846	54.0	23542.215	73.0	34381.158	8.021667e+06	35494.103150	6.207434e+05	3.392125e+09	1.427202e+11	101.603315	103.71
2	185.597510	143.732785	153.0	68496.631	232.0	95582.302	2.290688e+07	34498.313006	2.370453e+06	4.215049e+09	8.927286e+11	90.931057	103.00
3	191.984262	153.093318	101.0	50327.844	182.0	77924.499	1.876070e+07	36930.511700	1.767045e+06	3.858039e+09	4.439128e+11	101.978580	98.87
4	196.647206	151.389749	70.0	33273.709	110.0	51775.294	1.322518e+07	38783.525622	2.334464e+06	4.849117e+09	8.191247e+11	85.799017	97.29
...	...	...	...	...	...	...	...	...	...	...	...	...	...
43	479.242567	360.915802	87.0	92865.279	112.0	144366.321	9.186938e+07	230249.060261	6.129332e+07	2.297822e+11	1.850249e+14	210.348044	232.16
44	419.025595	316.267315	67.0	70150.079	98.0	103419.412	5.741546e+07	176121.046145	5.974516e+06	1.245755e+11	2.956703e+13	200.590448	214.52
45	415.444551	323.986423	88.0	83675.795	128.0	118579.031	6.316947e+07	173067.035538	2.188062e+05	1.040140e+11	1.053512e+13	196.843188	228.62
46	448.787073	338.593009	68.0	69090.133	99.0	107333.984	6.384692e+07	202047.209981	1.155405e+07	1.526728e+11	2.302610e+13	207.871167	217.94
47	528.500643	406.902735	63.0	93682.797	95.0	173747.468	1.192666e+08	279968.593364	3.990004e+07	2.721598e+11	1.224275e+14	217.433690	219.39

960 rows × 16 columns

Figura A.6: Archivo entrenamiento-testeo del canal total

A.3. Selección de características: RFE

Tabla donde se muestran las características escogidas por el algoritmo RFE para el diseño de los clasificadores.

Cuadro A.1: Características seleccionadas para cada canal

	N = 1	N = 2	N = 5
canal 1	IEMG	WL, IEMG	ZC, WL, IEMG, TM4, ACC
canal 2	IEMG	IEMG, MDF	ZC, WL, IEMG, VAR, MDF
canal 3	RMS	RMS, TM4	RMS, ZC, TM4, LOG, ACC
canal 4	MDF	SSI, MDF	ZC, SSI, TM3, TM4, MDF
canal total	IEMG	IEMG, SSI	RMS, WL, SSC, IEMG, SSI
canal 1,2	RMS	RMS, VAR	RMS, IEMG, VAR, TM4, TM5
canal 3,4	RMS	RMS, VAR	RMS, MAV, SSI, VAR, MDF

A.4. Extracción de características

A.4.1. Implementación

A continuación el listado de todas las características implementadas junto con el código fuente correspondiente.

Con respecto a las características del dominio del tiempo:

- Root mean square (RMS)

```
def RMS(datos,ratio):

    datosDevueltos=np.array([])

    if (ratio==0):
        return None

    i=0
    while i < datos.size:

        ratioN=0
        j=i
        i=i+int(ratio)
        sumaLocal=0
        #Para cada i , recorro sus "ratio" siguientes
        while j < i and j < datos.size:
            sumaLocal=sumaLocal+(datos[j]*datos[j])
            j=j+1
            ratioN=ratioN+1


        sumaTotal=sumaLocal/ratioN
        resultado=math.sqrt(sumaTotal)

        datosDevueltos=np.append(datosDevueltos,resultado)




    return range(datosDevueltos.size),datosDevueltos
```

- Mean absolute value (MAV)

```
def MAV(datos,ratio):


    datosDevueltos = np.array([])

    i=0
    while i < datos.size:

        ratioN=0
        j=i
        i=i+int(ratio)
        sumaLocal=0
        #Para cada i , recorro sus "ratio" siguientes
        while j < i and j < datos.size:
            sumaLocal=sumaLocal+abs(datos[j])
            j=j+1
            ratioN=ratioN+1


        datosDevueltos=np.append(datosDevueltos,sumaLocal/ratioN)

    return range(datosDevueltos.size),datosDevueltos
```

- Zero crossing (ZC) con umbral de 0,01

```
def ZC(datos,ratio,voltajeUmbral=0.01):


    datosDevueltos=np.array([])
    i=0
    while i < datos.size:

        j=i
        contadorLocal=0
        i+=int(ratio)

        while j < i-1 and j< datos.size-1:

            if (((datos[j] > 0 and datos[j+1]<0) or (datos[j]<0
                and datos[j+1]>0)) and (abs(datos[j]-datos[j+1])
                >= voltajeUmbral)):
                contadorLocal+=1

            j+=1


        #if (contadorLocal != 0):
        datosDevueltos=np.append(datosDevueltos,contadorLocal)



    return range(datosDevueltos.size),datosDevueltos
```

- Waveform length (WL)

```
def WL(datos,ratio):

    datosDevueltos=np.array([])
    i=0

    while i < datos.size:

        j=i
        sumaLocal=0
        i+=int(ratio)

        while j < i-1 and j< datos.size-1:
            sumaLocal=sumaLocal+abs(datos[j+1]-datos[j])
            j+=1



        datosDevueltos=np.append(datosDevueltos,sumaLocal)

    return range(datosDevueltos.size),datosDevueltos
```

- Slope sign change (SSC) con umbral de 0,01

```
def SSC(datos,ratio,voltajeUmbral=0.01):

    datosDevueltos=np.array([])
    #Dado que compara x(i) con x(i-1) y x(i+1)
    i=1
    while i < datos.size:

        j=i
        contadorLocal=0
        i+=int(ratio)
        #-2 dado que i comienza en 1
        while j < i-2 and j< datos.size-1:
            if(((datos[j]> datos[j-1] and datos[j]>datos[j+1]) or
                 (datos[j] < datos[j-1] and datos[j]<datos[j+1] )
                ) and (abs(datos[j]-datos[j+1])>= voltajeUmbral
                or abs(datos[j]-datos[j-1])>=voltajeUmbral)):
                contadorLocal+=1

            j+=1


        #if (contadorLocal != 0):
        datosDevueltos=np.append(datosDevueltos,contadorLocal)



    return range(datosDevueltos.size),datosDevueltos
```

- Integrated EMG (IEMG)

```
def IEMG(datos,ratio):

    datosDevueltos = np.array([])

    i=0
    while i < datos.size:

        j=i
        i=i+int(ratio)
        sumaLocal=0
        #Para cada i , recorro sus "ratio" siguientes
        while j < i and j < datos.size:
            sumaLocal=sumaLocal+abs(datos[j])
            j=j+1


        datosDevueltos=np.append(datosDevueltos,sumaLocal)

    return range(datosDevueltos.size),datosDevueltos
```

- Simple square integral (SSI)

```
def SSI(datos,ratio):

    datosDevueltos = np.array([])

    i=0
    while i < datos.size:

        j=i
        i=i+int(ratio)
        sumaLocal=0
        #Para cada i , recorro sus "ratio" siguientes
        while j < i and j < datos.size:
            sumaLocal=sumaLocal+(datos[j]*datos[j])
            j=j+1


        datosDevueltos=np.append(datosDevueltos,sumaLocal)

    return range(datosDevueltos.size),datosDevueltos
```

- Variance of EMG (VAR)

```
def VAR(datos,ratio):

    datosDevueltos=np.array([])

    i=0
    while i < datos.size:

        ratioN=0
        j=i
        i=i+int(ratio)
        sumaLocal=0
        #Para cada i , recorro sus "ratio" siguientes
        while j < i and j < datos.size:
            sumaLocal=sumaLocal+(datos[j]*datos[j])
            j=j+1
            ratioN=ratioN+1


        if (ratioN == 1):
            ratioN=2

        resultado=sumaLocal/(ratioN-1)

        datosDevueltos=np.append(datosDevueltos,resultado)




    return range(datosDevueltos.size),datosDevueltos
```

- Absolute value 3rd (TM3)

```
def TM3(datos, ratio):

    datosDevueltos=np.array([])


    i=0
    while i < datos.size:
        ratioN=0
        j=i
        i=i+int(ratio)
        sumaLocal=0
        while j< i and j< datos.size:
            sumaLocal=sumaLocal+pow(datos[j],3)
            j=j+1
            ratioN=ratioN+1

        resultado=abs(sumaLocal/ratioN)

        datosDevueltos=np.append(datosDevueltos,resultado)

    return range(datosDevueltos.size),datosDevueltos
```

- Absolute value 4th (TM4)

```
def TM4(datos, ratio):

    datosDevueltos=np.array([])


    i=0
    while i < datos.size:
        ratioN=0
        j=i
        i=i+int(ratio)
        sumaLocal=0
        while j< i and j< datos.size:
            sumaLocal=sumaLocal+pow(datos[j],4)
            j=j+1
            ratioN=ratioN+1

        resultado=sumaLocal/ratioN

        datosDevueltos=np.append(datosDevueltos,resultado)

    return range(datosDevueltos.size),datosDevueltos
```

- Absolute value 5th (TM5)

```
def TM5(datos, ratio):

    datosDevueltos=np.array([])


    i=0
    while i < datos.size:
        ratioN=0
        j=i
        i=i+int(ratio)
        sumaLocal=0
        while j< i and j< datos.size:
            sumaLocal=sumaLocal+pow(datos[j],5)
            j=j+1
            ratioN=ratioN+1

        resultado=abs(sumaLocal/ratioN)

        datosDevueltos=np.append(datosDevueltos,resultado)

    return range(datosDevueltos.size),datosDevueltos
```

- Log detector (LOG)

```
def LOG(datos,ratio):

    datosDevueltos=np.array([])

    i=0
    while i < datos.size:
        ratioN=0
        j=i
        i=i+int(ratio)
        sumaLocal=0
        while j< i and j< datos.size:
            if(datos[j] != 0):
                sumaLocal=sumaLocal+math.log(abs(datos[j]))
            j=j+1
            ratioN=ratioN+1

        resultado=math.exp(sumaLocal/ratioN)

        datosDevueltos=np.append(datosDevueltos,resultado)

    return range(datosDevueltos.size),datosDevueltos
```

- Average amplitude change (ACC)

```
def ACC(datos,ratio):

    datosDevueltos=np.array([])
    i=0

    while i < datos.size:

        j=i
        sumaLocal=0
        ratioN=1
        i+=int(ratio)

        while j < i-1 and j< datos.size-1:
            sumaLocal=sumaLocal+abs(datos[j+1]-datos[j])
            j+=1
            ratioN+=1



        datosDevueltos=np.append(datosDevueltos,sumaLocal/ratioN)

    return range(datosDevueltos.size),datosDevueltos
```

Con respecto al dominio de la frecuencia :

- Median frecuency (MDF)

```
def MDF(datos,ratio,fs=1024.0):

    datosDevueltos=np.array([])
    lista=[]
    datosProcesados=[]

    if (ratio==0):
        return None


    i=0
    while i< len(datos):

        lista.append(datos[i])
        if (len(lista)==ratio or i == len(datos)-1):
            frecuencia,poder = signal.welch(lista,fs,nperseg=len(
                lista))
            #Lista de vectores
            datosProcesados.append(poder)
            lista.clear()
        i=i+1


    for seccion in datosProcesados:
        suma=seccion.sum()/2
        datosDevueltos=np.append(datosDevueltos,suma)

    return range(datosDevueltos.size),datosDevueltos
```

- Mean frecuency (MNF)

```
def MNF(datos,ratio,fs=1024.0):

    datosDevueltos=np.array([])
    lista=[]
    datosPod=[]
    datosFre=[]

    if (ratio==0):
        return None


    i=0
    while i< len(datos):

        lista.append(datos[i])
        if (len(lista)==ratio or i == len(datos)-1):
            frecuencia,poder = signal.welch(lista,fs,nperseg=len(
                lista))
            datosPod.append(poder)
            datosFre.append(frecuencia)
            lista.clear()
        i=i+1


```

```
24      suma=[]
25
26      for i,j in zip(datosPod, datosFre):
27          sumaNumerador=0
28          sumaDenominador=0
29          for datosP,datosF in zip(i,j):
30              sumaNumerador=sumaNumerador+(datosP*datosF)
31              sumaDenominador=sumaDenominador+datosP
32
33
34          datosDevueltos=np.append(datosDevueltos,sumaNumerador/
                sumaDenominador)
35
36
37      return range(datosDevueltos.size),datosDevueltos
```

A.4.2. Evaluación

Lista de gráficas de ejemplo para demostrar el funcionamiento de la extracción de características. Se ha tomado la muestra del sujeto número 16, en la serie 3 del día 1. A continuación las gráficas relativas a cada una de las características implementadas:

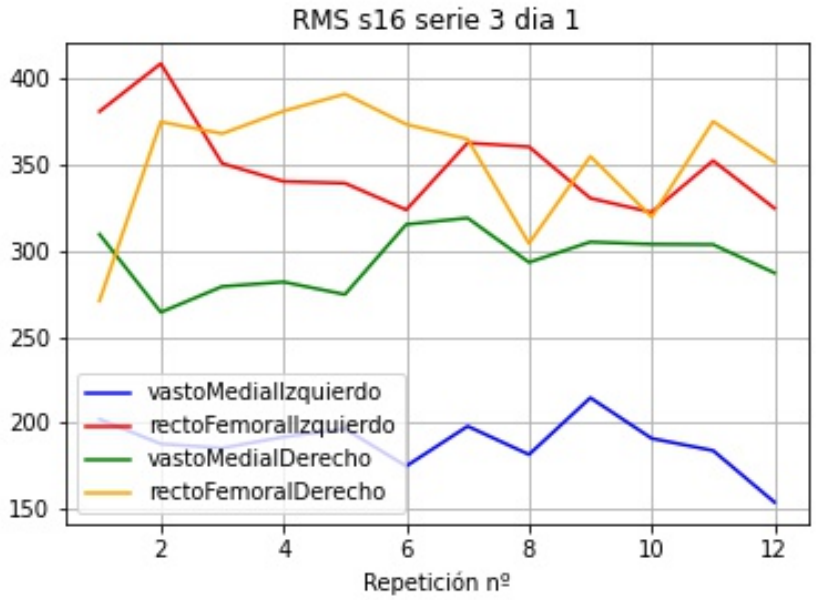

Figura A.7: RMS

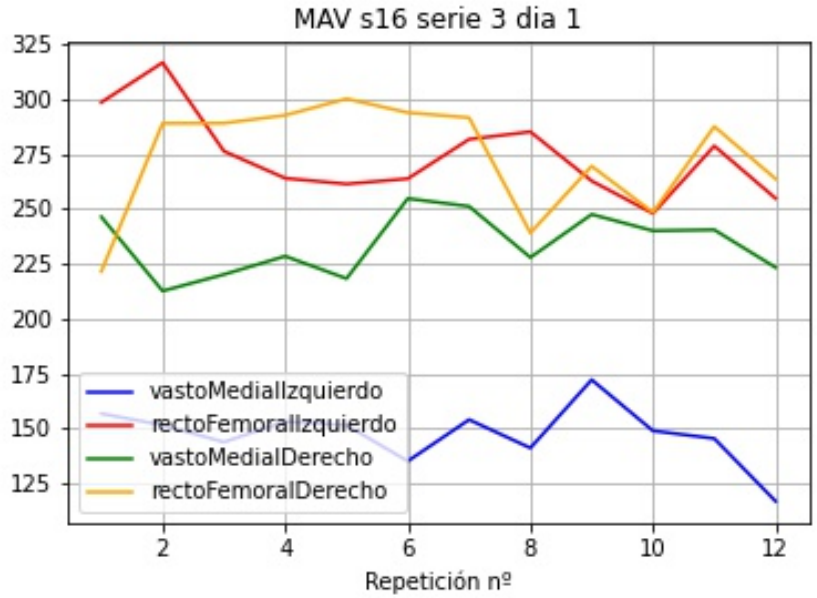

Figura A.8: MAV

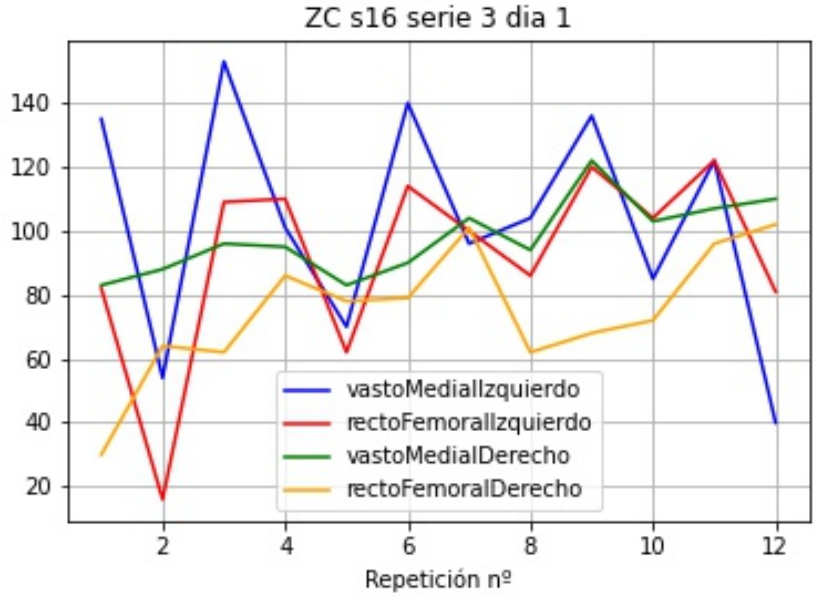

Figura A.9: ZC con umbral de 0,01

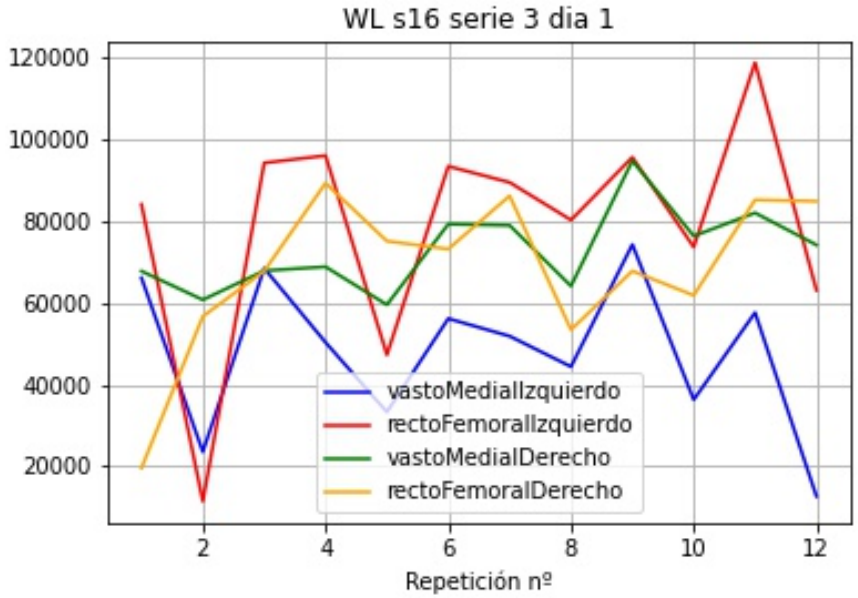

Figura A.10: WL

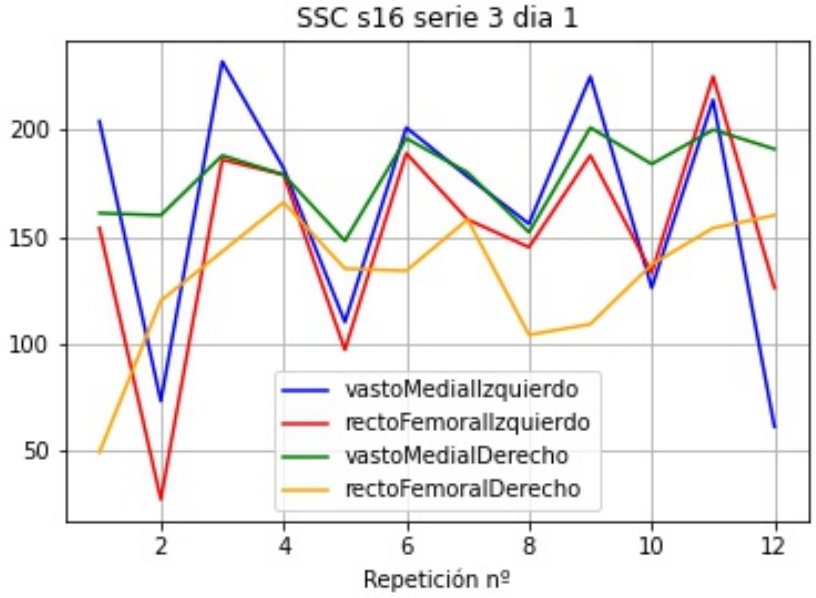

Figura A.11: SSC con umbral de 0,01

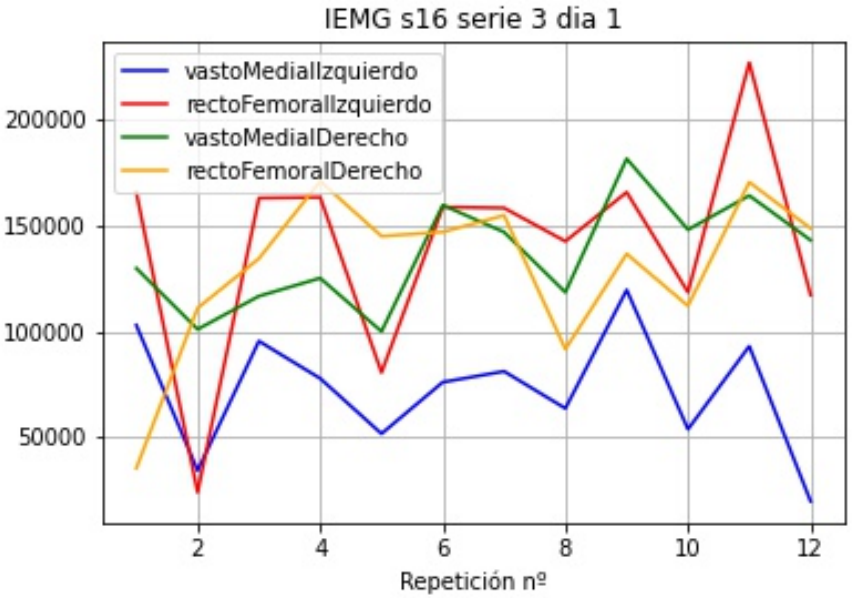

Figura A.12: IEMG

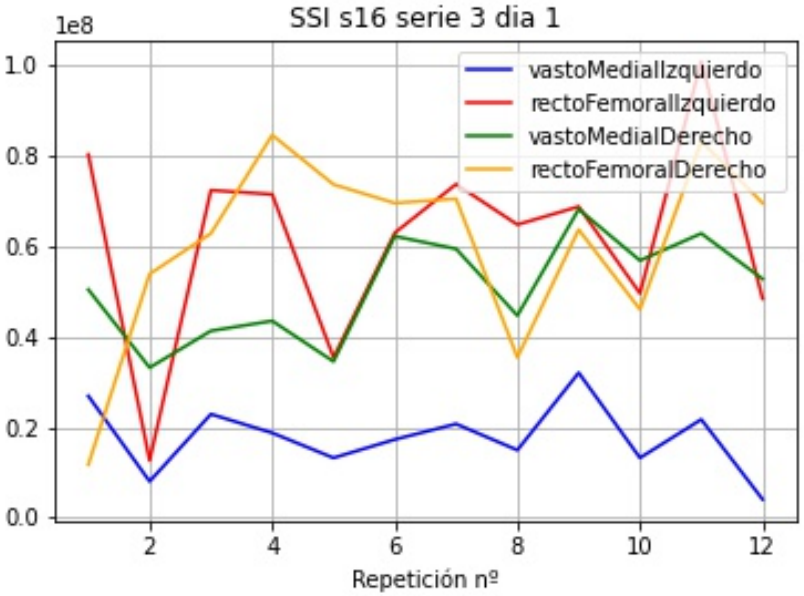

Figura A.13: SSI

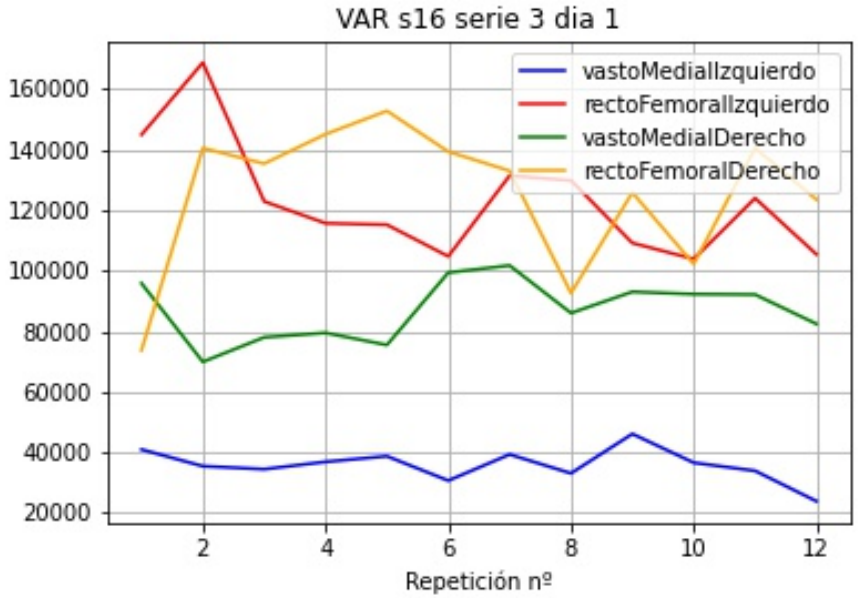

Figura A.14: VAR

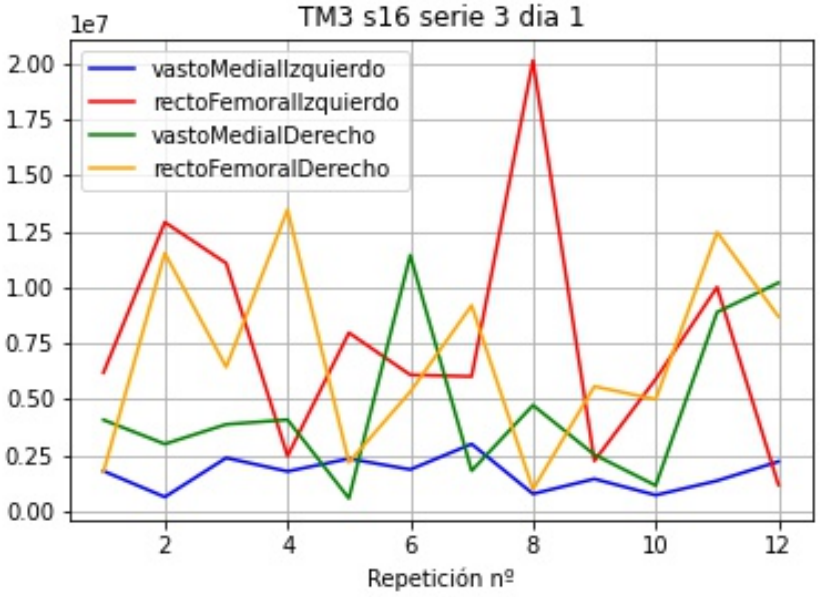

Figura A.15: TM3

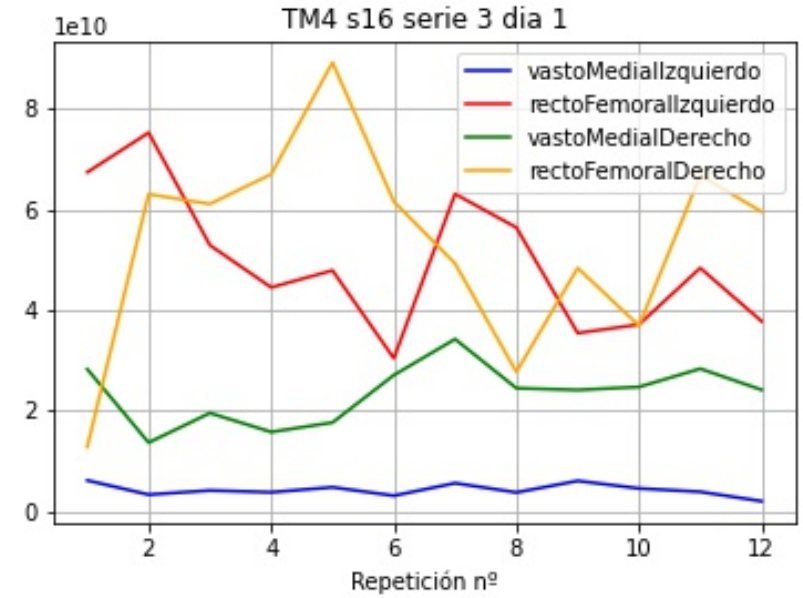

Figura A.16: TM4

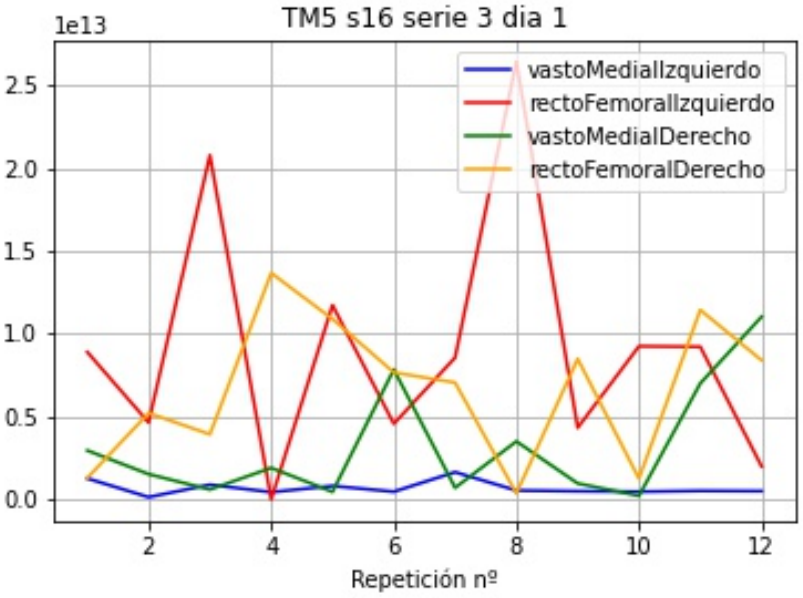

Figura A.17: TM5

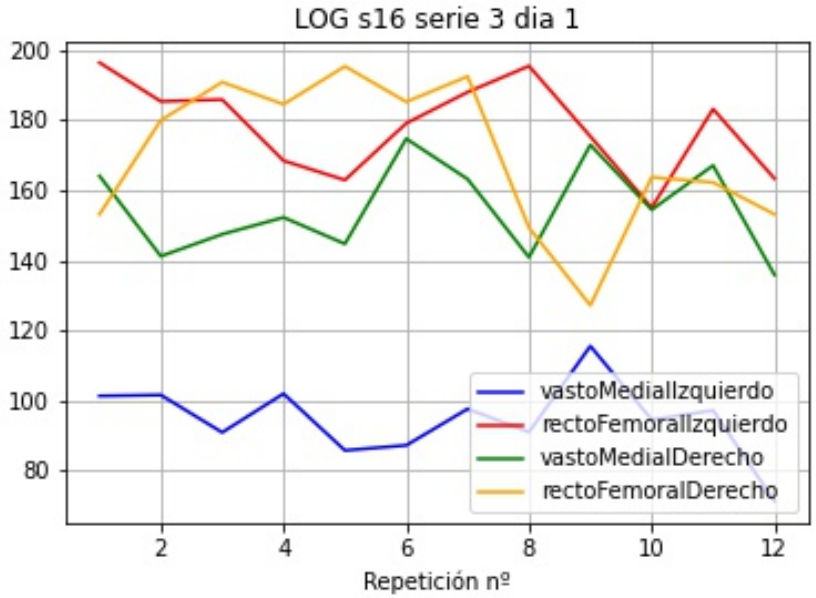

Figura A.18: LOG

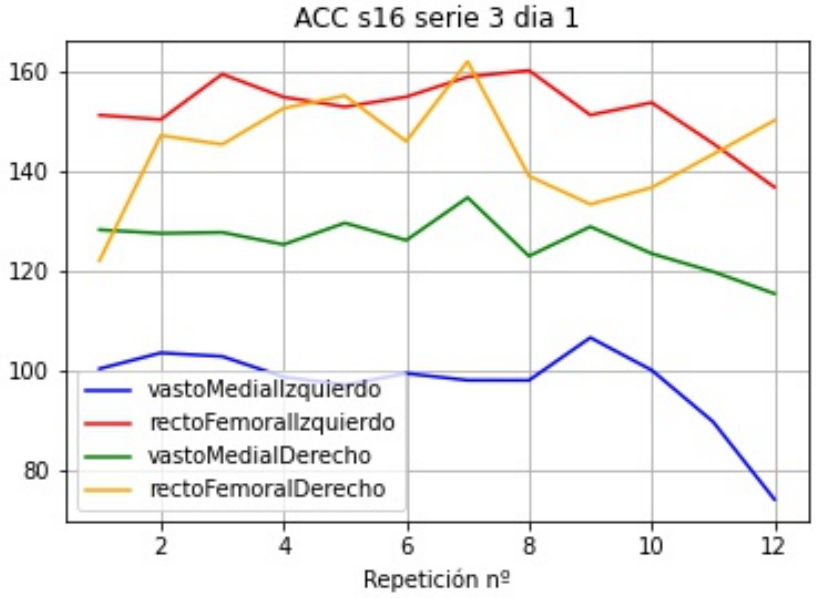

Figura A.19: ACC

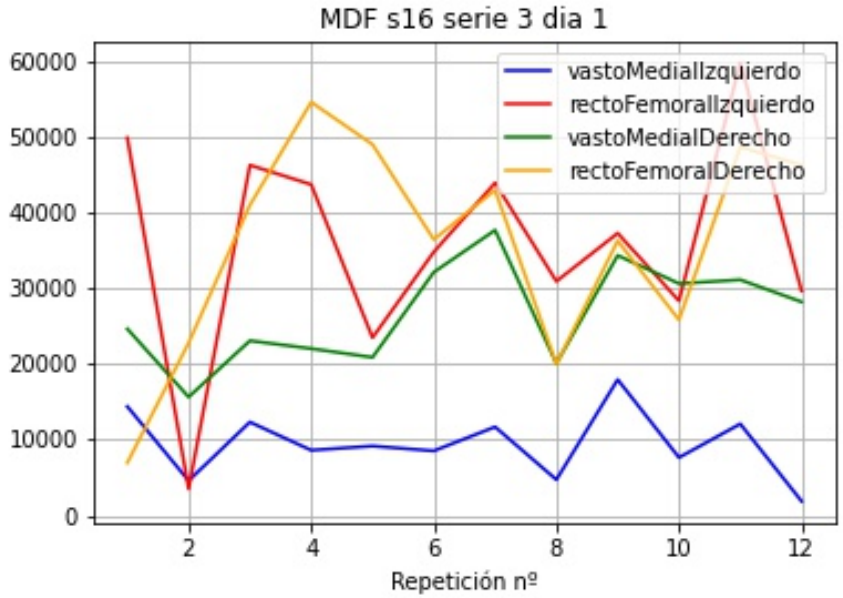

Figura A.20: MDF

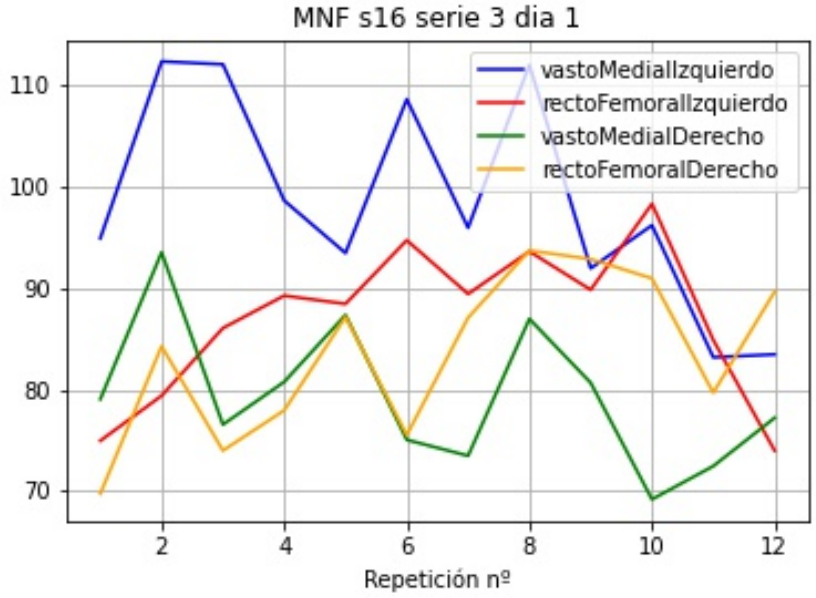

Figura A.21: MNF