Trabajo Fin de Grado
en Ingeniería de las Tecnologías de Telecomunicación

# Procesado del EEG y técnicas de machine learning para la detección de pacientes con esclerosis múltiple

Autor: Lucía Balsa Picado

Tutor: Rubén Martín Clemente

**Dpto. Teoría de la Señal y Comunicaciones**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2024

Trabajo Fin de Grado
en Ingeniería de las Tecnologías de Telecomunicación

# Procesado del EEG y técnicas de machine learning para la detección de pacientes con esclerosis múltiple

Autor:
Lucía Balsa Picado

Tutor:
Rubén Martín Clemente
Catedrático

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Procesado del EEG y técnicas de machine learning para la detección de pacientes con esclerosis múltiple

Autor: Lucía Balsa Picado

Tutor: Rubén Martín Clemente

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

*A mi familia*

# Agradecimientos

En primer lugar, querría agradecer a Manuel Vázquez Marrufo y Rocío Caballero Díaz del Departamento de Psicología Experimental de la Facultad de Psicología de la US —gracias, Manuel, por la implicación y el entusiasmo de tus explicaciones sobre el proyecto aquel primer día en el laboratorio; y gracias, Rocío, por aclararme duda, tras duda, *tras duda*, siempre con tanta paciencia y consideración.

También querría agradecer a Rubén Martín Clemente del Departamento de Teoría de la Señal y Comunicaciones, mi tutor durante este trabajo, por presentarme la oportunidad de formar parte del proyecto y de aprender un poquito sobre el área de la psicofisiología y algo más sobre el procesado digital de señal.

Aprovecho además para dar las gracias a todo el profesorado que me ha dado clases durante estos años, en especial, a quienes han hecho de sus asignaturas algo enriquecedor y memorable a través de una docencia entregada y dispuesta.

Por último, necesito agradecer a mi familia —gracias por los ánimos, por los toques de atención, por estar ahí, por haber(me) aguantado; gracias por todo.

*Lucía Balsa Picado*

*Sevilla, 2024*

# Resumen

La esclerosis múltiple (EM) es una enfermedad autoinmune que provoca una degeneración de la transmisión de los impulsos nerviosos, lo que ocasiona al paciente un amplio abanico de síntomas relacionados con la falta de coordinación o pérdida de control muscular, pero también con la capacidad cognitiva. Es en esta última línea en la que se propone contribuir con este proyecto, y para ello se ha planteado el diseño de un 'ojo clínico' que proporcione una detección fiable de casos de EM en base a una serie de características de la actividad cerebral del sujeto sometido a evaluación.

A partir del electroencefalograma (EEG) registrado durante una prueba de carácter atencional, se construye un protocolo de procesado de señal para extraer un número de parámetros que pueden aportar información sobre el estado de determinados mecanismos cognitivos. Estas características pasan a conformar la base de datos necesaria para generar un modelo de aprendizaje automático o *machine learning* (ML) capacitado para discernir entre sujetos con EM y sin ella con una precisión significativa.

# Abstract

Multiple sclerosis (MS) is an autoimmune disease that causes a decline of nerve impulse transmission, which in turn results in a wide range of symptoms that circle back to a lack of coordination or loss of muscle control, but to cognitive ability as well. It is to the latter that this project aims to contribute, and thus it proposes the implementation of a tool capable of reliable detection of MS cases making use of a series of features obtained from the brain activity of the subject being evaluated.

Starting with the electroencephalogram (EEG) recorded during an attention test, the study builds a signal processing protocol in order to extract a set of parameters that guarantees information on the state of certain cognitive mechanisms. These features become the database that is required to generate a machine learning model (ML) that is able to discriminate with significant accuracy between subjects with and without MS.

# ÍNDICE ABREVIADO

# Índice

# ÍNDICE DE FIGURAS

# ÍNDICE DE ABREVIATURAS

# 1. INTRODUCCIÓN

## 1.1. Motivación

La esclerosis múltiple (EM) es una enfermedad autoinmune que provoca el decaimiento de las vainas de mielina que recubren los axones de las neuronas [1]. En un sistema nervioso no afectado por EM, la forma que tiene la mielina de recubrir los cuerpos de las células nerviosas puede modelarse de forma simplificada como una línea de transmisión de parámetros distribuidos [2], estableciendo un símil estructural con un cable coaxial donde el axón sería el conductor interior, la mielina constituiría el dieléctrico y el fluido intercelular asumiría el papel de conductor externo [3]. La desaparición de mielina que sufren los pacientes con EM se interpretaría como un aumento de la distorsión, de manera que los receptores presentan más complicaciones para interpretar la señal.

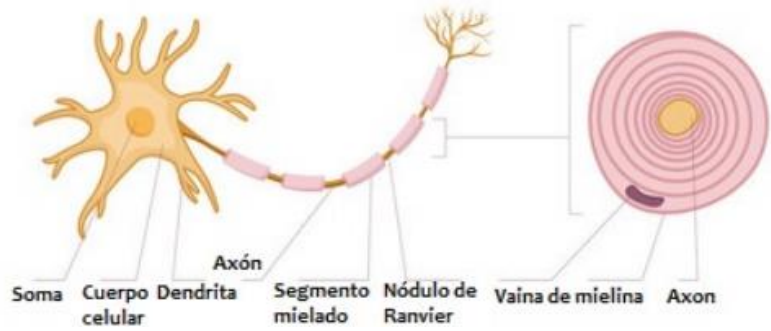


**Figura 1.1.** Estructura de una neurona y corte transversal del axón [2]

Esta sería entonces la explicación para la falta de coordinación o pérdida de control muscular, ampliamente estudiada. Sin embargo, aunque más obviado hasta hace relativamente poco, la EM también es responsable del deterioro cognitivo en un vasto número de pacientes. Así pues, lo que se busca con este trabajo es conocer los efectos de esta patología sobre los procesos cognitivos y explotar dicha información para la construcción de un modelo de ML que sea capaz de distinguir entre sujetos con y sin EM.

## 1.2. Objetivos

Los objetivos de este proyecto se pueden concretar de forma más detallada como sigue:

- Análisis de las consecuencias de la EM en mecanismos cognitivos.
- Implementación de técnicas de procesado de señal tradicional con aplicación al EEG.
- Utilización de técnicas de ML para el desarrollo de una máquina de soporte vectorial o *support vector machine* (SVM).

# 2. EL EEG

## 2.1. Introducción al EEG

El electroencefalograma (EEG) es el registro de la actividad eléctrica generada en el cerebro que se recoge a través de unos electrodos colocados sobre la cabeza que están conectados a un electroencefalógrafo.

**Figura 2.1.** Casco EEG [4]

Uno de los fenómenos más estudiados en el EEG es lo que se conoce como potencial evocado o respuesta evocada, y consiste en el promediado de la actividad eléctrica que se genera en el cerebro ante la aparición de un tipo de estímulo. La suposición con la que se trabaja es que números muy elevados de neuronas se sincronizan para procesar los eventos percibidos, y es por esta coordinación entre células nerviosas (*i. e.*, las señales oscilan en coherencia de fase) que la respuesta evocada también recibe el nombre de respuesta en fase.

Se entiende así que los instantes de interés para el análisis de los procesos cognitivos se dan inmediatamente después de la aparición de estímulos. Es por ello que este estudio se centra en la caracterización de esos intervalos temporales mediante distintos parámetros.

Concretamente, dos de las variables que parecen verse alteradas por la EM son la amplitud y la latencia de dicha respuesta evocada —en pacientes de EM, las amplitudes suelen ser menores y las latencias, más largas— [5][6]. Sin embargo, también podrían existir mecanismos subyacentes que afectaran a otras métricas menos revisadas en la literatura, por lo que también se explorarán en este trabajo.

# 3. TÉCNICAS EMPLEADAS

En este capítulo se explican de forma más minuciosa algunas de las técnicas que se comentarán en secciones posteriores.

## 3.1. ICA

ICA (*Independent Component Analysis*) forma parte de la familia de métodos de separación de fuentes ciega, que se caracterizan por descomponer la señal recibida en cada elemento de la matriz de receptores (esto es, en cada electrodo) en una combinación lineal de distintas fuentes estadísticamente independientes. De esta manera, es posible identificar una de estas fuentes si se tiene una referencia de esta con la que comparar.

En el caso concreto del apartado *4.3.1.1. Corrección de los artefactos oculares*, la fuente de interés sería la componente ocular que contamina el EEG, y puede identificarse comparando todas las fuentes encontradas por ICA con la diferencia de los electrodos Veog+ y Veog_ (que constituyen una referencia bipolar de la actividad ocular), y así eliminar dicha componente de la señal en todo electrodo que la presentara.

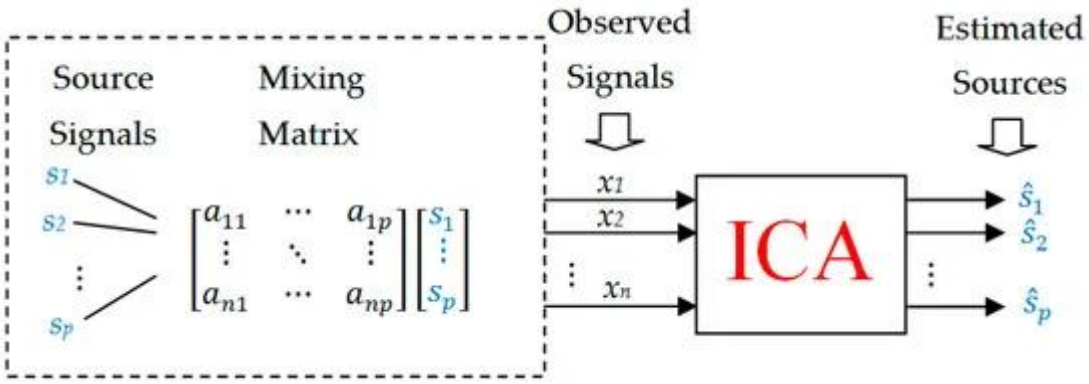


**Figura 3.1.** Diagrama de bloques de ICA [7]

Atendiendo a la notación presentada en la *Figura 3.1*, el objetivo de ICA es que el vector de fuentes estimadas $\hat{\boldsymbol{s}}$ se parezca lo máximo posible al vector de fuentes $\boldsymbol{s}$, es decir, $\hat{\boldsymbol{s}} \approx \boldsymbol{s}$.

Por otro lado, en ese modelo de caja negra de la imagen anterior, ICA se puede considerar una transformación $\boldsymbol{W}$ que convierte el vector de observación $\boldsymbol{x}$ al vector de fuentes estimadas $\hat{\boldsymbol{s}}$, es decir, $\hat{\boldsymbol{s}} = \boldsymbol{W}\boldsymbol{x}$.

Con todo, se tiene que $\hat{\boldsymbol{s}} = \boldsymbol{W}\boldsymbol{x} \approx \boldsymbol{s}$ ; $\boldsymbol{W}\boldsymbol{x} \approx \boldsymbol{s}$. Luego el problema radica en encontrar la matriz $\boldsymbol{W}$ que al multiplicar por las señales registradas en los electrodos $\boldsymbol{x}$, resulte lo más cercana posible a las fuentes independientes.

Así, la forma que tiene ICA de separar las distintas componentes se reduce a la maximización de la 'no gaussianidad' o independencia de las señales recibidas en los electrodos. Esto se explica al considerar el teorema central del límite, que establece que la suma de variables aleatorias independientes se aproxima a una gaussiana cuando el número de variables tiende a infinito. Se deduce pues, que cualquier superposición de componentes de la naturaleza mencionada se parecerá más a una gaussiana que cualquiera de sus componentes por separado; esto es, la combinación lineal de fuentes que se mide en los electrodos es más gaussiana que cada una de dichas fuentes, y como lo que se busca es aislar estas últimas (que equivale a deshacer la suma de variables independientes que tiende a la gaussiana), se deduce que el algoritmo ICA se puede entender como un intento de invertir el teorema del límite.

Por consiguiente, la implementación de ICA se consigue con un algoritmo de optimización —son frecuentes infomax y FastICA— que minimice la gaussianidad —cuya cuantificación puede realizarse mediante distintas métricas, como la curtosis (*apartado 4.3.2.4. Curtosis*)— de la estimación de las fuentes. Una

posibilidad sería por tanto: max( curtosis( $\hat{\boldsymbol{s}}$ ) ) = max( curtosis( $\boldsymbol{Wx}$ ) ), siendo los elementos de $\boldsymbol{W}$ las variables de la función objetivo.

## 3.2. SVM

La máquina de soporte vectorial o *support* vector *machine* (SVM) es un tipo de algoritmo supervisado de ML que construye uno o más hiperplanos para separar las distintas categorías de la base de datos [7]. El criterio que utiliza para seleccionar uno de entre los múltiples hiperplanos o fronteras de decisión posibles es la maximización del margen, o lo que es lo mismo, la maximización de la distancia a las muestras más cercanas de cualquiera de las clases. Estos valores que se emplean para maximizar el margen son lo que se denominan vectores de soporte o *suppor vectors*.

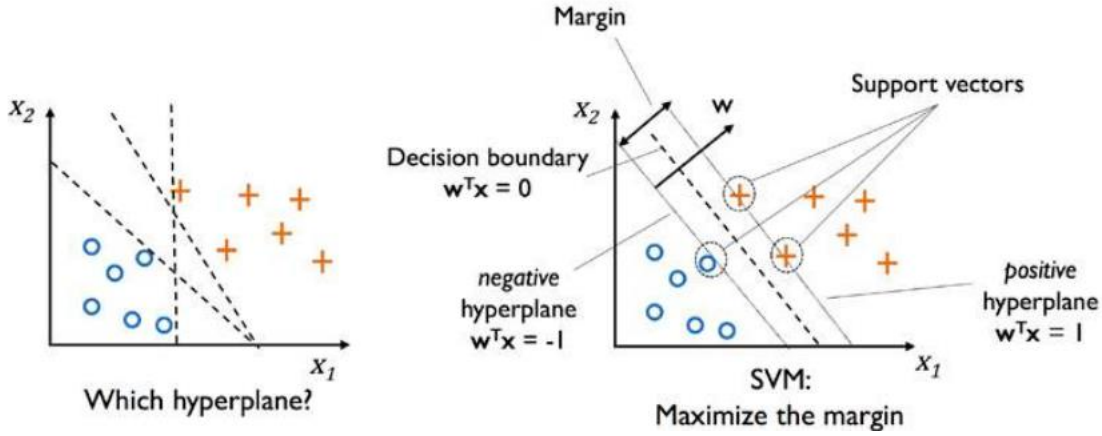


**Figura 3.2.** Representación gráfica y ecuaciones del margen y los hierplanos [8]

Según el núcleo o *kernel* elegido, el algoritmo de optimización escoge una función objetivo distinta con parámetros internos diferentes (aunque siempre se trate en cierta manera de una versión de la maximización del margen). Por ejemplo, el núcleo lineal solo incluye el hiperparámetro *C* (al igual que el resto de *kernels*), mientras que el núcleo RBF (*Radial Basis Function*) además tiene en cuenta la influencia que se le asigna a una muestra del conjunto de entrenamiento a través de $\gamma$ (cuanto mayor es $\gamma$, más cerca deben encontrarse las muestras para verse afectadas por el resto; es decir, más localizado está el efecto de cada muestra).

La SVM implementa además una serie de hiperparámetros de regularización para evitar el sobreaprendizaje u *overfitting*. Concretamente, con independencia del núcleo, toda SVM cuenta con el hiperparámetro *C* o parámetro de regularización inversa, que controla la dicotomía entre la maximización del margen y la minimización del error de clasificación durante el entrenamiento. Esto quiere decir que cuanto mayor es *C*, menor es el margen (por su posición en la función objetivo [8]), más se reduce la robustez de la regularización (por la definición de *C* como inversa de la regularización) y las clasificaciones erróneas durante el entrenamiento se dan con menor frecuencia.

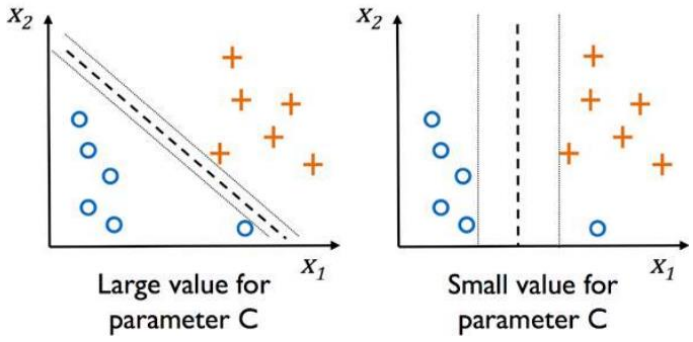


**Figura 3.3.** Comparación de tendencias al *overfitting* (izq.) o al error de clasificación (der.) según *C* [8]

# 4. MATERIAL Y MÉTODOS

En este capítulo se detallan los materiales (librerías y base de datos) que se han utilizado para la implementación de la extracción de características y el modelado del clasificador, así como también se indaga en la metodología seguida en cada sección del proceso (procesado de señal y machine learning).

## 4.1. Librerías

El lenguaje de programación escogido para la realización de este estudio ha sido *Python*, que cuenta con un amplio repertorio de librerías relevantes para el análisis de datos. Así, para la implementación de este proyecto se ha recurrido a las siguientes:

- **mne**: análisis y representación de datos neurofisiológicos [9].
- **numpy**: manejo de matrices y funciones matemáticas elementales [10].
- **os**: interacción con el sistema operativo para la manipulación de archivos y directorios [11].
- **scipy**: funciones matemáticas, principalmente del paquete **stats** (estadística) [12].
- **mne_features**: extracción de características del EEG [13].
- **antropy**: extracción de características de señales temporales [14].
- **pickle**: serialización y deserialización (para almacenar resultados parciales y evitar desaprovechar tiempo durante el desarrollo de código, dado que se utilizan algoritmos con exigencias temporales considerables) [15].
- **matplotlib**: representación gráfica de resultados [16].
- **sklearn**: machine learning [17].

## 4.2. Base de datos

El conjunto de datos de partida ha sido facilitado por el Departamento de Psicología Experimental de la Facultad de Psicología de la Universidad de Sevilla. Se trata de una selección de sesenta electroencefalogramas, de los cuales treinta corresponden a pacientes con EM y otros treinta, a sujetos control (sin EM). Todos los EEG fueron recogidos siguiendo el *paradigma oddball*, que consiste en la presentación de una sucesión de estímulos auditivos o visuales en la que se alternan estímulos denominados estándar con estímulos menos frecuentes que reciben el nombre de *targets*. Ante estos últimos, el sujeto debe reaccionar realizando una tarea concreta, y la actividad cerebral registrada en torno a esos instantes constituye la información de interés.

En el caso de este estudio en particular, la duración de cada EEG es de entre 5.5 y 6.5 minutos, con una proporción estándar-*target* del 75-25%, respectivamente. Asimismo, se emplearon estímulos visuales en la pantalla de un ordenador: un tablero de ajedrez con casillas blancas y negras como estímulo estándar; y un tablero de ajedrez con casillas blancas y rojas como *target*. Ambos tipos de estímulo pueden aparecer a la izquierda o a la derecha de una cruz de fijación que existe en el centro de la pantalla a la que el sujeto debe mirar durante toda la prueba, y la reacción que se espera ante cada *target* consiste en pulsar la flecha del ratón correspondiente al lado de la pantalla donde se muestra el estímulo.

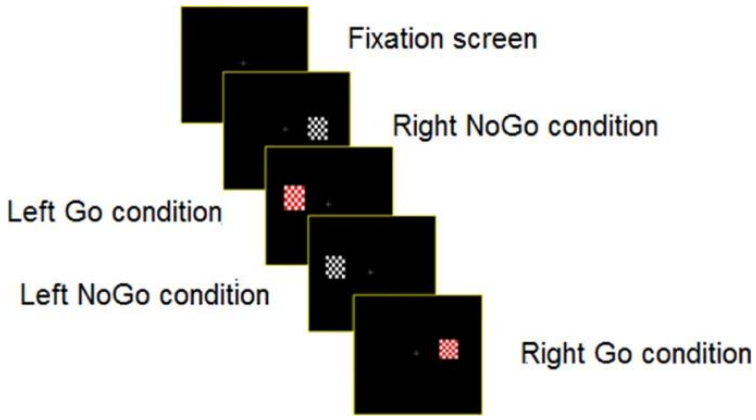


**Figura 4.1.** Las cuatro posibles condiciones del experimento [18]

Por otro lado, los EEG fueron recopilados con un electroencefalógrafo BrainVision Recorder, que distribuye cada registro en tres archivos diferentes [19]:

- **Header file (*.vhdr)**: Fichero de texto que agrupa los metadatos necesarios para interpretar el archivo *.eeg. Destacan los parámetros de registro como la frecuencia de muestreo de 500 Hz y el número de canales, 62 (el número de electrodos que tiene el casco que lleva el sujeto).
- **Marker file (*.vmrk)**: Fichero de texto que contiene la sucesión temporal de los eventos que se han recopilado durante el registro del EEG. Se distinguen los marcadores S 51, S 52, S 53 y S 54, que se corresponden con la aparición en pantalla de los estímulos *target* a la izquierda, *target* a la derecha, estándar a la izquierda y estándar a la derecha, respectivamente; además de los marcadores S1 y S2, que señalan los instantes en que el sujeto pulsa el botón de la izquierda y el de la derecha, respectivamente. Puesto que lo que resulta de interés es parametrizar el procesamiento cognitivo desencadenado por los estímulos a los que debe reaccionar el sujeto, son los eventos S 51 y S 52 los que serán utilizados en el análisis de este proyecto.
- **Raw EEG data file (*.eeg)**: Archivo binario con los datos del EEG en bruto.

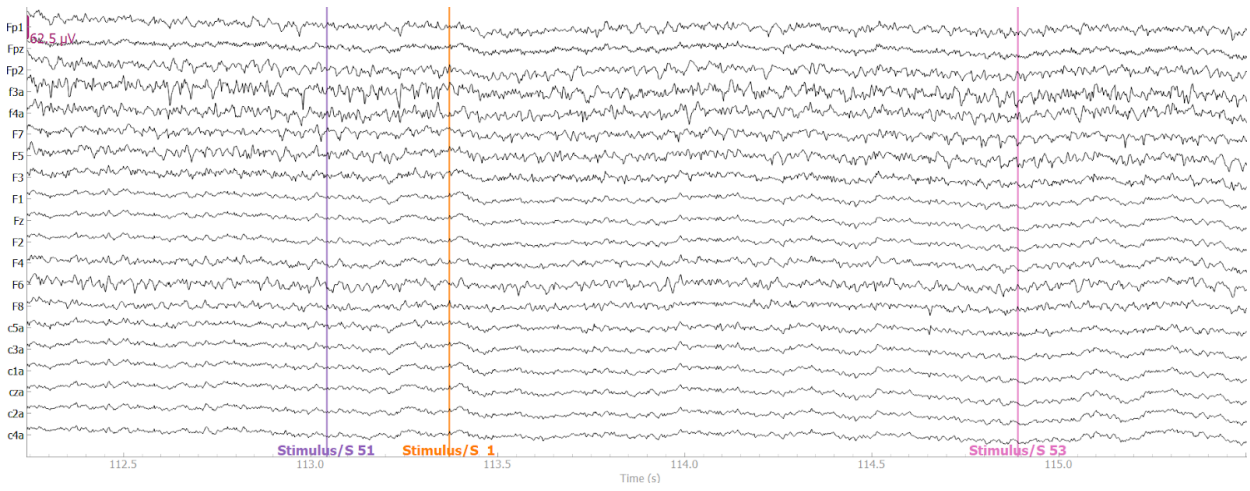


**Figura 4.2.** Segmento de algunos canales de un EEG crudo

En cuanto a la composición frecuencial de los EEG proporcionados, se estudiarán las bandas estandarizadas en neurología, a saber: delta (0.5-4 Hz), theta (4-8 Hz), alfa o mu —según se detecte en el córtex visual (lóbulo occipital) o en el córtex motor (región supra-lateral del cerebro)— (8-12 Hz), beta (12-35 Hz) y gamma (35-45 Hz).

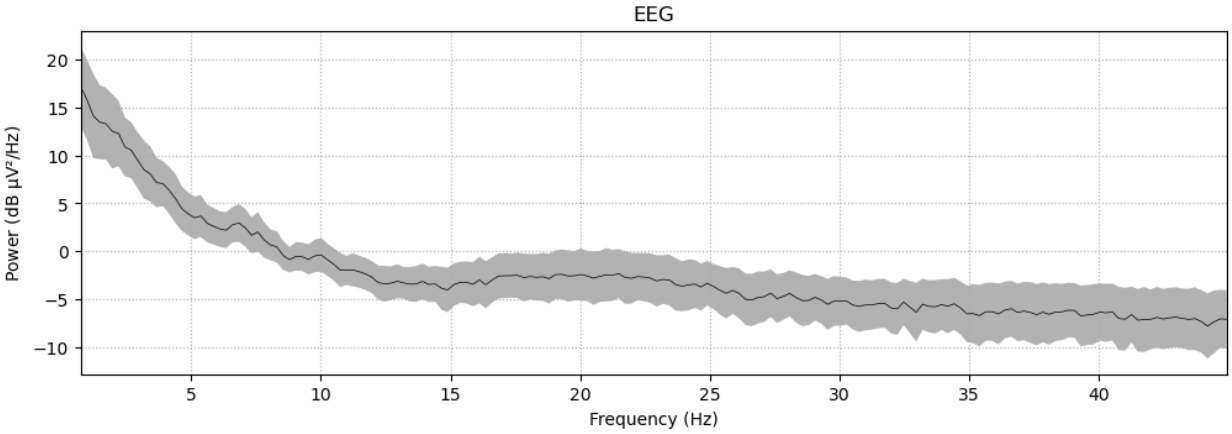


**Figura 4.3.** Densidad espectral de potencia en las bandas de estudio de un EEG

## 4.3. Procesado de señal

Para obtener el conjunto de datos con el que entrenar y validar la máquina de soporte vectorial, se ha realizado una transformación de los registros originales en crudo en la que pueden distinguirse dos fases diferenciadas:

- **Protocolo para la obtención de la respuesta evocada**: Se trata del preprocesado del EEG que se sigue en el laboratorio del Área de Psicobiología del Departamento de Psicología Experimental de la US para hallar la envolvente de la respuesta evocada (envolvente de la respuesta promedio de un sujeto ante cada tipo de estímulo *target*). Se realiza en el fichero **evoked_pipeline.py**, que hace uso de las funciones definidas en **custom_functions.py**.
- **Extracción de los parámetros**: Sección en la que, a partir de la respuesta evocada, se calculan las distintas *features* con las que se modelará y testeará el clasificador. Se recoge en el fichero ***features*.py**.

### 4.3.1. Protocolo para la obtención de la respuesta evocada

#### 4.3.1.1. Corrección de los artefactos oculares

Los electrodos colocados sobre la cabeza del sujeto registran, además de la actividad neuronal procedente del cerebro que se busca analizar, toda clase de respuesta de naturaleza eléctrica que llegue a ellos, como puede ser la procedente del bombeo del corazón o los movimientos oculares. Estos últimos son capaces de generar voltajes de suficiente amplitud (artefactos oculares) como para opacar considerablemente la actividad cerebral en los electrodos más cercanos a los ojos, lo que hace especialmente necesaria la eliminación de estos efectos para realizar un correcto estudio de la actividad cognitiva.

De este modo, se colocan unos electrodos alrededor de un ojo con el fin de tener una referencia de la actividad generada por los movimientos oculares y así poder mitigar su actividad eléctrica sobre el EEG por medio del procesado de señal. Si bien el paradigma empleado en este estudio no propicia que el sujeto mueva los ojos hacia los lados al deber mirar constantemente a la cruz de fijación que se encuentra en el centro de la pantalla, la electricidad estática de las pestañas y el fenómeno de Bell (movimiento hacia arriba de la parte central del ojo al cerrar los ojos) durante el parpadeo son suficientes como para introducir este tipo de alteraciones en el electroencefalograma. Así, aunque no se precise registrar movimiento horizontal en los ojos, sí que resulta necesario recoger el vertical, por lo que los electrodos colocados en torno al ojo son solamente dos y se disponen encima (Veog+) y debajo de este (Veog_), consiguiendo una referencia bipolar de la actividad ocular que genera el sujeto.

En cuanto al procedimiento empleado para la corrección de los artefactos oculares, existen diversas técnicas, como, por ejemplo, el algoritmo de regresión desarrollado por Gratton et al. [20] que utilizan en el Departamento de Psicología Experimental. Sin embargo, existen otros métodos ampliamente extendidos que ofrecen mejores resultados, como el algoritmo ICA [21], el cual se ha empleado en este estudio y cuya

explicación se detalla en la sección *3.1. ICA*.

Aunque no resulta imprescindible eliminar la componente de corriente continua (esto es, media estadística) de los datos a los que se les va a aplicar ICA, sí es cierto que favorece a una mayor simplicidad de los algoritmos [22]. Esto se debe a que al cancelar el nivel de continua (0 Hz), se suprimen tendencias o desviaciones lentas que afectan a múltiples electrodos (como las producidas por el movimiento de un cable o una sudoración fuerte) [23] y por tanto no son estadísticamente independientes. Así, una previa eliminación de la media consigue aumentar la independencia de las fuentes asumidas como independientes (que, en la práctica, en muchos casos no son absolutamente independientes).

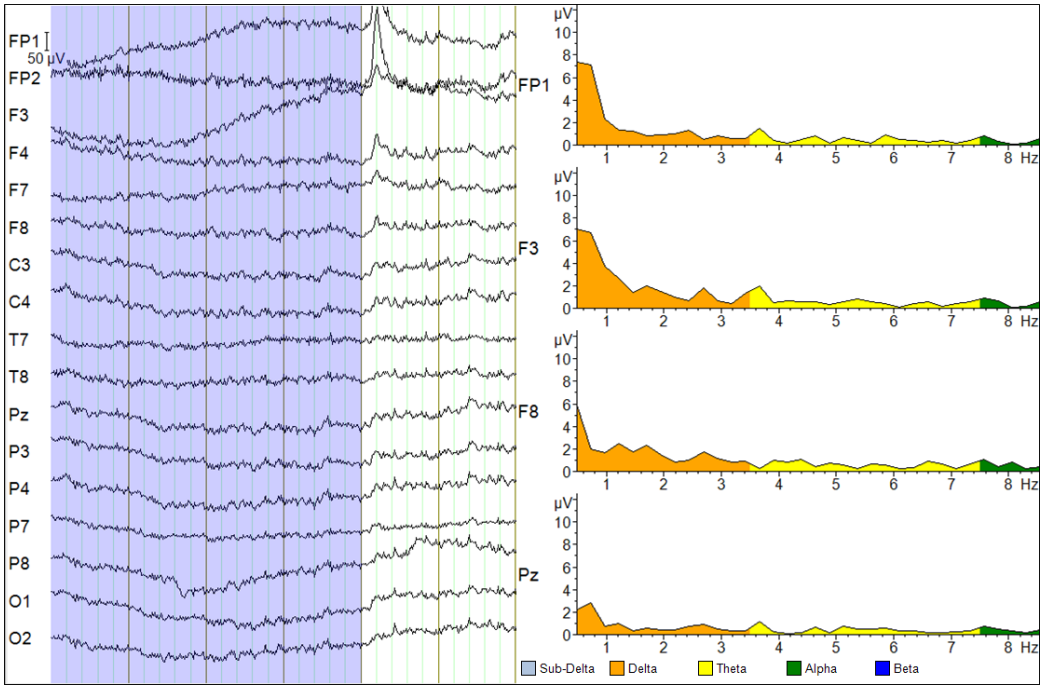


**Figura 4.4.** Desviación lenta en algunos electrodos debido al sudor [24]

Con esto, tras la corrección del offset de continua con la función *raw_noDC* del EEG en bruto, la siguiente operación que se ejecuta en **evoked_pipeline.py** es una llamada a la función *artifact_correction*, que en primer lugar realiza una decorrelación o blanqueado (*whitening*) para reducir de nuevo la complejidad computacional. Esto se consigue por medio de la descomposición en autovectores de la matriz de covarianza de los datos, que permite utilizar una nueva matriz de mezcla ortogonal y por ello estimar un número de parámetros significativamente menor con ICA [22].

Una vez completado el blanqueado, se procede a aplicar ICA al conjunto de datos, y posteriormente se toma la diferencia entre los electrodos oculares empleados como referencia para buscar la componente ocular entre todas las identificadas por ICA. Por último, se reconstruyen las señales como la superposición de todas las componentes detectadas, a excepción de la ocular.

#### 4.3.1.2. Segmentación

Dado que se pretende estudiar la respuesta promedio de la reacción de los sujetos ante los dos tipos de estímulos *target* (S 51 y S 52), a continuación se divide cada EEG en múltiples segmentos de la misma longitud en torno a los marcadores de los eventos mencionados. La duración de estos segmentos, también denominados épocas o *trials*, es de 3200 ms, empezando 1200 ms antes de la aparición del *target* en cuestión y terminado por tanto 2000 ms después de esta. El intervalo temporal de verdadero interés para el Departamento de Psicología Experimental no es superior al segundo desde el marcador del estímulo, pero se toma un segmento de mayor longitud para evitar que los transitorios provocados por el filtrado afecten a las muestras centrales de las que se va a hacer uso para el cálculo de *features* y la corrección de la línea base.

#### 4.3.1.3. Corrección de línea base

Con el fin de relativizar la respuesta eléctrica a un evento externo (como es la aparición de un *target*) con respecto a la actividad cerebral intrínseca del estado de reposo durante una tarea cognitiva, resulta habitual el procedimiento conocido como corrección de la línea base [25]. Este se reduce a calcular la media de la señal durante un intervalo de tiempo anterior a la presentación del estímulo (de -200 a 0 ms en el caso de este estudio en particular) y restarla a todas las muestras pertenecientes a la época en cuestión.

#### 4.3.1.4. Rechazo de artefactos

Puesto que las condiciones teóricas que establece el algoritmo ICA no se transfieren de forma estricta a los casos prácticos, pueden darse casos en los que determinados artefactos oculares persisten tras la aplicación de dicho procesado. Además, también podrían existir otras clases de artefactos puntuales cuya eliminación no era objeto de ICA. Así, es frecuente el rechazo de épocas donde se considera que está presente este tipo de alteraciones. El criterio para la eliminación de *trials* es la superación de un doble umbral de tensión de ±150 μV en el intervalo de 0 a 300 ms, bien en los canales Pz, Cz y/o Fpz (los principales electrodos centrales).

#### 4.3.1.5. Promediado

Una vez se dispone de una selección consistente de épocas para cada uno de los *target*s, se lleva a cabo un promedio de estas, consiguiendo, tanto para S 51 como para S 52, lo que se conoce como respuesta evocada o respuesta en fase.

#### 4.3.1.6. Filtrado paso de banda

Si bien es cierto que se podría decir que los paradigmas de tipo *oddball* están diseñados para el estudio de procesos cognitivos (modulación gamma), sensoriales (modulación alfa) y del sistema motor (modulación mu), se ha decidido analizar todas las bandas de frecuencia que se trabajan en neurología por si se diera el supuesto de que existieran mecanismos diferenciales para distinguir entre pacientes y controles en las frecuencias a las que no suele prestarse tanta atención en paradigmas de esta clase.

Teniendo como objetivo la replicabilidad de los resultados que obtienen en el laboratorio del Departamento de Psicología Experimental a través del software Brainvision Analyzer, se ha realizado un filtrado del mismo modo: aplicando dos filtros Butterworth de orden 8 en serie; primero un paso de baja con frecuencia de corte igual a la frecuencia superior de la banda deseada, y el segundo, un paso de alta con frecuencia de corte igual a la frecuencia inferior de la banda de interés. Además, al igual que el software comercial del laboratorio, cada uno de los dos filtros se aplican hacia delante y hacia detrás para conseguir una función de transferencia conjunta con fase nula. Esta última es sin embargo de orden 32, lo que conlleva unos transitorios no despreciables tanto al principio como al final de la señal de salida que deben descartarse.

Asimismo, se hace uso de secciones de segundo orden en cascada con el objetivo de mejorar la estabilidad del filtro y reducir los errores numéricos asociados a la precisión finita de las aplicaciones prácticas [26].

A excepción de la amplitud y la latencia, el resto de *feautres* que se calcularán en la segunda fase del procesado para entrenar al clasificador partirán de estas señales filtradas.

#### 4.3.1.7. Cálculo de la envolvente

Cuando se dispone de la respuesta en fase para cada banda, se procede a calcular la envolvente de la misma para estimar la amplitud y latencia más adelante. Para ello, se implementa un detector de envolvente como el que incluyen los receptores de radio AM, es decir, un rectificado de onda completa y un filtro paso de baja.

- **Rectificado:** Lo lleva a cabo la función *abs*, que calcula el valor absoluto de la señal de cada banda de frecuencias.
- **Suavizado**: También conocido como *smoothing*, se implementa a través de un filtro paso de baja, que permite observar los cambios más lentos de la señal de entrada para poder advertir tendencias más complicadas de estimar cuando están presentes las componentes de más alta frecuencia. En otras

palabras, recupera la envolvente.

El filtro utilizado, al igual que cada uno de los empleados en el filtrado paso de banda, tiene orden 8, se convoluciona hacia delante y hacia detrás (fase nula), compensa el retardo, utiliza secciones de segundo orden y, en este caso, la frecuencia de corte es de 5 Hz.

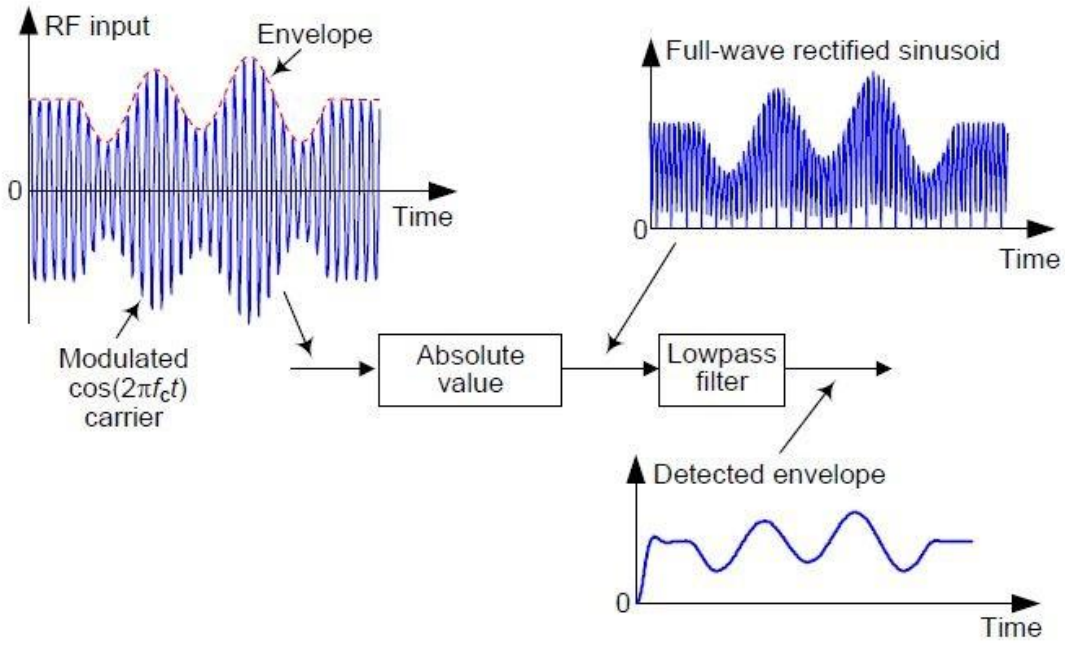


**Figura 4.5.** Diagrama de bloques del detector de envolvente AM [27]

#### 4.3.1.8. Corrección de línea base

Finalmente, se acostumbra a terminar el procesado que antecede al cálculo de amplitudes y latencias con una última corrección de la línea base para situar la actividad cerebral intrínseca del reposo durante tareas cognitivas en los 0 V.

#### 4.3.1.9. Reducción del intervalo temporal

Si se tiene en cuenta la estructura del experimento cognitivo, se puede observar que en ocasiones se suceden dos eventos distintos en algo menos de 2s. Así pues, se ha decidido optar por acortar la longitud de las señales tras el filtrado (apartado *4.3.1.6. Filtrado paso de banda*) y de las señales tras la última corrección de la línea base (apartado *4.3.1.8. Corrección de línea base*). Aunque esta operación no forma parte del protocolo definido por el Departamento de Psicología Experimental, no afecta a las amplitudes y latencias, y además asegura que la respuesta a otro estímulo no falsee el cómputo de las demás *features*.

El nuevo intervalo temporal empieza en los 0 ms y termina en los 1500 ms, de modo que las modulaciones más lentas puedan completar un periodo de oscilación, pero sin llegar a comprometer el análisis de la actividad correspondiente a un estímulo por la intrusión del siguiente.

### 4.3.2. Extracción de los parámetros

En primer lugar, tanto para la respuesta en fase como para su envolvente, se selecciona un electrodo sobre el que calcular las *features*. Se ha decidido escoger el Pz, donde la 'P' hace referencia a 'parietal' y la 'z', a 'zero'; luego se ha recurrido a un electrodo representativo de la zona en la que se desarrolla el procesamiento de atención espacial y la transformación visual-motora [28]. A continuación se exponen las distintas *features* que se han obtenido a partir de esta selección.

#### 4.3.2.1. Amplitud y latencia

Como se ha comentado en apartados anteriores, estos dos parámetros son los únicos que se extraen de la envolvente en lugar del promedio. Se agrupan ambos en la función *peak*, donde se busca el máximo local significativo de tensión que corresponde a la percepción del estímulo, cuyo valor de pico e instante de tiempo asociados son la amplitud y la latencia, respectivamente. Para asegurar que no se escoge un máximo local que venía creciendo desde antes de la presentación del *target* o comenzaba a hacerlo exactamente en ese momento, se limita el intervalo de búsqueda desde los 20 ms hasta los 1500 ms.

#### 4.3.2.2. Varianza

Una medida de dispersión, la varianza se define como el momento central de segundo orden, esto es, el valor esperado del cuadrado de la desviación de una variable aleatoria respecto de su media. Por tanto, cuanto menor sea la varianza, menos dispersos se encuentran los datos respecto a la media del conjunto.

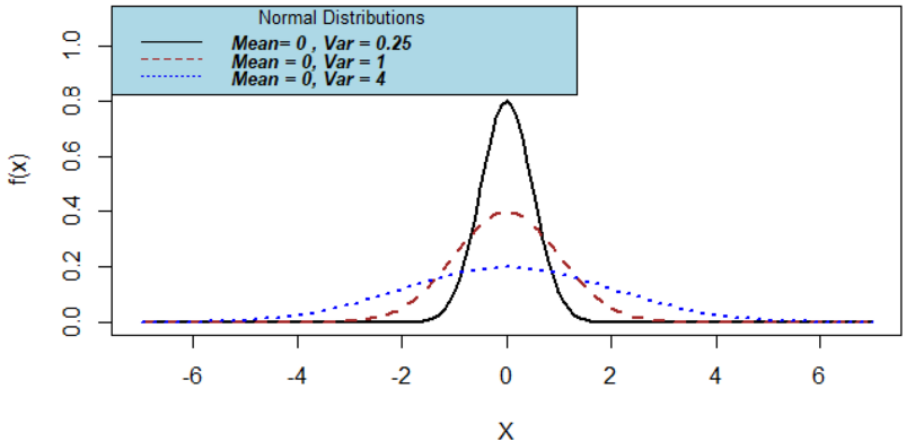


**Figura 4.6.** Comparación de varianzas [29]

#### 4.3.2.3. Asimetría estadística

También conocida como *skewness*, la asimetría estadística es el nombre que recibe el momento central de tercer orden. Puede ser positiva o negativa, y es indicativo de una distribución oblicua (o con la cola) hacia la derecha e izquierda, respectivamente. Constituye por tanto una medida que en cierto modo contempla mejor los valores extremos que la varianza.

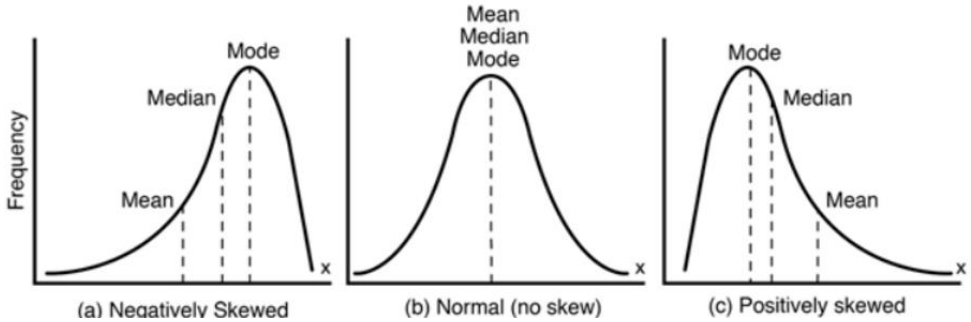


**Figura 4.7.** Comparación de asimetrías [30]

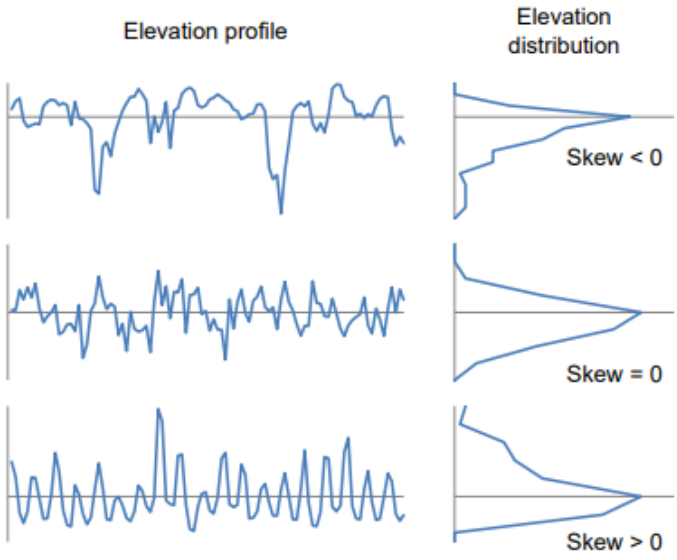


**Figura 4.8.** Perfiles con distinta simetría y sus correspondientes distribuciones [29]

#### 4.3.2.4. Curtosis

Se trata del momento central de cuarto orden y se entiende como una medida de independencia o 'no gaussianidad'. La curtosis puede ser tanto positiva (variable leptocúrtica) como negativa (variable platicúrtica) [22]. La primera suele corresponder a distribuciones picudas con colas con algo más de peso que las de la normal (como la distribución de Laplace), mientras que la segunda suele asociarse a funciones de densidad de probabilidad más planas (como lo es la distribución uniforme).

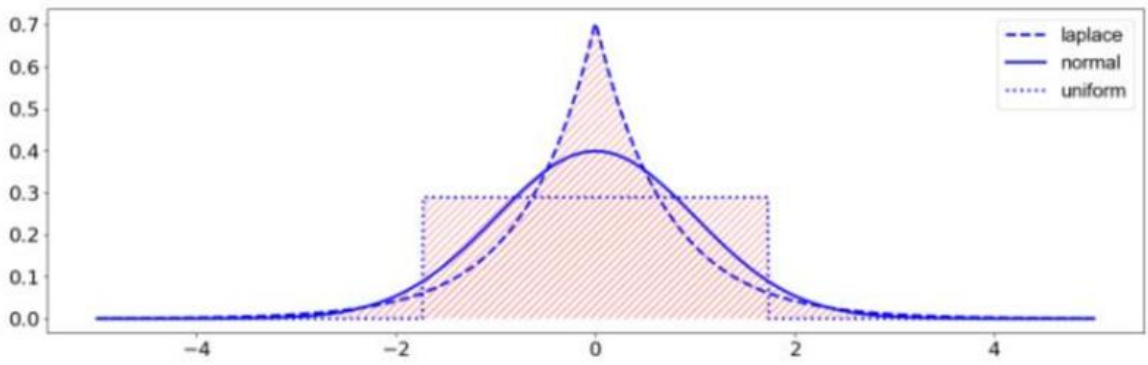


**Figura 4.9.** Comparación de curtosis [29]

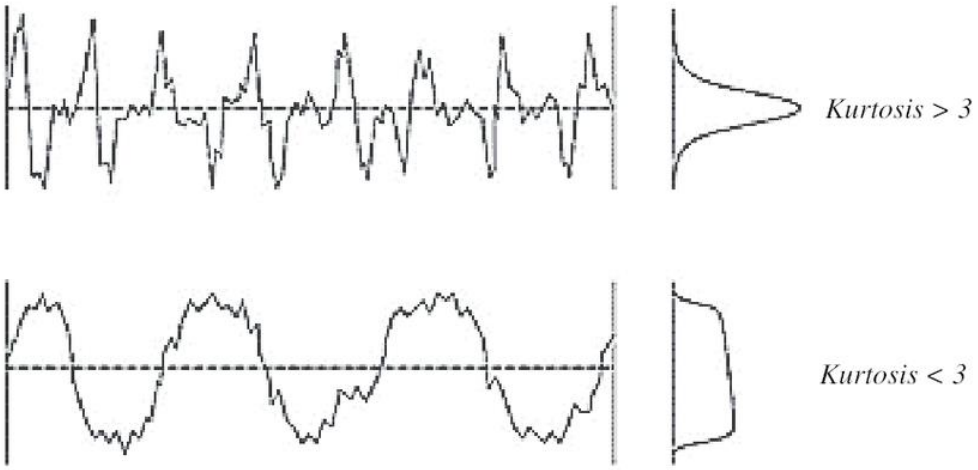


**Figura 4.10.** Perfiles con distinta curtosis y sus correspondientes distribuciones [31]

#### 4.3.2.5. Exponente de Hurst

El exponente de Hurst (*H*) sirve para cuantificar la inercia que posee una señal temporal para volver a la media o agruparse en una dirección [32] y puede tomar los siguientes valores:

- ***H* = 0.5**: que se atribuye a procesos independientes.
- ***H* ∈ (0.5,1]**: que indica cierta tendencia o memoria a largo plazo, como el caso de una serie en la que tras valores grandes aparecen más valores grandes durante un cierto periodo de tiempo.
- ***H* ∈ [0,0.5)**: se asocia a señales con tendencia a la alternancia de valores altos y bajos de manera sucesiva, lo que implica una reversión a la media por medio de una tendencia a la fluctuación entre extremos a largo plazo (anti-tendencia).

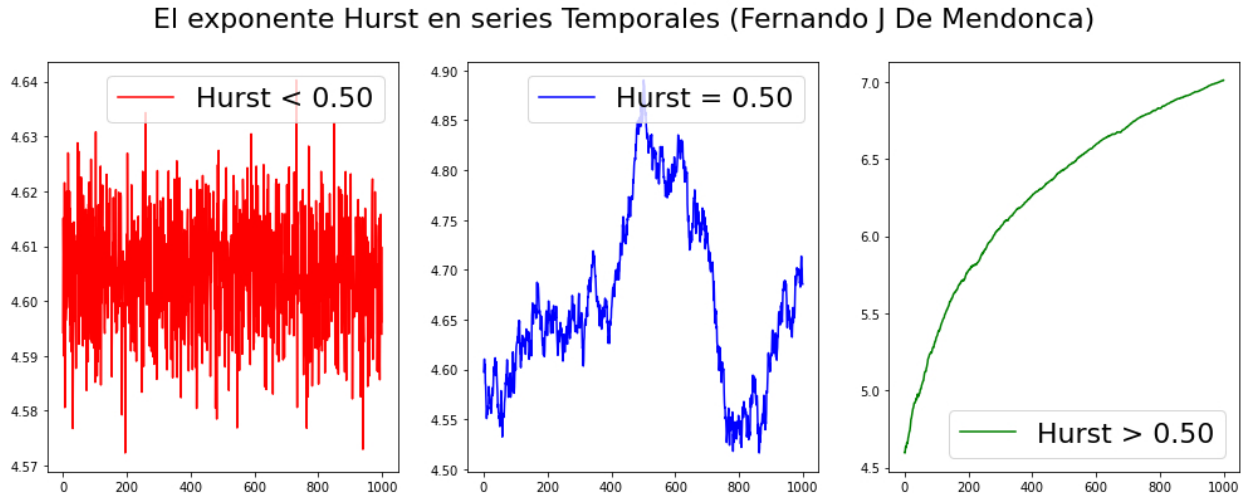


**Figura 4.11.** Ejemplificación de los tres tipos de exponente de Hurst [33]

#### 4.3.2.6. Entropía espectral

La entropía espectral se define como la aplicación de la entropía definida por Shannon a la densidad de potencia espectral (PSD). Y puesto que la entropía se puede interpretar como una estimación de la incertidumbre de una fuente de información (el grado de información promedio que aporta la observación de la variable aleatoria), la entropía espectral —esto es, la entropía del espectro de la señal— puede considerarse como la

cantidad de información que contiene el espectro de frecuencias de la variable aleatoria en cuestión [22]. Este parámetro será mayor, pues, cuanto mayor sea su independencia o gaussianidad [34].

En la *Figura 4.12* puede observarse la expresión de la entropía espectral de una señal *x(t)* con frecuencia de muestreo $f_s$ (*sf*) y PSD normalizada *P(f)*.

$$H(x, sf) = -\sum_{f=0}^{f_s/2} P(f) \log_2[P(f)]$$

**Figura 4.12.** Expresión de la entropía espectral [35]

#### 4.3.2.7. Potencia

Para el cálculo de la potencia media en cada banda se ha optado por la integración del área bajo la curva de la PSD. Esta última se consigue mediante el método de Welch, que consiste en la división de la señal temporal en segmentos que se solapan y a los que se les aplica una ventana (en este caso concreto, de Hanning), para después calcular la DFT sobre cada segmento enventanado y elevar el resultado de cada uno de ellos al cuadrado, finalizando con el promediado de todos [36].

El hecho de que se apliquen ventanas a los segmentos permite minimizar el fenómeno denominado *spectral leakage* (fuga espectral). Este se conoce por manifestarse cuando se computa la DFT de señales con discontinuidades en sus límites —lo que puede interpretarse como un ruido de conmutación—, que da a lugar a componentes frecuenciales espurias [37] [38]. Así, una ventana adecuada (esto es, aquella que reduce significativamente la amplitud de las primeras y últimas muestras de la señal por la que se multiplica) contribuye a forzar una cierta continuidad en las fronteras de los segmentos.

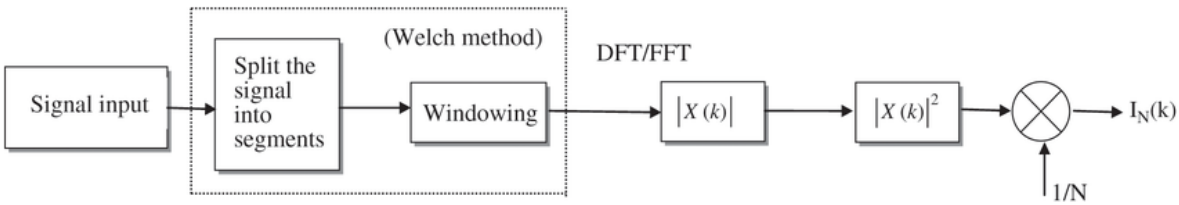


**Figura 4.13.** Diagrama de flujo del método de Welch [38]

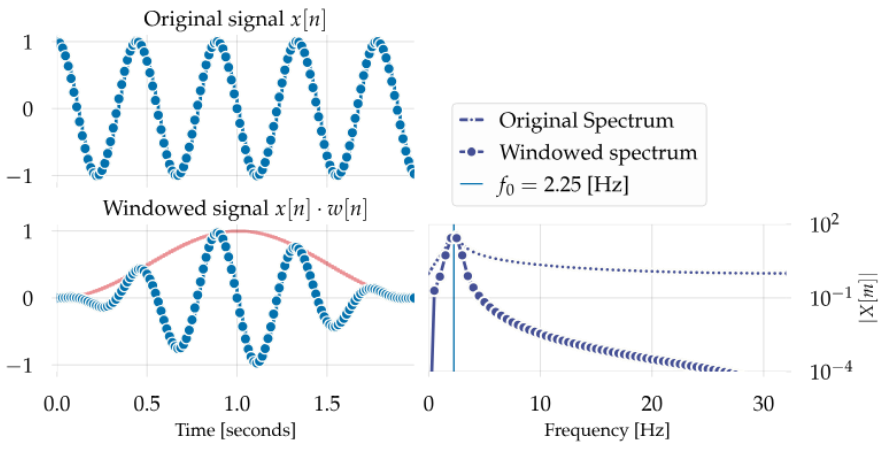


**Figura 4.14.** Reducción del *spectral splatter* con el uso de ventanas [39]

Por otra parte, una vez se dispone de la PSD, para estimar el área que abarca basta con utilizar algún método numérico que aproxime la integral definida, como, por ejemplo, la regla de Simpson, que se sirve de parábolas para ello.

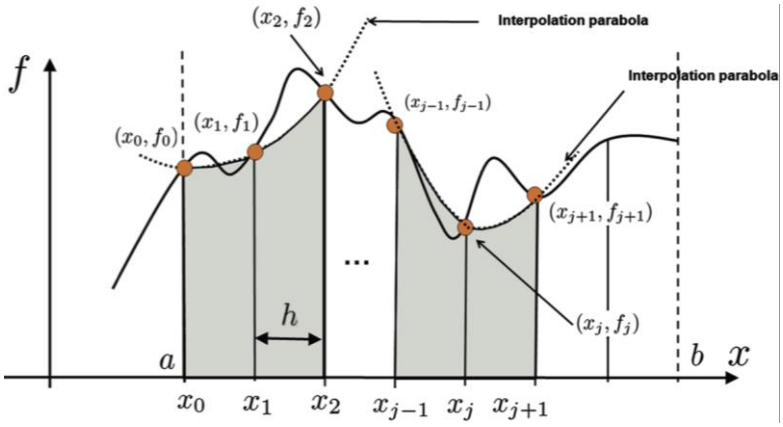


**Figura 4.15.** Aproximación de integral mediante la regla de Simpson [40]

## 4.4. Machine Learning

Una vez se dispone de suficientes datos que caracterizan a cada sujeto que realizó la tarea atencional, se puede proceder a la implementación de un clasificador binario que distinga entre pacientes con EM y sujetos controles en base a esas *features*.

Para ello, en el fichero ***features*.py** se estructura una matriz de datos donde cada fila corresponde a un sujeto y a cada una de estas se le asignan todas las características extraídas (amplitud, latencia, varianza, asimetría, curtosis, exponente de Hurst, entropía espectral y potencia) calculadas para cada banda (delta, theta, alfa, beta y gamma) y para cada clase de estímulo *target* (izquierda y derecha), de forma que se obtiene una matriz de características de dimensiones 60×2×5×8. Asimismo, se crea un vector de *labels* o etiquetas de longitud 60 que contiene *'ms'* o *'control'* de acuerdo con la condición de paciente con EM o control (respectivamente) del sujeto en la posición de mismo valor que la fila de la matriz de características.

Además, se ha escrutado dicha matriz de *features* en busca de posibles valores *NaN* o *NULL* con el fin de eliminar dichas entradas de la base de datos.

### 4.4.1. Partición de los datos

Se conoce que la evaluación del rendimiento de un modelo de aprendizaje automático por medio del uso del mismo conjunto de datos que se utilizó para entrenarlo puede opacar el fenómeno conocido como *overfitting* o sobreajuste, que consiste en una sobre-adecuación del modelo a la base de datos con la que se entrenó, de manera que pierde la capacidad de generalización. Es por esto que la base de datos disponible se divide en un conjunto de entrenamiento (aquel que se utiliza para optimizar los parámetros internos del clasificador) y otro de menor tamaño conocido como conjunto de test (empleado en la valoración de la exactitud del modelo). Estos datos de test se introducen como entrada al clasificador tras haberlo ajustado con los datos de entrenamiento, de modo que la exactitud que se reporta procede de examinar valores que el clasificador no ha recibido con antelación. En consecuencia, si la exactitud de la predicción es alta, se deduce que no se ha incurrido en *overfitting*.

Así pues, se ha realizado una partición de la base de datos asignando un 80% de las entradas al conjunto de entrenamiento (48 sujetos) y el 20% restante al conjunto de test (12 sujetos), cambiando el orden de sujetos de forma pseualeatorioa y manteniendo la proporción inicial de las dos clases a diferenciar.

Además, como se da el caso de que el clasificador a implementar es una máquina de soporte vectorial, se deben redimensionar las matrices de entrenamiento y test para que sean bidimensional. Estas quedan por tanto en una matriz de 48×80 y otra de 12×80, respectivamente.

### 4.4.2. Estandarización

La máquina de soporte vectorial inicializa los pesos de la función objetivo a 0 o algún valor pseualeatorio cercano a este, por lo que si se procura que las columnas de *features* tengan media 0 y varianza 1, se facilitaría el aprendizaje de los pesos al propiciar que no existan características de magnitudes significativamente mayores (o menores) que las demás y dominen sobre estas en la optimización de distancias [8]. Este proceso se denomina estandarización, y como puede adivinarse, se consigue restando a cada valor la media de cada columna de *features* y dividiendo el resultado entre la correspondiente desviación típica. Tras esta operación, los datos estarían ya preparados para emplearse como entrada de la SVM.

### 4.4.3. Validación

Como se comentó en el apartado *4.4.1. Partición de los datos*, los modelos de aprendizaje automático no están exentos de no desarrollar la capacidad de generalizar. Para evitar este efecto se utilizan las técnicas de regularización (apartado *3.2. SVM*) y validación, que controlan el compromiso entre el error de clasificación y la habilidad para detectar el patrón que comparten los datos [41].

En casos en los que se dispone de una considerable cantidad de datos iniciales es posible realizar una partición adicional a la que divide la base de datos en conjunto de entrenamiento y conjunto de test, obteniendo un nuevo conjunto de validación —esto es, para el ajuste de los parámetros externos del modelo, es decir, aquellos parámetros que no se aprenden (ni, por tanto, actualizan) durante la optimización de la función objetivo del algoritmo de aprendizaje, también conocidos como hiperparámetros. Sin embargo, el número de sujetos que conforman la base de datos de partida no es lo bastante alto como para que los datos destinados al entrenamiento sean suficientes para aprender el modelo sin depender de la partición concreta que se haya realizado. Es por ello que se ha adoptado el procedimiento de la validación cruzada o *cross-validation* (CV), concretamente, con el enfoque conocido como *k-folds*, que solo requiere de dos conjuntos de datos; el de entrenamiento y el de testeo. Esto se debe a que el propio grupo de datos de entrenamiento se subdivide en *k* subconjuntos o *folds*, de modo que se utilizan *k*-1 de estos *folds* para el entrenamiento del modelo y este último se valida (testea) en el *fold* restante para obtener la exactitud correspondiente. Esto se repite *k* veces, eligiendo como *fold* de validación un subconjunto distinto cada vez, y la exactitud del modelo vendría dada por la media de las exactitudes de todas las iteraciones, como se ilustra en la *Figura 4.16.*

No obstante, el párrafo anterior no ha hecho referencia alguna al ajuste de hiperparámetros. Si se hiciese uso de los *folds* de validación para la optimización de los parámetros externos del modelo, al ser también subconjuntos que evalúan la exactitud del algoritmo, se estría produciendo ese tipo de incidencia denominado *data leakage* o fuga de información hacia el modelo, sesgando los resultados de la métrica de rendimiento y pudiendo perder así capacidad de generalización [42]. En consecuencia, se recurre a la validación cruzada anidada.

#### 4.4.3.1. Validación cruzada anidada

La validación cruzada anidada o *nested cross-validation* consiste en, como podría deducirse de su nombre, anidar dos niveles de validación cruzada. Esto resulta pues en un bucle exterior que fragmenta el conjunto de datos de entrenamiento en *k*-1 *folds* de entrenamiento y 1 de validación, y un bucle interno que emplea los *k*-1 *folds* de entrenamiento del bucle exterior como conjunto que segmentar en *l*-1 *folds* de entrenamiento y 1 de validación para realizar el ajuste de los hiperparámetros del modelo [42]. Así, una vez se tienen los modelos de hiperparámetros óptimos proporcionados por el bucle interior, el bucle exterior puede evaluar la exactitud de dichos modelos, por lo que finalmente se consigue tanto la optimización del modelo como la estimación de su métrica haciendo uso único de dos subconjuntos de la base de datos inicial (entrenamiento y test; sin necesidad de un tercer subconjunto exclusivo para la validación) —a cambio, eso sí, de un incremento (aunque no especialmente significativo) en la carga computacional.

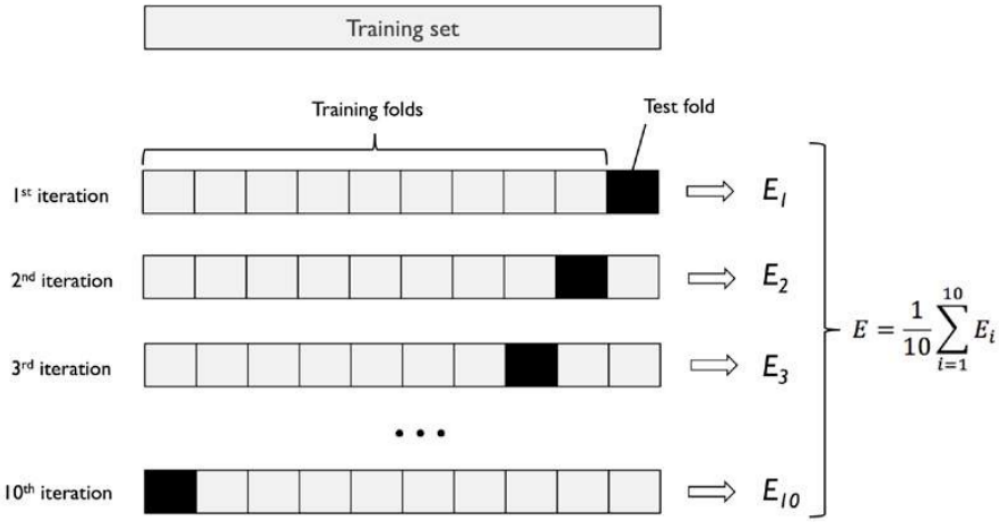


**Figura 4.16.** Ejemplificación de validación cruzada con 10 *folds* [8]

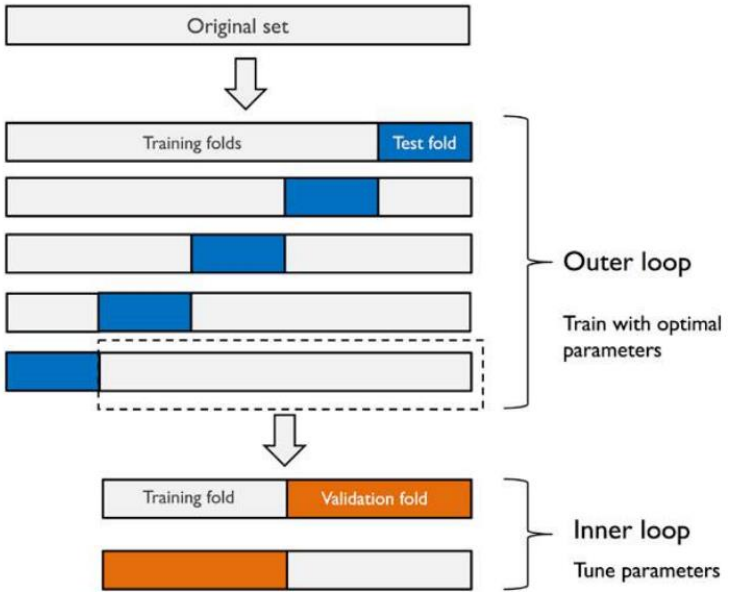


**Figura 4.17.** Ejemplificación de validación cruzada anidada de 5×2 *folds* [8]

En cuanto al objeto de *Python* que se encarga del ajuste de los hiperparámetros, se trata de *GridSearchCV*, el cual implementa un paradigma de búsqueda exhaustiva que valora durante el entrenamiento todas las combinaciones de hiperparámetros que se le proporcionan (*parameter grid*) y selecciona la que maximiza la exactitud durante la validación [43]. Para el *parameter grid* se ha optado por valores frecuentes del parámetro de regularización $C$ para el núcleo *lineal*, y por los parámetros $C$ y $\gamma$ para el núcleo *rbf* o de función de base radial (no lineal).

La función que realiza la validación cruzada que evalúa el rendimiento de los modelos optimizados es *cross_val_score*, a la que se le pasa como entrada el objeto *GridSearchCV*, que retiene los hiperparámetros óptimos en la variable *best_params_*.

Cabe añadir que, dado el reducido número de sujetos de la matriz de entrenamiento (48), se han fijado los números de *folds* en 5 para el bucle exterior y 2 para el interior con el fin de no reducir demasiado la cantidad de sujetos por *fold*.

Una vez se conoce la exactitud media de los 5 modelos, se procede a entrenar la máquina de soporte vectorial final (función *fit*) con la totalidad del conjunto de datos de entrenamiento y los valores optimizados de los hiperparámetros.

### 4.4.4. Testeo

El último paso del proceso consiste en probar la exactitud del modelo (función *predict*) con el conjunto de datos de test, que no se había empleado hasta este momento.

# 5. RESULTADOS

A continuación, se exponen los resultados principales conseguidos a través de la ejecución de la implementación detallada en el capítulo *4. Material y métodos*.

## 5.1. Procesado de señal

Comenzando con la corrección de artefactos oculares, se pude comprobar como ICA detecta el parpadeo de en torno a los 39.9 s y lo elimina adecuadamente gracias a la referencia que establecen los electrodos oculares.

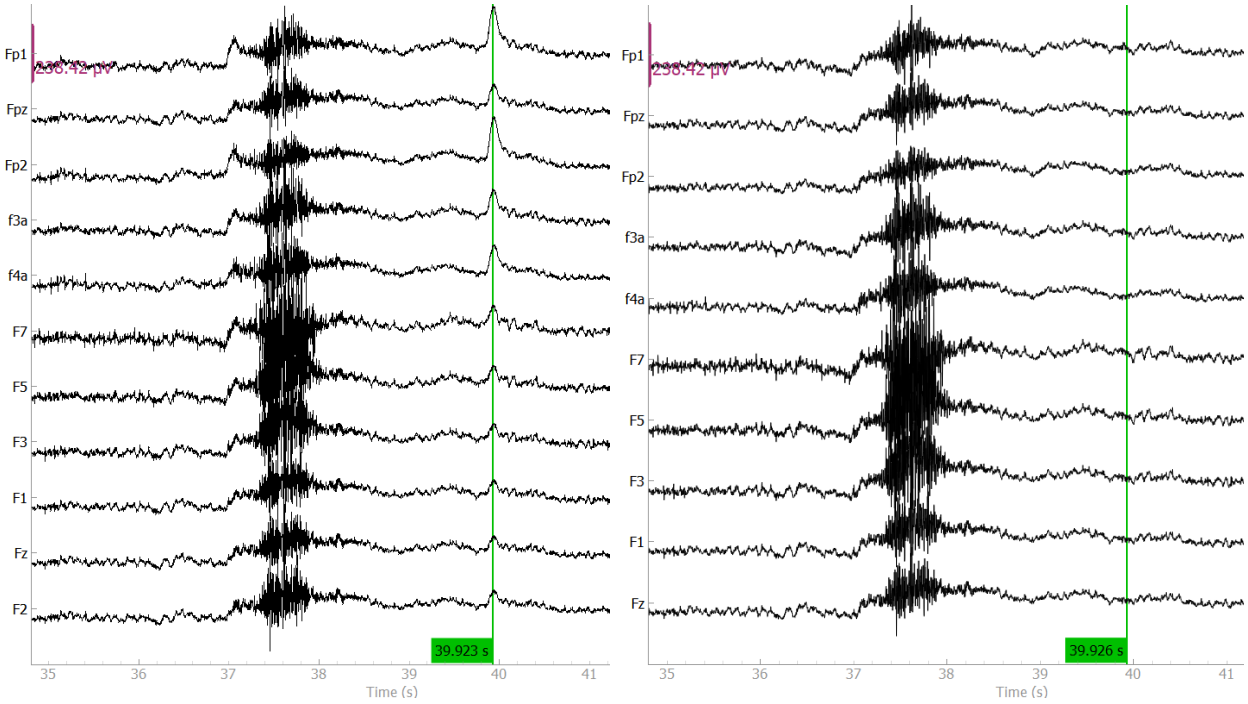


**Figura 5.1.** Corrección de artefactos oculares (pre-ICA a la izquierda; post-ICA a la derecha)

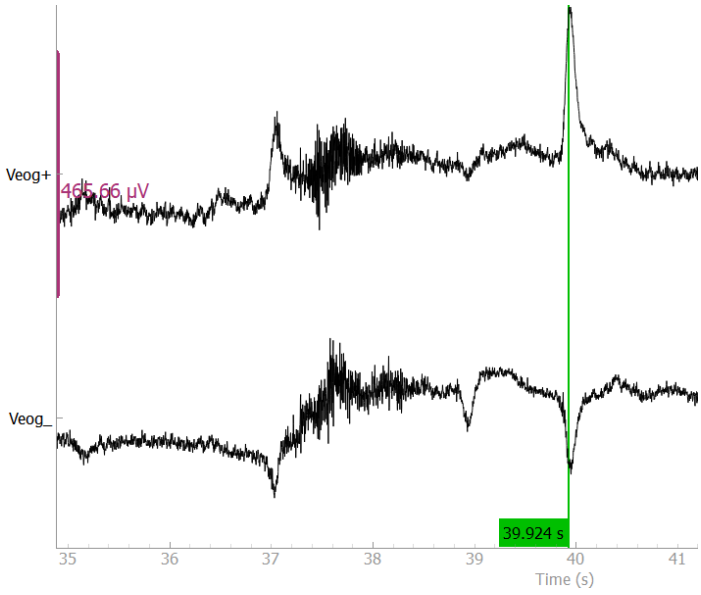


**Figura 5.2.** Electrodos oculares

Después, se procede a segmentar en *trials* y se sigue entonces con la corrección de la línea base, como lo que se puede ver en la figura *Figura 5.3.*

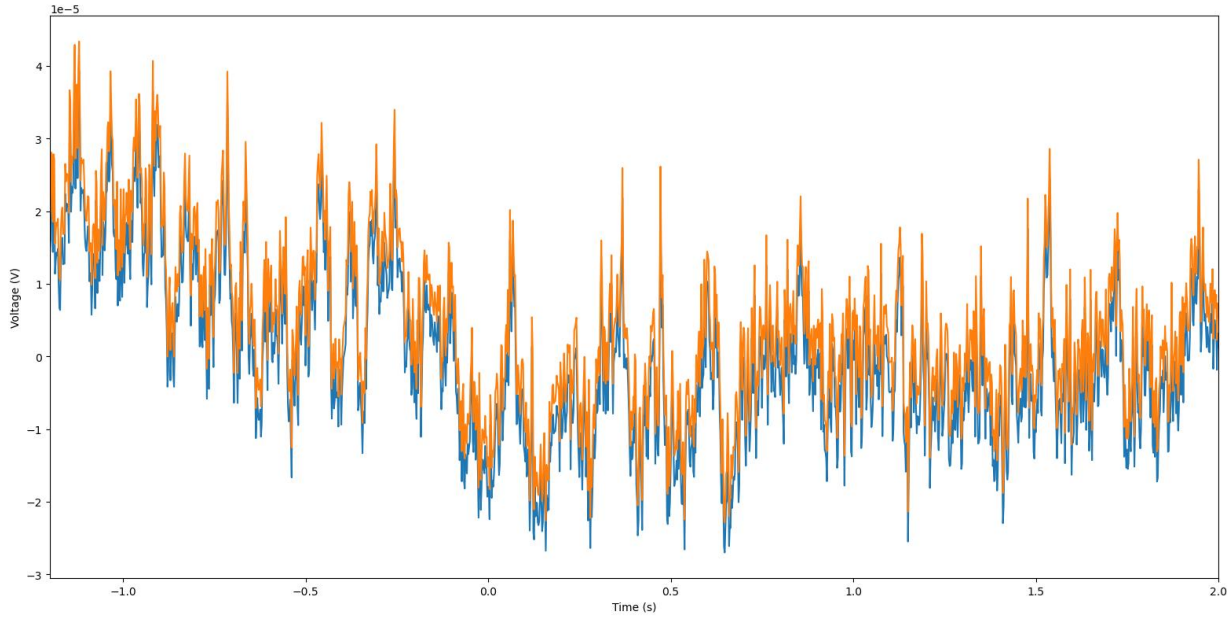


**Figura 5.3.** Época correspondiente a un *target* S 51 en Pz (post-corrección de la línea base en naranja)

El siguiente paso lo constituía el rechazo de épocas, que descarta segmentos como el de la siguiente figura.

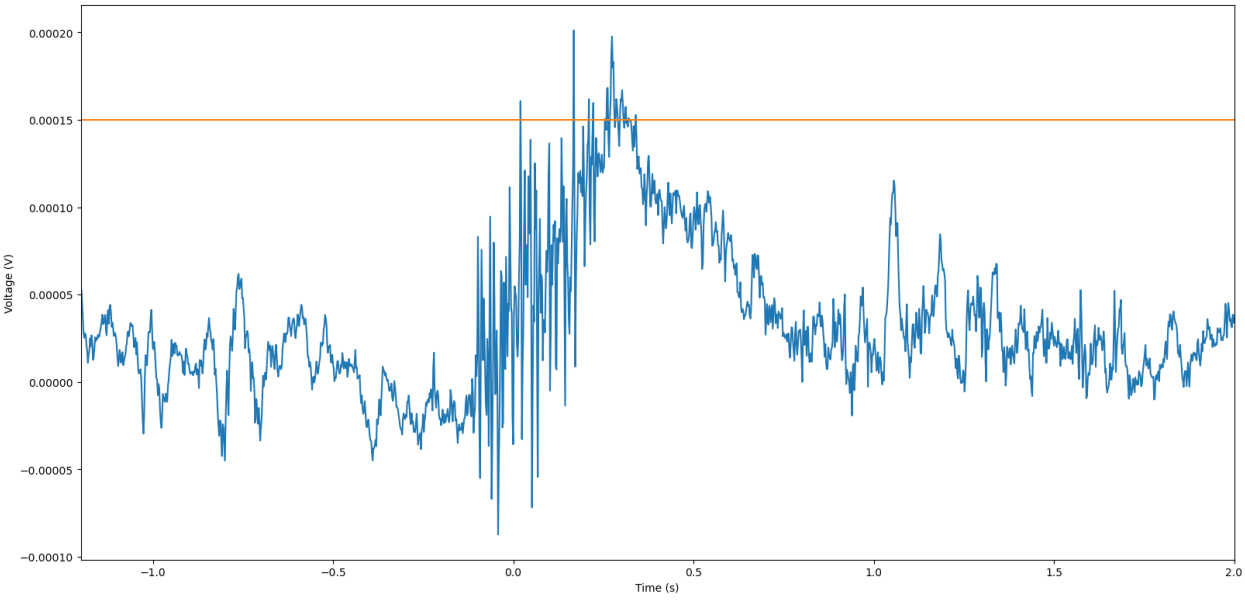


**Figura 5.4.** Época que supera el umbral superior establecido (naranja)

Justo después se promedian todas las épocas para cada sujeto, quedando una respuesta como la que sigue:

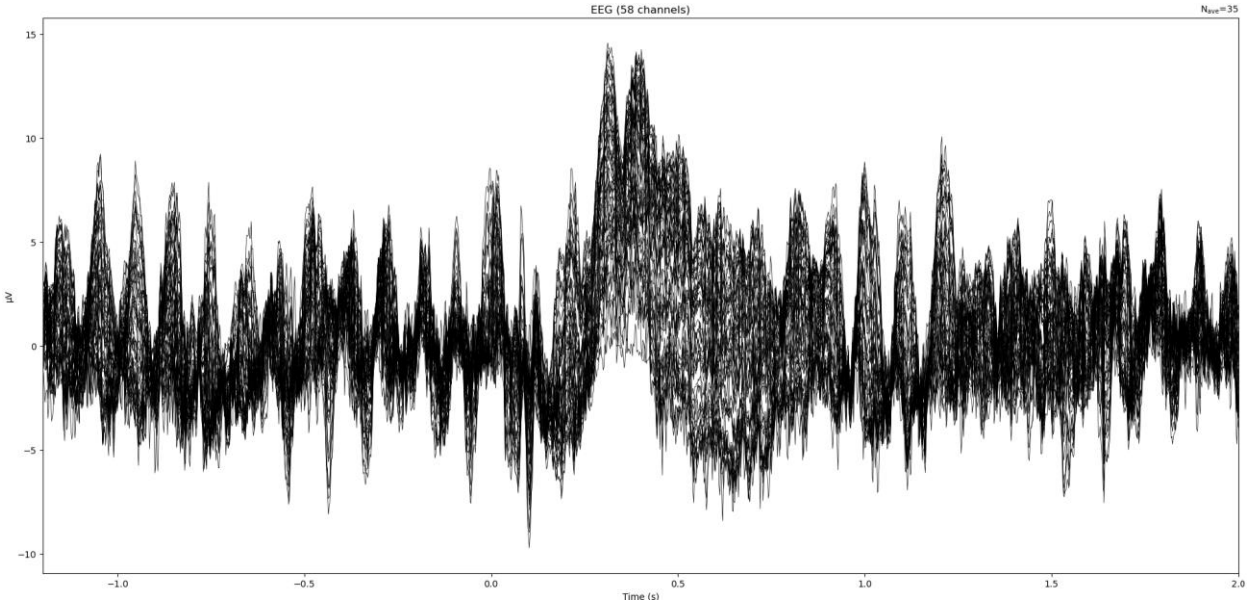


**Figura 5.5.** *Butterfly view* (superposición de todos los canales) del promedio

A continuación, se filtra y se obtienen gráficas con la actividad en cada banda de frecuencia, como por ejemplo:

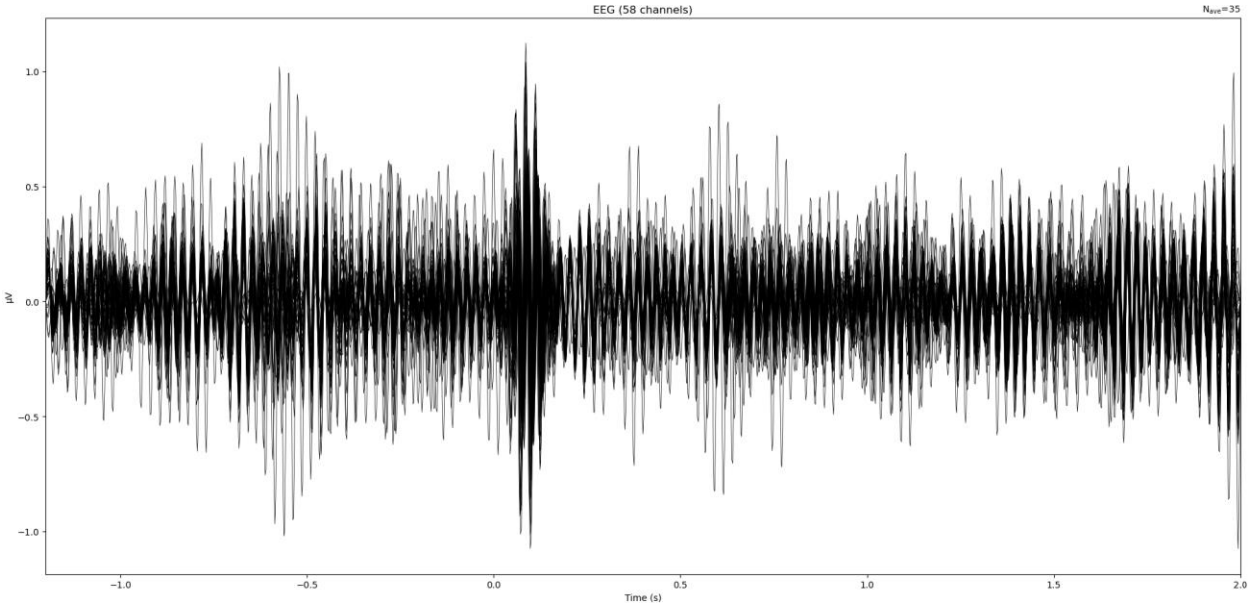


**Figura 5.6.** *Butterfly view* del promedio en la banda gamma

Se continua con el cálculo de la envolvente (*Figura 5.8*) a través del rectificado (*Figura 5.7*) y el filtrado paso de baja.

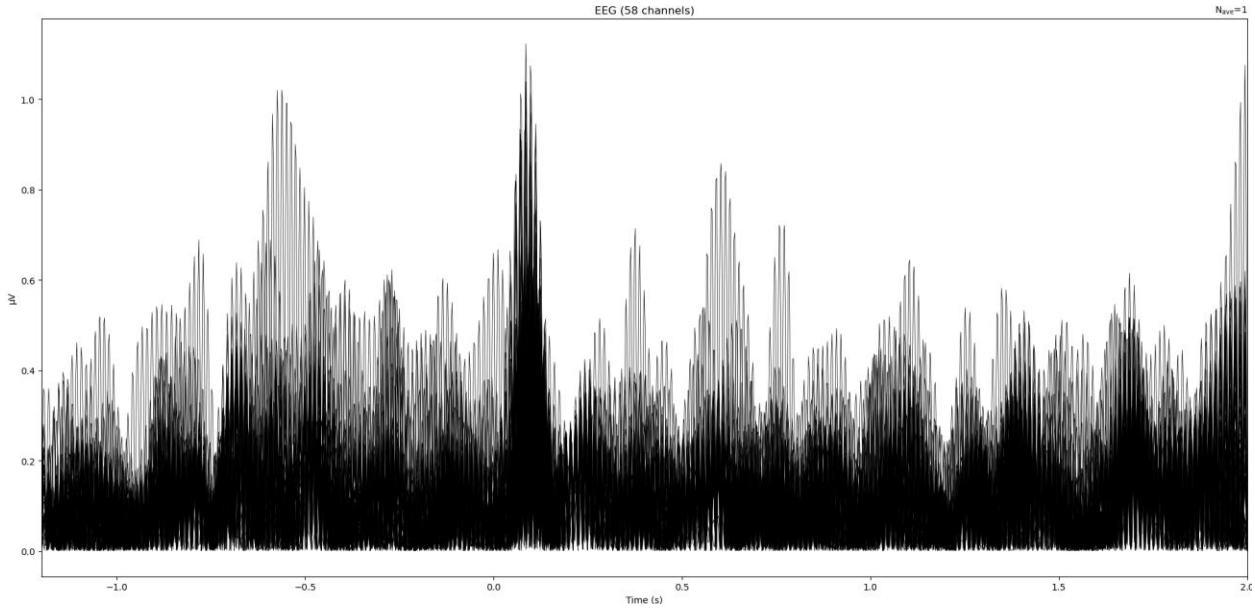


**Figura 5.7.** Rectificado (gamma)

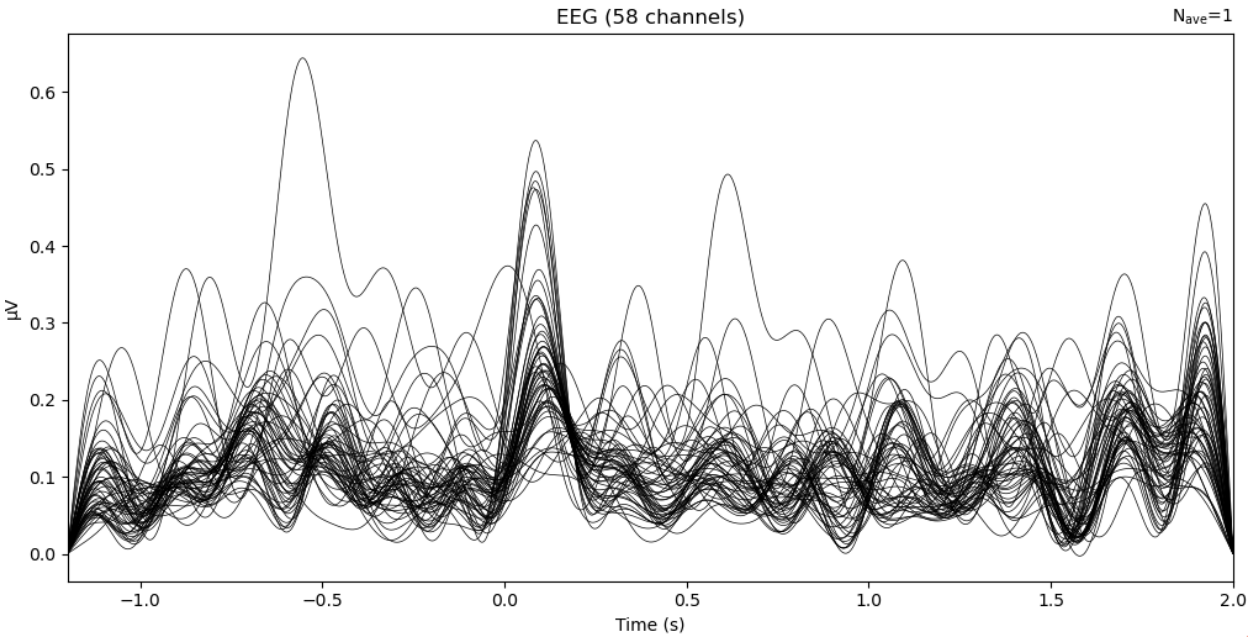


**Figura 5.8.** Envolvente (gamma)

Por último, antes de pasar a la extracción de características, se aplica otra corrección de línea base y se selecciona el intervalo de interés:

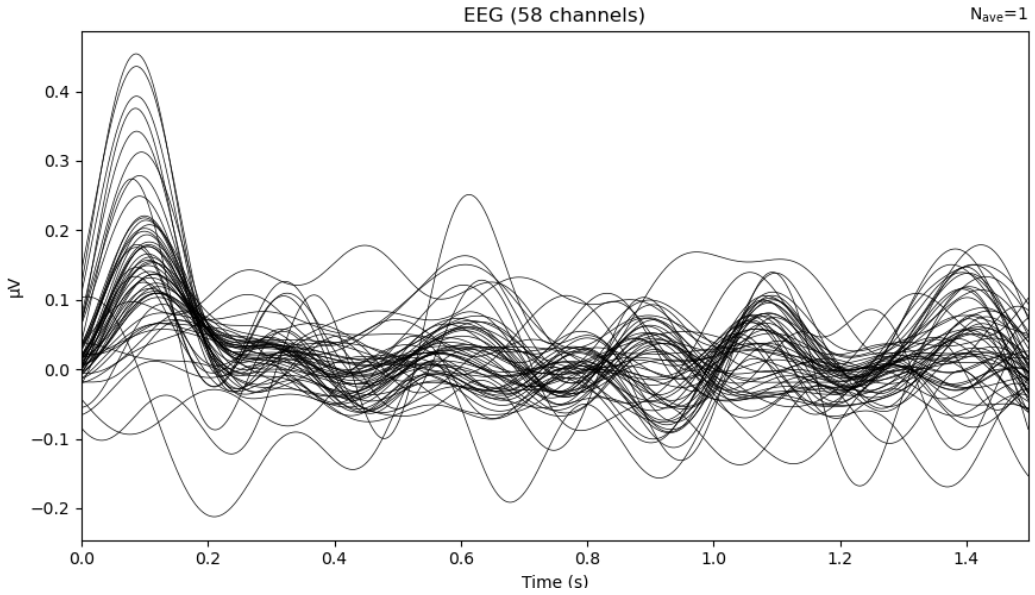


**Figura 5.9.** Envolvente de la respuesta evocada (gamma)

Para el cálculo de la latencia y de la amplitud, se escoge el electrodo Pz de la envolvente. Para el caso del sujeto y la banda con los que se ha ido ejemplificando, la primera es de 0.1 s y la segunda, de 0.218 µV.

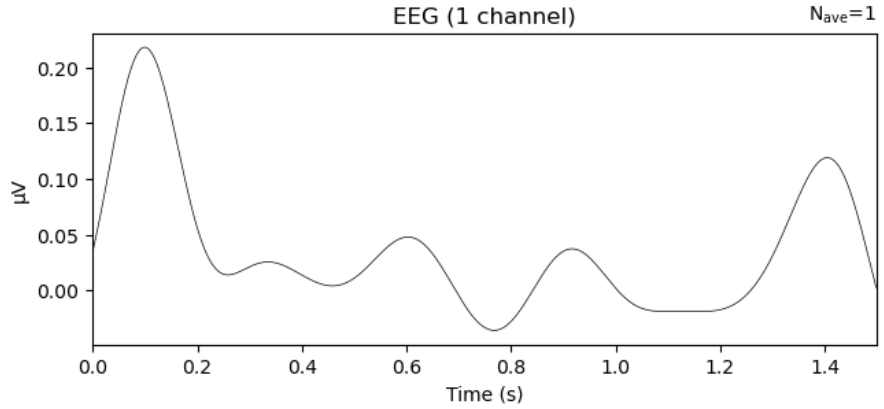


**Figura 5.10.** Electrodo Pz (envolvente)

El resto de medidas se realizan sobre el electrodo Pz del promedio, pero, dado que su relación con la representación gráfica de la señal resulta poco intuitiva al ojo humano, tampoco resulta útil su concreción.

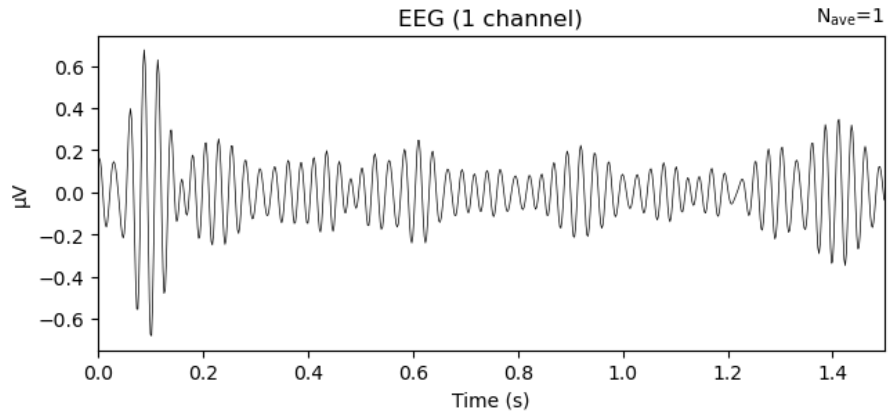


**Figura 5.11.** Electrodo Pz (promedio)

## 5.2. Machine Learning

La exactitud de la SVM es del 100%, es decir, el modelo obtenido clasifica a sujetos control y a pacientes con EM sin error, como se aprecia de forma gráfica en la matriz de confusión de la *Figura 5.12*, que indica una diferenciación absoluta entre ambos tipos de sujeto al coincidir las etiquetas de las predicciones con las del vector de etiquetas del subconjunto de datos de test tanto para la clase 0 (control) como para la clase 1 (MS).

Atendiendo a la *Figura 5.13*, se pueden comprobar además las exactitudes comprendidas entre 0.89 y 1 de los 5 modelos propuestos por la validación cruzada anidada; la exactitud media de 0.98 ± 0.04 que esta sugiere; que el estimador óptimo es una SVM con núcleo RBF, $C = 10$ y $\gamma = 0.001$ (previa estandarización de los datos de test de acuerdo a los valores decididos durante la estandarización de los datos de entrenamiento (cambio de variable)); y que las distintas métricas sobre el clasificador final arrojan resultados inmejorables.

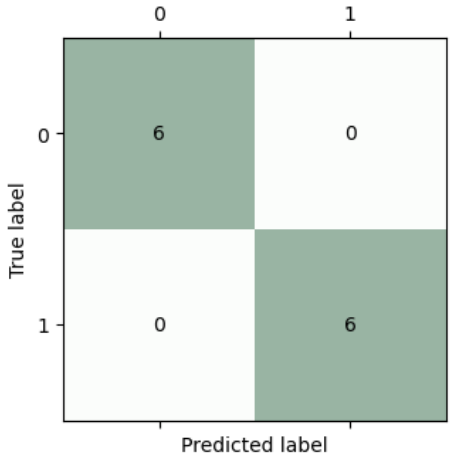


**Figura 5.12.** Matriz de confusión

```
Accuracy of each nested-cross-validated model:
 [1.         1.         1.         0.88888889 1.        ]

0.98 average accuracy with a standard deviation of 0.04


Optimal hyperparameters:
 {'svc__C': 10, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}

Optimal estimator:
 Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('svc', SVC(C=10, gamma=0.001))])


              precision    recall  f1-score   support

     control       1.00      1.00      1.00         6
          ms       1.00      1.00      1.00         6

    accuracy                           1.00        12
   macro avg       1.00      1.00      1.00        12
weighted avg       1.00      1.00      1.00        12

########
1.00 accuracy with a standard deviation of 0.00
########
```

**Figura 5.13.** Resultados **svm.py**

Los resultados reportados van asociados a una semilla o *seed* (*random_state* = 0 en la función *train_test_split*) que genera la misma partición de datos cada vez que se ejecuta el fichero **svm.py**. Esta semilla se escogió de forma aleatoria para facilitar el análisis del sistema durante el desarrollo del código en materia de reproductibilidad, depuración y comparación, pero no debe utilizarse como hiperparámetro del modelo. Por ello, se prueba finalmente a ejecutar el proceso explicado hasta ahora una serie de veces para obtener un valor medio, que para 100 iteraciones resulta ser de $0.97 \pm 0.05$.

# 6. CONCLUSIONES

En vista a los resultados, se demuestra que es posible la distinción de pacientes con EM de sujetos sanos en base a la observación de características cognitivas del EEG recogido durante tareas atencionales de tipo *oddball*.

Aplicando un procesado de señal —separación de fuentes, filtrados, promediados, detección de envolventes, entre otros— al EEG crudo, se obtiene la respuesta evocada de un tipo de estímulo, a partir de la cual se calculan una serie de parámetros que caracterizan la señal —latencia, amplitud, varianza, asimetría, curtosis, exponente de Hurst, entropía espectral, potencia— y se utilizan para la implementación de un clasificador —estandarización, SVM, ajuste de hiperparámetros, validación cruzada— capaz de diferenciar entre los dos perfiles mencionados.

Si bien es cierto que la exactitud del modelo ($0.97 \pm 0.05$) se considera razonablemente positiva, tal vez se podría aumentar la probabilidad de una predicción correcta poniendo en práctica una serie de medidas. Así, las propuestas que se plantean de cara a líneas de investigación futuras serían las siguientes:

- **Búsqueda de nuevas *features***: Probar a incluir otras formas de caracterizar la respuesta evocada con mayor capacidad de discriminar entre pacientes y controles.
- **Aumento de la base de datos y deep learning:** Un mayor número de observaciones en la base de datos podría permitir el uso de algoritmos más complejos y con mayor potencial para mejorar las métricas, como lo son las redes neuronales. Concretamente, podrían implementarse modelos mediante técnicas de aprendizaje profundo para diferenciar con todavía más porcentaje de aciertos.
- **Respuesta inducida:** Resultaría interesante explorar la caracterización de la respuesta inducida o respuesta no-fase del EEG (aquella actividad que se elimina del EEG al promediar para obtener la respuesta evocada) del mismo modo que se ha hecho con la respuesta en fase. Aunque aún no se conoce con absoluta certeza qué tipo de actividad neuronal puede estar detrás de lo que tradicionalmente se ha tratado como ruido, existen indicios que apuntan a la respuesta inducida como evidencia de ciertos mecanismos cognitivos que hasta recientemente se han visto opacados por la respuesta evocada [5] [6].

# REFERENCIAS

[1] R. H. B. Benedict, M. P. Amato, J. DeLuca, and J. J. G. Geurts, "Cognitive impairment in multiple sclerosis: clinical management, MRI, and therapeutic avenues," *The Lancet Neurology*, vol. 19, no. 10, pp. 860–871, Oct. 2020, doi: https://doi.org/10.1016/s1474-4422(20)30277-5.

[2] J. Reina Tosina and L. M. Roa Romero, "Revisitando el problema de la transmisión en axones: algunas certezas e incertidumbres," in *URSI 2024*, 2024.

[3] M. M. Villapecellin-Cid, L. M. Roa, and J. Reina-Tosina, "Transverse magnetic waves in myelinated nerves," *2001 Conference Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 1, pp. 888–891, doi: https://doi.org/10.1109/iembs.2001.1019085.

[4] "casco EEG 32 canales ANT Neuro," *http://twitter.com/MedicalExpoNews*, 2015. https://www.medicalexpo.es/prod/ant-neuro/product-84713-544841.html (accessed Sep. 26, 2024).

[5] M. Vázquez-Marrufo, R. Caballero-Díaz, R. Martín-Clemente, A. Galvao-Carmona, and J. J. González-Rosa, "Individual test-retest reliability of evoked and induced alpha activity in human EEG data," *PLOS ONE*, vol. 15, no. 9, p. e0239612, Sep. 2020, doi: https://doi.org/10.1371/journal.pone.0239612.

[6] M. Vázquez-Marrufo, Macarena García-Valdecasas, Rocío Caballero-Diaz, R. Martin-Clemente, and A. Galvao-Carmona, "Multiple evoked and induced alpha modulations in a visual attention task: Latency, amplitude and topographical profiles," *PloS one*, vol. 14, no. 9, pp. e0223055–e0223055, Sep. 2019, doi: https://doi.org/10.1371/journal.pone.0223055.

[7] D. Mika, G. Budzik, and J. Józwik, "Single Channel Source Separation with ICA-Based Time-Frequency Decomposition," *Sensors*, vol. 20, no. 7, p. 2019, Apr. 2020, doi: https://doi.org/10.3390/s20072019.

[8] S. Raschka and V. Mirjalili, *Python machine learning : machine learning and deep learning with python, scikit-learn, and tensorflow 2*. Birmingham: Packt Publishing, Limited, 2019.

[9] "MNE — MNE 1.0.2 documentation," *mne.tools*. https://mne.tools/stable/index.html (accessed Sep. 26, 2024).

[10] Numpy, "NumPy," *Numpy.org*, 2009. https://numpy.org/ (accessed Sep. 26, 2024).

[11] "os — Interfaces misceláneas del sistema operativo — documentación de Python - 3.10.11," *docs.python.org*. https://docs.python.org/es/3.10/library/os.html (accessed Sep. 26, 2024).

[12] SciPy, "SciPy.org — SciPy.org," *Scipy.org*, 2020. https://scipy.org/ (accessed Sep. 26, 2024).

[13] "MNE-Features — mne_features 0.2 documentation," *Mne.tools*, 2018. https://mne.tools/mne-features/ (accessed Sep. 26, 2024).

[14] "Installation — antropy 0.1.6 documentation," *Raphaelvallat.com*, 2020. https://raphaelvallat.com/antropy/build/html/index.html (accessed Sep. 26, 2024).

[15] Python Software Foundation, "pickle — Python object serialization — Python 3.7.3 documentation," *Python.org*, 2019. https://docs.python.org/3/library/pickle.html (accessed Sep. 26, 2024).

[16] Matplotlib, "Matplotlib: Python plotting — Matplotlib 3.1.1 documentation," *Matplotlib.org*, 2012. https://matplotlib.org/ (accessed Sep. 26, 2024).

[17] Scikit-learn, "scikit-learn: machine learning in Python," *Scikit-learn.org*, 2019. https://scikit-learn.org/stable/ (accessed Sep. 26, 2024).

[18] E. Sarrias-Arrabal, R. Martín-Clemente, A. Galvao-Carmona, M. L. Benítez-Lugo, and M. Vázquez-Marrufo, "Effect of the side of presentation in the visual field on phase-locked and nonphase-locked alpha and gamma responses," *Scientific Reports*, vol. 12, no. 1, Aug. 2022, doi: https://doi.org/10.1038/s41598-022-15936-7.

[19] "BrainVision Core Data Format 1.0 | Brain Products GmbH," *Brain Products GmbH*, Apr. 30, 2024. https://www.brainproducts.com/support-resources/brainvision-core-data-format-1-0/ (accessed Sep. 26, 2024).

[20] G. Gratton, M. G. H. Coles, and E. Donchin, "A new method for off-line removal of ocular artifact," *Electroencephalography and Clinical Neurophysiology*, vol. 55, no. 4, pp. 468–484, Apr. 1983, doi: https://doi.org/10.1016/0013-4694(83)90135-9.

[21] C. Jutten and J. Herault, "Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture," *Signal Processing*, vol. 24, no. 1, pp. 1–10, Jul. 1991, doi: https://doi.org/10.1016/0165-1684(91)90079-x.

[22] A. Hyvärinen and E. Oja, "Independent Component Analysis: Algorithms and Applications," *Neural Networks*, vol. 13, no. 45, pp. 411–430, 2000, Available: https://www.cs.helsinki.fi/u/ahyvarin/papers/NN00new.pdf

[23] M. Klug and K. Gramann, "Identifying key factors for improving ICA-based decomposition of EEG data in mobile and stationary experiments," *European Journal of Neuroscience*, vol. 54, no. 12, pp. 8406–8420, Oct. 2020, doi: https://doi.org/10.1111/ejn.14992.

[24] Brain Products, "Getting to know EEG artifacts and how to handle them in BrainVision Analyzer," *Brain Products Press Release*, Dec. 09, 2022. https://pressrelease.brainproducts.com/eeg-artifacts-handling-in-analyzer/ (accessed Sep. 26, 2024).

[25] W. Wan *et al.*, "Resting state EEG complexity as a predictor of cognitive performance," *Physica A: Statistical Mechanics and its Applications*, vol. 624, p. 128952, Aug. 2023, doi: https://doi.org/10.1016/j.physa.2023.128952.

[26] J. Dattorro, "The Implementation of Recursive Digital Filters for High-Fidelity Audio* H(z) (2)," *J. Audio Eng. Soc*, vol. 36, no. 11, 1968, Accessed: Sep. 26, 2024. [Online]. Available: https://ccrma.stanford.edu/~dattorro/HiFi.pdf

[27] "Digital Envelope Detection: The Good, the Bad, and the Ugly - Rick Lyons," *www.dsprelated.com*. https://www.dsprelated.com/showarticle/938.php (accessed Sep. 26, 2024).

[28] J. W. Bisley, "Parietal Lobe," *Encyclopedia of Animal Cognition and Behavior*, pp. 1–5, 2017, doi: https://doi.org/10.1007/978-3-319-47829-6_1252-1.

[29] C. A. Sabillon-Orellana, "Evaluation of data imputation techniques in pavement texture processing," *Utexas.edu*, Jan. 26, 2021. https://repositories.lib.utexas.edu/items/8f5dbb93-d04e-41e6-8515-f8a96d937ebc (accessed Sep. 26, 2024).

[30] "Quantile regression | GLabStat," *GLabStat*, 2019. https://www.glabstat.com/quantile-regression (accessed Sep. 26, 2024).

[31] Z. Kovács, "The Investigation of Tribological Characteristics of Surface Improved by Magnetic Polishing and Roller Burnishing," *Procedia Engineering*, vol. 149, pp. 183–189, 2016, doi: https://doi.org/10.1016/j.proeng.2016.06.654.

[32] M. D. las N. López García and J. P. Ramos Requena, "Different methodologies and uses of the hurst exponent in econophysics," *Estudios de Economía Aplicada*, vol. 37, no. 2, p. 96, Oct. 2019, doi: https://doi.org/10.25115/eea.v37i2.2603.

[33] F. J. De Mendonça, "El exponente de Hurst: ¿Conocías esta técnica estadística?" https://es.investing.com/analysis/el-exponente-de-hurst-conocias-esta-tecnica-estadistica-200458826 (accessed Sep. 26, 2024).

[34] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2005. doi: https://doi.org/10.1002/047174882x.

[35] "entropy.spectral_entropy — entropy 0.1.3 documentation," *Raphaelvallat.com*, 2018. https://raphaelvallat.com/entropy/build/html/generated/entropy.spectral_entropy.html (accessed Sep. 26, 2024).

[36] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, Jun. 1967, doi: https://doi.org/10.1109/tau.1967.1161901.

[37] "Speactral Leakage in the DFT," *w.astro.berkeley.edu*. https://w.astro.berkeley.edu/~jrg/ngst/fft/leakage.html (accessed Sep. 26, 2024).

[38] D.-J. Jwo, W.-Y. Chang, and I-Hua. Wu, "Windowing Techniques, the Welch Method for Improvement of Power Spectrum Estimation," *Computers, Materials & Continua*, vol. 67, no. 3, pp. 3983–4003, 2021, doi: https://doi.org/10.32604/cmc.2021.014752.

[39] "6.4. Spectral leakage and windowing — Digital Signals Theory," *brianmcfee.net*. https://brianmcfee.net/dstbook-site/content/ch06-dft-properties/Leakage.html (accessed Sep. 26, 2024).

[40] T. Bayen, *PYTHON PROGRAMMING AND NUMERICAL METHODS : a guide for engineers and scientists.* S.L.: Elsevier Academic Press, 2020. Available: https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html

[41] Q. Wang, "Support Vector Machine Algorithm in Machine Learning," *2022 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, Jun. 2022, doi: https://doi.org/10.1109/icaica54878.2022.9844516.

[42] "Nested versus non-nested cross-validation," *scikit-learn*, 2024. https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html#id2 (accessed Sep. 26, 2024).

[43] "3.2. Tuning the hyper-parameters of an estimator — scikit-learn 0.23.2 documentation," *scikit-learn.org*. https://scikit-learn.org/stable/modules/grid_search.html#grid-search (accessed Sep. 26, 2024).

# ANEXO A

A continuación se relaciona el código implementado durante el proyecto:

- **custom_functions.py**: agrupación de todas las funciones necesarias para el procesado del EEG de acuerdo con el protocolo del laboratorio de psicofisiología de la US, así como otras empleadas para la obtención de las *features* requeridas por la SVM.

```
# -*- coding: utf-8 -*-
from scipy import signal
import numpy as np
import mne
import os


def getFiles(data_path):
    '''
    Retrieves all subjects' recordings (BrainVision format) from the selected
    directory.

    Paramaters:
        - data_path: str
            Directory in which the recording files are.

    Returns:
        - raw: list of instances of Raw
            Raw data for each subject.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.
        -  vhdr_files: list of str
            Names of the subjects.
    '''

    # Retrieve available file names
    vhdr_files = [os.path.splitext(f)[0] for f in os.listdir(data_path)
                  if (os.path.isfile(os.path.join(data_path, f))
                      and os.path.splitext(f)[1]=='.vhdr')]
    vmrk_files = [os.path.splitext(f)[0] for f in os.listdir(data_path)
                  if (os.path.isfile(os.path.join(data_path, f))
                      and os.path.splitext(f)[1]=='.vmrk')]
    eeg_files = [os.path.splitext(f)[0] for f in os.listdir(data_path)
                  if (os.path.isfile(os.path.join(data_path, f))
                      and os.path.splitext(f)[1]=='.eeg')]

    fNum = range(len(vhdr_files))
```

```
# Raise warning if there seems to be any missing files
try:
    flag = 1
    if not all(vhdr_files[i] == vmrk_files[i] == eeg_files[i] for i in fNum):
        # Same number of .vhdr, .vmrk and .eeg files, but they don't all match
        if len(vhdr_files) == len(vmrk_files) == len(eeg_files):
            # Same dimensions --> Loop through all files
            flag = 0

            missing_vhdr = []
            for f in vmrk_files:
                if f not in vhdr_files:
                    missing_vhdr.append(f+'.vhdr')
            for f in eeg_files:
                if f not in vhdr_files:
                    missing_vhdr.append(f+'.vhdr')

            missing_vmrk = []
            for f in vhdr_files:
                if f not in vmrk_files:
                    missing vmrk.append(f+'.vmrk')
            for f in eeg_files:
                if f not in vmrk_files:
                    missing vmrk.append(f+'.vmkr')

            missing_eeg = []
            for f in vmrk_files:
                if f not in eeg_files:
                    missing_eeg.append(f+'.eeg')
            for f in vhdr_files:
                if f not in eeg files:
                    missing_eeg.append(f+'.eeg')

            if (len(missing_vhdr) + len(missing_eeg) + len(missing_vmrk)) > 1:
                print('It seems like the following files are missing:\n',
                      *missing_vhdr,'\n', *missing_vmrk,'\n', *missing_eeg)
            if (len(missing_vmrk) + len(missing_eeg) + len(missing_vmrk)) == 1:
                print('It seems like {} is missing.'. format(*missing_vhdr,
                                                             *missing_vmrk,
                                                             *missing_eeg))
except IndexError:
    # Incompatible dimensions
    pass
finally:
    if flag:
        # Incompatible dimensions --> Loop through some files
        dim = [len(vhdr_files), len(vmrk_files), len(eeg_files)]
        match dim.index(max(dim)):
```

```
            case 0:
                if ((not(len(vmrk_files) == len(eeg_files))
                    and len(vmrk_files) < len(vhdr_files)) # Just .vmrk files missing
                or len(vmrk_files) == len(eeg_files)): # .vmrk and .eeg files missing
                    missing_vmrk = [f+'.vmrk' for f in vhdr_files if f not in vmrk_files]

                if ((not(len(vmrk_files) == len(eeg_files))
                    and len(eeg_files) < len(vhdr_files)) # Just .eeg files missing
                or len(vmrk_files) == len(eeg_files)): # .vmrk and .eeg files missing
                    missing_eeg = [f+'.eeg' for f in vhdr_files if f not in eeg_files]

                if (len(missing_vmrk) + len(missing_eeg)) > 1:
                    print('It seems like the following files are missing:\n',
                          *missing_vmrk,'\n', *missing_eeg)
                if (len(missing_vmrk) + len(missing_eeg)) == 1:
                    print('It seems like {} is missing.'. format(*missing_vmrk,
                                                                 *missing eeg))
            case 1:
                if ((not(len(vhdr_files) == len(eeg_files))
                    and len(vhdr_files) < len(vmrk_files)) # Just .vhdr files missing
                or len(vhdr files) == len(eeg files)): # .vhdr and .eeg files missing
                    missing_vhdr = [f+'.vhdr' for f in vmrk_files if f not in vhdr_files]

                if ((not(len(vhdr files) == len(eeg files))
                    and len(eeg_files) < len(vmrk_files)) # Just .eeg files missing
                or len(vhdr_files) == len(eeg_files)): # .vhdr and .eeg files missing
                    missing_eeg = [f+'.eeg' for f in vmrk_files if f not in eeg_files]

                if (len(missing_vhdr) + len(missing_eeg)) > 1:
                    print('It seems like the following files are missing:\n',
                          *missing vhdr,'\n', *missing eeg)
                if (len(missing_vmrk) + len(missing_eeg)) == 1:
                    print('It seems like {} is missing.'. format(*missing_vhdr,
                                                                 *missing_eeg))
            case 2:
                if ((not(len(vmrk_files) == len(vhdr_files))
                    and len(vmrk_files) < len(eeg_files)) # Just .vmrk files missing
                or len(vmrk_files) == len(vhdr_files)): # .vmrk and .vhdr files missing
                    missing_vmrk = [f+'.vmrk' for f in eeg_files if f not in vmrk_files]

                if ((not(len(vmrk_files) == len(vhdr_files))
                    and len(vhdr_files) < len(eeg_files)) # Just .vhdr files missing
                or len(vmrk_files) == len(vhdr_files)): # .vmrk and .vhdr files missing
                    missing_vhdr = [f+'.vhdr' for f in eeg_files if f not in vhdr_files]

                if (len(missing_vmrk) + len(missing_vhdr)) > 1:
                    print('It seems like the following files are missing:\n',
                          *missing_vmrk,'\n', *missing_vhdr)
                if (len(missing_vmrk) + len(missing_eeg)) == 1:
```

```
                    print('It seems like {} is missing.'. format(*missing_vmrk,
                                                                  *missing_vhdr))

    # Retrieve raw EEG data
    vhdr_fnames = [os.path.join(data_path, vhdr_files[i]+'.vhdr') for i in fNum]
    raw = [mne.io.read_raw_brainvision(vhdr_fnames[i]) for i in fNum]

    return raw, fNum, vhdr_files



def getEvents(raw, fNum):
    '''
    Retrieves all the marked events for each recording.

    Parameters:
        - raw:  list of instances of Raw
            Raw data for each subject.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Returns:
        - events: list of ndarray
            Events for each subject.
        - event_id: dict
            Event dictionary (for cog10-5) for easier manipulation.
    '''

    try:
        events = mne.find_events(raw[0])
    except ValueError:
        events = [mne.events_from_annotations(raw[i])[0] for i in fNum]

    event_id = {'new run':99999, 'left response':1, 'right response':2,
                'left stimulus':51, 'right stimulus':52,
                'left standard':53, 'right standard':54}

    return events, event_id



def remove_DC(raw, fNum):
    '''
    Removes DC offset from un-epoched data.

    Parameters:
        - raw: list of instances of Raw
```

```
            Raw data for each subject.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Returns:
        - raw: list of instances of Raw
            Baseline-fixed raw data for each subject.
    '''

    raw = [mne.io.RawArray(mne.baseline.rescale
                           (raw[i].get_data(), raw[i].times, baseline=(None,0.2),
                            mode='mean'), raw[i].info) for i in fNum]

    return raw



def artifact_correction(raw, fNum):
    '''
    Ocular artifacts correction with ICA.

    Parameters:
        - raw: list of instances of Raw
            Raw data for each subject.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Returns:
        - reconst_raw: list of instances of Raw
            Recontructed data without ocular component.
    '''

    # Fitting ICA
    ica = [mne.preprocessing.ICA(n_components=raw[i].info['nchan'],
                                 max_iter='auto') for i in fNum]
    ica = [ica[i].fit(raw[i]) for i in fNum]

    for i in fNum:
        raw[i].load_data()

    # Re-reference vertical ocular electrodes
    raw_bip = [mne.set_bipolar_reference(
        raw[i], 'Veog+', 'Veog_', ch_name='VEOG', drop_refs=False) for i in fNum]

    # Detect VEOG related components (returns index and scores for each iteration)
    veog = [ica[i].find_bads_eog(raw_bip[i], ch_name='VEOG')
                             for i in fNum]
```

```
    # Reconstruction of the data without the ocular component
    for i in fNum:
        ica[i].exclude = veog[i][0]

    reconst_raw = [raw[i].copy() for i in fNum]
    reconst_raw = [ica[i].apply(reconst_raw[i]) for i in fNum]

    return reconst_raw



def segmentation(raw_filt, tmin, tmax, events, event_id, fNum):
    '''
    Segmentation of recordings into different group of epochs based on the
    stimuli of interest.

    Parameters:
        - raw: list of instances of Raw
            Raw data for each subject.
        - tmin: float
            Starting time of the epochs in seconds, relative to the event.
        - tmax: float
            Ending time of the epochs in seconds, relative to the event.
         - events: list of ndarray
             Events for each subject.
         - event_id: dict
             Event dictionary (for cog10-5) for easier manipulation.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Returns:
        - epochs_left: list of Epoch
            Epochs centered around the left stimulus.
        - epochs_right: list of Epoch
            Epochs centered around the right stimulus.
    '''

    # Segmentation
    epochs = []
    for i in fNum:
        epochs.append(mne.Epochs(raw_filt[i], events[i], event_id, tmin=tmin,
                                 tmax=tmax, baseline=None))

    # Separate epochs by condition
    epochs_left = [epochs[i]['left stimulus'] for i in fNum]
    epochs_right = [epochs[i]['right stimulus'] for i in fNum]
```

```
    # Drop VEOG so that it will not be used in peak selection when
    # solving latencies
    for i in fNum:
        epochs_left[i].load_data()
        epochs_right[i].load_data()
        epochs_left[i].drop_channels(['Veog+','Veog_','Heog+','Heog_'])
        epochs_right[i].drop_channels(['Veog+','Veog_','Heog+','Heog_'])

    return epochs_left, epochs_right



def artifact_rejection(x, threshold, time, ch, tmin, fNum):
    '''
    Rejects epochs that bypassed the ICA algorithm.

    Parameters:
        - x : list of Epoch
            Epochs to be examined for artifact rejection after artifact
            correction.
        - threshold : float
            Voltage amplitude beyond which any epoch will be discarded.
        - time : list of float
            Interval in which voltage amplitude is to be examined.
        - ch : list of str
            Channels to apply the threshold condition to.
        - tmin: float
            Starting time of the epochs in seconds, relative to the event.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Returns:
        - y : list of Epoch
            Epochs after performing artifact rejection.
    '''

    # Data required to determine artifact rejection needs
    reference = [x[i].get_data(picks=ch, tmin=time[0], tmax=time[1])
                 for i in fNum]

    nTrials = [reference[i].shape[0] for i in fNum]

    # Original segmentation data (without dropped trials)
    full_data = [x[i].get_data() for i in fNum]

    # Segmentation data (after dropped trials)
    y = []
    for i in fNum:
        for j in range(nTrials[i]):
```

```
            if (np.abs(reference[i][j]) > threshold).any():
                full_data[i] = np.delete(full_data[i], j, axis=0)
        y.append(mne.EpochsArray(full_data[i], x[i].info, tmin = tmin))

    return y



def bandpassFilter(x, f_cutoff, fNum):
    '''
    Selects specified frequency band by lowpass-filtering the input data first
    & highpass-filtering the result afterwards.

    Parameters:
        - x: list of Evoked | list of Epoch
            Data for each subject.
        - f_cutoff: ndarray
            Cutoff frequencies.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Reutrns:
        - x: list of Evoked | list of Epoch
            Filtered data for each subject.
    '''

    # Filter parameters
    iir_params = dict(order=8, ftype='butter', output='sos')

    # Bandpass filtering
    x = [x[i].filter(l freq=None,h freq=f cutoff[1], method='iir',
                     iir_params=iir_params, phase='zero') for i in fNum]

    x = [x[i].filter(l_freq=f_cutoff[0],h_freq=None, method='iir',
                     iir_params=iir_params, phase='zero') for i in fNum]

    return x



def abs(x, tmin, fNum):
    '''
    Rectifies the input.

    Parameters:
        - x: list of Epoch | list of Evoked
            Data to be rectified.
```

```
        - tmin: float
            Starting time of the epochs in seconds, relative to the event.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Returns:
        - y: list of EpochsArray | list of EvokedArray
            Rectified data.
    '''

    y = []

    if type(x[0])==mne.epochs.Epochs or type(x[0])==mne.epochs.EpochsArray:
        for i in fNum:
            ep = x[i].__len__()
            e = np.ndarray([ep,x[i].info['nchan'],len(x[i].times)])
            for j in range(ep):
               e[j] = np.abs(x[i].get_data()[j])
            y.append(mne.EpochsArray(e, x[i].info, tmin = tmin))
    else:
        y = [mne.EvokedArray(np.abs(x[i].get_data()), x[i].info, tmin = tmin)
             for i in fNum]
    return y



def lowpassFilter(x, fNum):
    '''
    Lowpass filters input data at 5 Hz.

    Parameters:
        - x: list of Evoked
            Data for each subject.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Reutrns:
        - x: list of Evoked
            Filtered data for each subject.
    '''
    # Filter parameters
    iir_params = dict(order=8, ftype='butter', output='sos')

    # Filtering
    x = [x[i].filter(0,5, method='iir', iir_params=iir_params,
                     phase='zero') for i in fNum]

    return x
```

```
def shorten(x, tmin, tmax, fNum):
    '''
    Shortens epochs to specified length.

    Parameters:
        - x: list of Epoch
            Epochs to be shortened.
        - tmin: float
            Starting time of the epochs in seconds, relative to the event.
        - tmax: float
            Ending time of the epochs in seconds, relative to the event.
        - fNum: range
            Corresponds to the number of subjects and therefore will be used to
            loop through them.

    Returns:
        - short: list of Epoch
            Shortened epochs.
    '''

    mat = [x[i].get_data(tmin=tmin,tmax=tmax) for i in fNum]

    short = [mne.EvokedArray(mat[i], info=x[i].info) for i in fNum]

    return short



def peak(x):
    '''
    Computes the latency of the evoked response.

    Parameters:
        - x: Evoked
            Evoked response whose latency is to be determined.

    Returns:
        - latency: float
            Latency.
        - amplitude: float
            Amplitude.
    '''

    # Extract time segment of interest
    mat = x.get_data(tmin=0.02,tmax=1.5)
```

```
    # Samples among t = 0 & t = 0.02
    time_increment = x.time_as_index(0.02)

    latency = []
    amplitude = []

    # Search for peaks
    peaks, dic = signal.find_peaks(mat[0], height=-150e-6, prominence=0.01e-6)

    # Index to seconds & shift to input data's time reference
    peaks_seconds = []
    for j in range(len(peaks)):
        peaks_seconds.append(x.times[peaks[j] + time_increment])

    try:
        # Only intrested in the fisrt peak
        latency = peaks_seconds[0][0]

        # Amplitude
        amplitude = dic['peak heights'][0]
    except:
        pass

    return latency, amplitude
```

- **evoked_pipeline.py**: simulación del protocolo de procesado de datos del EEG del laboratorio de psicofisiología para hallar la media y la respuesta evocada (envolvente de la media) tanto para los sujetos control como para los pacientes.

```
# -*- coding: utf-8 -*-
from custom_functions import getFiles, getEvents, remove_DC, artifact_correction
from custom_functions import segmentation, artifact_rejection, bandpassFilter
from custom_functions import abs, lowpassFilter, shorten
import numpy as np
import mne
import os


# Control & MS subjects' datasets directories
datasets = {'control':r'C:\Users\balsa\OneDrive\Documentos\brainstorm_unzip\sujetos_rocio',
            'ms':r'C:\Users\balsa\OneDrive\Documentos\brainstorm_unzip\sujetos_rocio_ms'}

# Pipeline epoch interval
tmin_aux = -1.2
tmax_aux = 2

# Interval for SVM features
tmin = 0
tmax= 1.5

# Frequency bands
delta = np.array([0.5,4])
theta = np.array([4,8])
alpha = np.array([8,12])
beta = np.array([12,35])
gamma = np.array([35,45])
bands = {"delta": delta,"theta": theta,"alpha": alpha,"beta": beta,
         'gamma':gamma}


for data_path in datasets:

    # Create directories to store some intermidiate and final processed data
    try:
        os.mkdir(fr'{datasets[data_path]}\analysis')
        os.mkdir(fr'{datasets[data_path]}\analysis')
    except FileExistsError:
        pass

    try:
        os.mkdir(fr'{datasets[data_path]}\analysis\envelope')
        os.mkdir(fr'{datasets[data_path]}\analysis\average')
```

```
    os.mkdir(fr'{datasets[data_path]}\analysis\raw')
except FileExistsError:
    pass

'''
Psychology laboratory protocol pipeline for evoked response:
'''
#%% Retrieve all subjects' raw data files (BrainVision)

raw, fNum, vhdr_files = getFiles(datasets[data_path])

#%% Event dictionary (for cog10-5)

events, event_id = getEvents(raw, fNum)

#%% DC offset correction

raw_noDC = remove_DC(raw,fNum)

#%% ICA

# First time:
reconst_raw = artifact_correction(raw_noDC,fNum)

# # Rest of the times:
# reconst_raw = [mne.io.read_raw_fif(
#     fr'{datasets[data_path]}\analysis\raw\{i}_raw.fif') for i in fNum]
# for i in fNum: reconst_raw[i].load_data()

#%% Segmentation

epochs_left, epochs_right = segmentation(reconst_raw,tmin_aux,tmax_aux,
                                         events,event_id,fNum)

#%% Baseline correction

epochs_left_baseline = [epochs_left[i].copy() for i in fNum]
epochs_right_baseline = [epochs_right[i].copy() for i in fNum]

epochs_left_baseline = [epochs_left_baseline[i].apply_baseline(baseline=(-0.2,0))
                    for i in fNum]
epochs_right_baseline = [epochs_right_baseline[i].apply_baseline(baseline=(-0.2,0))
                     for i in fNum]

#%% Artifact rejection

threshold = 150e-6 # in V
ch = ['Pz', 'Cz', 'Fpz']
time = [0, 300e-3] # in seconds
```

```
epochs_left_reject = artifact_rejection(epochs_left_baseline, threshold, time,
                                        ch, tmin_aux, fNum)
epochs_right_reject = artifact_rejection(epochs_right_baseline, threshold, time,
                                         ch, tmin_aux, fNum)

#%% Average for each channel

avg_left = [epochs_left_reject[i].average(method='mean') for i in fNum]
avg_right = [epochs_right_reject[i].average(method='mean') for i in fNum]

#%% for:

for f_cutoff in bands:
    #% Bandpass filter

    avg left filt = [avg left[i].copy() for i in fNum]
    avg_right_filt = [avg_right[i].copy() for i in fNum]

    avg_left_filt = bandpassFilter(avg_left_filt,bands[f_cutoff],fNum)
    avg right filt = bandpassFilter(avg right filt,bands[f cutoff],fNum)


    #%% Envelope: rectify & 5 Hz LP filter

    rect_left = abs(avg_left_filt, tmin_aux, fNum)
    rect_right = abs(avg_right_filt, tmin_aux, fNum)

    env_left = [rect_left[i].copy() for i in fNum]
    env_right = [rect_right[i].copy() for i in fNum]

    env_left = lowpassFilter(env_left,fNum)
    env_right = lowpassFilter(env_right,fNum)

    #%% Baseline correction

    evoked_left = [env_left[i].copy() for i in fNum]
    evoked_right = [env_right[i].copy() for i in fNum]

    evoked_left = [evoked_left[i].apply_baseline(baseline=(-0.2,0)) for i in fNum]
    evoked_right = [evoked_right[i].apply_baseline(baseline=(-0.2,0)) for i in fNum]

    #%% Only need the 0-1.5s interval

    evoked_left = shorten(evoked_left,tmin,tmax,fNum)
    evoked_right = shorten(evoked_right,tmin,tmax,fNum)
    avg_left_filt = shorten(avg_left_filt,tmin,tmax,fNum)
    avg_right_filt = shorten(avg_right_filt,tmin,tmax,fNum)
```

```
    #%% Store data

    # Store evoked response (envelope of the average) & average
     for i in fNum:
        evoked_left[i].save(
            fr'{datasets[data_path]}\analysis\envelope\{f_cutoff}_E_left_{i}_ave.fif',
            overwrite=True)
        evoked_right[i].save(
            fr'{datasets[data_path]}\analysis\envelope\{f_cutoff}_E_right_{i}_ave.fif',
            overwrite=True)
        avg_left_filt[i].save(
            fr'{datasets[data_path]}\analysis\average\{f_cutoff}_left_{i}_ave.fif',
            overwrite=True)
        avg_right_filt[i].save(
            fr'{datasets[data_path]}\analysis\average\{f_cutoff}_right_{i}_ave.fif',
            overwrite=True)

# Store reconstructed data (given that ICA takes relatively long to run)
for i in fNum:
    reconst_raw[i].save(fr'{datasets[data_path]}\analysis\raw\{i}_raw.fif',
                        overwrite=True)
```

- **features.py**: cálculo de las *features* y arreglo de las matrices de datos y etiquetas.

```
# -*- coding: utf-8 -*-
from mne_features.univariate import compute_hurst_exp
from custom_functions import getFiles, peak
from scipy.stats import kurtosis, skew
from scipy.integrate import simpson
from scipy import signal
import antropy as ant
import numpy as np
import pickle
import mne


# Control & MS subjects' datasets directories
datasets = {'control':r'C:\Users\balsa\OneDrive\Documentos\brainstorm_unzip\sujetos_rocio',
            'ms':r'C:\Users\balsa\OneDrive\Documentos\brainstorm_unzip\sujetos_rocio_ms'}

# Frequency bands
delta = np.array([0.5,4])
theta = np.array([4,8])
alpha = np.array([8,12])
beta = np.array([12,35])
gamma = np.array([35,45])
bands = {"delta": delta,"theta": theta,"alpha": alpha,"beta": beta,
         'gamma':gamma}

stimuli = ['left','right']

evoked_dic = {} # Evoked response (envelope of the average)
avg_dic = {} # Average response
amp_dic = {}
lat_dic = {}
var_dic = {}
kurtosis_dic = {}
skewness_dic = {}
hurst_dic = {}
entropy_dic = {}
power_dic = {}


for data_path in datasets:

    #%% Data loading for each data path, stimulus, band & subject

    evoked_dic[data_path] = {}
    avg_dic[data_path] = {}
```

```
amp_dic[data_path] = {}
lat_dic[data_path] = {}
var_dic[data_path] = {}
kurtosis_dic[data_path] = {}
skewness_dic[data_path] = {}
hurst_dic[data_path] = {}
entropy_dic[data_path] = {}
power_dic[data_path] = {}

raw, fNum, vhdr_files = getFiles(datasets[data_path])

for stim in stimuli:

    evoked_dic[data_path][stim] = {}
    avg_dic[data_path][stim] = {}
    amp_dic[data_path][stim] = {}
    lat dic[data path][stim] = {}
    var_dic[data_path][stim] = {}
    kurtosis_dic[data_path][stim] = {}
    skewness_dic[data_path][stim] = {}
    hurst dic[data path][stim] = {}
    entropy_dic[data_path][stim] = {}
    power_dic[data_path][stim] = {}

    for f_cutoff in bands:

        evoked_dic[data_path][stim][f_cutoff] = []
        avg_dic[data_path][stim][f_cutoff] = []
        amp_dic[data_path][stim][f_cutoff] = []
        lat_dic[data_path][stim][f_cutoff] = []
        var dic[data path][stim][f cutoff] = []
        kurtosis_dic[data_path][stim][f_cutoff] = []
        skewness_dic[data_path][stim][f_cutoff] = []
        hurst_dic[data_path][stim][f_cutoff] = []
        entropy_dic[data_path][stim][f_cutoff] = []
        power_dic[data_path][stim][f_cutoff] = []

        for i in fNum:
            evoked_dic[data_path][stim][f_cutoff].append(mne.read_evokeds(
                fr'{datasets[data_path]}\analysis\envelope\{f_cutoff}_E_{stim}_{i}_ave.fif',
                condition=0))
            avg_dic[data_path][stim][f_cutoff].append(mne.read_evokeds(
                fr'{datasets[data_path]}\analysis\average\{f_cutoff}_{stim}_{i}_ave.fif',
                condition=0))

        #%% Features

            # Select desired channel
            sfreq = evoked_dic[data_path][stim][f_cutoff][i].info['sfreq']
```

```
                evoked_dic[data_path][stim][f_cutoff][i] =
evoked_dic[data_path][stim][f_cutoff][i].pick('Pz')
                avg_dic[data_path][stim][f_cutoff][i] = avg_dic
[data_path][stim][f_cutoff][i].pick('Pz')

                # Amplitude & latency
                lat, amp = peak(evoked_dic[data_path][stim][f_cutoff][i])

                lat_dic[data_path][stim][f_cutoff].append(lat)
                amp_dic[data_path][stim][f_cutoff].append(amp)

                # Variance
                avg = avg_dic[data_path][stim][f_cutoff][i].get_data()

                var_dic[data_path][stim][f_cutoff].append(np.var(avg[0]))

                # Kurtosis
                kurtosis_dic[data_path][stim][f_cutoff].append(kurtosis(avg[0]))

                # Skewness
                skewness dic[data path][stim][f cutoff].append(skew(avg[0]))

                # Hurst exponent
                hurst dic[data path][stim][f cutoff].append(compute hurst exp(avg)[0])

                # Spectral entropy
                entropy_dic[data_path][stim][f_cutoff].append(
                    ant.spectral_entropy(avg[0], sf=sfreq, method='welch', normalize=True))

                # Power
                freqs, psd = signal.welch(avg[0])
                 power_dic[data_path][stim][f_cutoff].append(simpson(y=psd, x=freqs))

#%% Feature & label matrixes

X = []
y = []

# Control
for i in fNum:
    bands_per_stim = []
    for stim in stimuli:
        features_per_band = []
        for f_cutoff in bands:
            features_per_band.append(np.array([amp_dic['control'][stim][f_cutoff][i],
            lat_dic['control'][stim][f_cutoff][i],
            var_dic['control'][stim][f_cutoff][i],
            kurtosis_dic['control'][stim][f_cutoff][i],
```

```
            skewness_dic['control'][stim][f_cutoff][i],
            hurst_dic['control'][stim][f_cutoff][i],
            entropy_dic['control'][stim][f_cutoff][i],
            power_dic['control'][stim][f_cutoff][i]], dtype=object))

        bands_per_stim.append(features_per_band)
    X.append(bands_per_stim)
    y.append('control')

# MS
for i in fNum:
    bands_per_stim = []
    for stim in stimuli:
        features_per_band = []
        for f_cutoff in bands:
            features_per_band.append(np.array([amp_dic['ms'][stim][f_cutoff][i],
            lat dic['ms'][stim][f cutoff][i],
            var_dic['ms'][stim][f_cutoff][i],
            kurtosis_dic['ms'][stim][f_cutoff][i],
            skewness_dic['ms'][stim][f_cutoff][i],
            hurst dic['ms'][stim][f cutoff][i],
            entropy_dic['ms'][stim][f_cutoff][i],
            power_dic['ms'][stim][f_cutoff][i]], dtype=object))

        bands_per_stim.append(features_per_band)
    X.append(bands_per_stim)
    y.append('ms')

X = np.array(X, dtype=object)
y = np.array(y)

# Drop samples with non-existent features
index = []
drop_flag = 0
for i in range(len(X)):
    for j in range(len(X[i])):
        for k in range(len(X[i][j])):
            if not(all(X[i][j][k])):
                drop_flag = 1
    if drop_flag:
        index.append(i)
        drop_flag = 0
X = np.delete(X,index,axis=0)
y = np.delete(y,index,axis=0)

# Store final dataset
with open('features_dataset', 'wb') as fp:
  pickle.dump(X, fp)
with open('labels_dataset', 'wb') as fp:
```

```
  pickle.dump(y, fp)
```

- **svm.py**: implementación de la máquina de soporte vectorial y comprobación de su exactitud.

```
# -*- coding: utf-8 -*-
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import numpy as np
import pickle

# Load processed dataset
with open('features_dataset ', 'rb') as fp:
  X = pickle.load(fp)
with open('labels_dataset ', 'rb') as fp:
  y = pickle.load(fp)

n_trials = 100
accuracy = []

for n in range(n_trials):
    # Data subsets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y)

    # SVM only accepts 2-D feature matrixes
    X_train = X_train.reshape(len(X_train),-1)
    X_test = X_test.reshape(len(X_test),-1)

    # Standardization and SVM algorithm
    pipe = make_pipeline(StandardScaler(), SVC(kernel='linear'))

    #%% Hyperparameter tuning

    param_grid = [
        {"svc__kernel": ["rbf"], "svc__gamma": [1e-3, 1e-4], "svc__C": [1, 10, 100, 1000]},
        {"svc__kernel": ["linear"], "svc__C": [1, 10, 100, 1000]}]

    # Inner loop for hyperparameter tuning
    gs = GridSearchCV(pipe, param_grid, scoring='accuracy', cv=2, refit=True)

    # Outer loop for model evaluation
    scores = cross_val_score(gs, X_train, y_train, scoring='accuracy', cv=5,
                             error_score='raise', n_jobs=-1)

    # Accuracy of each model
    print(scores,'\n')
```

```
    # Average accuracy & standard deviation
    print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(),
                                                                 scores.std()))

    #%% Final model

    # Fit final model on all training data with the optimal hyperparameters
    gs.fit(X_train, y_train)
    print('Optimal hyperparameters:\n', gs.best_params_,'\n')
    print('Optimal estimator:\n', gs.best_estimator_,'\n\n')

    # Evaluate performance of model on test data
    y_pred = gs.predict(X_test)
    accuracy.append(accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))

    # Confusion matrix
    mat = confusion_matrix(y_true=y_test, y_pred=y_pred)
    print(mat,'\n')
    fig, ax = plt.subplots(figsize=(3.5, 3.5))
    ax.matshow(mat, cmap=plt.cm.Greens, alpha=0.4)
    for i in range(mat.shape[0]):
        for j in range(mat.shape[1]):
            ax.text(x=j, y=i,
                    s=mat[i, j],
                    va='center', ha='center')
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.show()

# Average accuracy & standard deviation
print("\n%0.2f accuracy with a standard deviation of %0.2f" % (np.mean(accuracy),
                                                               np.std(accuracy)))
```