Universidad Nacional de Córdoba
Facultad de Ciencias Exactas, Físicas y Naturales
Facultad de Ciencias Médicas
Escuela de Ingeniería Biomédica

Estudio comparativo de algoritmos de clasificación de señales electroencefalográficas en el paradigma del deletreador P300

María Sofía Sappia

Asesor: Dr. Elmer A. Fernández

Co-asesora: Ing. Florencia Garro

Córdoba, Diciembre de 2018

Agradecimientos

A mis directores, Florencia Garro y Elmer Fernández, por su invaluable aporte de conocimientos y herramientas que me permitieron llevar a cabo este proyecto.

A mi familia, por brindarme el apoyo y sustento necesarios durante la carrera y el desarrollo de este proyecto. En particular a mi madre y a mi hermana Paula, por escucharme y aconsejarme; a mi padre, por su ayuda al conseguir herramientas útiles para el desarrollo de la carrera; a mi abuela; por su paciencia durante los meses de estudio intensivo; y a mi hermano Mariano, por su desinteresado servicio de consultas presenciales y a distancia en todo lo concerniente a software.

A mis amistades, por haberme siempre acompañado durante este proceso y por haberme ayudado a relajarme y desconectarme en los momentos de estrés en que nada más podía hacerse. En particular a Yael, fiel compañera de carrera; y a Amanda y Maia, que me supieron escuchar y aconsejar durante este proyecto, en los momentos en que más lo necesitaba.

A la Universidad Nacional de Córdoba, casa de altos estudios que me ha brindado educación pública, gratuita y de calidad no sólo durante mi formación universitaria, sino también secundaria. En particular, a aquella juventud de jóvenes estudiantes de 1918, gracias a cuya lucha se consagró la Reforma Universitaria que hoy, habiéndose cumplido ya 100 años, continúa permitiendo que esto sea posible y me recuerda que también hoy, más que nunca, es necesario seguir luchando para que las futuras generaciones de estudiantes que deseen ejercer su derecho a una Educación Pública, Gratuita y de Calidad, puedan hacerlo.

Resumen

Afecciones como la Esclerosis Lateral Amiotrófica o el Síndrome de Enclaustramiento generan, en los pacientes, limitaciones motrices severas, en que el sujeto es incapaz de realizar movimientos musculares, pero conserva habilidades cognitivas. Si bien existen diversas formas alternativas de comunicación, como sistemas basados en comandos de voz o gestos, no son apropiados para aquellos individuos que presentan afecciones neuromusculares severas. Las Interfaces Cerebro Computadora (BCIs, del inglés) surgen, entonces, como herramientas de comunicación alternativa que, sin necesidad de recurrir a la actividad de nervios periféricos o músculos para discernir y ejecutar la acción deseada, pueden valerse de señales provenientes del sistema nervioso central, estableciendo un camino directo entre la actividad cerebral de una persona y el mundo externo.

Uno de los principales desafíos a la hora de implementar esta tecnología radica en la clasificación correcta y eficiente de los patrones cerebrales utilizados como comando, debido a que presentan una gran variabilidad. Para sortearlos, habitualmente se recurre al uso de algoritmos de aprendizaje automático que detectan y clasifican estos patrones. En el presente Proyecto Integrador, se lleva a cabo el estudio de algoritmos de clasificación de señales electroencefalográficas aplicados al Deletrador P300. Éste consiste en un sistema BCI compuesto por una pantalla donde se disponen los caracteres a deletrear en forma matricial, y se permite al usuario, a partir de la intensificación aleatoria de las filas y columnas de la matriz, seleccionar un carácter. El sistema se basa en la detección del potencial evocado P300, para lo cual es necesaria la implementación de algoritmos de clasificación automática.

Los objetivos específicos de investigación son la implementación del paradigma del deletrador P300 en sus etapas de entrenamiento y predicción, aplicando algoritmos de clasificación; el estudio comparativo de su desempeño y capacidad predictiva; y la evaluación de la disminución del tiempo de selección del carácter durante la calibración y el deletreo.

El estudio fue realizado en dos etapas: 1) correspondiente a la clasificación binaria, en que se discrimina entre los estímulos que producen un P300 y los que no; 2) predicción de caracteres en un entorno online simulado. Se experimentó con los siguientes algoritmos: Análisis Discriminante Lineal (LDA), Máquina de Vectores de Soporte (SVM) Lineal y Gausiana, Perceptrón Multicapa y Redes Neuronales Convolucionales. Los mejores desempeños se obtuvieron para los tres primeros, que constituyen clasificadores lineales y que lograron una tasa de detección de caracteres comprendida entre el 80 % y 90 %, variando de acuerdo a los sujetos evaluados.

Palabras clave: P300, Interfaz Cerebro Computadora, EEG, Máquina de Vectores de Soporte, Análisis Discriminante Lineal, Perceptrón Multicapa, Redes Neuronales, Redes Convolucionales.

Simplemente para mí lo fantástico es la indicación súbita de que, al margen de las leyes aristotélicas y de nuestra mente razonante, existen mecanismos perfectamente válidos, vigentes, que nuestro cerebro lógico no capta, pero que en algunos momentos irrumpen y se hacen sentir.

Julio Florencio Cortázar

Índice general

Índice de figuras

Índice de tablas

Acrónimos

BCI Interfaz Cerebro Computadora.

CNN Red Neuronal Convolucional.

EEG Electroencefalograma.

EP Potencial Evocado.

ERP Potencial Relacionado a Eventos.

ITR Tasa de Transferencia de Información.

LDA Análisis Discriminante Lineal.

MLP Perceptrón Multicapa.

SGD Descenso de Gradiente Estocástico.

SNC Sistema Nervioso Central.

SNP Sistema Nervioso Periférico.

SVM Máquina de Vectores de Soporte.

TC Totalmente Conectadas.

Introducción

Motivación

Una Interfaz Cerebro Computadora (BCI, por sus siglas en inglés) puede constituir el único medio de comunicación para individuos que sufren limitaciones motrices severas, como sucede en el caso de pacientes con Esclerosis Lateral Amiotrófica o con síndrome de enclaustramiento. Si bien existen diversas formas alternativas de comunicación para personas con discapacidad, como sistemas basados en comandos de voz o gestos, no son apropiados para aquellos individuos que presentan afecciones neuromusculares severas. En estos casos, el sujeto es normalmente incapaz de realizar prácticamente movimiento muscular alguno, pero conserva habilidades cognitivas [31]. Ante esto, surgen las interfaces cerebro computadora como herramientas de comunicación que, sin necesidad de recurrir a la actividad de nervios periféricos o músculos para discernir y ejecutar la acción deseada, pueden valerse de señales provenientes del sistema nervioso central, estableciendo un camino directo entre la actividad cerebral de una persona y el mundo externo [38].

El principio de funcionamiento de una BCI consiste, normalmente, en la adquisición y análisis de señales electroencefalográficas, y su traducción a un comando, que es utilizado para controlar un determinado sistema. Diversos tipos de señales son empleados en los sistemas BCI, entre las cuales se puede citar el potencial P300. Éste constituye un Potencial Relacionado a Eventos (ERP, del inglés), asociado a estímulos inesperados que proporcionan información relevante para la tarea que el usuario está desarrollando. Sin embargo, la detección de este potencial es actualmente ineficaz e ineficiente, lo que imposibilita su utilización en un dispositivo de uso cotidiano dada su falta de confiabilidad.

La interfaz propuesta en este proyecto corresponde, en particular, al deletreador P300. En él, el uso del P300 permite al usuario seleccionar caracteres alfanuméricos de manera secuencial, al focalizar su atención durante varios segundos sobre el carácter deseado. Los distintos caracteres se encuentran agrupados en una matriz a modo de teclado, visualizada en la pantalla de una computadora. Sus filas y columnas son repetidamente resaltadas de manera aleatoria, a fin de provocar una respuesta de tipo P300 en aquellas que contengan el carácter deseado. Éste puede, entonces, ser identificado al determinar la fila y columna en que el sujeto dedica su atención, caracterizadas por la presencia del potencial P300 [22].

La motivación de este proyecto surge en el marco de la empresa OTTAA Project, responsable del desarrollo de un sistema aumentativo y alternativo de comunicación basado en el uso de pictogramas. Este sistema, sin embargo, consiste en una interfaz táctil, por lo cual su uso queda limitado a personas que cuentan con algún grado de capacidad motriz. Actualmente, la empresa se encuentra en desarrollo de un módulo de adquisición de señales electroencefalográficas, que permita establecer una interfaz entre la aplicación desarrollada y la actividad cerebral; pudiendo constituir una herramienta alternativa de comunicación para aquellos usuarios que no puedan beneficiarse de las ya disponibles. Es en este contexto que surge el presente trabajo

de investigación, que constituye una instancia inicial en el desarrollo de tal interfaz, abordando el análisis e implementación de diferentes algoritmos de clasificación de señales electroencefalográficas para la detección del potencial P300, aplicados a la interfaz clásica del deletreador. Los principales problemas de estos sistemas son: 1) su lentitud, por lo cual resultan agotadores para el usuario; 2) su baja precisión. Estas dos dificultades generan frustración y cansancio en el usuario, por lo cual resulta de baja usabilidad. Se pretende disminuir estos inconvenientes a partir de la implementación de algoritmos que resulten precisos y que, a su vez, precisen de un tiempo menor de selección del carácter, aumentando la usabilidad del sistema.

Objetivos generales

El objetivo general del Proyecto Integrador reside en el análisis e implementación de algoritmos de clasificación de señales electroencefalográficas para interfaces cerebro-computadora en el contexto del paradigma del deletreador P300, y la evaluación de su poder predictivo dentro de un entorno controlado.

Objetivos particulares

- Implementación del paradigma del deletreador P300 en sus etapas de entrenamiento y predicción, empleando como interfaz una matriz alfanumérica de 6 filas y 6 columnas.
- Implementación de algoritmos clasificadores en la etapa de entrenamiento del paradigma.
- Estudio comparativo del desempeño y capacidad predictiva de los algoritmos, simulando una sesión en línea a partir de la utilización de una base de datos.
- Evaluación de la disminución del tiempo de selección del carácter, tanto en la etapa de calibración como de deletreo.

Capítulo 1

Marco teórico

1.1. Interfaces Cerebro Computadora

Las Interfaces Cerebro Computadora (BCI, del inglés Brain Computer Interface) son sistemas que interpretan la actividad cerebral del usuario a fin de controlar una acción o grupo de acciones. Proporcionan a sus usuarios canales de comunicación y control que remplazan o aumentan los canales de control normales del cuerpo humano, como los músculos o nervios periféricos. Por lo tanto, existe actualmente gran interés en su desarrollo, debido a que estas tecnologías pueden constituir una herramienta de comunicación aumentativa y/o alternativa para aquellas personas que presentan discapacidades motrices severas, que les impiden utilizar tecnologías asistivas ya que requieren algún grado de control muscular voluntario [69].

Se conciben, entonces, como sistemas de comunicación y/o control que permiten una interacción en tiempo real entre el cerebro humano y dispositivos externos. La intención del usuario está representada por las señales cerebrales, que constituyen la entrada al sistema y son traducidas por éste a una salida deseada (comando o comunicación). Una BCI mide y analiza señales cerebrales, convirtiéndolas en tiempo real a salidas que no dependen de músculos o nervios periféricos, siendo estos reemplazados con *hardware* y *software*. En consecuencia, quedan excluidos de la definición aquellos sistemas basados en la medición de señales mioeléctricas [36].

En general, un sistema BCI consta de cuatro bloques constitutivos: 1) adquisición de señales, 2) procesamiento de señales, 3) dispositivo de salida y 4) retroalimentación, que puede o no estar presente. El módulo de procesamiento de señales está subdividido, a su vez, en otros tres bloques: preprocesamiento, extracción de características y clasificación (figura 1.1).

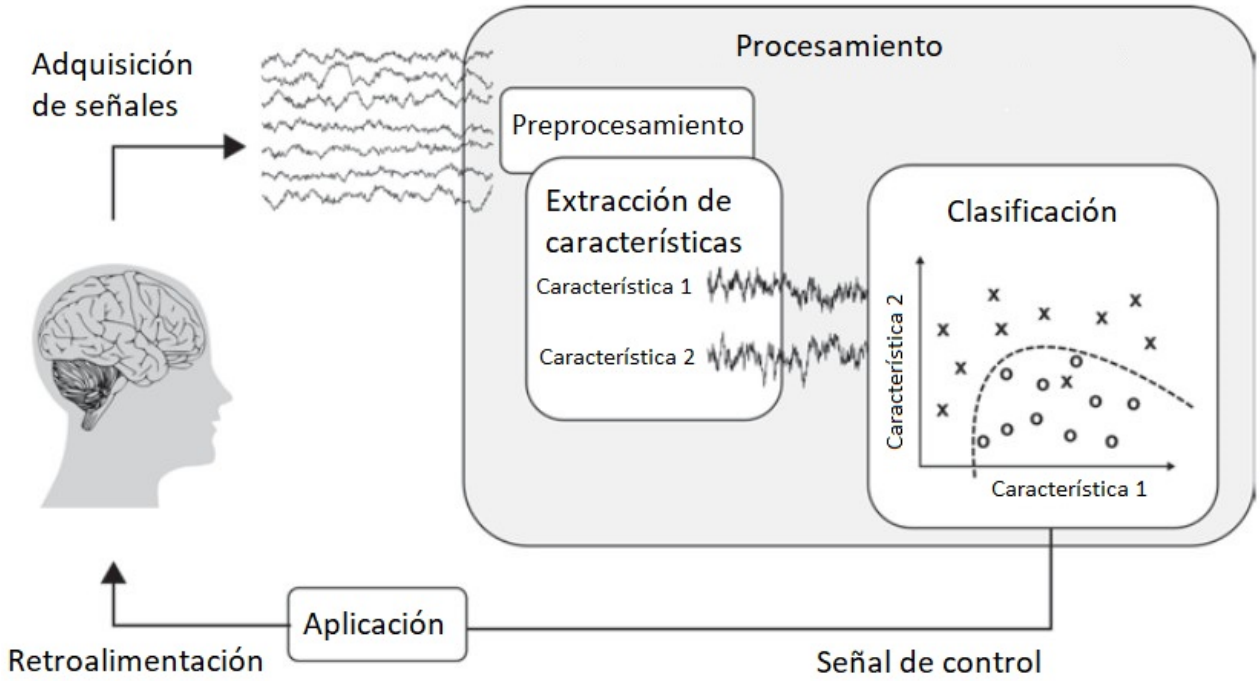


Figura 1.1: Bloques constitutivos de una Interfaz Cerebro Computadora.[1]

1.1.1. Fundamentos Biológicos

En esta sección se describen los conceptos básicos de la anatomía y fisiología del cerebro humano, encargado de generar las señales de las que se nutre la BCI en estudio.

1.1.1.1. La neurona

Una neurona es una célula dotada de la capacidad de transferir información de una parte del cuerpo a otra en forma de impulsos electroquímicos, y es generalmente concebida como la unidad funcional básica del sistema nervioso. Las neuronas de distintas regiones del cuerpo poseen estructuras morfológicas diferentes, pero todas están constituidas por tres unidades funcionales:

- *Cuerpo neuronal* o *soma*: tiene un núcleo con información genética y un plasma que sustenta los componentes moleculares usados para producir el material necesitado por la célula.
- *Dendritas*: reciben las señales producidas por otras neuronas y las transmiten al soma.
- *Axón*: recibe las señales del soma y las transmite mediante sinapsis a las dendritas de neuronas vecinas.

Estas células residen en un medio acuoso con una mayor concentración de iones sodio (Na^+), cloruro (Cl^-) y calcio (Ca^+), y menor concentración de potasio (K^+) e iones orgánicos respecto del interior de la célula. Como resultado de este desbalance, existe una diferencia de potencial de aproximadamente -65 o -70 mV a través de la membrana celular, llamado *potencial de reposo*, que es mantenido a partir de bombas activas situadas en ella.

[1] Imagen extraída de [48].

1.1.1.1.1 Potencial de Acción

La membrana de una neurona está constituida por una bicapa lipídica que permite el pasaje selectivo de estos iones. Cuando la neurona recibe estímulos provenientes de otras neuronas cuya suma espacial y temporal es mayor a un umbral determinado, se desata una cascada de eventos: se produce un rápido flujo de iones a la célula, causando un incremento acelerado de su potencial hasta lograr la apertura de los canales de K^+, que ocasiona un flujo hacia afuera de estos iones y, por ende, una caída del potencial de membrana. Este rápido ascenso y descenso del potencial de membrana recibe el nombre de *potencial de acción* o *espiga* y es el modo predominante de comunicación entre dos neuronas. Constituye un evento estereotipado que se rige por la ley de *'todo o nada'*, y transmite poca o ninguna información. Se cree que la información es transmitida, en cambio, por la tasa de disparo (número de espigas por segundo) y/o la duración de las espigas. Por su parte, la ley del todo o nada establece que existe una mínima intensidad de corriente estimulante que, al actuar durante un tiempo determinado, produce un potencial de acción en una neurona. Éste no se produce si la magnitud del estímulo es menor al umbral, y ocurre con amplitud y forma constantes, sin importar la fuerza del estímulo que lo produjo (figura 1.2).

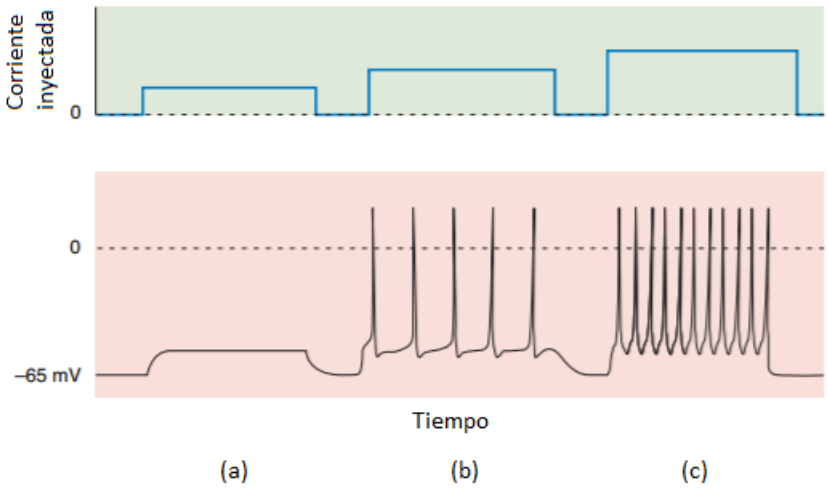


Figura 1.2: Efecto de inyectar corriente (gráfico superior) en una neurona. El gráfico inferior representa el potencial de membrana de la neurona en respuesta a los diferentes estímulos. (a) Si la corriente no despolariza la membrana hasta alcanzar el potencial umbral, no se genera ningún potencial de acción. (b) Si la corriente despolariza la membrana más allá del umbral, se producen los potenciales de acción. (c) La tasa de disparo del potencial de acción se incrementa a medida que la corriente de despolarización aumenta [4].[2]

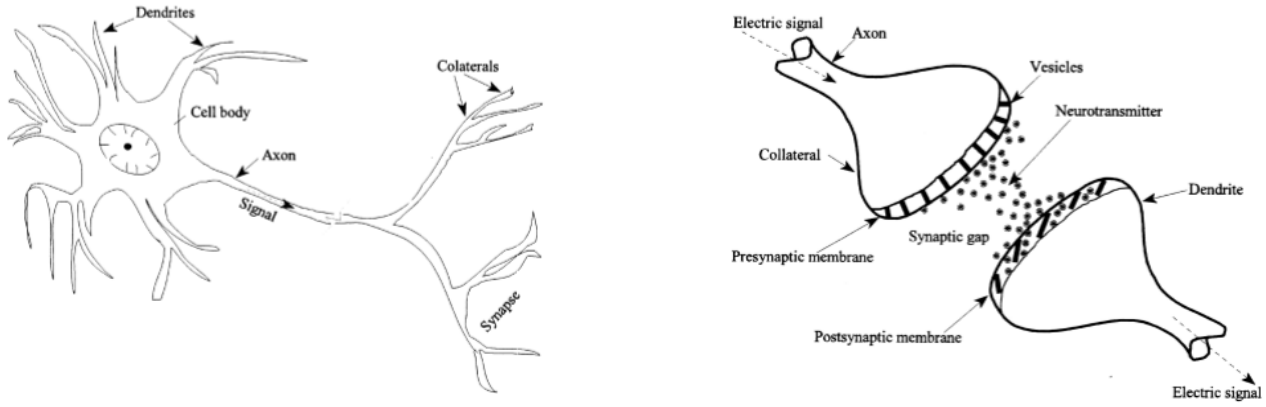
1.1.1.1.2 Sinapsis

Las neuronas se comunican entre sí a través de conexiones conocidas como sinapsis, que pueden ser de naturaleza eléctrica o química, siendo estas últimas las más habituales. Una sinapsis (figura 1.3b) consiste en un espacio submicroscópico (*hendidura sináptica*) entre el axón de una neurona, llamada *neurona presináptica*, y la dendrita de otra, llamada *neurona postsináptica*. Cuando el potencial de acción llega al terminal presináptico, se produce la liberación de neurotransmisores (sustancias químicas) a la hendidura sináptica, en cantidades proporcionales a la

[2]Imagen extraída de [4].

fuerza de la señal recibida. Los neurotransmisores difunden a través de la hendidura sináptica hacia las dendritas de la neurona postsináptica. Allí, se unen a los canales iónicos de la membrana, induciendo cambios locales en su potencial, que pueden desatar un nuevo potencial de acción en la neurona postsináptica.

Las sinapsis pueden ser excitatorias o inhibitorias. Las primeras causan un incremento momentáneo en el potencial local de la membrana del terminal postsináptico, aumentando la probabilidad de que se genere un potencial de acción en esta neurona. Las sinapsis inhibitorias producen el efecto opuesto, disminuyendo temporalmente el potencial local de la membrana [3, 16, 56].



(a) (b)

Figura 1.3: (a) Esquema de una neurona biológica. (b) Sinapsis entre dos neuronas.[3]

1.1.1.2. El Sistema Nervioso: Organización, Anatomía y Función

El sistema nervioso humano puede ser divido en Sistema Nervioso Central (SNC) y Sistema Nervioso Periférico (SNP). El SNP consta del sistema nervioso somático, formado por las neuronas conectadas a músculos esqueléticos, piel y órganos sensoriales; y el sistema nervioso autónomo, constituido por las neuronas que controlan las funciones viscerales, como el latido del corazón, la respiración, etc. El SNC consta del encéfalo y la médula espinal, que constituye el principal camino de descenso de las señales motoras de control desde el cerebro hacia los músculos, y de ascenso de la información sensorial de retroalimentación desde los músculos y piel hacia el cerebro.

La corteza cerebral está subdividida en lóbulos nombrados a partir del hueso del cráneo que se ubica sobre ellos: frontal, parietal, temporal y occipital. A grandes rasgos, sus funciones son:

- **Frontal:** razonamiento, planificación, lenguaje, movimientos, emociones y resolución de problemas. La parte más anterior, llamada área prefrontal, está encargada de las funciones de alto nivel: comportamientos cognitivos complejos, personalidad y toma de decisiones.
- **Parietal:** movimiento, orientación, reconocimiento, percepción de estímulos, procesamiento de emociones y habla.
- **Temporal:** percepción y reconocimiento de estímulos auditivos y visuales, memoria y habla.

[3]Imágenes tomadas de [3].

- **Occipital:** procesamiento visual.

Además, existe una porción escondida de corteza cerebral llamada 'ínsula', que se revela retirando hacia afuera los márgenes de la fisura lateral. La ínsula bordea y separa los lóbulos frontal y temporal, y está involucrada en la integración de funciones sensorimotoras, cognitivas y socioemocionales.

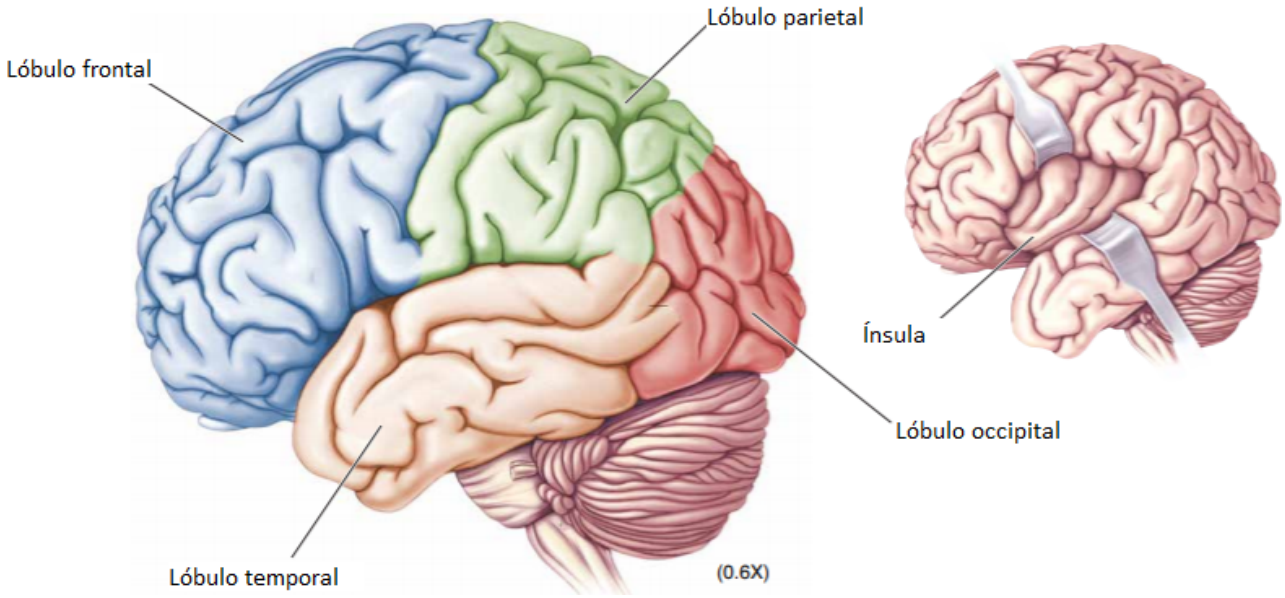


Figura 1.4: Vista lateral del cerebro mostrando sus diferentes lóbulos.[4]

El cerebro está dividido en dos hemisferios (hemisferios cerebrales), izquierdo y derecho, constituidos por: la neocorteza, la amígdala, los ganglios basales y el hipocampo. Los ganglios basales participan en el control motriz y elección de acciones; la amígdala está implicada en la regulación de emociones; y el hipocampo juega un rol crucial en el aprendizaje y la memoria, además de la cognición espacial.

La neocorteza es una estructura compleja que reside en la superficie del cerebro. Presenta un aspecto intrincado, mostrando fisuras y crestas conocidas como surcos y circunvoluciones, respectivamente. Muestra una especialización funcional, encontrándose dividida en diversas áreas, cada una de las cuales cumple una función particular. Estas áreas pueden reunirse en tres grandes grupos:

- **Áreas motoras:** las principales son el área motora primaria, el área premotora y el área motora suplementaria.
- **Áreas sensoriales:** las principales son la corteza visual, sómatosensorial, auditiva y gustativa.
- **Áreas de asociación:** comprende áreas relacionadas con funciones de orden superior. Las áreas de asociación situadas en las cortezas inferotemporal y prefrontal están involucradas en funciones cognitivas como el reconocimiento de rostros, lenguaje y planificación.

[4]Imagen extraída de [4].

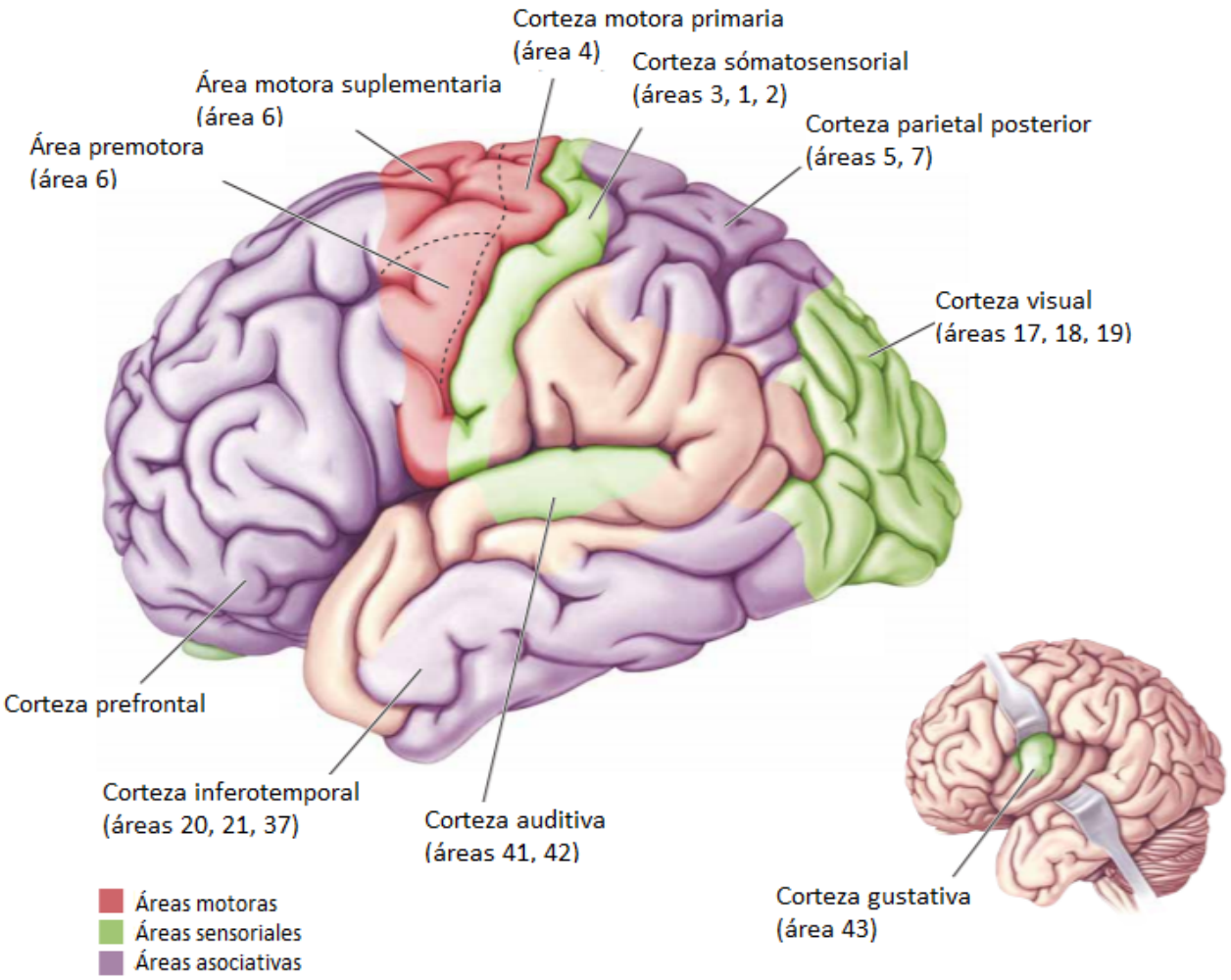


Figura 1.5: Imagen ilustrativa de las diferentes áreas de la corteza cerebral.[5]

Está constituida por alrededor de 30 mil millones de neuronas, dispuestas en seis capas. Cada neurona establece alrededor de diez mil sinapsis con otras neuronas, dando un total de 300 billones de conexiones. La información entra al área cortical a partir de las capas intermedias, principalmente; mientras que las salidas de las áreas se producen en las capas superiores e inferiores. Basándose en los patrones de entrada y salida de la información, es posible concebir a la corteza como una red de áreas motoras y sensoriales organizadas jerárquicamente [56, 4, 38].

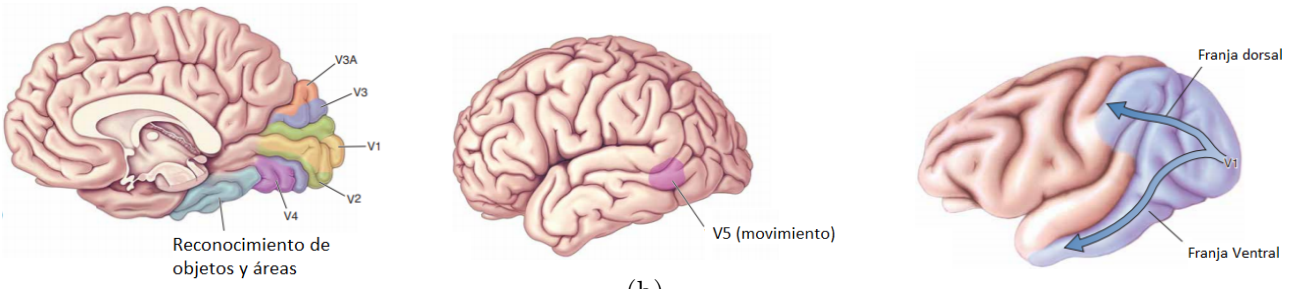
1.1.1.2.1 Procesamiento de la información visual

En el caso del procesamiento visual, la información de la retina llega a la corteza a partir de la región visual del tálamo, desde donde llega primero a las capas intermedias de la corteza visual primaria (área V1 o área 17). El área V1 contiene neuronas selectivas de características primitivas, como detección de movimiento y bordes. El procesamiento ulterior involucra tipos de procesamiento que se tornan progresivamente más complejos, implicando a las áreas visuales V2 y V4 y a la corteza inferotemporal (IT), a lo largo de la 'franja ventral'; y las áreas MT (o área V5), MST y corteza parietal, a lo largo de la 'franja dorsal'. La franja ventral se especializa en el procesamiento de la forma y color de los objetos y está implicada en el reconocimiento

[5]Imagen extraída de [4].

de rostros y objetos. La franja dorsal se especializa en el movimiento y razonamiento sobre relaciones espaciales.

A pesar de estas diferencias funcionales, las diferentes áreas de la corteza son notoriamente similares en su organización anatómica, lo que sugiere que la corteza emplea un algoritmo prototipo para el procesamiento de la información y que la especialización reside en el tipo de entrada recibida en cada área [56].



(a) (b) (c)

Figura 1.6: (a) y (b) Áreas visuales del cerebro Humano. (c) Franjas dorsal y ventral de procesamiento visual del mono macaque. [6]

1.1.2. Potenciales Relacionados a Eventos

Los Potenciales Relacionados a Eventos (ERP) o Potenciales Evocados (EPs), son manifestaciones de la actividad neuronal desencadenada por un evento específico, que puede ser sensorial, cognitivo o motor. Constituyen una respuesta neuroeléctrica cerebral específica, generada por la activación sincrónica de una población neuronal determinada, y se manifiestan mediante señales estereotipadas en el registro EEG. Dependiendo de la naturaleza del estímulo que la desencadena, se produce la activación selectiva del área cerebral encargada de su procesamiento.

Los ERPs se clasifican en dos tipos:

- **Exógenos:** reflejan la actividad en el sistema sensorial primario, por lo que corresponden a las componentes tempranas (primeros 150 ms posteriores a la aparición del evento).
- **Endógenos:** son los componentes de latencia de larga duración, dado que reflejan la actividad de procesamiento de la información. Tienen una naturaleza cognitiva, por lo que dependen en menor medida del estímulo, y están muy vinculados a la relevancia de la tarea para el sujeto.

Los ERPs son respuestas de muy pequeña amplitud, superpuestas y enmascaradas por la actividad electroencefalográfica de fondo, que es de mayor amplitud. Por lo tanto, es necesario promediar múltiples registros con el estímulo desencadenante para que la respuesta sincronizada sea discernible de la actividad de base.

Los ERPs se pueden definir por la polaridad y latencia de sus componentes, con la sigla de la polaridad P si es positiva y N si es negativa, seguida de la latencia o retardo posterior al estímulo que produce la aparición de dicho componente. Un ejemplo es el potencial P300, que se describe a continuación [46, 38].

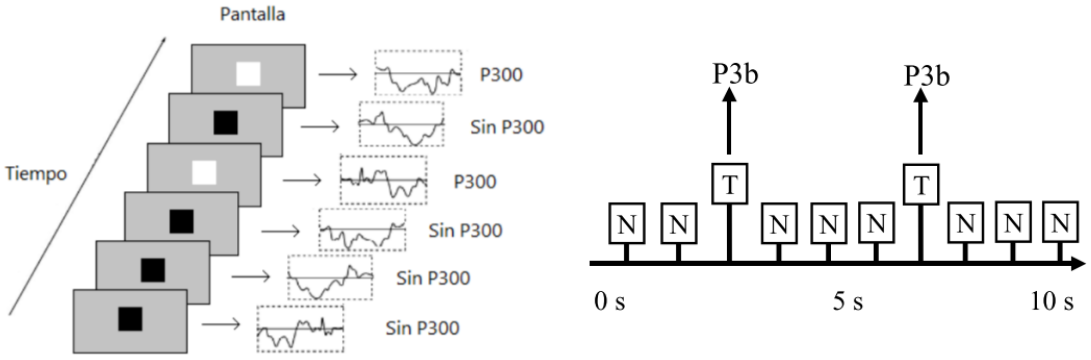
[6] Imágenes extraídas de [4].

1.1.2.1. Potencial P300

El potencial P300, también llamado P3, debe su nombre a su polaridad y latencia, dado que consiste en una desviación positiva en el registro electroencefalográfico que ocurre cerca de los 300 ms posteriores a la presentación del estímulo. Éste debe ser raro (de baja probabilidad) e impredecible, pero relevante para el sujeto. La amplitud del P300 dependerá directamente de qué tan relevante sea el estímulo, y varía inversamente a la probabilidad de éste. Este potencial es más frecuentemente observado en el lóbulo parietal, aunque algunas componentes también se originan en los lóbulos temporal y frontal. Los mecanismos neuronales responsables de la aparición del P300 son todavía inciertos, pero las estructuras cerebrales como la corteza parietal, la circunvolución del cíngulo y la corteza temporoparietal, al igual que las estructuras límbicas (amígdala e hipocampo) han sido implicadas como posibles fuentes [56].

1.1.2.1.1 El Paradigma Oddball

El método más comúnmente utilizado en el diseño de BCIs basadas en EEG para la evocación y detección de potenciales P300 es el *paradigma oddball*, que establece que los estímulos que generan este potencial deben tener dos cualidades: ser relevantes para la tarea y 'raros' (de baja probabilidad). En experimentos basados en este paradigma, se presenta al usuario un estímulo de este tipo (relevante para la tarea pero infrecuente), alternado con estímulos frecuentes y rutinarios; ambos pueden ser de tipo auditivo, visual o somatosentitivo. Mientras más raro sea este estímulo, mayor será la amplitud del P300. En la figura 1.7 se presenta un ejemplo de aplicación del paradigma [38, 56].



(a) (b)

Figura 1.7: (a) Ejemplo de experimento para la obtención del P300 a partir del paradigma oddball. Se presentan al usuario dos tipos de estímulos visuales: cuadrados negros y cuadrados blancos. Estos últimos aparecen con menor frecuencia y constituyen el evento objetivo, dado que se asigna al usuario la tarea de contar el número de veces que aparecen en la pantalla. De esta manera, cada una de sus apariciones genera un P300 en la actividad cerebral del observador. Éste puede ser percibido en el registro EEG, en una ventana temporal tomada a partir de la presentación del estímulo, y a través del promediado de varias instancias de este tipo [38]. (b) Secuencia de estímulos correspondientes al paradigma oddball (T: target, N: non-target) [27].[7]

[7]Imágenes extraídas de [38] (a) y [27] (b).

1.1.2.1.2 Variabilidad del P300

El P300 no consiste en un fenómeno fijo, sino que se trata de una respuesta variable del cerebro. En este sentido, se encuentra influenciado por diversos factores:

- **Probabilidad del evento target:** la amplitud pico del P300 varía de manera inversa con la probabilidad de aparación del estímulo esperado. Mientras más baja sea la probabilidad del estímulo target, mayor será la amplitud de la onda P300 que induzca su aparición.
- **Intervalo inter-estímulo:** la amplitud del P300 es directamente proporcional a la longitud del intervalo comprendido entre dos estímulos consecutivos.
- **Atención:** la amplitud del P300 depende de la atención que los usuarios prestan al estímulo mientras éste se presenta, y de su estado de concentración.
- **Dificultad de la tarea:** la latencia de la onda P300 es directamente proporcional a la dificultad de la tarea en proceso, mientras que su amplitud es inversamente proporcional. Por ejemplo, estímulos target muy diferentes a los non-target suscitan respuestas caracterizadas por un P300 de mayor amplitud que aquellos que presentan pocas diferencias entre sí [19].

Además, por tratarse de un potencial endógeno, producto de un procesamiento cognitivo del cerebro, su latencia es muy variable entre personas [27, 38].

1.1.3. Adquisición de señales

En esta sección se describe la técnica de adquisición de señales más habitualmente empleada en la construcción de BCIs, correspondiente al electroencefalograma.

1.1.3.1. Electroencefalograma

La electroencefalografía (EEG) es una técnica no invasiva de medición de señales cerebrales a partir de la utilización de electrodos ubicados en el cuero cabelludo. Las señales electroencefalográficas reflejan la sumatoria de los potenciales postsinápticos de miles de neuronas que están orientadas radialmente a éste. Las corrientes tangenciales no son detectadas mediante el EEG, al igual que las corrientes originadas en la profundidad del cerebro. Esto último es debido a que la intensidad de los voltajes decae de manera proporcional al cuadrado de la distancia desde la fuente, a lo que se suma la atenuación generada por los huesos del cráneo y las meninges. En consecuencia, el EEG captura predominantemente la actividad eléctrica en la corteza cerebral, cuyo arreglo en columnas de neuronas y su proximidad al cuero cabelludo favorecen su adquisición mediante esta técnica. La resolución espacial del EEG es típicamente pobre (en el rango de los centímetros cuadrados), pero la resolución temporal es buena (en el rango de los milisegundos) [56].

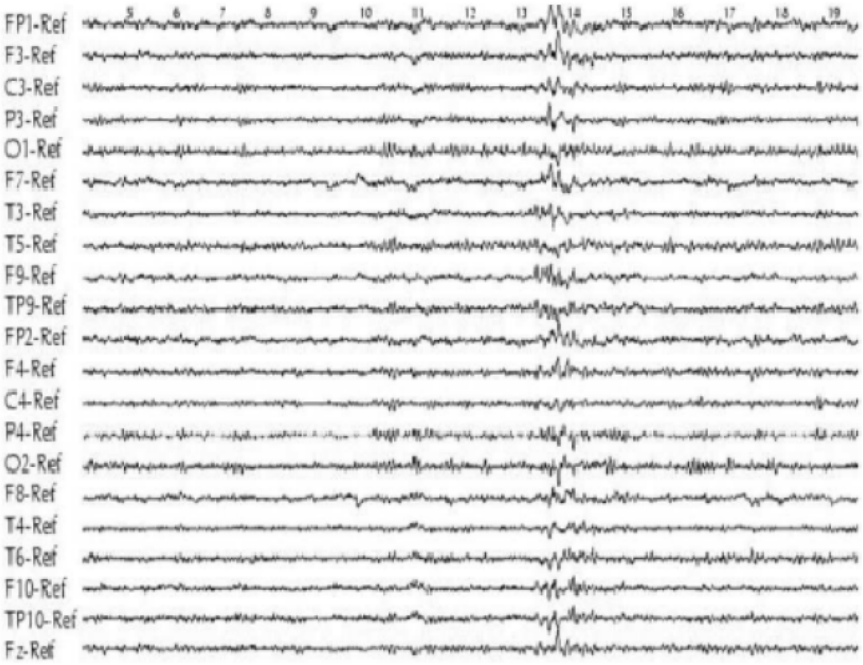


Figura 1.8: Registro electroencegalográfico. Se observan las amplitudes recogidas en varios canales a lo largo del tiempo. Los canales llevan nombre según su ubicación espacial. [8]

1.1.3.1.1 Artefactos EEG

La pobre resolución espacial se debe principalmente a las diferentes capas de tejido (meninges, líquido cefalorraquídeo, cráneo, cuero cabelludo) interpuestas entre la fuente de la señal, constituida por la actividad neuronal en la corteza, y los sensores ubicados en el cuero cabelludo. Estas capas actúan como un filtro pasa bajo que degrada la señal. Las señales medidas se encuentran en el rango de algunas decenas de microvoltios, requiriendo por consecuencia el uso de amplificadores poderosos y procesamiento de señal para amplificarla y filtrar el ruido. Esta baja amplitud también implica que las señales EEG pueden ser fácilmente corrompidas por la actividad muscular y contaminadas por los dispositivos eléctricos circundantes (señal de línea de 50 Hz). Por ejemplo, los movimientos oculares, parpadeo, movimientos de cejas, hablar, masticar o mover la cabeza pueden provocar artefactos de gran amplitud en las señales EEG. Por lo tanto, normalmente se instruye a los sujetos que eviten todo tipo de movimiento y se utilizan técnicas de eliminación de artefactos que filtren las porciones de señales corrompidas por artefactos musculares. Existen, además, fuentes adicionales de ruido, como la impedancia variable de los electrodos y los estados psicológicos variables del usuario debido a aburrimiento, distracción, fatiga, estrés o frustración [56].

1.1.3.1.2 Adquisición de las señales EEG

Para llevar a cabo los registros electroencefalográficos, se disponen los electrodos sobre el cuero cabelludo del sujeto, habitualmente valiéndose de un casco o red que los contenga. El Sistema Internacional 10-20, adoptado y recomendado por la Federación Internacional de Electroencefalografía, establece un método sistemático para la colocación, ubicación y nomenclatura de los electrodos en el cuero cabelludo, asegurando que sea consistente entre los diferentes laboratorios. Se miden los perímetros transversal (entre los huesos mastoides, detrás de cada

[8] Imagen extraída de [38].

oreja) y medio (entre el *nasion* [9] y el *inion* [10]); y se determinan las posiciones de los electrodos dividiendo estos perímetros en intervalos de 10 % y 20 %. El número de electrodos varía desde algunos pocos, para aplicaciones BCI específicas, hasta 256, en arreglos de alta densidad. Se utilizan letras para identificar las distintas regiones, y números pares para el hemisferio derecho e impares para el izquierdo. Las letras utilizadas para cada región son:

- F: lóbulo frontal
- P: lóbulo parietal
- T: lóbulo temporal
- O: lóbulo occipital
- C: línea central
- z: línea media
- A: oreja

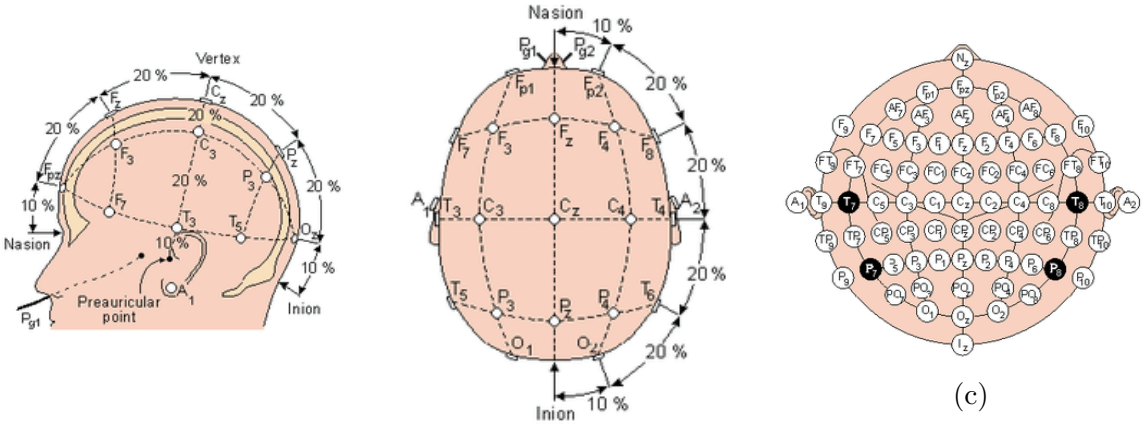


Figura 1.9: Distribución de los electrodos de acuerdo al Sistema 10-20.(a) Detalle del cálculo de posicionamiento, vista de perfil. (b) Detalle del cálculo de posicionamiento, vista superior. (c) Vista superior de la distribución de un sistema con 64 electrodos. [11]

Para la medición de este tipo de señales, se pueden emplear electrodos monopolares o bipolares, siendo estos últimos los más comunes. Para la medición bipolar, se mide la diferencia de potencial entre cada par de electrodos; mientras que, para la medición monopolar, el potencial de cada electrodo es comparado con el de un electrodo neutro, o con el promedio de todos los electrodos, técnica conocida como CAR (*Common Average Referencing*). En una disposición típica, cada electrodo es conectado a una entrada de un amplificador diferencial, cuya entrada restante es conectada al electrodo de referencia: por ejemplo, el nasion, las mastoides, el

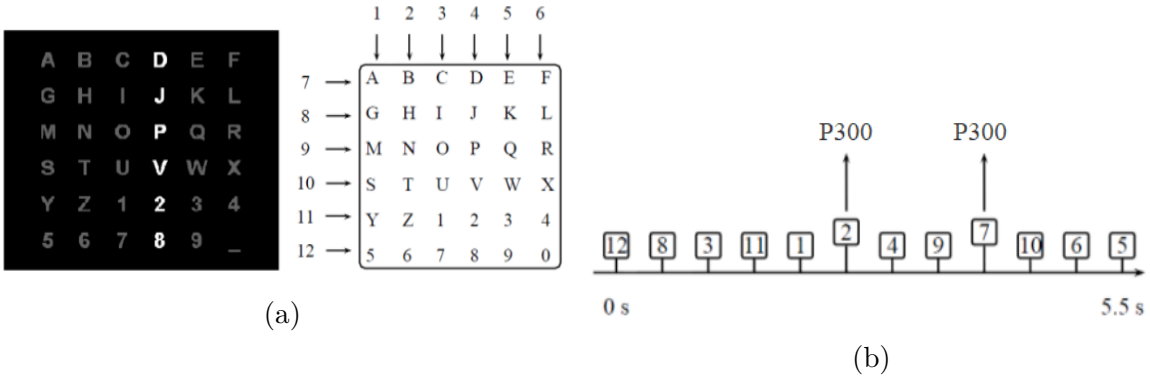
[9] Referencia anatómica ubicada en lo alto de la nariz, a nivel de los ojos.
[10] Referencia anatómica situada en la base del cráneo.
[11] Imagen extraída de [44].

lóbulo de una oreja. La diferencia de potencial entre el electrodo activo y el de referencia es amplificada en el orden de 1.000 a 100.000 veces. La señal amplificada es digitalizada por un conversor A/D (analógico/digital) a frecuencias de muestreo de hasta 20 kHz (las más típicas en aplicaciones BCI están comprendidas entre 256 Hz y 1 kHz). Luego de la digitalización, la señal puede ser adicionalmente filtrada por un filtro pasabanda de habitualmente 1-50 Hz, que excluye el ruido y artefactos de movimiento que se producen en frecuencias muy bajas o muy altas. Adicionalmente, puede aplicarse un filtro *notch* (filtro eliminador de banda) para eliminar la interferencia de línea (50 Hz en Argentina) [56, 38].

1.1.4. El Deletreador P300

La primer BCI basada en el P300 fue presentada por Donchin et al en [14]. En ésta, se presenta al usuario, en la pantalla de una computadora, una matriz de 6x6, en la que se distribuyen letras y otro tipo de caracteres. Sus filas y columnas son repetidamente resaltadas de manera aleatoria, a fin de provocar una respuesta de tipo P300 en aquellas que contengan el carácter deseado. Para que esto suceda, y de acuerdo al paradigma oddball, cada secuencia debe contener un número bajo de *targets* (objetivos) frente a uno alto de estímulos no objetivo (*non-target*). En consecuencia, las intensificaciones de la fila y la columna que contienen el carácter deseado constituyen el estímulo target e inducen un potencial P300; mientras que el resto de las intensificaciones constituye un estímulo estándar o non-target, de modo que no suscita este potencial.

La selección de cada carácter se lleva a cabo al concentrar el sujeto su atención sobre el carácter deseado, lo que provocará la aparición de un potencial P300. El carácter puede, entonces, ser determinado al identificar en los registros EEG, obtenidos durante la intensificación de las diferentes filas y columnas, los potenciales P300 que fueron evocados. A partir de estos se pueden identificar la fila y la columna correspondientes al carácter en cuestión, que constituyen sus coordenadas dentro de la matriz [22, 27].



(a) (b)

Figura 1.10: Principio de funcionamiento del deletreador P300. (a) Ejemplo de una matriz de caracteres usadas en el deletreador P300. Las intensificaciones de las filas y columnas constituyen los estímulos, que son numerados del 1 al 12. (b) Secuencia de estímulos aleatoria. Ej: si el usuario concentra su atención sobre la letra 'B', un potencial P300 será evocado para los estímulos 2 y 7 [27].[12]

[12]Imágenes extraídas de [27].

Las pruebas se realizan con el usuario con los electrodos colocados sobre el cuero cabelludo, sentado frente a una pantalla, donde se muestra la matriz de caracteres. Las diferentes etapas que comprende son:

- **Montaje**: sobre el cuero cabelludo se colocan y ajustan los electrodos, que pueden estar dispuestos individualmente o en un casco diseñado *ad hoc*, y se evalúa la calidad de las señales provenientes de los diferentes canales.
- **Sesión de calibración (*online*):** se adquieren señales EEG a partir de la ejecución del deletreador, para generar la base de datos de entrenamiento de los algoritmos clasificadores. La sesión consta de determinado número de pruebas, en cada una de las cuales el usuario debe concentrarse en un carácter asignado arbitrariamente, que es debidamente indicado en la pantalla. Luego de un número determinado de repeticiones, correspondiente al número de veces que cada fila y columna son resaltadas, se prosigue con el siguiente caracter, previa pausa intermedia de algunos segundos de descanso.
- **Entrenamiento de algoritmos clasificadores (*offline*):** se entrenan los algoritmos clasificadores para lograr discriminar entre dos clases, P300 y no P300.
- **Deletreador online:** el usuario deletrea el texto al tiempo que los clasificadores ya entrenados diferencian entre los eventos P300 y no P300, para determinar el carácter deseado. Luego de cierto número de repeticiones, el algoritmo muestra el carácter reconocido.

1.2. Preprocesamiento de la señal EEG

Las señales electroencefalográficas son de baja amplitud y muy susceptibles al ruido eléctrico, ya sea del ambiente, como de la actividad ocular o muscular. Por lo tanto, al trabajar con ellas, es vital la implementación de técnicas de preprocesamiento que permitan eliminar el ruido y otros artefactos de los registros EEG, como así también resaltar información importante contenida en las señales. Luego, el preprocesamiento consiste en la aplicación de una o varias de estas técnicas a fin de transformar un conjunto de señales '*crudas*' en un nuevo conjunto de señales con menor ruido, aumentando la relación señal-ruido.

Los métodos de preprocesamiento más habitualmente usados en interfaces cerebro computadora incluyen filtros espaciales, temporales y frecuenciales, y métodos más avanzados como la extracción de los componentes más representativos de las señales (ICA: Análisis de Componentes Independientes, por sus siglas en inglés y PCA: Análisis de Componentes Principales, por sus siglas en inglés).

1.2.1. Filtros temporales

En el tratamiento de señales electroencefalográficas, la operación de filtrado es habitualmente llevada a cabo para eliminar aquellas variaciones de voltaje de muy baja o muy alta velocidad: $f < 0{,}01 - 0{,}1\ Hz$ y $f > 15 - 100\ Hz$, respectivamente. Esto se debe a que los voltajes recogidos del cuero cabelludo que se encuentran comprendidos entre estos rangos de frecuencias corresponden, generalmente, a ruido aportado por fuentes no neuronales [34].

Luego, para remover componentes de alta frecuencia, ruido y artefactos de la señal, se la somete a filtros temporales pasa-banda o pasa-bajo. También estos filtros son empleados para limitar el análisis a las bandas frecuenciales que contienen las señales neurofisiológicas de interés. Para implementar este tipo de filtro, la señal cruda es primero transferida al dominio

frecuencial, donde se retienen sólo los componentes correspondientes a la banda escogida, y finalmente se la transfiere de vuelta al dominio temporal.

1.2.1.1. Filtros de Respuesta Infinita al Impulso (IIR)

Son filtros lineales recursivos que, además de hacer uso de las últimas M muestras de la señal cruda, como es el caso de los filtros FIR (Respuesta Finita al Impulso) emplean las salidas de las últimas P muestras filtradas:

$$y(n) = \sum_{k=0}^{M} a_k s(n-k) + \sum_{k=1}^{P} b_k y(n-k) \tag{1.1}$$

En comparación con los FIR, los fitros IIR llevan a cabo el filtrado con un menor número de coeficientes, pero muestran un desempeño menor en el dominio frecuencial. Un ejemplo de este tipo de filtro empleado en el preprocesamiento de señales BCI es el filtro de Butterworth [2].

1.2.1.2. Filtro de media móvil

El filtro de media móvil se emplea para suavizar las señales, eliminando los componentes de alta frecuencia. Este filtro disminuye el ruido aleatorio, al tiempo que retiene una fuerte respuesta al escalón, lo que lo hace apropiado para ser utilizado en señales que se encuentren en el dominio temporal, no así en el frecuencial, donde carece de la habilidad de separar señales según sus bandas de frecuencia.

El filtro opera promediando un cierto número de puntos de la señal de entrada para producir cada punto de la señal de salida. A esto lo hace a partir de la convolución de la señal con una ventana móvil, cuyos elementos son de magnitud igual a $\frac{1}{ancho\ ventana}$ [61].

1.2.2. Submuestreo y decimación

La operación de submuestreo consiste, simplemente, en la reducción de la cantidad de muestras de la señal; lo que se traduce en una disminución de la frecuencia de muestreo original. Un criterio que debe respetarse para evitar el solapamiento de las señales es el criterio de Nyquist, que establece que la frecuencia de muestreo debe ser superior al doble de la frecuencia máxima de interés de la señal.

La decimación, por su parte, es también una técnica de reducción de la frecuencia de muestreo de una señal, en la que se emplea el filtrado para mitigar la distorisión por solapamiento. El factor de decimación es habitualmente un número entero o una fracción racional mayor que uno. Este factor multiplica el tiempo de muestreo o, lo que es equivalente, divide la frecuencia de muestreo de la señal.

1.3. Extracción de características

El registro de señales electroencefalográficas implica la adquisición de una gran cantidad de datos, variable según sea la frecuencia de muestreo y el número de canales empleados. A la hora de trabajar con algoritmos de aprendizaje automático, es necesario introducir al sistema vectores que contengan un número reducido de valores, que representen información relevante de las señales. Estos valores se conocen como "características", y pueden corresponder, por ejemplo, a propiedades temporales, espaciales o frecuenciales de la señal. Estas características se agrupan en vectores, llamados "vectores de características". Por lo tanto, el proceso de extracción de

características puede concebirse como una operación que reúne una o varias propiedades de la señal en un vector de características.

En el diseño de una interfaz cerebro computadora, es importante que las características extraídas sean representativas y relevantes para la señal neuro-fisiológica en estudio, dado que, de otra manera, los algoritmos clasificadores no podrían identificar correctamente las clases a las que pertenecen. Además, es deseable que el número de características sea reducido, ya que influye sobre el tiempo de cómputo de los algoritmos.

Las técnicas habitualmente empleadas en extracción de características para BCIs pueden agruparse en cuatro grupos según el tipo de información que explotan: temporal, frecuencial, híbridos (temporal y frecuencial) y espacial. Dado que el P300 tiene una impronta temporal específica, los métodos más habitualmente utilizados para su estudio son los temporales. Estos métodos se valen de las variaciones temporales de las señales como características representivas. En particular, la información temporal más simple y ampliamente empleada en el caso del P300, debido a su eficiencia, es la variación en el tiempo de la amplitud de la señal EEG. Por lo tanto, las amplitudes recolectadas a partir de los diferentes electrodos, luego de ser preprocesadas, son segmentadas en épocas de interés y concatenadas en un vector de características.

1.4. Clasificación binaria de potenciales P300

La determinación de la presencia o ausencia de un potencial evocado P300 a partir de las características extraídas de un registro EEG constituye un problema de clasificación binaria. Es decir, existen dos clases posibles: P300 y No-P300, y se busca poder discernir entre ambas a partir de la información contenida en los vectores de características creados. Esto se puede llevar a cabo con una función discriminante cuyo hiperplano esté definido por:

$$w.f(x) + b = 0 \tag{1.2}$$

donde x es el vector de características, $f(\cdot)$ es una función de transformación, w es un vector de pesos de clasificación y b corresponde al término bias. Esta ecuación aplica tanto para problemas linealmente separables, como para no linealmente separables. Esto es posible porque, para este último caso, la función $f(\cdot)$ puede representar una transformación que mapea los vectores de entrada al modelo, de dimensionalidad igual al número de características, en un espacio de mayor dimensión, a fin de crear un conjunto linealmente separable. Este tipo de función se conoce como *función kernel*. Para métodos lineales, $f(\cdot)$ corresponde simplemente a la función identidad: $f(x) = x$.

El problema de clasificación consiste, entonces, en obtener, a partir de una etapa de entrenamiento, los valores w y b que definen la función de decisión (ecuación 1.4) del clasificador. Al aplicarla, luego, a los vectores de características del conjunto de prueba, se obtienen los puntajes correspondientes a las dos clases.

A partir de los puntajes devueltos por cada modelo, que representan la probabilidad de pertenencia a una u otra clase, y dado que se asume que el potencial P300 sólo es provocado por la intensificación de la fila y la columna que contienen el carácter deseado, es posible determinar los caracteres objetivo. Para ello, para cada época de caracteres, se toma el máximo de la suma de los puntajes obtenidos para las filas y las columnas por separado, que corresponderán a la fila y columna objetivo, obteniendo de su intersección el carácter correspondiente [29]:

$$
\begin{aligned}
fila\ predicha &= \underset{\text{filas}}{arg\ max} \left(\sum_{i_{fila}} w \cdot f(x)_{i_{fila}} \right) \\
columna\ predicha &= \underset{\text{columnas}}{arg\ max} \left(\sum_{i_{columna}} w \cdot f(x)_{i_{columna}} \right)
\end{aligned}
\tag{1.3}
$$

1.4.1. Métricas en clasificación binaria

En un problema de clasificación binaria, existen cuatro posibles casos de clasificación de un determinado ejemplo:

1. Ejemplo positivo, clasificado como positivo: se trata de un verdadero positivo (TP, *True Positive*)
2. Ejemplo positivo, clasificado como negativo: corresponde a un falso negativo (FN, *False Negative*).
3. Ejemplo negativo, clasificado como negativo: se trata de un verdadero negativo (TN, *True Negative*).
4. Ejemplo negativo, clasificado como positivo: corresponde a un falso positivo (FP, *False Positive*).

A partir de estos valores, es posible construir una matriz denominada *'matriz de confusión'*:

$$
\text{Matriz de confusión} = \begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \tag{1.4}
$$

que representa el desempeño de un algoritmo, permitiendo visualizar su capacidad de distinción entre ambas clases. Sus filas representan las clases predichas, mientras que sus columnas representan las verdaderas clases. Por lo tanto, para un clasificador ideal, que clasifica correctamente la totalidad de los casos, la matriz es diagonal.

De esta matriz surgen, a su vez, diversas métricas:

- *Accuracy*: conocida como exactitud en español, indica el porcentaje de predicciones correctas.

$$
\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{1.5}
$$

- *Recall*: también conocida como tasa de verdaderos positivos (TPR, *True Positive Rate*) o sensibilidad, indica la capacidad del clasificador de encontrar todos los ejemplos positivos.

$$
\text{Recall} = \frac{TP}{TP + FN} \tag{1.6}
$$

- Tasa de falsos positivos (FPR *False Positive Rate*): indica la cantidad relativa de ejemplos erróneamente clasificados como positivos.

$$
\text{FPR} = \frac{FP}{TP + FN} \tag{1.7}
$$

- *Especificidad*: también conocida como tasa de verdaderos negativos (TNR, *True Negative Rate*), indica la proporción de ejemplos correctamente etiquetados como negativos.

$$\text{Especificidad} = \frac{TN}{FP + TN} = 1 - FPR \tag{1.8}$$

- *Precision*: indica la habilidad del clasificador de no etiquetar como positivo un ejemplo que es negativo.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1.9}$$

- *Puntaje F1*: constituye el promedio ponderado de las métricas precision y recall. La contribución relativa de cada una al puntaje F1 es la misma.

$$\text{F1} = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{1.10}$$

En todos los casos, las anteriores métricas están comprendidas entre 0 y 1, siendo estos el peor y mejor valor, respectivamente [49].

1.5. Algoritmos clasificadores

En esta sección, se introducen los fundamentos de los algoritmos de clasificación empleados para la detección de potenciales P300. Para cada uno, se mencionan los parámetros más importantes a ajustar y/o definir durante el entrenamiento.

1.5.1. Análisis Discriminante Lineal

El clasificador Análisis Discriminante Lineal (LDA) traza un hiperplano para separar datos pertenecientes a diferentes clases. En un problema de clasificación binaria, la clase del vector de características depende del lado del hiperplano en que se encuentra. Esto se puede ver en la figura 1.11, en que la clase 1 (círculos) y la clase 2 (cruces) están separadas por una recta de ecuación $w_o + w^T x = 0$. Si el cálculo del vector no etiquetado es mayor que 0, es un círculo; de lo contrario, es una cruz.

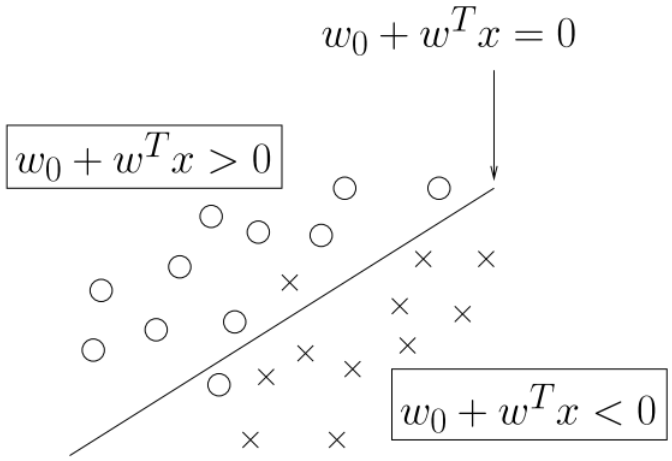


Figura 1.11: Hiperplano que separa dos clases: círculos y cruces.[13]

El LDA asume una distribución normal de los datos para ambas clases, y el vector separador es obtenido al buscar la proyección que maximice la distancia entre las medias de ambas clases y, a su vez, minimice la varianza dentro de cada una. Este clasificador es simple de utilizar, tiene un bajo costo computacional y provee buenos resultados, por lo cual ha sido usado en numerosas implementaciones de BCI [33].

En términos matemáticos, dada una base de datos de n muestras $\{(x_i, y_i)\}_{i=1}^{n}$, donde $x_i \in \mathbb{R}^d$ y $y_i \in \{1, 2, \ldots, k\}$ denota la etiqueta de clase del i-ésimo ejemplo, n es el número de muestras y k el número de clases; se particiona la matriz $X = [x_1, x_2, \ldots, x_n]$ en k clases: $X = [X_1, X_2, \ldots, X_k]$, donde $X_j \in \mathbb{R}^{dxn_j}$, siendo n_j el tamaño de la clase j, y $\sum_{j=1}^{k} n_j = n$.

El LDA computa luego una transformación lineal $G \in \mathbb{R}^{dxl}$ que mapea x_i en el espacio d-dimensional a un vector x_i^L en el espacio l dimensional de la siguiente manera: $x_i \in \mathbb{R}^d \rightarrow x_i^L = G^T x_i \in \mathbb{R}^l$ $(l < d)$. Para ello, se definen tres matrices de dispersión llamadas *within-class* (intra-clases), *between-class* (entre-clases), y *total*:

$$S_w = \frac{1}{n} \sum_{j=1}^{k} \sum_{x \in X_j} (x - c^{(j)})(x - c^{(j)})^T \tag{1.11}$$

$$S_b = \frac{1}{n} \sum_{j=1}^{k} n_j (c^{(j)} - c)(c^{(j)} - c)^T \tag{1.12}$$

$$S_b = \frac{1}{n} \sum_{i=1}^{n} (x_i - c)^T \tag{1.13}$$

donde $c^{(j)}$ es el *centroide* de la clase j, y c es el centroide global. De la definición se sigue que

$$S_t = S_w + S_b \tag{1.14}$$

La matriz S_w mide la cohesión intra-clase; mientras que la matriz S_b mide la separación entre clases. En el espacio de menor dimensión resultante de aplicar la transformación lineal G, las matrices de dispersión se tornan

$$S_w^L = G^T S_w G, \quad S_b^L = G^T S_b G, \quad S_t^L = G^T S_t G \tag{1.15}$$

Una transformación óptima G minimizaría la matriz S_w^L y maximizaría S_b^L simulatáneamente; lo que resulta equivalente a maximizar S_b^L y minimizar S_t^L, dado que $S_t^L = S_w^L + S_b^L$. Luego, la transformación óptima estará dada por

$$G = \arg\max_{G} \{S_b^L (S_t^L)^{-1}\} \tag{1.16}$$

que consiste en los vectores propios (*eigen vectors*) de mayor valor de $S_t^{-1} S_b$ correspondientes a los valores propios (*eigen values*) no nulos [14].

[13] Imagen extraída de [33].

[14] Desarrollo extraído de [72]

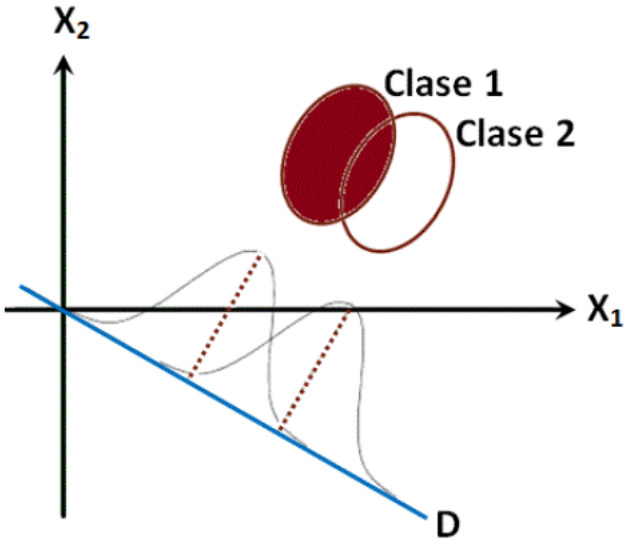


Figura 1.12: Diagrama de dispersión de dos clases para un problema de clasificación binario. Se aprecia la proyección de éstas sobre el hiperplano D, correspondiente a un espacio de menor dimensioanlidad. [15]

1.5.2. Máquina de Vectores de Soporte

Dentro de la tarea de clasificación, la Máquina de Vectores de Soporte (SVM) [16] pertenece a la categoría de clasificadores lineales, en tanto se basa en el establecimiento de un hiperplano óptimo de separación entre clases, ya sea en el espacio original de los ejemplos de entrada, si estos son separables o cuasi-separables (ruido), o en un espacio transformado, si los ejemplos no son separables linealmente en el espacio original.

Las SVMs se basan en la definición de un hiperplano de separación que equidiste de los ejemplos más cercanos de cada clase para, de esta forma, conseguir un *margen máximo* a cada lado del hiperplano; entendiéndose por margen a la distancia comprendida entre ambos. Estos puntos de entrenamiento son los que definen el hiperplano óptimo y se conocen como *'vectores de soporte'* (ver figura 1.13). Es decir que los únicos ejemplos de entrenamiento considerados en la definición del hiperplano, son aquellos que caen justo en la frontera de dichos márgenes. Desde un punto de vista práctico, cuanto mayor sea el margen de separación, mayor será la capacidad de generalización del clasificador.

[15] Imagen extraída de [46].

[16] El desarrollo matemático presentado en esta sección fue extraído de [62].

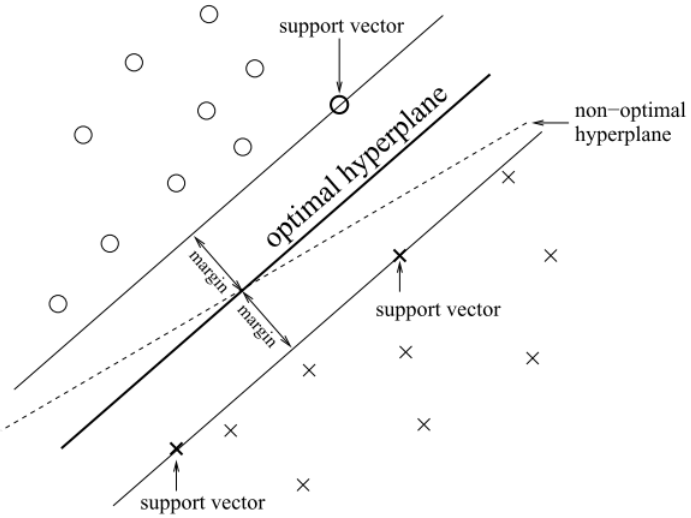


Figura 1.13: Hiperplano óptimo para la generalización en método SVM. [17]

En los procesos de clasificación puede ocurrir que los datos sean linealmente separables, que exista un cierto nivel de ruido, o que no sean linealmente separables. En consecuencia, se pueden definir distintos tipos de SVM: 1) SVM lineal con margen duro, 2) SVM lineal con margen blando, 3) SVM para la clasificación no lineal [46, 62].

1.5.2.1. SVM Lineal de Margen Duro

Dado un conjunto separable de ejemplos $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, donde $x_i \in \mathbb{R}$, se puede definir un hiperplano de separación

$$D(x) = (w_1 x_1 + \ldots + w_d x_d) + b =< \boldsymbol{w}, \boldsymbol{x} > + b \tag{1.17}$$

donde w_i y b son coeficientes reales. El hiperplano de separación cumplirá las siguientes restricciones para todo x_i del conjunto de ejemplos:

$$\begin{aligned} &< \boldsymbol{w}, \boldsymbol{x} > + b \geq 0 \; si \; y_i = +1 \\ &< \boldsymbol{w}, \boldsymbol{x} > + b \leq 0 \; si \; y_1 = -1, i = 1, \ldots, n \end{aligned} \tag{1.18}$$

o también,

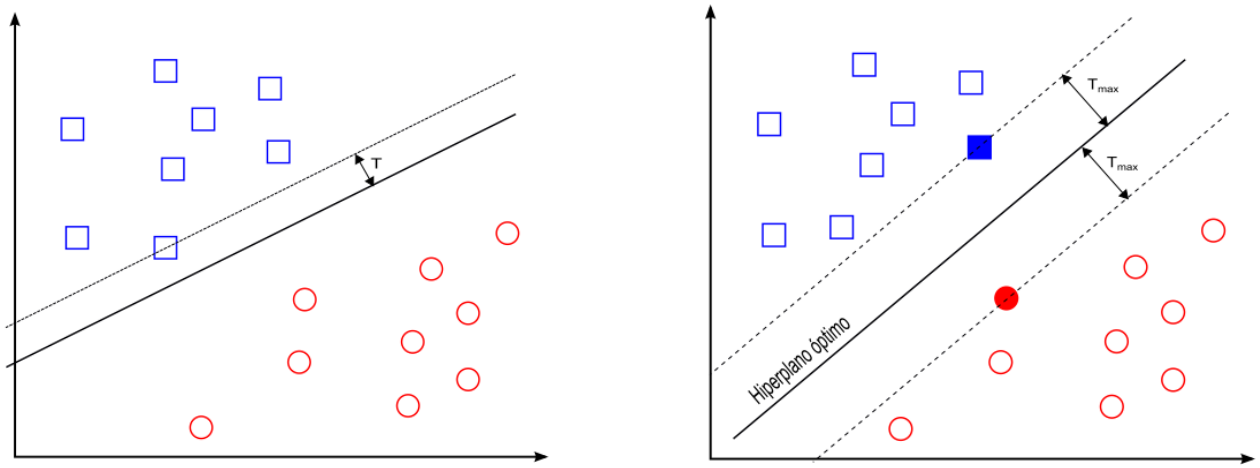
$$\begin{aligned} y_i(< \boldsymbol{w}, \boldsymbol{x} > + b) \geq 0, \\ y_i D(x_i) \geq 0, \; i = 1, \ldots, n \end{aligned} \tag{1.19}$$

Sin embargo, el hiperplano de separación no es único, existiendo infinitos hiperplanos posibles que lleven a cabo esta tarea. Para encontrar el hiperplano óptimo, se define el margen de un hiperplano de separación

$$\tau = \frac{1}{\parallel \boldsymbol{w} \parallel} \tag{1.20}$$

que representa la mínima distancia entre dicho hiperplano y los ejemplos más cercanos a ambas clases. En estos términos, encontrar el hiperplano óptimo es equivalente a encontrar el valor de $\boldsymbol{w}$ que maximiza el margen.

[17] Imagen extraída de [33].



(a) (b)

Figura 1.14: Margen de un hiperplano de separación: (a) hiperplano de separación no óptimo y su margen asociado (no máximo) (b) hiperplano de separación óptimo y su margen asociado (máximo) [62].[18]

1.5.2.1.1 Problema de optimización primal

De manera equivalente, la búsqueda del hiperplano óptimo para el caso separable puede ser formalizada como el problema de encontrar el valor de $\boldsymbol{w}$ y b que minimiza el funcional $f(\boldsymbol{w})$ sujeto a las restricciones 1.21:

$$\begin{aligned} min \quad & f(\boldsymbol{w}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 = \frac{1}{2} < \boldsymbol{w}, \boldsymbol{w} > \\ s.a. \quad & y_i(< \boldsymbol{w}, \boldsymbol{x} > + b) - 1 \geq 0, \ i = 1, \ldots, n \end{aligned} \tag{1.21}$$

Este problema de optimización con restricciones corresponde a un problema de programación cuadrático y es abordable mediante la *teoría de la optimización*. Esta teoría establece que un problema de optimización, denominado primal, tiene una forma dual si la función a optimizar y las restricciones son funciones estrictamente convexas. En estas circunstancias, resolver el problema dual permite obtener la solución del problema primal.

1.5.2.1.2 Problema de optimización dual

Tras la introducción de los multiplicadores de Lagrange, el problema puede ser formulado en su forma dual, que consiste en maximizar

$$L(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j < \boldsymbol{x_i}, \boldsymbol{x_j} > \tag{1.22}$$

sujeto a las restricciones

$$\begin{aligned} \sum_{i=1}^{n} \alpha_i y_i = 0 \\ \alpha_i \geq 0, \ i = 1, \ldots, n \end{aligned} \tag{1.23}$$

donde $\alpha_i > 0$ son los multiplicadores de Lagrange o coeficientes duales.

[18] Imagen extraída de [62].

Si bien este problema también es abordable mediante técnicas de programación cuadrática, su ventaja frente al primal reside en el hecho de que, mientras que el problema de optimización primal escala con la dimensionalidad d, el problema dual lo hace con el número de muestras n. Por lo tanto, el coste computacional asociado a su resolución lo hace factible incluso para problemas con un número muy alto de dimensiones.

1.5.2.1.3 Solución del problema primal

La solución del problema dual, $\boldsymbol{\alpha^*}$ permite obtener la solución del problema primal:

$$D(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i^* y_i < \boldsymbol{x}, \boldsymbol{x_i} > + b^* \tag{1.24}$$

Por definición, los ejemplos que satisfacen las restricciones expresadas en 1.21, considerando el caso 'igual que', son los vectores soporte. Sólo los ejemplos que tengan asociado un $\alpha_i > 0$ serán vectores soporte. De este resultado, también puede afirmarse que el hiperplano de separación 1.24 se construirá como una combinación lineal de sólo los vectores soporte del conjunto de ejemplos, ya que el resto de ejemplos tendrán asociado un $\alpha_i = 0$.

1.5.2.2. SVM Lineal de Margen Blando

El problema planteado en la sección anterior tiene escaso interés práctico porque los problemas reales se caracterizan normalmente por poseer ejemplos ruidosos y no ser perfecta y linealmente separables. La estrategia para este tipo de problemas reales es relajar el grado de separabilidad del conjunto de ejemplos, permitiendo que haya errores de clasificación en algunos de los ejemplos del conjunto de entrenamiento. Sin embargo, sigue siendo un objetivo el encontrar un hiperplano óptimo para el resto de ejemplos que sí son separables.

Para ello, se introduce un conjunto de variables reales positivas $\xi_i,\ i = 1, \ldots, n$, denominadas *'variables de holgura'*, que permiten cuantificar el número de ejemplos no separables que se está dispuesto a admitir:

$$y_i(< \boldsymbol{w}, \boldsymbol{x_i} > + b) \geq 1 - \xi_i,\ i = 1, \ldots, n \tag{1.25}$$

Por lo tanto, para un ejemplo $(\boldsymbol{x_i}, y_i)$, ξ_i representa la desviación del caso separable, medida desde el borde del margen correspondiente a la clase y_i. En consecuencia:

- $\xi_i = 0 \longrightarrow$ ejemplos separables
- $\xi_i > 0 \longrightarrow$ ejemplos no separables
- $\xi_i > 1 \longrightarrow$ ejemplos no separables y mal clasificados

La suma de todas las variables de holgura $\sum_{i=1}^{n} \xi_i$ permite medir el coste asociado al número de ejemplos no separables. Cuanto mayor sea esta suma, mayor será el número de ejemplos no separables. Relajadas las restricciones, la función a optimizar debe incluir, de alguna forma, los errores que está cometiendo el hiperplano de separación. Es decir:

$$f(\boldsymbol{w}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + C \sum_{i=1}^{n} \xi_i \tag{1.26}$$

donde C, denominada *Constante de Margen Blando*, es una constante elegida por el usuario, que permite controlar en qué grado influye el término del coste de ejemplos no separables en la minimización de la norma. Es decir, permite regular el compromiso entre el grado de sobreajuste

del clasificador final y la proporción del número de ejemplos no separables. En consecuencia, un valor de C muy grande permitiría valores de ξ_i muy pequeños. En el límite $C \to \infty$, se está ante el caso de ejemplos perfectamente separables ($\xi_i \to 0$). En el caso contrario, un valor de C muy pequeño permitiría valores de ξ_i muy grandes, por lo que se admitiría un número grande de ejemplos mal clasificados. En el caso límite $C \to 0$, se permitiría que todos los ejemplos estuvieran mal clasificados ($\xi_i \to \infty$).

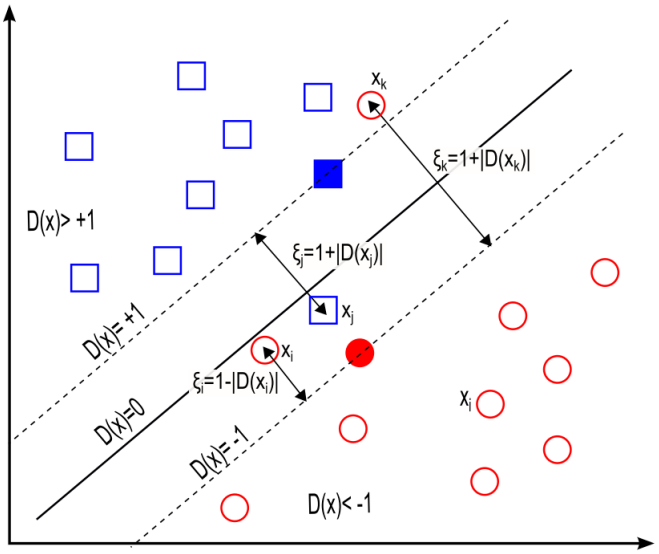


Figura 1.15: En el caso de los ejemplos no separables, las variables de holgura miden la desviación desde el borde del margen de la clase correspondiente. Los ejemplos x_i, x_j, x_k son no separables ($xi_i, \xi_j, \xi_k > 0$). Sin embargo, x_i está correctamente clasificado; mientras que x_j y x_k en el lado incorrecto de la frontera y, por tanto, mal clasificados [62].[19]

1.5.2.2.1 Problema de optimización primal

En consecuencia, el nuevo problema de optimización consistirá en encontrar el hiperplano definido por $\boldsymbol{w}$ y b, que minimiza el funcional 1.26 y esté sujeto a las restricciones dadas por 1.25. Es decir,

$$\begin{aligned} min \quad & \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + C \sum_{i=1}^{n} \xi_i \\ s.a \quad & y_i(< \boldsymbol{w}, \boldsymbol{x_i} > + b) + \xi_i - 1 \geq 0 \\ & \xi_i \geq 0, \ i = 1, \ldots, n \end{aligned} \tag{1.27}$$

El hiperplano así definido recibe el nombre de *hiperplano de separación de margen blando.*

1.5.2.2.2 Problema de optimización dual

Al igual que en el caso anterior, si el problema de optimización corresponde a un espacio de características de muy alta dimensionalidad, se lo transforma a su forma dual para facilitar su resolución. Luego, el problema a maximizar es

$$L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j < \boldsymbol{x_i}, \boldsymbol{x_j} > \tag{1.28}$$

[19]Imagen extraída de [62].

sujeto a las restricciones

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$
$$0 \leq \alpha_i \leq C,\ i = 1, \ldots, n \tag{1.29}$$

La solución del problema dual permitirá expresar el hiperplano de separación en términos de $\boldsymbol{\alpha}^*$ que, haciendo las sustituciones correspondientes, adoptará la misma forma que en la ecuación 1.24.

Todos los ejemplos x_i cuyo α_i asociado sea igual a cero corresponden a ejemplos separables ($\xi_i = 0$). Todos los ejemplos x_i cuyo $\alpha_i = C$ corresponden a ejemplos no separables ($\xi_i > 0$). Finalmente, un ejemplo x_i es vector de soporte sólo si $0 \leq \alpha_i \leq C$.

1.5.2.3. SVM para la Clasificación de Ejemplos No Lineales

El clasificador SVM permite, entonces, clasificar datos a partir del establecimiento de límites de decisión lineales. No obstante, es posible también crear límites de decisión no lineales, a partir de lo que se conoce como "el truco del kernel". Éste consiste en mapear el espacio de entradas a otro espacio (espacio de características), de dimensionalidad superior, a través de una función $K(x, x')$ conocida como *kernel*.

La función de decisión se obtiene modificando la expresión de la frontera de decisión en 1.24

$$D(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i^* y_i K(\boldsymbol{x}, \boldsymbol{x_i}) \tag{1.30}$$

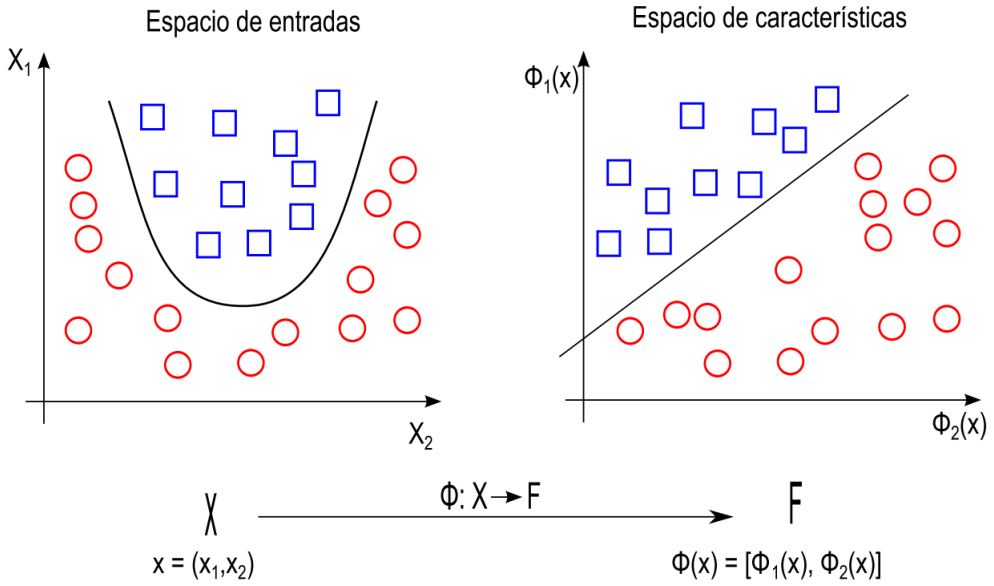


Figura 1.16: En la clasificación de ejemplos no lineales, el problema de búsqueda de una función de decisión lineal en el espacio del conjunto de ejemplos original (espacio de entradas) puede ser transformado a un nuevo problema, consistente en la búsqueda de una función de decisión lineal en un nuevo espacio transformado (espacio de características). La función ϕ es quien mapea el conjunto de entradas en el nuevo espacio del conjunto de características [62].[20]

[20]Imagen extraída de [62].

1.5.2.3.1 Problema de optimización dual

El problema a resolver sigue siendo encontrar el valor de los parámetros $\alpha_i^*,\ i = 1, \ldots, n$ que optimiza el problema dual 1.28, pero expresado ahora como

$$\begin{aligned} max \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \\ s.a. \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C,\ i = 1, \ldots, n \end{aligned} \tag{1.31}$$

Al igual que en los casos abordados anteriormente, el hiperplano de separación en términos de $\boldsymbol{\alpha^*}$ estará dado por 1.24.

Dado que un pequeño aumento en la dimensionalidad del espacio de entrada puede provocar un gran aumento en la dimensionalidad del espacio de características, el problema de optimización del caso no lineal se expresa sólo en su forma dual, que no depende de la dimensionalidad del espacio.

Las funciones kernel más habituales son:

- Lineal:
$$< x, x' > \tag{1.32}$$
- Polinómica:
$$\gamma < x, x' > + r)^d \tag{1.33}$$
- Gaussiana:
$$e^{-\gamma ||x - x'||^2} \tag{1.34}$$
- Sigmoidea:
$$\tanh(\gamma < x, x' > + r) \tag{1.35}$$

1.5.3. Redes Neuronales Artificiales

Una Red Neuronal Artificial (Artificial Neural Network, ANN), es un clasificador no lineal basado en una analogía con el sistema nervioso. Tiene por finalidad emular su capacidad de aprendizaje, de manera que la ANN pueda aprender a identificar un patrón de asociación entre los valores de un conjunto de variables predictoras (entradas) y los estados que se consideran dependientes de dichos valores (salidas) [46].

Las ANNs pueden ser definidas como estructuras constituidas por elementos simples de procesamiento conectados densamente entre sí, llamados '*neuronas artificiales*', capaces de ejecutar operaciones computacionales con un alto grado de paralelismo. Si bien estas redes constituyen una abstracción de sus contrapartes biológicas, su finalidad no es replicar la forma de operación de estos sistemas, sino aplicar los conocimientos que se tienen sobre ellos a la resolución de problemas. El gran atractivo de las ANNs reside en las excepcionales características de procesamiento de la información que presenta el sistema biológico: no linealidad, alto paralelismo, robustez, tolerancia a las fallas y errores, y su capacidad de generalización. Luego, los modelos artificiales inspirados en ellas son muy beneficiosos gracias a que:

1. la no linealidad permite un mejor ajuste de los datos,

2. la insensibilidad al ruido permite predicciones precisas ante la presencia de datos inciertos o errores de medición,

3. un alto paralelismo implica una alta velocidad de procesamiento,

4. el aprendizaje y adaptabilidad permiten al sistema adaptar su estructura interna en respuesta a los cambios producidos en el entorno,

5. y la generalización permite la aplicación del modelo a nuevos datos no aprendidos [3].

Para que un sistema sea considerado como una red neuronal artificial, debe contener una estructura de grafo dirigido y etiquetado en el que cada nodo realice algún cómputo simple. Estos grafos están constituidos por un conjunto de nodos y de conexiones que los unen entre sí. Cada conexión transporta una señal de un nodo a otro, que constituye la salida de la operación computacional, etiquetada por un peso que indica el grado en que la señal es amplificada o atenuada. Aquellas conexiones con grandes pesos positivos amplifican la señal, indicando que es muy importante para realizar cierta clasificación; mientras que lo opuesto sucede para los pesos muy negativos. Este sistema constituye entonces una red neuronal, si los pesos de sus conexiones son modificables mediante algoritmos de aprendizaje [59]. La analogía entre una neurona artificial y una biológica reside en que las conexiones entre los nodos representan a los axones y dendritas, los pesos de las conexiones representan las sinapsis y el umbral establecido representa la actividad dentro del soma [3].

1.5.3.1. La unidad neuronal

La neurona artificial, que se ilustra en la figura 1.17, constituye el componente computacional básico de las redes neuronales. Consiste en una unidad de umbral binaria, que computa una sumatoria de n señales de entrada x_j, $j = 1, 2, \ldots, n$, ponderadas por sus pesos w_j; generando una salida igual a 1, si la suma está por encima de determinado umbral b, o a 0, en caso contrario [21].

$$y = \sigma\left(\sum_{j=1}^{n} w_j x_j - b\right) \tag{1.36}$$

La suma ponderada $\xi = \sum \boldsymbol{w} \cdot \boldsymbol{x}$ constituye el potencial postsináptico de la neurona, al cual se aplica una función de activación σ, que define su umbral de excitación. Cuando la salida de la neurona es igual a 1, se dice que está activada o encendida y presenta el estado 1, mientras que si su salida es igual a cero, se dice que está desactivada o apagada, presentando el estado 0. La señal de salida y para n señales de entrada se puede expresar como:

$$y = \begin{cases} 1, & si \ \sum_{j=1}^{n} w_i x_i \geq b \\ 0, & si \ \sum_{j=1}^{n} w_i x_i < b \end{cases} \tag{1.37}$$

A fin de simplificar el entrenamiento, el umbral b se expresa como un peso sináptico más, de magnitud $w_0 = -b$, pero asociado a una neurona siempre activa (x_0). Esta neurona se denomina *bias*, y se sitúa en la capa de entrada de las redes neuronales [57].

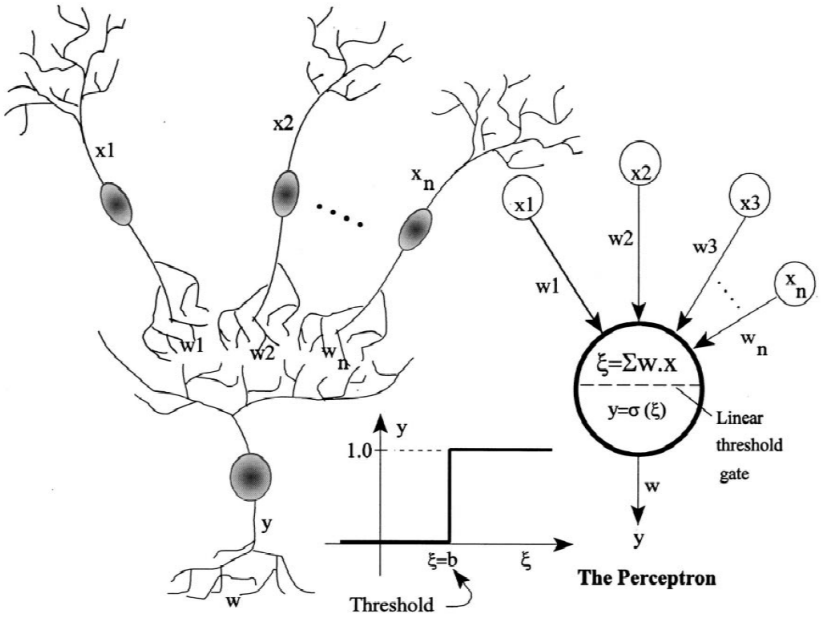


Figura 1.17: Interacción entre n neuronas biológicas (izquierda) y sumatoria de señales en una neurona artificial constituida por el perceptrón simple (derecha) [3]. [21]

1.5.3.2. Perceptrón Multicapa

El perceptrón multicapa (Multilayer Perceptron, MLP) es un tipo de red neuronal artificial derivada del perceptrón simple o de capa única, que sólo puede realizar clasificaciones binarias sencillas y en datos linealmente separables. Estas limitaciones son superadas por el perceptrón multicapa que, gracias a su arquitectura, puede resolver problemas más complejos o no linealmente separables. Se caracteriza por estar compuesto por neuronas agrupadas en diferentes capas, de las que se distinguen tres tipos: de entrada, ocultas y de salida. La entrada de cada neurona está conectada con las salidas de las neuronas de la capa precedente, constituyendo lo que se conoce como arquitectura *feed-forward*, es decir, de propagación hacia adelante. En ésta, los datos fluyen siempre en una única dirección, desde la capa de entrada, propagándose por las capas ocultas, hasta alcanzar la capa de salida. Las neuronas de la capa de salida determinan la clase del vector de características recibido como entrada por la red.

Una configuración típica de este tipo de MLP se muestra en la figura 1.18, donde cada neurona es representada por un círculo. Las neuronas están organizadas en tres capas: capa de entrada (*input layer*), capa oculta (*hidden layer*) y capa de salida (*output layer*). La primera capa no constituye verdaderamente una capa de neuronas, ya que las unidades que la componen no actúan como neuronas propiamente dichas, sino que simplemente se encargan de recibir la información y propagarla a la siguiente capa. Las neuronas de la capa oculta procesan los patrones recibidos y los propagan a la capa siguiente que, en este caso, corresponde a la capa de salida. Finalmente, la capa de salida entrega los resultados de la clasificación para cada uno de los patrones de entrada [50].

[21] Imagen extraída de [3].

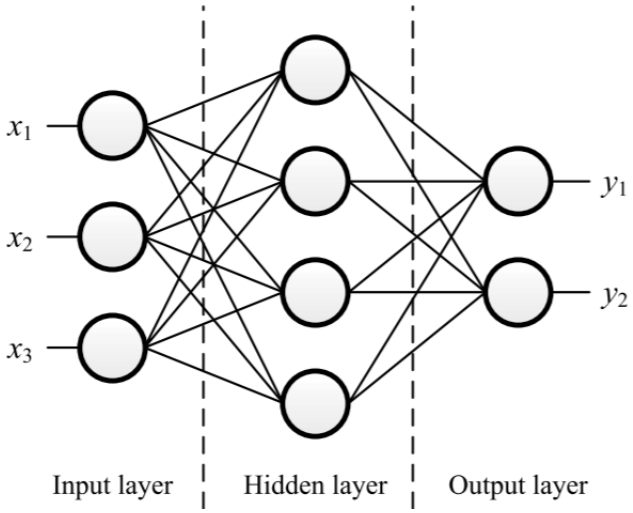


Figura 1.18: Configuración típica de un MLP de 3 capas. [22]

1.5.3.3. Redes Convolucionales

Las Redes Neuronales Convolucionales (CNNs) son variantes de los MLPs. A diferencia de estos últimos, en que cada neurona en la capa de entrada está conectada a cada neurona de la siguiente capa (capas *fully-connected*, totalmente conectadas, en español), en las CNNs se utilizan capas 'localmente conectadas'. En éstas, las neuronas se conectan únicamente a una pequeña región de la capa anterior. A cada capa se le aplica una función de activación, como las definidas en 1.5.3.4. Cada capa en una CNN aplica una serie de filtros y combina sus respectivos resultados, alimentando la salida a la siguiente capa en la red. La sucesión de capas convolucionales, luego, constituye una cadena de filtros no lineales, en que la operación de convolución permite manipular datos bidimensionales; al contrario de los MLPs, que sólo pueden operar en vectores unidimensionales. Las capas totalmente conectadas, por su parte, no se emplean sino hasta el final de la red, constituyendo la o las últimas capas de la arquitectura. [59, 64].

1.5.3.3.1 La operación de convolución

En términos de aprendizaje profundo (*deep learning*), la convolución es una multiplicación lugar a lugar de dos matrices, seguida de la suma de los productos resultantes. Las matrices que se multiplican son la matriz de entrada X, y la matriz *kernel* K, que constituye el filtro de la matriz de entrada, la que habitualmente corresponde a una imagen. En términos de operaciones convolucionales, un kernel puede ser visualizado como una pequeña matriz que se desplaza a lo largo de una matriz de mayor tamaño, de izquierda a derecha, y de arriba hacia abajo. Para cada elemento de la matriz, los elementos circundantes ('vecinos') son convolucionados con el kernel y el resultado obtenido es almacenado [59].

[22] Imagen extraída de [74].

Figura 1.19: Ilustración del desplazamiento del kernel, en rojo, sobre la matriz durante la operación de convolución.[23]

1.5.3.3.2 Capas convolucionales

La capa convolucional es el bloque constitutivo de una red CNN. Consta de una serie de k filtros (kernels), donde cada uno tiene una altura y ancho definidos, y son casi siempre cuadrados. Estos kernels se desplazan a lo largo de la región de entrada, computando la operación de convolución y almacenando el valor en un *'mapa de activación'* bidimensional. La magnitud del desplazamiento del kernel sobre la matriz está dada por el parámetro *stride*, que regula el desplazamiento tanto en alto como en ancho. Valores pequeños de stride resultan en el solapamiento de los campos receptivos y valores de salida grandes; mientras que lo opuesto sucede para valores de stride grandes. A diferencia de las redes neuronales estándar, las capas de una CNN están organizadas en un volumen 3D: ancho, alto y profundidad. Las neuronas de las capas subsecuentes se conectan únicamente a una pequeña región de la capa anterior, 'conectividad local', que recibe el nombre de *campo receptivo F*. Finalmente, la capa de salida será un vector que contenga las probabilidades de pertenencia a cada clase.

[23]Imagen extraída de [59].

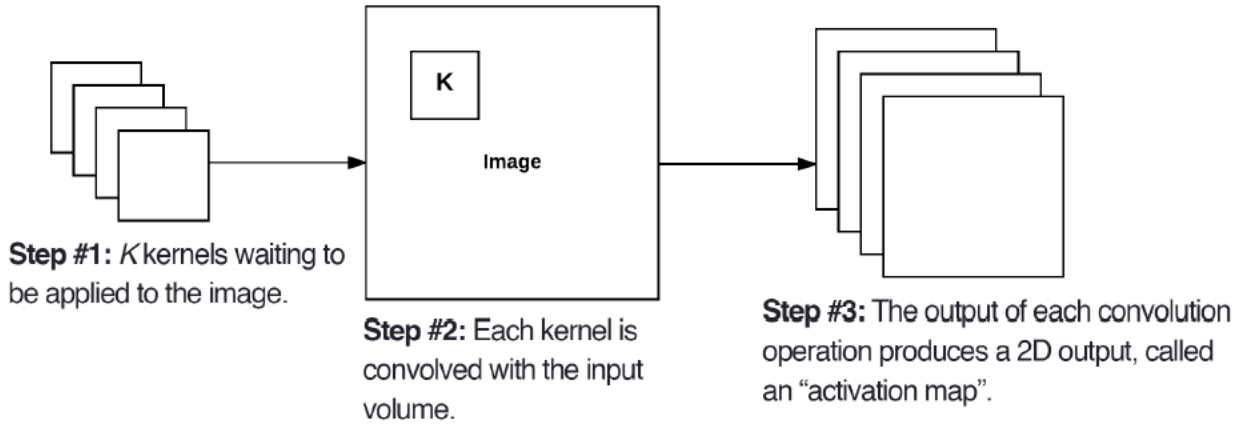


Figura 1.20: Izquierda: para cada capa en una CNN, hay k kernels que se aplican al volumen de entrada. Medio: cada uno de los k kernels lleva es convolucionado con el volumen de entrada. Derecha: cada kernel produce una salida 2D, llamada 'mapa de activación' [59].[24]

1.5.3.4. Funciones de activación

Las funciones de activación más ampliamente utilizadas en las redes neuronales son:

- Lineal (linear):
$$f(x) = x \tag{1.38}$$
- Sigmoidea (sigmoid):
$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.39}$$
- Unidad Rectificadora Lineal (ReLU):
$$f(x) = max(0, x) \tag{1.40}$$
- Tangente hiperbólica (tanh):
$$f(x) = \tanh x \tag{1.41}$$

1.5.3.4.1 La capa softmax

En los problemas de clasificación, es habitual emplear una arquitectura de red neuronal con una salida categórica que entregue las probabilidades de pertenencia de los datos analizados a cada una de las clases. Este tipo de clasificadores recibe el nombre de '*clasificador softmax*', y se construye colocando en su capa de salida tantas neuronas como clases tenga el problema, y utilizando la función de activación *softmax*. Esta capa, llamada 'capa softmax', normaliza las salidas de la capa previa a fin de que su suma sea igual a la unidad. De esta manera, la salida de cada neurona representa la probabilidad de pertenencia a la clase correspondiente [26].

[24]Imagen extraída de [59].

1.5.3.5. Aprendizaje de las redes neuronales

El aprendizaje de una red neuronal es el proceso mediante el cual se van ajustando los pesos sinápticos para minimizar la función de costo elegida y así lograr que las entradas recibidas produzcan las salidas deseadas. Se emplea un algoritmo de aprendizaje supervisado que, para lograrlo, modifica los pesos de las conexiones de la red, minimizando así la función de costo.

Las redes feed-forward se entrenan mediante el método de retropropagación, que consta de dos fases. En la primera, se propaga la entrada por las sucesivas capas hasta producir una salida, a partir de la cual se calcula el error de la red de acuerdo a la función de costo elegida. En la segunda fase, este error se propaga hacia atrás, partiendo de la capa de salida hacia las neuronas de las capas ocultas. De esta manera, se van ajustando sucesivamente los pesos de la red. El proceso se repite en cada época (iteración) hasta lograr que el error se aproxime lo máximo posible a cero, o hasta alcanzar algún límite fijado durante el entrenamiento (número de épocas, ausencia de cambio en cierto número de épocas, entre otros).

1.5.3.5.1 Función de Costo

Las funciones de costo cuantifican el desempeño de un modelo en la clasificación de los datos. En el caso de los clasificadores softmax, en que las salidas son categóricas y corresponden a un valor de probabilidad, se utiliza la función de costo de entropía cruzada (*Cross-entropy loss*). Esta función se basa en el principio de máxima verosimilitud, en el que se intenta maximizar las probabilidades de pertenencia a la clase correspondiente para cada patrón del conjunto de datos de entrenamiento. De manera equivalente, y para simplificar los cálculos, usualmente se minimiza el producto del logaritmo de las probabilidades de pertenencia. Su valor se incrementa a medida que la probabilidad predicha diverge del valor de la verdadera etiqueta de clase [39]. La contraparte binaria de esta función de costo recibe el nombre de *Binary Cross-entropy* (entropía cruzada binaria).

1.5.3.5.2 Optimización

La no linealidad de las redes neuronales trae como consecuencia el hecho de que las funciones de costo se tornen no convexas, lo que implica que no poseen un único mínimo. En consecuencia, las redes neuronales son entrenadas usando optimizadores iterativos basados en el gradiente de descenso de la función de costo, a fin de llevarla a un valor tan bajo como sea posible.

La figura 1.21 muestra un ejemplo de superficie de optimización en dos dimensiones, en que en el eje x se representan los pesos y en el eje y la función de costo correspondiente a esos pesos. Esta superficie presenta numerosos picos y valles de acuerdo a los valores que adopten los parámetros del modelo. Cada pico corresponde a un máximo local, que representa regiones de pérdida muy altas. El máximo local con la mayor pérdida a lo largo de toda la superficie de optimización es el máximo global. Análogamente, los mínimos locales representan regiones de pérdida pequeñas y, aquel que presente la menor pérdida a lo largo de toda la superficie es el mínimo global.

La solución *ideal* del problema de optimización reside en encontrar el mínimo global en la superficie de optimización. Sin embargo, esta superficie es desconocida para el algoritmo y sólo se conocen de ella valores puntuales, obtenidos luego de cada iteración. Por lo tanto, si bien un algoritmo de optimización no garantiza encontrar siquiera un mínimo local, usualmente encuentra valores muy bajos de la función de costo lo suficientemente rápido como para que resulten de utilidad[18, 59].

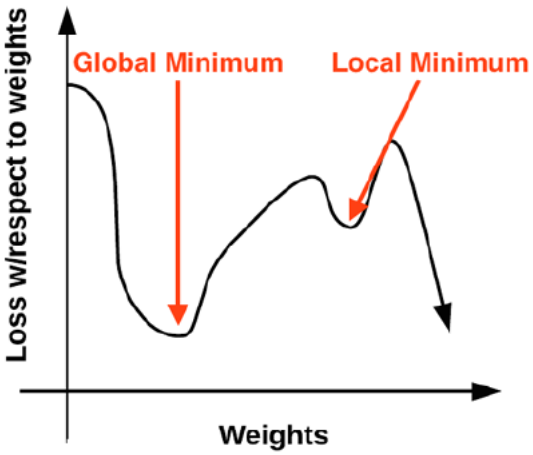


Figura 1.21: Superficie de optimización visualizada en dos dimensiones.[25]

1.5.3.5.2 Descenso de gradiente estocástico

Entrenar una red neuronal es equivalente a resolver el siguiente problema de optimización no-convexo:

$$\min_{w \in \mathbb{R}^n} f(w) \tag{1.42}$$

donde f es la función de costo. El Descenso de Gradiente Estocástico (SGD) es un algoritmo iterativo de optimización que opera sobre la suferficie de optimización definida por f. Las iteraciones del SGD pueden ser descriptas como:

$$w_k = w_{k-1} - \alpha_{k-1} \nabla f(w_{k-1}) \tag{1.43}$$

donde w_k denota la k-ésima iteración, α_k es el tamaño del paso de avance, llamada *learning rate* (tasa de aprendizaje) y $\nabla f(w_{k-1})$ denota el gradiente estocástico computado en w_k [25].

1.5.3.6. Dropout

El *dropout* es una forma de regularización que pretende evitar el sobreajuste mediante el incremento de la exactitud en el conjunto de prueba, a expensas de una disminución de ésta en el conjunto de entrenamiento. Para cada bloque de datos en el conjunto de entrenamiento, las capas de dropout desconectan aleatoriamente un cierto número de entradas de la capa precedente, a partir de un factor p que indica la fracción de conexiones a anular en esa instancia del entrenamiento.

La figura 1.22 ilustra este concepto: se deconectan de manera aleatoria las conexiones entre dos capas totaltamente conectadas de un cierto lote de entrenamiento (*minibatch*), con una tasa de dropout del 50 %. Luego de que los pasos hacia delante y hacia atrás se computen para este minibatch, se reconectan las conexiones descartadas y, en la siguiente iteración, se muestrea otro conjunto de conexiones para desechar. Esta técnica se aplica con el objeto de reducir el sobreajuste mediante la alteración explícita de la arquitectura de la red al momento del entrenamiento. La eliminación aleatoria de conexiones asegura que ningún nodo en la red es responsable de su activación cuando se presenta una característica en particular. Por el

[25] Imagen extraída de [59].

contrario, el dropout asegura la existencia de múltiples y redundantes nodos que se activen en presencia de entradas similares, lo que ayuda al modelo a generalizar [59].

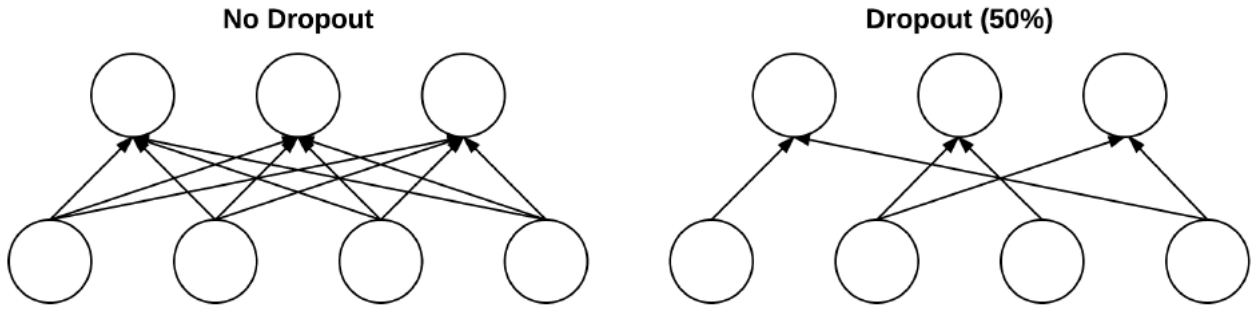


Figura 1.22: Izquierda: dos capas de una red neuronal están completamente conectadas, sin dropout. Derecha: las mismas dos capas, con un factor de dropout del 50 %, en que la mitad de las conexiones fueron descartadas [59].[26]

1.6. Estado del arte de los sistemas BCI basados en el P300

En esta sección se realiza una breve revisión de la literatura sobre las técnicas y algoritmos aplicados en el diseño de sistemas BCI similares al propuesto en este trabajo, es decir, basados en la detección del potencial P300.

1.6.1. Acondicionamiento de la señal

En el acondicionamiento de las señales EEG para la posterior identificación del P300, se emplean numerosas técnicas, dentro de las cuales las más típicas comprenden las operaciones de filtrado [24, 27, 10, 31, 70, 9, 60, 65, 58, 15], para la eliminación de ruido y artefactos de la señal y conservación de las bandas de frecuencia más relevantes; submuestreo [9, 33, 10, 65, 58, 15], para reducir la dimensionalidad del problema a abordar; y segmentación en épocas [24, 27, 31, 60, 9, 33, 10, 65, 15], para definir una ventana de tiempo dentro de la cual esté comprendido.

1.6.2. Extracción de características

Al ser el P300 una señal que depende temporalmente del estímulo, en la gran mayoría de los abordajes de su estudio se extraen características temporales, concatenando en un vector los diferentes segmentos de señal obtenidos en cada canal [65, 67]. Algunos autores, no obstante, también extraen características espaciales, mediante la aplicación de filtros espaciales [46, 58, 10]; o frecuenciales, a partir de la utilización de la función ondita [41, 53, 6, 13, 33], potencia de las bandas frecuenciales [52, 33], entre otras.

1.6.3. Algoritmos de clasificación

Existen numerosos algoritmos de clasificación aplicados al estudio del potencial P300. Los más ampliamente usados, que fueron seleccionados para su implementación en el presente es-

[26]Imagen extraída de [59].

tudio, se pueden dividir en dos categorías: clasificadores lineales y redes neuronales. Dentro de los clasificadores lineales, los más difundidos son el LDA [52, 12, 46, 67, 42] y el SVM [46, 52, 29, 40, 54]. Respecto de las redes neuronales, los más empleados son el perceptrón multicapa [42, 27, 63] y las redes neuronales convolucionales [9, 32, 60, 37], que encuentran actualmente un creciente atractivo en el abordaje de BCIs, dado que permiten analizar las señales originales en los dominios temporal y espacial, sin necesidad de preprocesamiento previo.

En general, de la revisión del estado del arte de este tipo de BCI, se puede concluir que la mayoría de las BCIs basadas en el P300 implementan métodos de filtrado y submuestreo en el acondicionamiento de la señal, y luego extraen segmentos de señal comprendidos en ventanas temporales lo suficientemente amplias para abarcar el potencial P300. Si bien muchas veces se extraen características frecuenciales y espaciales, las más usuales en este tipo de BCI consisten en características temporales, dada la naturaleza de la onda. Los clasificadores más ampliamente implementados corresponden al LDA y SVM, pero en los últimos años se ha mostrado un creciente interés en la utilización de redes neuronales para su detección. En el caso particular del deletreador P300, se presenta siempre el desafío de obtener un alto porcentaje de predicción correcta de los caracteres, con un tiempo de entrenamiento tan breve como sea posible. En general, los porcentajes de predicción de caracteres en los trabajos analizados varían de aproximadamente 75 % a 95 % [9, 60].

Capítulo 2

Materiales y métodos

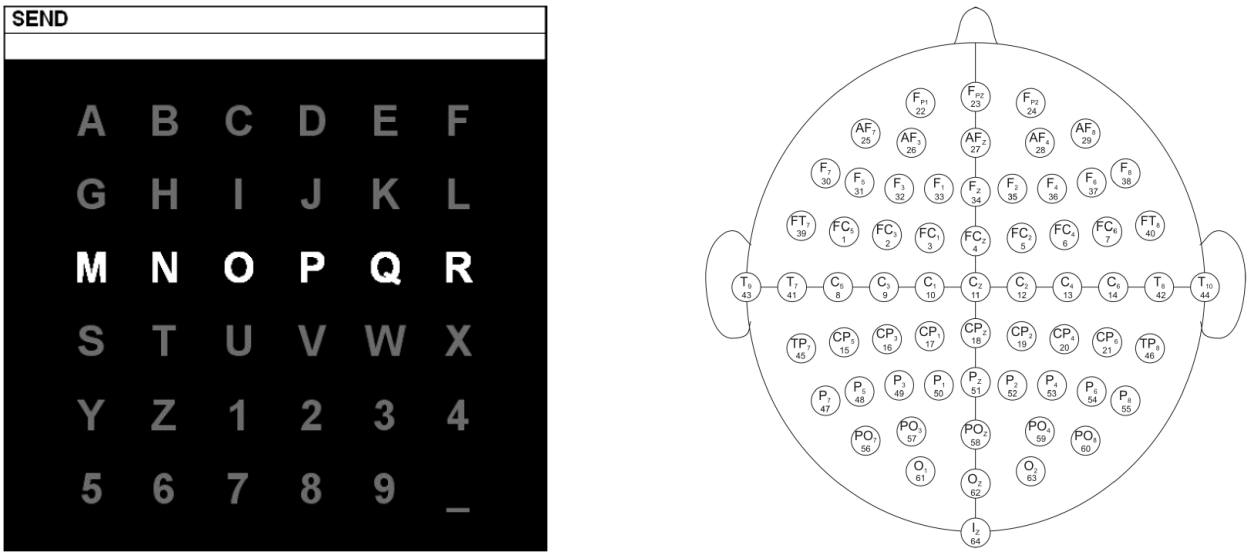
Los experimentos llevados a cabo en la presente investigación tuvieron como fin evaluar diferentes algoritmos de clasificación propuestos en la revisión del estado del arte para la detección del potencial evocado P300 a partir de registros electroencefalográficos; y poder predecir a partir de ellos los caracteres. En primer lugar, se presenta una descripción de la base de datos, seguida de las etapas de preprocesamiento, extracción de características y clasificación de los diferentes algoritmos.

2.1. Base de datos empleada

Los experimentos se llevaron a cabo sobre la base de datos II de la tercer competencia de BCI (*"BCI Competition III"*), realizada en el año 2004, correspondiente a un deletreador P300. En esta competencia, organizada por Benjamin Blankertz, los datos fueron adquiridos en el laboratorio de Interfaces Cerebro Computadora del Centro Wadsworth del Departamento de Salud del Estado de Nueva York, encabezado por J. R. Wolpaw. Una descripción completa de la base de datos, incluyendo los vínculos de descarga, está disponible online [5]. La base de datos consiste en registros EEG de 64 canales tomados de dos sujetos a partir de la plataforma de Interfaces Cerebro Computadora BCI2000. Las señales fueron adquiridas durante diversas sesiones de deletreo, basadas en el paradigma clásico del deletreador P300 descripto por Dochin et al [14].

2.1.1. Descripción del diseño experimental

En el experimento, se presenta al usuario una matriz de caracteres (letras, números y símbolos) de 6 filas por 6 columnas (figura 2.1a), y se le asigna la tarea de concentrar su atención sobre los caracteres de una determinada palabra, de a uno por vez. Todas las filas y columnas son sucesivamente intensificadas de manera aleatoria a una frecuencia de 5.7 Hz. De estos 12 estímulos (intensificación de las 6 filas y 6 columnas), sólo dos corresponden a la fila y columna que contienen el carácter deseado. Por lo tanto, y de acuerdo al paradigma oddball, estos dos estímulos, por ser infrecuentes y relevantes para la tarea en desarrollo, inducirán un potencial P300 en el registro electroencefalográfico del usuario. En cambio, la respuesta generada por el resto de los estímulos, correspondiente a las intensificaciones de las filas y columnas que no contienen el carácter en cuestión, será diferente ya que, al ser de mayor frecuencia e irrelevantes para el usuario, no producirán dicho potencial.



(a) (b)

Figura 2.1: Interfaz gráfica empleada y disposición de los electrodos en la adquisición de datos de la base de datos II de la competencia BCI III. (a)Matriz de caracteres presentada al usuario durante la sesión de deletreo. En el ejemplo, se puede ver el tipo de estímulo empleado en el experimento: intensificación de las filas (y columnas). (b) Disposición de los electrodos de acuerdo al sistema 10-20 y nómina de los canales correspondientes.[1]

2.1.2. Adquisición de los datos

En la adquisición de los datos que integran la base de datos empleada (sección 2.1) las señales fueron tomadas a partir de dos sujetos (A y B) en cinco sesiones cada uno. Para ello, se dispusieron 64 electrodos sobre su cuero cabelludo, de acuerdo al sistema 10-20 (figura 2.1b), obteniendo los correspondientes 64 canales en el registro EEG. Las señales recogidas fueron sometidas a un filtro pasa banda de 0.1 - 60 Hz y digitalizadas a 240 Hz. En la descripción de la base de datos empleada no se especifican las características del equipamiento utilizado.

Por cada época de caracteres, correspondiente al tiempo requerido para la selección de cada carácter, el procedimiento seguido consistió en la siguiente serie de pasos: primero, se mostraba la matriz en blanco (todos los caracteres con igual intensidad) durante 2.5 segundos. Luego, se intensificaban de manera aleatoria cada una de las filas y columnas, con un período de 175 ms (5.7 Hz), constituyendo un total de 12 estímulos diferentes. Cada período constaba de 100 ms en que la fila/columna era intensificada y 75 ms posteriores, en que la matriz permanecía en 'blanco' hasta la intensificación de la siguiente fila/columna. Este conjunto de 12 intensificaciones, de aquí en adelante referido como '*trial*', fue repetido 15 veces por cada época de caracteres, dando un total de 180 intensificaciones por carácter. Finalmente, la matriz era nuevamente mostrada en blanco, dando inicio a una nueva época de caracteres. Luego, el tiempo de selección por carácter es de $175\ ms \cdot 12 \cdot 15 + 2{,}5\ s = 34$ segundos.

[1] Imágenes extraídas de [5].

2.1.3. Estructura de la base de datos

El dataset contiene información de los dos sujetos (A y B) y consta de un conjunto de entrenamiento '*train*' (85 caracteres, etiquetados) y uno de prueba '*test*' (100 caracteres, sin etiquetar) para cada uno. El tiempo total de los registros para cada sujeto es de $85 \cdot 34\ s = 2890\ s = 48$ minutos, para el conjunto de entrenamiento; y $100 \cdot 34\ s = 3400\ s = 57$ minutos, para el conjunto de prueba. Además de los registros de EEG de los 64 canales, contenidos en una matriz de datos (*Signal*), la base de datos contiene una serie de variables auxiliares que codifican los eventos asociados a cada muestra contenida en Signal:

Flashing:	1 cuando la fila/columna era intensificada 0 en caso contrario
StimulusCode:	0 cuando ninguna fila/columna era intensificada 1...6 columnas intensificadas, de izquierda a derecha 7...12 filas intensificadas, de arriba a abajo
StimulusType:	0 si ninguna fila/columna era intensificada o la fila/columna intensificada no contenía el carácter deseado 1 si la fila/columna intensificada contenía el carácter deseado
TargetChar:	Etiqueta del carácter correspondiente a cada época de caracteres

La estructura de cada una de las variables se muestra en la siguiente tabla:

Tabla 2.1: Dimensiones de las variables contenidas en la base de datos.

Variable	Dimensión 1	Dimensión 2	Dimensión 3
Signal	Épocas de caracteres	Muestras	Canales
Flashing	Épocas de caracteres	Muestras	-
StimulusCode	Épocas de caracteres	Muestras	-
StimulusType	Épocas de caracteres	Muestras	-
TargetChars	Épocas de caracteres	-	-

2.2. Herramientas de software empleadas

La presente investigación fue llevada a cabo enteramente en el lenguaje de programación Python, debido a su amplio uso en ciencia de datos y aprendizaje automático, la gran variedad de librerías que ofrece y la posibilidad de interactuar directamente con el código a partir de la terminal o a través de herramientas como Jupyter Notebook [51]. Dentro de las librerías empleadas, cabe destacar las más relevantes:

- Numpy[23] es una librería numérica que permite una manipulación rápida y sencilla de vectores y matrices, e introduce una estructura de datos llamada *array*, que puede adoptar n dimensiones y permite manipular datos del mismo tipo. Además, los *numpy arrays* constituyen la estructura básica empleada en la librería scikit-learn.

- Matplotlib [20] permite generar una variedad de gráficos a partir de datos contenidos en listas o arrays, ofreciendo una interfaz y funcionalidades similares a las empleadas en MatLab.

- Wyrm [66] librería de código abierto para interfaces cerebro computadora, apta para ejecutar experimentos BCI on-line y realizar análisis off-line de datos electroencefalográficos.

- Scikit-learn [49] es una librería ampliamente difundida en el ámbito de machine learning y presenta la ventaja de que, al ser un proyecto de código abierto, es posible acceder al código fuente para comprender el funcionamiento detrás de cada función. Además, está ampliamente documentada y cuenta con una comunidad activa que facilita el proceso de aprendizaje.

- Tensorflow [1] es una librería de código abierto para computación numérica de alto desempeño. Consta de una arquitectura flexible que permite desplegar las tareas computacionales en diferentes plataformas: CPUs, GPUs, TPUs. Es empleada para definir gráficos computaciones abstractos de propósito general, aunque también es muy empleada en machine learning y, fundamentalmente, en deep learning.

- Keras [11] es una librería de código abierto diseñada para redes neuronales, que permite crear de manera sencilla una gran variedad de modelos de Deep Learning, usando de fondo librerías como TensorFlow, Theano o CNTK.

Por su parte, Jupyter Notebook es un entorno interactivo de ejecución de código en el navegador. Soporta diversos lenguajes de programación y consta de diferentes celdas que permiten integrar en un mismo documento código, texto, imágenes, videos y ecuaciones.

2.3. Tratamiento de la base de datos

Como se describió en la sección 2.1.3, la base de datos está compuesta por la matriz Signal, que contiene los datos de los registro EEG obtenidos durante los experimentos, y una serie de variables auxiliares, que constituyen la *metadata* y permiten interpretar cada una de las muestras contenidas en Signal.

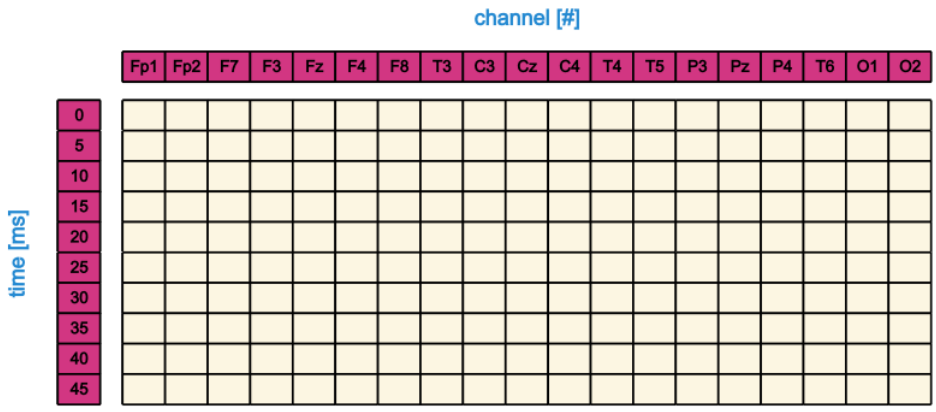


Figura 2.2: Estructura del objeto `Data` de la librería Wyrm. Se puede apreciar en el centro, la matriz de datos signal, con los correspondientes atributos `time` y `channel`.[2]

Para facilitar su manipulación, se decidió reunir todas estas variables en un único objeto llamado `Data`, correspondiente a la estructura de datos empleada en Wyrm. Este objeto contiene como base un arreglo $n-$dimensional para almacenar los datos, '`data`', y un set de atributos que los describen y que constituyen la metainformación. Estos son: `axes`, que describe las filas y columnas de los datos contenidos en `Data`; `names`, arreglo de caracteres que almacena el nombre de cada una de sus dimensiones y `units`, unidades físicas de los datos.

2.3.1. Visualización de las señales

En la figura 2.3 se presentan las señales EEG obtenidas durante las sesiones de entrenamiento, expresadas en una época de 1200 ms. Ésta fue definida a partir de una ventana que se extiende desde los 400 ms anteriores a la aparición del estímulo, representado por una línea roja vertical en $t = 0$, hasta los 800 ms posteriores. Las señales se presentan para ambos sujetos y se dividen de acuerdo al tipo de estímulo que las produjo: target y nontarget.

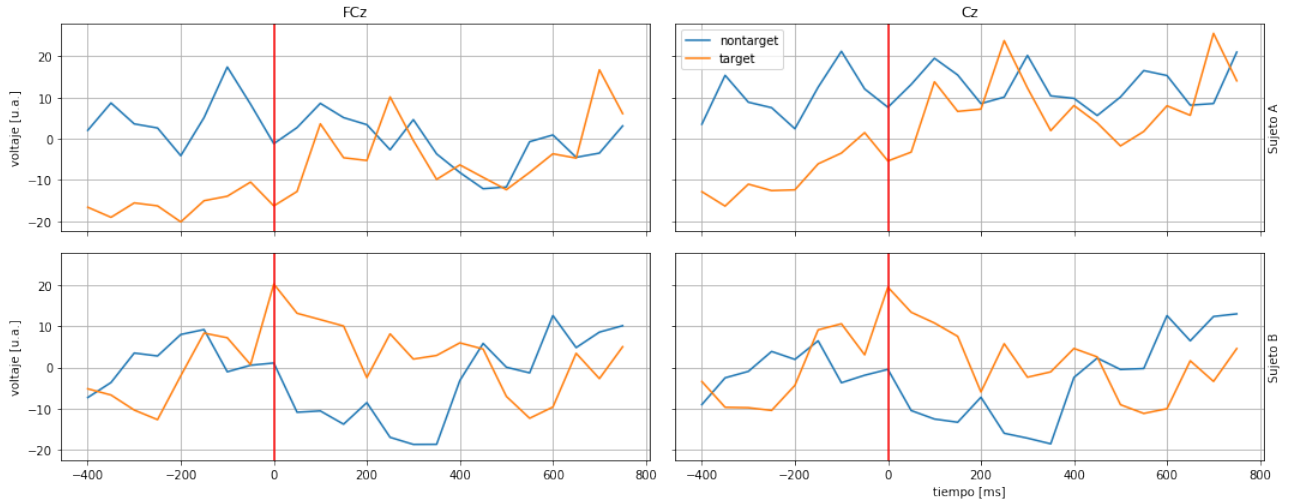


Figura 2.3: Señales obtenidas por clase para una época de caracteres, sobre los canales FCz y Cz. Las líneas rojas verticales en $t = 0$ indican el momento de presentación del estímulo. En la fila superior se presentan los gráficos correspondientes al sujeto A, y en la inferior al sujeto B.

De acuerdo al paradigma oddball, se sabe que los estímulos target deberían generar como respuesta una deflexión positiva aproximadamente 300 ms después de la aparición del estímulo, correspondiente al P300. Sin embargo, estos potenciales son de muy pequeña amplitud, por lo que quedan enmascarados por la actividad electroencefalográfica de fondo, como se puede apreciar en la figura, por lo que es muy difícil identificarlos. En el caso del sujeto A, se identifica un pico para $t = 250$ ms pero, por su baja magnitud, no resulta evidente a simple vista. Se puede ver que este pico es más pronunciado para el canal Cz. En cuanto al sujeto B, si bien en las señales correspondientes al evento target se puede identificar un pico positivo, éste, por comenzar a generarse alrededor de 300 ms antes, no está ligado al estímulo en cuestión. Por lo tanto, o bien no se produce un potencial P300, o bien queda enmascarado en el pico que comenzó a gestarse antes del evento, correspondiente posiblemente a un estímulo target anterior.

Esta aparente contradicción con respecto a la teoría tiene por fundamento dos motivos principales: 1) la señal contiene mucho ruido, lo que entorpece la extracción de información relevante; 2) la generación del potencial P300 está íntimamente ligada al sujeto, tanto en la

[2]Imagen extraída de [66].

forma y latencia con que se presenta, como en el mismo hecho de que se presente en sí. Esto último hace referencia al estado de concentración del sujeto, que es determinante en la aparición y magnitud del potencial P300. Es un factor fundamental a tener presente durante el transcurso de la presente investigación, dado que refleja que el paradigma se basa en la asunción de que los estímulos target siempre producirán un P300.

2.3.1.1. Grandes promedios

Como se observa en la sección anterior, el potencial P300 es difícil de observar en una sola época, debido a su variabilidad y baja amplitud. Por ende, se recurre al promediado de todas las instancias de la base de datos correspondientes a esa clase, lo que recibe el nombre de 'grandes promedios'. En la figura 2.8 se representan los grandes promedios de las respuestas a los estímulos target y nontarget para ambos sujetos, obtenidas a partir de los electrodos FCz y Cz. Se entiende por gran promedio al promedio computado sobre todas las instancias correspondientes a cada clase contenidas en la base de datos [38].

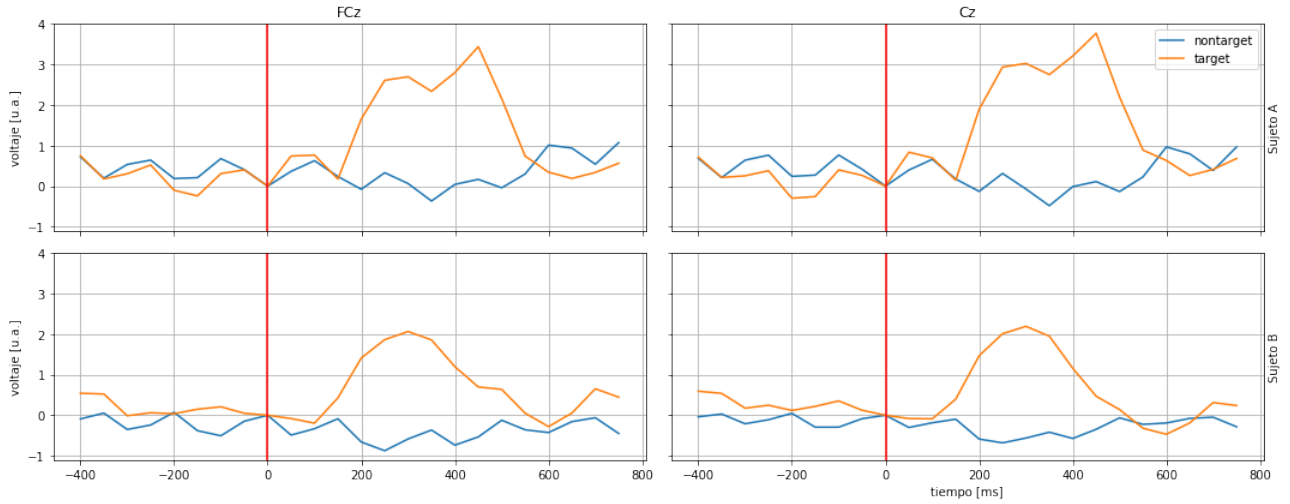


Figura 2.4: Grandes promedios calculados por clase sobre todas las épocas de caracteres del conjunto de entrenamiento, sobre los canales FCz y Cz. Las líneas rojas verticales en $t = 0$ indican el momento de presentación del estímulo. En la fila superior se presentan los gráficos correspondientes al sujeto A, y en la inferior al sujeto B.

En el caso de las señales nontarget, se aprecian oscilaciones de período comprendido entre 150 y 200 ms aproximadamente, lo que se condice con la frecuencia de aparición de los estímulos durante la ejecución del experimento (5.7 Hz). Este fenómeno se conoce con el nombre de Potenciales Visuales Evocados de Estado Estacionario (SSVEP, por sus siglas en inglés) y refleja la actividad cerebral producto del procesamiento visual de estímulos de frecuencia superiores a 6 Hz [17].

Respecto de las curvas obtenidas para la clase target, se puede observar claramente la presencia de los potenciales P300 para ambos sujetos, comprendidos dentro de una ventana de 150 a 500 ms. Asimismo, se hace evidente el carácter fuertemente ligado al sujeto del potencial en cuestión. Son notorias las diferencias de la forma que adopta para cada uno de ellos, pudiendo distinguir una curva más roma y de menor ancho para el sujeto B; y una curva en forma de 'M' y de mayor amplitud y extensión en el dominio del tiempo para el sujeto A. En particular, las características descriptas para las curvas de este último son producto de una mayor variación

en la latencia del P300, por lo cual, al promediar, se extiende más en el tiempo y se obtienen los dos picos que dan la forma de 'M' a la curva.

De lo analizado en estas secciones se puede concluir que no es posible discernir los eventos P300 en una sola época, sino que es preciso repetir el experimento para un mismo carácter un cierto número de veces para, a partir del promediado de las diferentes instancias, lograr discriminar este potencial de la actividad electroencefalográfica de fondo. Asimismo, la variabilidad intersujeto dificulta el establecimiento de un método estándar de reconocimiento del potencial para diferentes sujetos, ya que la forma en que se presenta para cada uno es muy variable. Estas dificultades constituyen desafíos a sortear en la clasificación de los datos recolectados a partir de la interfaz del deletreador, por lo cual resulta indispensable aplicar métodos de aprendizaje automático que, una vez entrenados con suficiente cantidad de datos, detecten patrones en las señales que permitan discriminar los potenciales P300 de la actividad cerebral de fondo.

2.4. Ampliación de la base de datos mediante el método de Bootstrapping

La base de datos analizada cuenta únicamente con datos obtenidos a partir de dos sujetos, A y B. Esto resulta inconveniente a la hora de analizar los algoritmos, dada la baja cantidad de sujetos de prueba para evaluar su desempeño y obtener resultados globales que reflejen el grado de robustez del modelo. En consecuencia, se decidió ampliar la base de datos de entrenamiento mediante la 'generación de sujetos', aplicando la técnica de Bootstrapping.

El método de bootstrap constituye una técnica estadística aplicada a la estimación de cantidades de una población, mediante el promediado de múltiples muestras de datos. Estas muestras se construyen tomando ejemplos de un conjunto de datos grande, de a uno por vez, y devolviéndolos a este conjunto luego de ser seleccionados. Esto permite que un determinado ejemplo sea incluido más de una vez dentro de la muestra considerada. Este enfoque de muestreo se denomina *remuestreo con reposición*. Así, mediante el remuestreo con reposición, cada *remuestra* podrá incluir algunos de los datos originales más de una vez y será algo diferente de la original. En consecuencia, un cálculo estadístico ϕ computado a partir de cada una de las remuestras tomará valores diferentes entre sí y respecto del original $\phi_{original}$. En consecuencia, una distribución de frecuencias de cada uno estos valores ϕ calculados a partir de las remuestras es una estimación de la distribución muestral de $\phi_{orginal}$[8, 43]. Sin embargo, cuando se trabaja con tamaños de muestras pequeños, la distribución bootstrap puede ser muy variable; por lo cual su forma y dispersión reflejan las características de la muestra y pueden no estimar con precisión la forma y la dispersión de la distribución muestral [55]. El proceso de constitución de las muestras se resume en los siguientes pasos:

1. Selección del tamaño de la muestra.

2. Mientras el tamaño de la muestra sea menor al elegido, tomar un ejemplo aleatorio de la base de datos.

3. Añadir el ejemplo a la muestra.

Para cada sujeto se tomó un 'sujeto de referencia', que cuenta con los datos de la base de datos completa (85 épocas de caracteres), y se generaron 10 sujetos extra aplicando la técnica. Para cada uno se prosiguió de la siguiente manera: primero, se generó un arreglo de números entre $[0, 85)$, sobre el cual se realizó un remuestreo aleatorio con remplazo y se computaron los índices únicos del arreglo resultante. Estos fueron utilizados para seleccionar

las correspondientes épocas de caracteres y generar el nuevo sujeto, añadiendo en el objeto `Data` un nuevo atributo (`indexes`) que permite trazar y reproducir cada nuevo sujeto. Sobre los conjuntos de prueba no se realizaron modificaciones porque serán empleados en su totalidad para evaluar a los sujetos correspondientes. Esto es, el conjunto de prueba del sujeto A será utilizado en la evaluación de los modelos entrenados con el sujeto A de referencia y sus derivados obtenidos mediante boostrapping. Lo mismo aplica, de manera análoga, para el sujeto B y sus derivados.

Luego, la base de datos ampliada quedó constituida de la siguiente manera:

Tabla 2.2: Constitución de la base de datos ampliada [3]

	Entrenamiento			Prueba		
Sujetos	P300	No-P300	Épocas	P300	No-P300	Épocas
A0	2550	12750	85			
A1	1530	7650	51			
A2	1800	9000	60			
A3	1650	8250	55			
A4	1470	7350	49			
A5	1590	7950	53	3000	15000	100
A6	1590	7950	53			
A7	1650	8250	55			
A8	1680	8400	56			
A9	1440	7200	48			
A10	1500	7500	50			
B0	2550	12750	85			
B1	1650	8250	55			
B2	1710	8550	57			
B3	1620	8100	54			
B4	1650	8250	55			
B5	1680	8400	56	3000	15000	100
B6	1560	7800	52			
B7	1470	7350	49			
B8	1530	7650	51			
B9	1560	7800	52			
B10	1530	7650	51			

2.5. Acondicionamiento de las señales EEG

En el acondicionamiento de señales EEG para la detección de potenciales P300, las etapas habitualmente implementadas consisten en: 1) filtrado, para atenuar el ruido y conservar las bandas de frecuencia relevantes; 2) submuestreo o decimación, para reducir la cantidad de datos; 3) segmentación en épocas, para definir una ventana en que se encuentre comprendido el P300.

A fin de evaluar la serie de técnicas más convenientes a implementar para realizar una buena preparación de los datos, se compararon dos métodos diferentes de preprocesamiento propuestos en la literatura, tomando en consideración la calidad de la señal obtenida y el tiempo de cómputo requerido.

2.5.1. Método 1: Filtro de media móvil y decimación

Para el método de filtro de media móvil y decimación implementado, utilizando como referencia el diseño realizado en [30], las técnicas de preprocesamiento aplicadas fueron, en primer lugar, la segmentación de los datos en épocas de 800 ms posteriores a cada evento (intensificación de fila/columna), seguido de la aplicación de un filtro de media móvil y la posterior decimación de la señal resultante, con un factor de 10, obteniendo una frecuencia de muestreo final de 24 Hz. La figura 2.5 muestra las señales obtenidas luego de cada una de estas etapas.

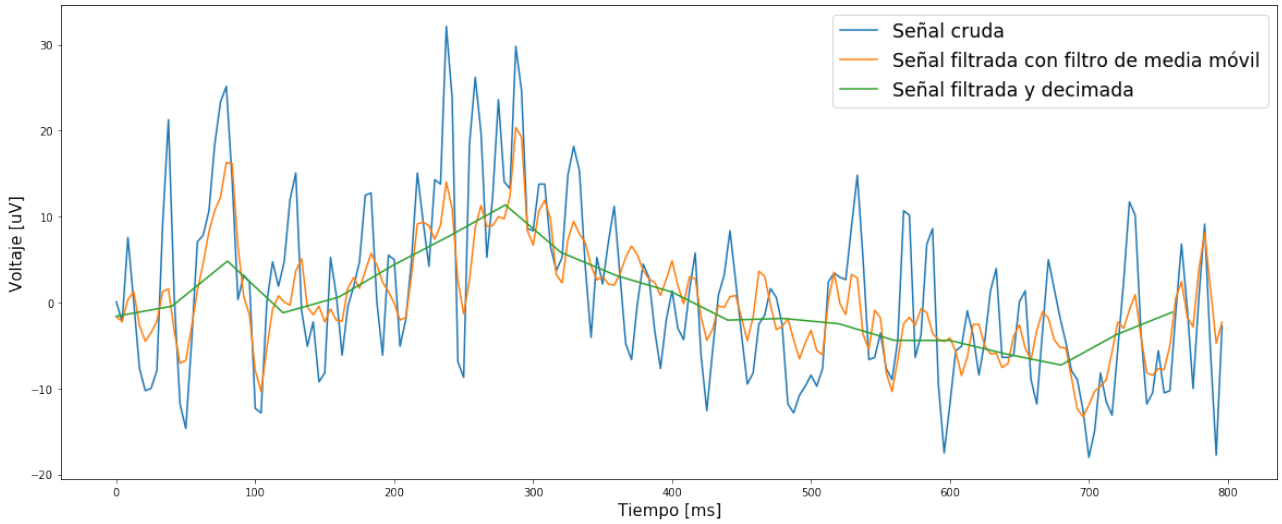


Figura 2.5: 800 ms de señal posteriores al estímulo. Se pueden apreciar tres señales, correspondientes al primer método de preprocesamiento: la señal cruda (en azul), la señal filtrada con un filtro de media móvil (en naranja) y, finalmente, la señal filtrada y decimada (en verde).

2.5.2. Método 2: Filtro Butterworth y submuestreo

Utilizando como referencia el método implementado en [66], donde la señal es primero filtrada con un filtro IIR Butterworth pasa-bajo de quinto orden, con frecuencia de corte de 10 Hz; luego submuestreada a 20 Hz y finalmente segmentada en épocas de 700 ms. En la figura 2.6 se muestran las señales obtenidas luego de cada etapa, pero sobre una ventana de 800 ms a fin de comparar el mismo segmento de señal en ambos métodos.

[3]Todos los sujetos fueron generados a partir de la técnica de bootstrapping, a excepción de los marcados en gris, que corresponden a los objetos tomados como referencia. El conjunto de prueba es igual para todos los sujetos. Se detalla el número de épocas de caracteres y de instancias P300/No-P300 para cada uno.

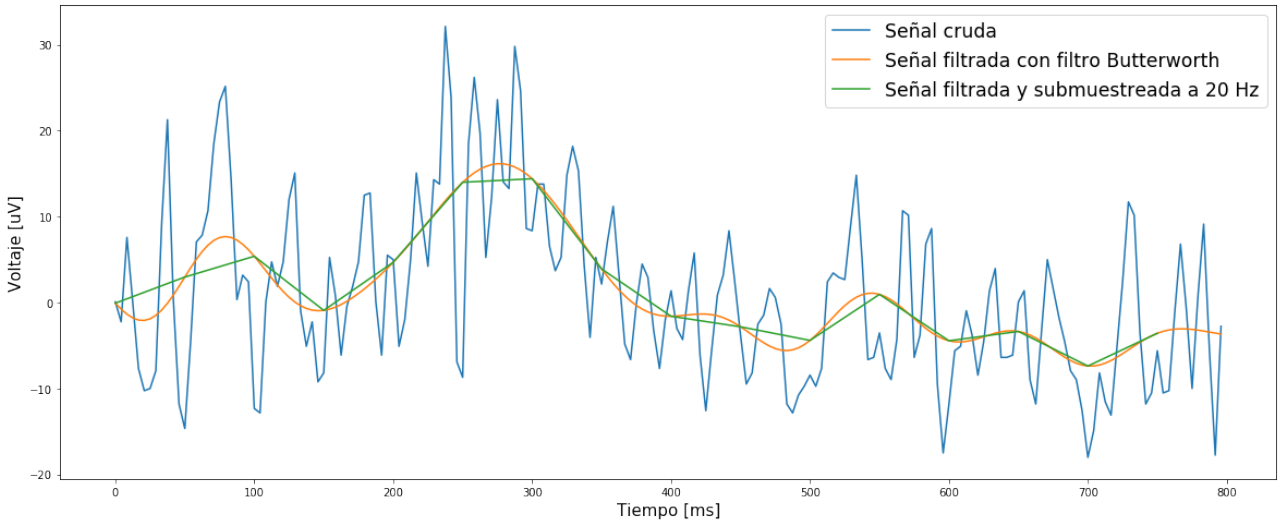


Figura 2.6: 800 ms de señal posteriores al estímulo. Pueden apreciarse tres señales, correspondientes a la aplicación del segundo método de preprocesamiento: la señal cruda (en naranja), la señal filtrada con filtro Butterworth (en azul) y, finalmente, la señal filtrada y submuestreada a 20 Hz (en verde).

2.5.3. Comparación de los métodos 1 y 2

En la comparación de ambos métodos, se tomó en consideración el tiempo de cómputo demandado y las señales resultantes obtenidas. La elección del método a implementar fue realizada de manera empírica, tomando en cuenta el tiempo de cómputo y el análisis visual de las señales obtenidas, para obtener la mejor representación del potencial buscado. La figura 2.7 muestra, para una época de 800 ms, la comparación entre la señal cruda original y sus versiones correspondientes obtenidas al aplicar los métodos 1 y 2. No se puede apreciar una diferencia notable entre las señales resultantes. Sólo se puede ver que, mientras el segundo método toma los valores medios en cada segmento de la curva; el primero toma valores más cercanos a los picos inferiores. Esto parece indicar que el método 2 conserva información más relevante de la señal, en comparación con el método 1, dado que toma muestras que representan valores intermedios entre los comprendidos en cada pico y valle; mientras que el método 1 es más sensible a los picos inferiores. Sin embargo, esto no constituye un criterio decisivo para la opción por uno u otro método, dado que ambos permiten identificar el potencial P300 en las señales, por lo cual se procede al análisis de los tiempos de cómputo demandados por cada uno.

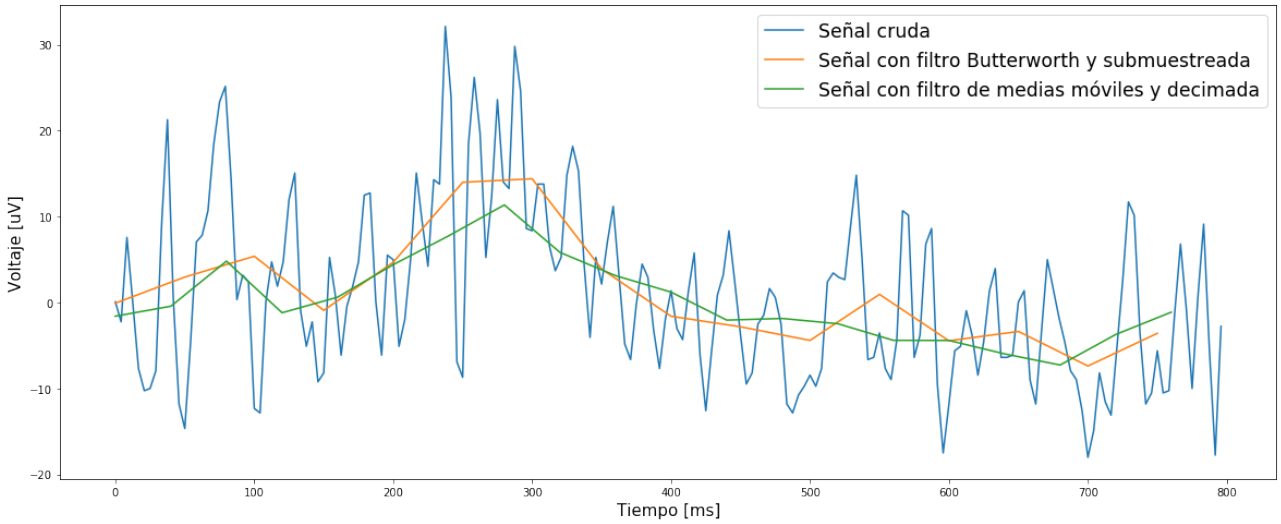


Figura 2.7: 800 ms de señal posteriores al estímulo. Se pueden apreciar tres señales: señal cruda (azul), señales obtenidas luego de aplicar los métodos de preprocesamiento 1 (verde) y 2 (naranja).

Luego, en lo concerniente al tiempo de cómputo, el segundo método demostró un desempeño notablemente superior, siendo cuatro veces más rápido que el primero; lo que constituye una ventaja si se tiene presente que el interés de toda BCI es la obtención y procesamiento en tiempo real de las señales. Por lo tanto, si bien en cuanto a las señales resultantes no existe una clara diferencia que oriente a la selección de uno u otro método; en lo referente al tiempo de cómputo demandado, el método 2 demostró ser claramente superior, por lo que se decidió adoptar la serie de técnicas en él implementadas (filtrado, submuestreo y segmentación) para llevar a cabo el acondicionamiento de las señales. Esta prueba, sin embargo, en posteriores investigaciones debería extenderse al estudio de un mayor número de casos. En las secciones subsiguientes se analizan, para cada una de ellas, los correspondientes parámetros de ajuste.

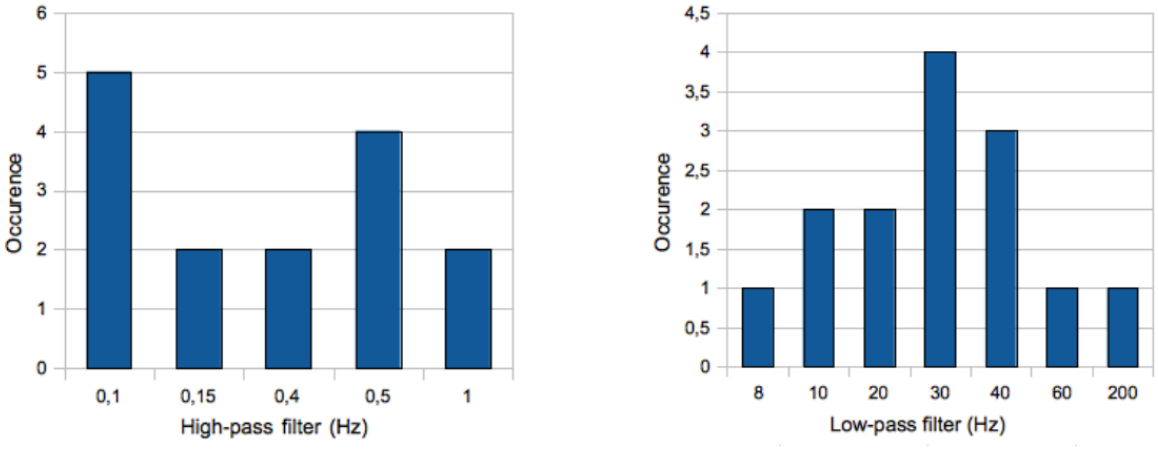
Tabla 2.3: Tiempo de cómputo de los métodos de preprocesamiento[4]

		Tiempo (s)			
Método	Operación	Una época de caracteres	Conjunto de entrenamiento	Conjunto de prueba	Conjuntos de entrenamiento y prueba
1	Segmentación	0,007	0,564	0,664	1,228
	Filtro de media móvil	0,107	9,055	10,653	19,709
	Decimación	0,121	10,258	12,069	22,327
	Total	**0,234**	**19,878**	**23,386**	**43,264**
2	Filtro Butterworth	0,044	3,758	4,421	8,179
	Submuestreo	0,005	0,443	0,521	0,964
	Segementación	0,007	0,564	0,664	1,228
	Total	**0,056**	**4,765**	**5,606**	**10,372**

[4]Se comapara el tiempo de cómputo (en segundos) de ambos métodos, medido para una época de caracteres y estimado para los conjuntos de entrenamiento y prueba completos, de 85 y 100 caracteres respectivamente.

2.5.4. Filtrado: definición de las frecuencias de corte

De los dos tipos de filtro evaluados, se decidió adoptar el implementado en [66], filtro Butterworth de 5^{to} orden, por los motivos ya expuestos, y se decidió estudiar los diferentes rangos de frecuencias de corte a definir. Dado que no existe actualmente un acuerdo sobre la mejor frecuencia de corte a emplear, al consultar la literatura se encontraron diversas combinaciones de valores, que se resumen en los siguientes histogramas:



¡

Figura 2.8: Histogramas mostrando la distribución de frecuencias de corte empleadas para filtros pasa-alto (izquierda) y pasa-bajo (derecha) en la detección de potenciales P300, tomados a partir de 27 artículos presentados en la conferencia de BCI llevada a cabo en Graz, en septiembre de 2011.[5]

Estos gráficos fueron obtenidos de un estudio comparativo del efecto de diferentes combinaciones de frecuencias de corte para filtros pasa-bajo y pasa-alto en la exactitud de la predicción de letras en un sólo trial, mediante un SVM [7]. Los resultados son presentados en la siguiente tabla, que fue tomada como referencia para definir las frecuencias de corte a emplear en el diseño de los filtros.

Tabla 2.4: Evaluación de la exactitud (en %) en la predicción obtenida para diferentes frecuencias de corte[6]

		Frecuencias de corte superiores						
		8 Hz	15 Hz	20 Hz	30 Hz	40 Hz	50 Hz	60 Hz
Frecuencias de corte inferiores	sin	51.7	52.56	53.62	53.62	52.12	50.41	48.25
	0.1 Hz	51.72	52.36	53.71	53.29	51.96	50.52	48.37
	0.5 Hz	52.14	52.87	53.19	52.94	51.36	50.13	47.72
	1 Hz	52.28	53.05	53.05	52.29	50.67	49.12	46.42

Como se puede apreciar, para rangos de frecuencias de corte superiores comprendidos entre 8 y 40 Hz, se obtienen resultados similares cualquiera sea la frecuencia de corte inferior, todos

[5]Imagen extraída de [7].
[6]Tabla extraída de [7].

por encima del 50 % de exactitud. Es preciso aclarar que los filtros pasa-alto son más propensos a generar distorsiones en la señal que los pasa-bajo. Estos últimos, si bien pueden generar un pequeño adelanto en la aparición de un evento en la señal, afectan típicamente en igual magnitud a todas las formas de onda dentro de un determinado experimento. Contrariamente, los filtros pasa-alto pueden introducir picos artificiales en la señal o algún otro tipo de distorsión. Sin embargo, esto normalmente ocurre para frecuencias superiores a 0.1 Hz. Es decir que el uso de una frecuencia de corte superior equivalente o menor a este valor no debería producir ninguna distorsión significativa. La figura 2.9 muestra cómo el aumento de la frecuencia de corte superior puede atenuar la onda P300 y crear un pico artificial al inicio de ésta[7]. Mientras que un filtro pasa-alto de frecuencia de corte igual a 0.1 Hz atenúa levemente la onda P300 y no produce artefactos discernibles; un filtro con $f_{corte} = 0{,}5\ Hz$ atenúa de manera evidente la señal e introduce una deflexión negativa de aproximadamente -50 a +100 ms, que constituye un artificio. Un filtro de $f_{corte} = 0{,}2\ Hz$ atenúa aún más la onda e incrementa la amplitud del artificio introducido en su inicio[34].

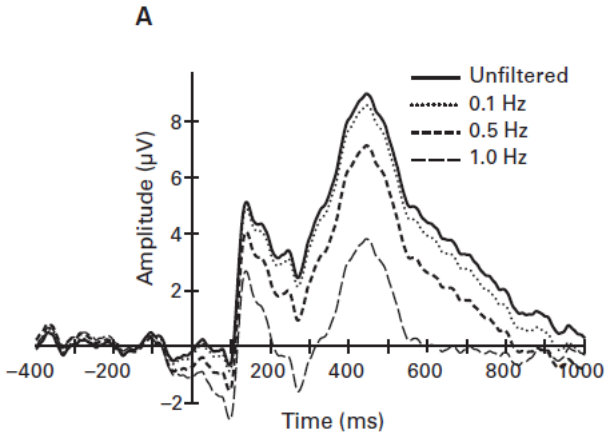


Figura 2.9: Efectos de diferentes filtros pasa alto en la onda P300 inducida mediante un estímulo visual de tipo oddball.[8]

2.5.5. Submuestreo

Con el propósito de disminuir la cantidad de datos y así reducir la dimensionalidad de los vectores de características a crear a partir de ellos, se llevó a cabo la operación de submuestreo. Ésta consiste en tomar una de cada x cantidad de muestras, lo que resulta en una reducción de la frecuencia de muestreo al valor x. Este valor debe ser definido tomando la precaución de respetar el criterio de Nyquist, a fin de evitar el solapamiento de las señales y la consecuente pérdida de información. Es decir que la nueva frecuencia de muestreo (x) debe ser superior al doble de la frecuencia máxima de interés de la señal.

2.5.6. Segmentación en épocas

Debido a que el P300 se caracteriza por depender temporalmente del estímulo que lo produce, definiéndose por el tiempo de latencia en que se evidencia, se decidió adoptar una ventana

[7]Estudio realizado en [35].
[8]Imagen extraída de [34].

temporal lo suficientemente amplia para comprenderlo dentro de ella. Esta ventana, definida en torno al momento de aparición del estímulo, ya sea target o no target, adopta diversos valores en la literatura: 600 ms [14], 650 ms [9], 667 ms [27], 700 ms [66] y 800 ms [30]. Se decidió adoptar este último valor que, por ser el peor caso, es más flexible a la variabilidad de la latencia de aparición del potencial para diferentes sujetos.

2.5.7. Extracción de características

Esta etapa consiste en la creación del vector de características para cada ejemplo presente en la base de datos, que se define en base a las propiedades de la señal que se desee incluir como atributos del vector de características. Es por ello que, habiendo realizado un análisis en cada una de las etapas previas de tratamiento de la señal, se explicitan aquí los métodos implementados para los diferentes algoritmos analizados.

2.5.7.1. Método de preprocesamiento de las señales de entrada a los algoritmos LDA, SVM y MLP

Las señales alimentadas a los algoritmos LDA, SVM gaussiano y lineal, y al perceptrón multicapa fueron filtradas con un filtro Butterworth pasa-bajo de 5^{to} orden, con frecuencia de corte en 10 Hz; submuestreadas a 20 Hz y segmentadas en épocas de 800 ms, tomados a partir del momento de aparición del estímulo. La frecuencia de submuestreo fue definida respetando el criterio de Nyquist: $f_s = 2f_{max}$.

Se optó por aplicar un filtro pasa-bajo en 10 Hz en lugar de uno pasa-banda entre, por ejemplo, 0.4 - 30 Hz, debido a lo expuesto en [66]. En el trabajo, se utilizan ambos filtros para preprocesar las señales que posteriormente se alimentan a un LDA, y se reportan mejores resultados en la predicción de caracteres con el pasa-bajo (96 % de accuracy) que con el pasa-banda (91 % de accuracy).

Además, de acuerdo a las consideraciones expresadas en la sección 2.5.4, no existen diferencias significativas a nivel de predicción de caracteres entre un filtro con frecuencia de corte superior de 10 o 30 Hz; y no siempre resulta conveniente aplicar un filtro pasa-alto, debido a las distorsiones que pueden introducir en la señal. Por otra parte, un filtro pasa-bajo con frecuencia de corte en 30 Hz implica un valor mínimo de frecuencia de submuestreo de 60 Hz, resultando en 60 muestras por segundo frente a las 20 que se pueden obtener al emplear $f_c = 10\ Hz$. En consecuencia, para establecer un compromiso entre la fijación de una frecuencia de corte que mantenga las componentes de principal interés en la señal, y la dimensionalidad del vector de características a crear a continuación, se decidió adoptar el filtro explicitado.

2.5.7.1.1 Creación del vector de características

Dado el carácter temporal del P300, la extracción de características se orientó a la captación de la variación de los potenciales en el tiempo. Por lo tanto, las épocas de datos resultantes fueron concatenadas por canal para cada intensificación, constituyendo un único vector de características X por estímulo. Luego, la longitud de cada uno fue de:

$$l = fs\ \Delta t\ C = 20\ Hz\ 800\ ms\ 64 = 1024 \tag{2.1}$$

donde f_s indica la frecuencia de muestreo, Δt el tamaño de la época y C el número total de canales. Para el caso de los vectores de características de entrenamiento, se generó, además, un vector y de etiquetas, de valores $\{1, 0\}$ para los eventos P300 y no P300, respectivamente.

2.5.7.2. Método de preprocesamiento de las señales de entrada a las redes convolucionales

Para el caso de las redes convolucionales analizadas en la presente investigación, se decidió de forma empírica respetar el diseño empleado en las implementaciones de referencia, ya que persiguen una estructura de vectores de características diferente a la empleada para el resto de los modelos.

Para las tres redes convolucionales implementadas, las señales fueron filtradas entre 0.1 y 20 Hz [9, 60]. En los tres casos se decidió adoptar un factor de submuestreo de 120 Hz, siguiendo con los lineamientos de los trabajos de referencia, para poder realizar una comparación acorde entre ellas y beneficiarse, igualmente, de la reducción de dimensionalidad de los datos. Se respetó el orden de cómputo indicado para ambas operaciones, por lo cual primero se realizó el submuestreo y posteriormente el filtrado.

2.5.7.2.1 Creación del tensor de características

El tensor de características creado adopta la forma (N, C), con $N = 120Hz \cdot 800\ ms = 96$, correspondiente al número de muestras contenidas en cada época y $C = 64$, correspondiente al número de canales. De esta manera, el vector de características deja de ser un vector para convertirse en un *tensor*, que pretende captar no sólo aspectos temporales en la producción del P300, sino también espaciales.

2.5.8. Preparación de los datos

Algunos algoritmos, como las redes neuronales y SVMs, son muy sensibles al escalamiento de los datos. La variación de éstos en diferentes escalas impacta negativamente en el desempeño del clasificador. Por lo tanto, antes de entregar los vectores de características como entrada a estos modelos, es necesario realizar una etapa de preparación de los datos en ellos contenidos; etapa que variará de acuerdo al algoritmo en cuestión.

A continuación se detallan dos técnicas, escalamiento y normalización, empleadas como etapa previa al entrenamiento de los SVM y redes neuronales, respectivamente. Se debe aplicar la misma transformación de los datos tanto al conjunto de entrenamiento, como al de prueba [45].

2.5.8.1. Escalamiento

Esta técnica de preprocesamiento fue aplicada a los SVM, que son sensibles al escalamiento de los datos y precisan que sus características varíen en una escala similar. En este trabajo se utilizó el escalamiento por valor mínimo y máximo, que asegura que los datos se encuentren comprendidos entre 0 y 1.

2.5.8.2. Normalización

Para el caso de las redes neuronales, se realizó una normalización estándar, que asegura que cada característica del vector tenga media cero y varianza 1. Ésta se computó de la siguiente manera:

$$I_{i,j} \leftarrow \frac{I_{i,j} - \overline{I}_i}{\sigma_i} \tag{2.2}$$

donde I_i y σ_i son, respectivamente, la media y la desviación estándar del electrodo i en el tiempo j. [9]

2.6. Métodos de evaluación del desempeño de los modelos

Los modelos construidos a partir de los distintos algoritmos de clasificación fueron sometidos a dos instancias diferentes de evaluación, correspondientes, en primer lugar, a la detección de los eventos P300; y, en segundo lugar, a la predicción de caracteres. En la primer instancia, se evalúa, para cada algoritmo, la capacidad de discriminación entre eventos P300 y No-P300 su poder de generalización. En la segunda, en cambio, se analiza la tasa de predicción de caracteres, y cómo se ve influenciada por el tiempo de selección de cada uno durante el entrenamiento y la simulación online.

2.6.1. Reducción del número de trials durante el entrenamiento

Debido a que la gran extensión en el tiempo de las sesiones de calibración puede resultar fatigante para el usuario, se decidió evaluar el desempeño de los algoritmos para diferentes tamaños de entrenamiento, a fin de dar con el mínimo necesario que permita obtener resultados aceptables. Para ello, se consideraron dos posibles enfoques:

1. Reducir el número de caracteres durante la calibración
2. Reducir el número de trials durante cada época de caracteres

La primera posibilidad fue descartada, al considerar que el número total de caracteres empleados en la construcción del conjunto de datos de entrenamiento puede ser dividido en tantas sesiones de calibración como se considere necesario, reduciendo la fatiga del usuario.

En cambio, se consideró que una reducción en el número total de trials empleados durante el entrenamiento, al repercutir sobre la duración de cada época de caracteres, no sólo disminuiría el cansancio del usuario, sino que también haría la tarea más amena. Esto se debe a que, en vez de prestar atención al mismo carácter por períodos prolongados de tiempo, tendría intervalos de descanso (matriz en blanco entre épocas de caracteres) con mayor frecuencia. Esto sumado al hecho de que traería aparejado, además, la reducción global del tiempo total de entrenamiento (ya sea dividido o no en diferentes sesiones); por lo cual no se consideró en esta investigación la cantidad de caracteres a asignar durante la calibración.

2.6.1.1. Definición de los tamaños de entrenamiento

Para cada sujeto (de referencia y bootstraps) se definieron tamaños de entrenamiento de a intervalos de 3 trials: [3, 6, 9, 12, 15]. Para ello, si la frecuencia de muestreo es $f_s = 240\ Hz$, y cada trial comprende 12 intensificaciones de período $T = 175\ ms$; el número de muestras de cada uno es:

$$175\ ms \cdot 12 = 2100\ ms \Longrightarrow 2100\ ms \cdot 240\ Hz = 504\ muestras \tag{2.3}$$

Luego, los tamaños de entrenamiento resultantes quedan detallados en la siguiente tabla.

Tabla 2.5: Número de muestras para cada tamaño de entrenamiento.[9]

Número de trials	3	6	9	12	15
Número de muestras	1512	3024	4536	6048	7794

2.6.2. Métricas utilizadas

Se definieron métricas diferentes para la evaluación del desempeño de los algoritmos en la clasificación binaria y en la predicción de caracteres, debido a la naturaleza distinta de cada problema. Para ambos casos, no obstante, las métricas fueron computadas mediante el método `sklearn.metrics`.

2.6.2.1. Clasificación binaria

En esta instancia, se buscó maximizar la capacidad de discernimiento entre ambas clases, haciendo énfasis en la detección de los eventos P300, que constituyen el interés de la BCI en cuestión. Sin embargo, al tratarse de una base de datos desbalanceada, en que la proporción de instancias de la clase P300 frente a la no-P300 es de 6:1, fue necesario establecer una métrica de evaluación del desempeño de los modelos alternativo a la exactitud (accuracy) de la predicción. Esto se debe a que, un modelo que clasifique todos los ejemplos como pertenecientes a la clase de mayor proporción dentro de los datos de entrenamiento, arrojaría un valor alto de exactitud, que sólo estaría reflejando la distribución de las clases.

Es por esto que se decidió computar un grupo amplio de métricas: precision, recall, F1, TP, TN, F y FN. De éstas, se prestó especial atención a las tres primeras y, en particular, al puntaje F1, que reúne los valores de precision y recall en uno solo. Este valor puede interpretarse como la capacidad del modelo de detectar con precisión la mayor cantidad posible de potenciales P300. Por lo tanto, el puntaje F1 fue empleado en la optimización de los modelos y en la evaluación de su poder de clasificación.

Para estas métricas, y en particular para la clasificación de bases de datos desbalanceadas, como es el caso de interés en este proyecto, no existe un valor umbral claro que establezca un límite entre lo que se considera un 'buen' o 'mal' clasificador. Por lo tanto, se recurre a la literatura para definir valores de referencia alcanzados por modelos que demuestren un buen desempeño en la etapa de clasificación posterior. En general, se obtienen modelos con puntajes F1 de 0.4 y 0.5 puntos, dados por valores de recall de 0.6 y 0.7, y de precision iguales a 0.3 y 0.4 [9]. El establecimiento de estos valores de referencia permitió definir un umbral a partir del cual un modelo se considera o no aceptable.

2.6.2.2. Predicción de caracteres

Para el caso de la predicción de caracteres, el problema es más sencillo debido a que, al no ser un problema de clasificación, no depende de la distribución de las clases dentro de la base de datos. Por lo tanto, puede ser evaluada sencillamente mediante la exactitud (o accuracy), que indica el porcentaje de casos que fueron correctamente predichos.

Asimismo, a fin de evaluar no sólo la exactitud lograda en la predicción, sino también la relación entre ésta y el tiempo de selección del carácter, se computó también la Tasa de Transferencia de Información (ITR). Ésta, de manera opuesta al porcentaje de aciertos, que aumenta en relación con el número de trials, toma en consideración el tiempo necesario para el reconocimiento del carácter. Está dada por la fórmula

$$ITR = \frac{60\left(P\log_2(P) + (1-P)\log_2\left(\frac{1-P}{N-1}\right) + \log_2 N\right)}{T} \tag{2.4}$$

[9] Para el caso de 15 trials, el número de muestras es 7794 en lugar de $504 \cdot 15 = 7560$, dado que se tomó el total de datos por época de carácter presentes en la base de datos, que incluyen el tiempo inter-caracteres en que la matriz está en blanco.

en donde P es la probabilidad de reconocimiento del carácter (accuracy), N es el número de opciones (36 caracteres), y T es el intervalo de tiempo para cada selección; es decir, el tiempo necesario para reconocer un carácter. La unidad de medida es de bits por minuto *bpm*.

Para el protocolo de la experimentación empleado en la recolección de los datos usados en la presente investigación, en que cada trial demanda 2.1 segundos (12 intensificaciones de 175 ms) y entre caracteres se muestra la matriz en blanco durante 2.5 segundos, T está dado por:

$$T = 2{,}5 + 2{,}1 \cdot n,\ 1 \leq n \leq 15 \tag{2.5}$$

donde n corresponde al número de trials usados durante la predicción online [9, 73].

2.6.3. División de la base de datos en conjuntos de entrenamiento, validación y prueba

Para cada sujeto, como se detalló en la sección 2.4, se tienen dos conjuntos principales de datos: un conjunto de entrenamiento, con los ejemplos etiquetados; y un conjunto de prueba, sin etiquetas.

Durante la etapa de clasificación binaria, se trabajó con el conjunto etiquetado para el entrenamiento, optimización y evaluación de los algoritmos. Se lo dividió en un nuevo conjunto de entrenamiento, correspondiente al 90 % del total, y en un conjunto de validación, correspondiente al 10 % restante, empleado en la evaluación del desempeño. Para el caso de aquellos algoritmos en cuyo entrenamiento se llevó a cabo un proceso de optimización (redes neuronales), de este nuevo conjunto de entrenamiento se designó, a su vez, un 20 % para ser empleado como conjunto de validación para optimización.

Durante la etapa de predicción de caracteres, en cambio, se hizo uso de un único conjunto, conjunto de prueba, correspondiente al set de datos no etiquetados.

2.7. Algoritmos implementados en la clasificación binaria de los potenciales P300

En esta primera instancia, se llevó a cabo el entrenamiento y optimización de los modelos, buscando maximizar la detección de potenciales P300, independientemente del carácter al que correspondieran.

2.7.1. Análisis Discriminante Lineal

La implementación de este clasificador fue realizada a partir del método `LinearDiscriminantAnalysis` de la librería scikit-learn [49]. Debido a que el método no presenta parámetros específicos a ajustar, simplemente se definió el algoritmo de resolución y la estrategia de *shrinkage* (contracción) a emplear. Los posibles algoritmos de resolución a definir son:

- `eigen`: descomposición en vectores y valores propios. Está basado en la optimización de la relación entre las dispersiones entre e intra clase.
- `svd`: descomposición en valores singulares. No se basa en el cálculo de la matriz de covarianza.
- `lsqr`: regresión por mínimos cuadrados.

Dado que, luego de evaluar cada uno de ellos, se obtuvieron similares resultados, se optó por utilizar el algoritmo `eigen`, que se corresponde con la definición original del método. El shrinkage, por su parte, es una herramienta empleada para mejorar la estimación de las matrices de covarianza en los casos en que el número de ejemplos de entrenamiento es pequeño frente al número de características, a partir de la estimación de una versión contraída de la matriz. Dado que en la base de datos empleada esta relación es de aproximadamente 15:1 (15300 ejemplos, 1024 características), y que al evaluar el clasificador con y sin shrinkage no se obtuvieron resultados distintivos, se decidió no utilizar shrinkage para la implementación del clasificador.

2.7.2. Máquinas de Vectores de Soporte

Se implementaron dos clasificadores SVM, Lineal y Gaussiano, a partir del módulo `sklearn.svm.SVC`. En ambos casos, se llevó a cabo un proceso de optimización de los parámetros e hiperparámetros[10] correspondientes, mediante el método de búsqueda de grilla `sklearn.model_selection.GridSearchCV`. Se ajustó el factor de validación cruzada en `cv` = 3, debido al excesivo tiempo de cómputo demandado por el proceso, y se definió como métrica de valoración el puntaje F1 (`scoring` = `'f1'`), por ser el adoptado como indicador del desempeño de los clasificadores en este trabajo.

La búsqueda se llevó a cabo para cada uno de los sujetos, empleando el conjunto de entrenamiento reducido de 3 trials, por ser el que contiene los ejemplos comunes al resto de los conjuntos, siendo entonces el más representativo. Primero se exploró sobre un rango amplio de valores para cada parámetro, a fin de identificar las 'regiones' de la grilla en que se obtuvieran los mejores resultados y, consecuentemente, llevar a cabo una búsqueda más fina en dicha región.

Posteriormente, para cada uno de los sujetos, se entrenaron ambos algoritmos empleando, para todos los tamaños de entrenamiento, los parámetros definidos durante la optimización. Se habilitó la estimación de probabilidades, ajustando la variable booleana `probability` en *True*, de modo que el clasificador devuelva la probabilidad de pertenencia a cada clase como predicción. Asimismo, para compensar el desbalance existente entre las clases, se seteó la variable `class_weight` en '*balanced*', que asigna al parámetro C (constante de margen blando) un coeficiente inversamente proporcional al número de instancias de cada clase.

Finalmente, además de computar las diferentes métricas de clasificación sobre el conjunto de validación, se recuperó el número de vectores de soporte definidos por el modelo para cada clase.

2.7.2.1. El método de búsqueda de grilla mediante validación cruzada

En la validación cruzada de k-particiones, se divide el conjunto de entrenamiento en k subconjuntos del mismo tamaño. Luego, k-1 subconjuntos son empleados para entrenar el modelo con ciertos parámetros y el conjunto restante es utilizado para validar su desempeño. Esta operación se repite k veces, de modo que el modelo es secuencialmente evaluado para cada uno de los k subconjuntos. En consecuencia, el puntaje de validación cruzada representa el promedio del obtenido para cada uno de los subconjuntos evaluados [71, 28]. Esta técnica permite reducir el nivel de sobreajuste de los datos, debido a que el modelo se entrena sucesivamente con diferentes particiones del conjunto de entrenamiento, lo que introduce variabilidad y una mejor estimación de su desempeño en comparación con el entrenamiento con todo el conjunto en una

[10] Parámetros que deben ser determinados antes de que el proceso de aprendizaje comience, sin poder ser aprendidos por el algoritmo.

sola vez. Sin embargo, es una técnica que tiene un costo computacional elevado, debido a que el número de operaciones a realizar escala de manera directamente proporcional al número de particiones elegidas.

El método de búsqueda de grilla, por su parte, implementa la técnica de validación cruzada para evaluar de manera exhaustiva todas las posibles combinaciones de parámetros que se desea entrenar. Para cada combinación, se almacena el puntaje obtenido por validación cruzada y, una vez finalizada la búsqueda, se determina el mejor modelo como aquel cuyos parámetros hayan arrojado el mejor puntaje.

2.7.2.2. SVM Lineal

El único parámetro a optimizar para el caso del kernel lineal corresponde a la constante de margen blando C. El rango de valores explorados, luego de una búsqueda inicial de valores comprendidos entre $[10^{-5}, 10^{6}]$, se redujo a $[1, 10, 100]$, rango en que se obtuvieron los mejores resultados (mayores puntajes F1). La tabla 2.6 presenta los valores resultantes de la optimización, posteriormente empleados en el entrenamiento.

Tabla 2.6: Valores de C empleados en el SVM lineal.[11]

Sujeto	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
C	100	100	10	100	100	100	100	100	100	10	100
F1	0.21	0.27	0.33	0.31	0.30	0.32	0.32	0.28	0.31	0.23	0.44
Sujeto	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
C	10	100	100	10	10	100	100	10	100	100	10
F1	0.57	0.38	0.47	0.40	0.49	0.36	0.44	0.42	0.45	0.48	0.32

2.7.2.3. SVM Gaussiano

En el caso del SVM de kernel gaussiano, de ecuación $e^{-\gamma||x-x'||^2}$, además del hiperparámetro C, se debe ajustar el parámetro γ. Éste define la importancia asignada a cada ejemplo de entrenamiento: valores bajos otorgan una gran influencia, mientras que lo opuesto sucede para valores altos. El rango de valores inicialmente explorado ($C = [10^{-5}, 10^{4}], \gamma = [10^{-9}, 1]$), fue acotado a los valores de $C = [1000, 4000, 7000, 10000]$ y $\gamma = [10^{-}3, 0{,}05, 0{,}1]$, en que se obtuvieron los puntajes F1 más altos.

[11] Valores obtenidos, para cada uno de los sujetos, en la optimización por búsqueda de grilla realizada para el SVM lineal, y sus respectivos puntajes F1.

Tabla 2.7: Valores de C y γ empleados para el SVM gaussiano.[12]

Sujeto	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
C	10^3	10^4	10^4	10^4	10^4	10^4	10^3	10^4	10^3	10^4	10^4
γ	0.0505	1	1	1	1	1	0.0505	1	0.0505	1	1
F1	0.35	0.35	0.32	0.38	0.35	0.34	0.34	0.35	0.34	0.38	0.33
Sujeto	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
C	7000	10^4	10^4	10^4	7000	10^4	10^4	10^4	7000	10^4	10^4
γ	10^-3	10^-3	10^-3	10^-3	10^-3	10^-3	10^-3	10^-3	10^-3	10^-3	10^-3
F1	0.52	0.52	0.49	0.49	0.47	0.51	0.51	0.53	0.49	0.50	0.49

2.7.3. Redes neuronales

Los dos tipos de redes neuronales abordados en esta investigación (MLP y CNN) fueron implementados en Keras, con Tensorflow corriendo en segundo plano. En primer lugar, se diseñaron y evaluaron diferentes arquitecturas, haciendo uso solamente de los datos correspondientes a los sujetos de referencia A0 y B0. Una vez definidas las de mejor desempeño, se procedió al entrenamiento de cada uno de los sujetos, con sus diferentes tamaños de entrenamiento. En el caso de las CNN, además, se replicaron en esta última instancia dos modelos de red propuestos en la literatura.

2.7.3.1. Proceso de implementación

La implementación, en ambos casos, fue llevada a cabo en cuatro etapas consecutivas: 1) creación del modelo, 2) compilación, 3) entrenamiento y 4) evaluación del desempeño.

2.7.3.1.1 Creación del modelo

En esta etapa se establecieron las arquitecturas para cada tipo de red, para lo cual se fijaron los siguientes parámetros:

- Número de capas ocultas.
- Número de neuronas por capa.
- Función de activación de la capa.

Los detalles sobre los diseños implementados para cada tipo de red serán abordados en las secciones 2.7.3.2 y 2.7.3.3, correspondientes a los MLPs y a las CNN, respectivamente.

[12]Valores obtenidos, para cada uno de los sujetos. en la optimización por búsqueda de grilla realizada para el SVM gaussiano, y sus respectivos puntajes F1.

2.7.3.1.2 Compilación

En esta etapa, se llevó a cabo la configuración del proceso de aprendizaje del modelo, a partir del método `compile`, que admite tres argumentos principales:

- `optimizer`: corresponde al algoritmo de optimización empleado por la red. Se utilizó el algoritmo de descenso de gradiente estocástico (`sgd`), con una tasa de aprendizaje fija de `lr` = 0,01, valor definido por defecto.
- `loss`: corresponde a la función de costo a minimizar durante el aprendizaje. En este caso, por tratarse de un problema de clasificación binaria, se hizo uso de la función `binary crossentropy`.
- `metrics`: el listado de métricas a computar. Se emplearon las definidas en la sección 2.6.2 de la página 67.

2.7.3.1.2 Elección del optimizador

Si bien existen algoritmos alternativos al SGD, como los métodos adaptativos (Adam, Adagrad o RMSprop), que presentan resultados superiores en el entrenamiento, existe evidencia de que estos métodos generalizan pobremente en comparación con el SGD. Tienden a mostrar un buen desempeño durante las etapas inciales del entrenamiento, pero luego son superados por el SGD en estadios tardíos [25]. Se ha encontrado que, incluso para un modelo simple de clasificación binaria, los algoritmos adaptativos convergen a una solución que clasifica incorrectamente nuevos datos con probabilidad cercana a la mitad; mientras que el SGD, para los mismos datos, encuentra una solución sin errores [68]. Es por esto que se optó por emplear este algoritmo en el proceso de optimización.

2.7.3.1.3 Entrenamiento

Esta operación se llevó a cabo a partir del método `fit`. Se definieron las entradas: `x`, vector de características, e `y`, sus correspondientes etiquetas. Se fijó en 128 el número de muestras a entrenar por cada actualización del gradiente (`batch_size` = 128), y en 100 el número de épocas de entrenamiento (`epochs` = 100). Para evaluar la función de costo y las métricas definidas en la compilación del modelo durante el proceso de aprendizaje, se destinó el 20 % de los ejemplos de entrenamiento `X` para ser empleados como conjunto de validación (`validation_split` = 0,2).

Finalmente, para compensar el desbalance de clases existente en los datos, se asignaron diferentes pesos a cada una de las clases a partir del argumento `class_weight`. A la clase P300 se le asignó un peso de $0{,}4 n_{noP300}/n_{P300}$, mientras que, para la clase noP300, el peso asignado fue n_{P300}/n_{noP300}. Estos pesos fueron asignados a fin de aumentar la tasa de reconocimiento de los eventos P300 que, por presentarse estos en menor cantidad que los No-P300, era muy baja. Fueron determinados de manera empírica, evaluando las métricas recall, precision y F1.

2.7.3.1.4 Evaluación del desempeño

Para la evaluación de las arquitecturas, se computaron las correspondientes métricas de clasificación binaria y se compararon, principalmente, los puntajes F1 obtenidos y el grado de sobreajuste presentado. En la comparación y definición de la mejor arquitectura de MLP se tuvo en cuenta, además, la complejidad de cada una como factor influyente en la toma de decisión.

2.7.3.2. Perceptrón Multicapa

Para el MLP, se diseñaron y evaluaron diversos modelos a partir de la clase `Sequential`, que permite crear arquitecturas que constan de múltiples capas de neuronas apiladas entre sí, en las que la salida de una de ellas constituye la entrada de la subsecuente. Se utilizaron capas del tipo `Dense`, que constituyen capas totalmente conectadas. Para el diseño de las diferentes arquitecturas, se experimentó con los parámetros enunciados en la sección 2.7.3.1.1.

Se crearon arquitecturas poco profundas, de entre 2 y 4 capas ocultas, con números arbitrarios de neuronas por cada una, comprendidos entre 3 y 1024, a fin de evaluar la respuesta de arquitecturas de diferente tamaño. Las arquitecturas desarrolladas se presentan en la tabla 2.8. Se asignaron funciones de activación clásicas: relu, linear, sigmoid y tanh.

Tabla 2.8: Detalle de las arquitecturas de MLP creadas[13]

Arquitectura	Capa	Número de neuronas	Función de activación
MLP_01	1	100	relu
	2	30	sigmoid
	salida	2	softmax
MLP_02	1	200	tanh
	2	150	tanh
	salida	2	softmax
MLP_03	1	1024	tanh
	2	512	tanh
	3	256	linear
	salida	2	softmax
MLP_04	1	1024	tanh
	2	512	tanh
	3	256	linear
	4	512	tanh
	salida	2	softmax
MLP_05	1	300	tanh
	2	150	tanh
	salida	2	softmax

[13]Por simplicidad, se omite la inclusión de las capas de dropout, ubicadas entre las diferentes capas.

La arquitectura final a implementar fue definida de acuerdo a la evaluación de los diferentes poderes de generalización, los puntajes F1 obtenidos y la complejidad de las arquitecturas. Este proceso se detalla en la sección 3.1.

2.7.3.3. Redes Convolucionales

La arquitectura general empleada en las redes convolucionales actuales para sistemas BCI basados en la detección del potencial P300, usa el tensor de entrada mostrado en la figura 2.10, donde N denota el número de muestras temporales, y C el número de canales del registro EEG. Esta arquitectura presenta tres niveles. En el primero, lleva a cabo una convolución en el eje espacial para aprender características espaciales. En el segundo, realiza una convolución en el tiempo, a fin de aprender características temporales. Finalmente, en el tercer nivel, utiliza capas totalmente conectadas para establecer una correlación entre las características aprendidas y una clase en particular [60].

En consecuencia, los modelos de red convolucional implementados en el presente trabajo se valen también de dicho tensor y de la utilización de operaciones de convolución tanto en el tiempo como en el espacio. En una primera instancia se replicaron dos modelos de red propuestos en la literatura, que realizan también un análisis no sólo temporal, sino también espacial de los datos, y los ejecutan con enfoques diferentes. Luego, habiendo estudiado e implementado estos modelos, se decidió proponer, bajo la misma línea de análisis, un modelo propio con otro enfoque de ejecución diferente.

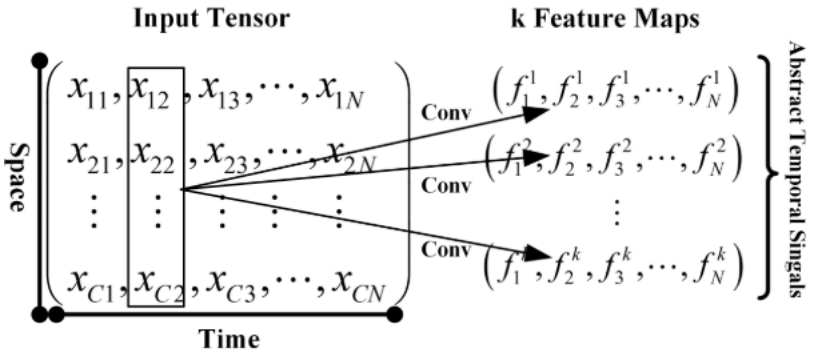


Figura 2.10: Tensor de entrada de la forma (N, C) habitualmente empleado en las arquiteturas de CNN para BCIs basadas en el P300 (izquierda); y convolución espacial. x denota una señal en el eje del tiempo y f una señal abstracta generada a partir de la convolución en el dominio espacial (derecha).[14]

2.7.3.3.1 Modelos propuestos en la literatura

Ambas redes fueron implementadas a partir de la clase `Sequential` descripta en 2.7.3.2. Si bien se intentó respetar al máximo los parámetros definidos en sus respectivas publicaciones, se introdujeron algunas modificaciones, a fin de hacerlos comparables entre sí y con el resto de los modelos evaluados. Esto se justifica en la medida en que el objetivo perseguido no radicaba en obtener exactamente los mismos resultados que los publicados, sino en poder comparar sus diferentes desempeños en circunstancias adoptadas como estándar. En consecuencia, si bien en ambas publicaciones se explicita con detalle el algoritmo de optimización empleado y sus parámetros, en la presente investigación se adoptó el optimizador definido en 2.7.3.1.2.

[14]Imagen extraída de [60].

2.7.3.3.1 CCNN

Esta red fue diseñada y propuesta por Cecotti y Graser en [9]. Consta de tres capas ocultas: dos convolucionales y una totalmente conectada. La primera capa está compuesta por 10 mapas de características de la forma (1, C), en el que cada uno constituye una combinación de canales. Esta capa lleva a cabo una convolución en una sola dimensión, la espacial. La segunda capa, a su turno, consta de 50 mapas de características de forma (13, 1), que ejecutan una convolución, también unidimensional, en la dimensión temporal. La tercer capa está constituida por 100 neuronas, densamente conectadas a los diferentes mapas de la capa 2. Finalmente, la capa de salida se compone de dos neuronas, densamente conectadas a la capa anterior, que representan las dos clases del problema.

Las dos primeras capas utilizan como función de activación una adaptación de la función sigmoidea, de la forma:

$$f(\sigma) = 1{,}7159\ tanh\left(\frac{2}{3}\ \sigma\right) \tag{2.6}$$

donde σ representa el potencial postsináptico de una neurona. La capa 3 utiliza la función sigmoidea clásica; y en la capa de salida se introdujo una modificación respecto de la utilizada en el modelo propuesto: se utilizó la función softmax en vez de la sigmoidea, a fin de trabajar con un clasificador softmax que devuelva un resultado en forma de probabilidades.

El orden de las capas convolucionales fue establecido con el fin de replicar las técnicas habitualmente empleadas en las BCIs. Mientras que la primera lleva a cabo un filtrado espacial, la segunda submuestrea las señales en el dominio temporal. La elección de los kernels como vectores y no como matrices, como es habitualmente el caso en las CNNs, pretende no mezclar en un mismo kernel características pertenecientes a estos dos dominios [9].

La arquitectura CCNN, al llevar a cabo una convolución espacial en la primer capa, genera diversos mapas de caraterísticas que son posteriormente alimentados a la capa de convolución temporal. Estos mapas presentan, entonces, señales temporales abstractas en lugar de señales crudas, lo que implica una pérdida de información temporal original e impide que la red aprenda correctamente las características temporales [60].

Tabla 2.9: Detalle de la arquitectura CCNN[15]

Capa	Operación	Función de Activación	Tamaño del kernel	Mapas de Características/ Neuronas
1	Convolución	Sigmoidea adaptada	(1, C)	10
2	Convolución	Sigmoidea adaptada	(13, 1)	50
3	TC	sigmoid	-	100
Salida	TC	softmax	-	2

[15]Esta arquitectura corresponde a la propuesta en [9], con la utilización de la función de activación softmax en lugar de sigmoid en la capa de salida.

2.7.3.3.1 OCLNN

La presente arquitectura consta de una sola capa convolucional que le da su nombre: One Convolutional Layer Neural Network (OCLNN) [60]. Se presenta como una alternativa superadora a la arquitectura CCNN, que presenta la limitación de extraer información temporal a partir de un mapa de características con información abstracta. En cambio, la OCLNN, en cuanto realiza en una misma capa una convolución tanto en tiempo como en espacio, puede extraer y aprender características de ambos dominios en su estado crudo.

La arquitectura de esta red está compuesta de la siguiente manera: su capa oculta contiene 16 mapas de características de la forma (N/15, C), que llevan a cabo una convolución en dos dimensiones, espacial y temporal. Aplica la función de activación `relu` y utiliza `dropout`, con un factor de 0.25, para reducir el sobreajuste. Esta capa genera 16 mapas de características con datos abstractos, que son alimentados a las dos neuronas softmax de la capa de salida.

Tabla 2.10: Detalle de la arquitectura OCLNN[16]

Capa	Operación	Función de Activación	Tamaño del kernel	Mapas de Características /Neuronas
1	Convolución	ReLU	(N/15, C)	16
	Dropout	-	-	-
Salida	TC	softmax	-	2

2.7.3.3.2 Modelo de diseño propio

Tomando como referencia los modelos anteriores, en que se extrae información temporal y espacial a partir de la operación de convolución, se ideó una arquitectura que también lleve a cabo estas operaciones, pero de manera paralela. En este sentido, se esperaba que esta arquitectura no sólo superara las limitaciones presentadas por la red CCNN en cuanto al procesamiento de características temporales abstractas, sino que además se beneficiara de la especialización de mapas neuronales específicos para los datos pertenecientes a cada tipo de dominio, a diferencia de la red OCLNN, que emplea las mismas neuronas para ambos.

2.7.3.3.2 TSMCNN

Esta arquitectura, a diferencia de las correspondientes a las demás redes neuronales (MLPs y CNNs), fue implementada haciendo uso de la clase `Model` de la librería Keras. Ésta permite llevar a cabo un diseño más flexible, conectando diferentes capas entre sí, sin que necesariamente se encuentren apiladas una encima de la otra. En consecuencia, el modelo creado toma como entrada los mismos tensores de características que las anteriores redes convolucionales, pero son enviados paralelamente a dos capas de mapas neuronales diferentes: 1a, que consta de 16 neuronas que realizan una convolución en el tiempo a partir del kernel (N/12, 1); y 1b, que realiza una convolución en el dominio espacial a partir de 16 mapas neuronales de kernel (1, 64). Posteriormente, la salida de ambas capas es llevada a una dimensión, mediante la

[16]Esta arquitectura corresponde a la propuesta en [60].

operación `Flatten` (aplanar) y es unida en una sola capa mediante la función `Concatenate`, que las alimenta a la capa de salida, constituida por las dos neuronas softmax. En consecuencia, esta arquitectura representa una especie de híbrido de las dos anteriores. Recibe su nombre, TSMCNN, de acuerdo a su arquitectura funcional *Time Space Merged Convolutional Neural Network*, que refleja el hecho de que es una red convolucional de convoluciones temporales (time) y espaciales (space) unidas (merged).

Tabla 2.11: Detalle de la arquitectura TSMCNN[17]

Capa	Operación	Función de activación	Tamaño del kernel	Mapas de características / Neuronas
1a	Convolución	relu	(N/12, 1)	16
	Flatten			
1b	Convolución	relu	(1, 64)	16
	Flatten			
Salida	Concatenate			
	TC	softmax	-	2

2.8. Simulación online

La motivación principal del presente estudio fue la determinación del algoritmo que mejor desempeño demostrara tener para, en futuros trabajos, ser implementado en una aplicación BCI para usuarios finales. Por lo tanto, se decidió simular un escenario de adquisición y procesamiento de datos, clasificación y predicción en tiempo real que permitiera evaluar su desempeño en un entorno más cercano a la realidad.

La implementación de la simulación online sigue los lineamientos establecidos en trabajos similares [66], y fue planteada en tres etapas: 1) adquisición de datos, 2) preprocesamiento y clasificación; y 3) predicción de caracteres.

2.8.1. Adquisición de datos

En un escenario online, los datos se adquieren en forma segmentada a medida que transcurre el tiempo, en forma de bloques de un número reducido de muestras. Para simular esta adquisición, se utilizó el pseudoamplificador `ReplayAmp` de la librería Mushu [65]. Éste permite cargar una base de datos completa, y devolver sólo la máxima cantidad de datos que es posible adquirir, dada la frecuencia de muestreo y el tiempo transcurrido desde la última adquisición. En consecuencia, al recibir sólo la cantidad de muestras correspondientes al tiempo transcurrido de experimento, se está ante un escenario similar a la ejecución en tiempo real, en la que los datos se reciben y deben ser procesados de a pequeños bloques.

Durante el submuestreo relizado en la siguiente etapa, es necesario que la cantidad de datos que se sometan a esta operación sea múltiplo del factor de submuestreo definido, a fin de evitar la

[17]La capa TC corresponde a una capa totalmente conectada.

pérdida de muestras. Dado que los factores de submuestreo empleados corresponden a $\frac{240\ Hz}{20\ Hz} = 12$, para los clasificadores lineales y MLPs, y a $\frac{240\ Hz}{120\ Hz} = 2$, para las redes convolucionales, se fijó el tamaño del *buffer* de datos en 12 muestras. Este número, correspondiente a 50 ms de registro, permitió adquirir datos al tiempo que se realizaba cada intensificación (175 ms de período entre encendido y apagado), lo que resulta ventajoso en la medida en que, al finalizar una intensificación, se tiene sólamente un retraso de 50 ms en la adquisición y procesamiento del último bloque de datos. Esto se traduce en un menor tiempo de cómputo en comparación con el que se precisaría si se debieran procesar en conjunto los 2100 ms totales de registro ($175\ ms \cdot 15$) que se obtienen al finalizar cada repetición.

2.8.2. Preprocesamiento y clasificación

Debido a que en el entorno simulado los datos se adquieren en forma de bloques de 12 muestras de longitud, durante el preprocesamiento se plantearon algunas diferencias respecto del escenario offline. En primer lugar, dado que el submuestreo fue realizado de a pequeñas cantidades de datos, fue preciso asignar un tamaño de buffer apropiado, de 12 muestras, de acuerdo a lo explicado en la sección anterior. Se introdujeron asimismo valores de *delay* (retraso) en los filtros, a fin de recibir los mismos resultados que se obtendrían de procesar la totalidad de los datos de una sola vez. Para ello, en el filtrado de cada nuevo bloque de datos, se asignaron los valores iniciales obtenidos en la etapa de filtrado de las últimas 12 muestras.

2.8.3. Predicción de caracteres

En el experimento del deletreador P300, como se explicó en la sección 2.1.1, cada repetición comprende una serie de 12 intensificaciones; luego de las cuales se está en condiciones de realizar una *estimación* del carácter en cuestión. Se utilizó la palabra 'estimación' en lugar de 'predicción', en la medida en que una sola época de caracteres resulta, en la mayoría de los casos, insuficiente para predecir con certeza el carácter. En este sentido, resulta de interés evaluar el número de repeticiones (trials) en que se logra predecir correctamente un porcentaje aceptable de caracteres. Para ello, para cada carácter, por cada 12 marcadores procesados, correspondientes a un trial, se realizó la predicción y se evaluó si correspondía o no al carácter en cuestión. Esto permitió, posteriormente, comparar el poder predictivo de cada clasificador para diferentes números de repeticiones durante el experimento online. Asimismo, a fin de no procesar más de una vez una misma época de caracteres, se almacenó el número de nuevas muestras que se obtenidas en cada instante, para poder determinar qué épocas habían sido evaluadas y sólo computar las nuevas.

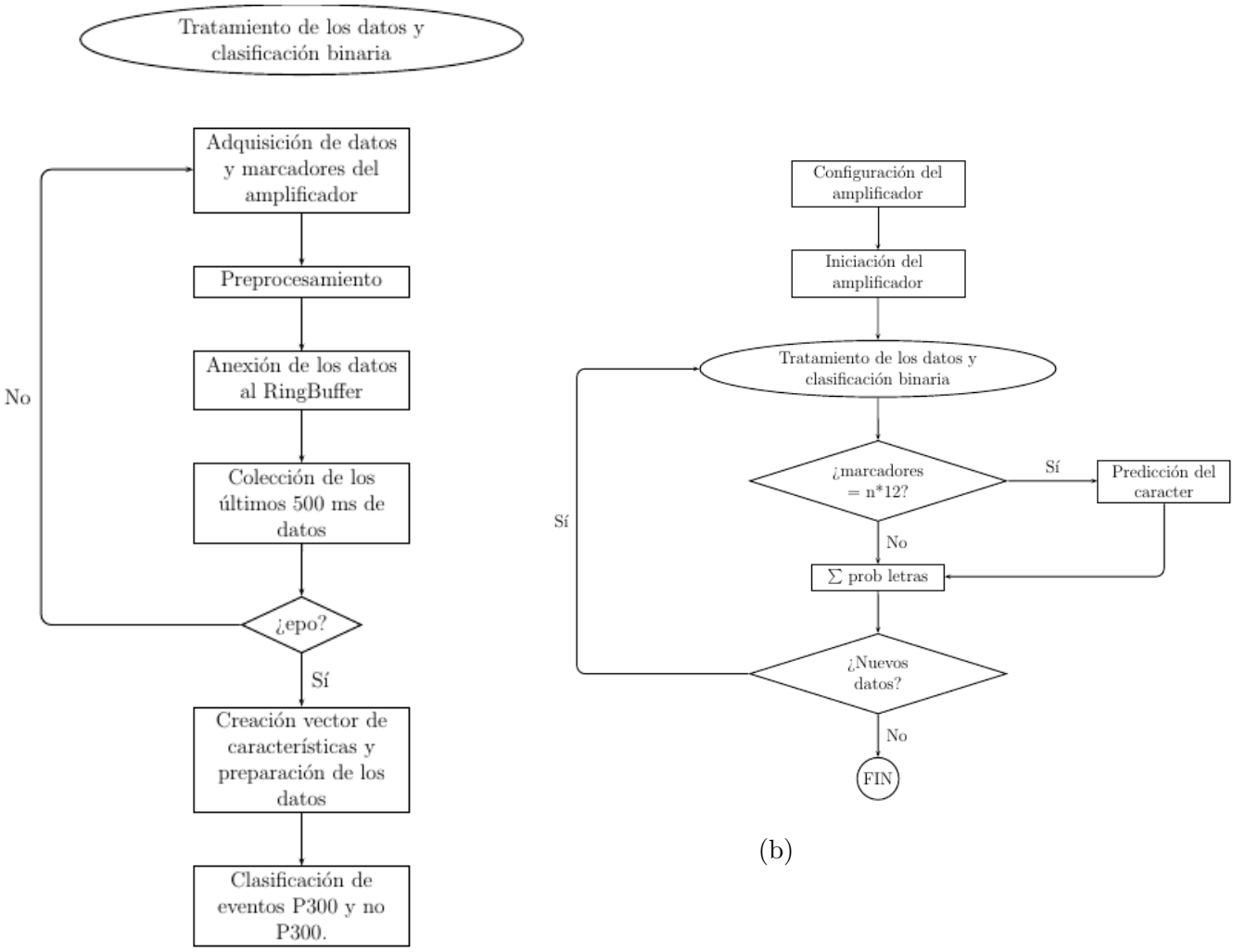


Figura 2.11: Esquema representativo de la simulación online implementada. (a) Adquisición y tratamiento de los datos, y clasificación de eventos P300. (b) Predicción de caracteres. La cantidad n hace referencia al número de trials evaluado ($n = (0, 15)$)

Capítulo 3

Resultados y discusión

3.1. Elección del MLP

En esta sección, se presenta el análisis realizado en la selección del modelo de MLP a implementar. Se evalúan las diferentes arquitecturas en base a su desempeño, poder de generalización y complejidad de la arquitectura.

3.1.1. Definición de los valores de dropout

A fin de obtener modelos con mayor poder de generalización, se evaluó, para cada arquitectura de MLP generada, la incorporación de diferentes factores de dropout luego de cada capa. Los valores evaluados fueron: [0, 0,1, 0,25, 0,5], y los resultados obtenidos se muestran en el anexo A. Para definir estos valores, se estableció una solución de compromiso entre el nivel de sobreajuste del modelo y el puntaje F1 obtenido. Los valores de dropout definidos para cada arquitectura fueron:

Tabla 3.1: Valores de dropout asignados en la construcción de cada una de las arquitecturas de MLP

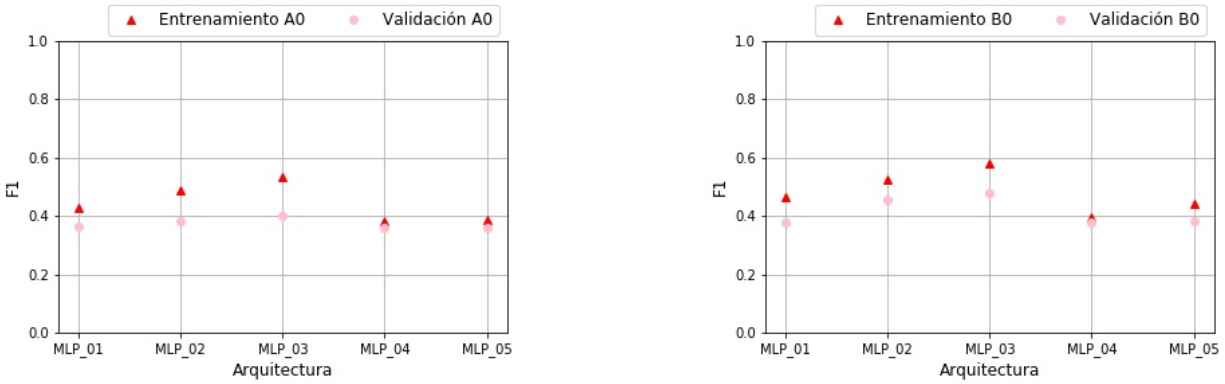
	MLP_01	MLP_02	MLP_03	MLP_04	MLP_05
Dropout	0	0.25	0.25	0.5	0.5

3.1.2. Selección de la mejor arquitectura

Una vez definidas las arquitecturas y sus respectivos valores de dropout, se procedió a su evaluación, computando, en cada caso, las métricas de clasificación binaria y seleccionando la mejor en base a su puntaje F1. Es preciso aclarar que, tanto en el proceso de selección del factor de dropout, como en la evaluación y selección de la mejor arquitectura, se empleó una relación entre los conjuntos de entrenamiento y validación de 0.7 y 0.3, respectivamente; diferente a la empleada en el resto de la investigación (0.9 y 0.1, respectivamente). En ambos casos estas proporciones de datos fueron tomadas del mismo conjunto de entrenamiento, independientemente del conjunto de prueba empleado durante la simulación. Esto se justifica en la medida en que una mayor cantidad de datos de validación permitió obtener una aproximación más certera del desempeño ulterior de cada modelo, debido a la presencia de mayor cantidad de ejemplos de cada clase durante su evaluación.

3.1.2.1. Evaluación del sobreajuste

En esta instancia, se compararon los resultados obtenidos para cada arquitectura en los conjuntos de entrenamiento y validación, a fin de evaluar la diferencia existente entre ambos e identificar los modelos con mayor capacidad de generalización (menor sobreajuste, ver tabla A.1, anexo A) . Como se puede apreciar en la figura 3.1, si bien el desempeño de cada modelo varía según el sujeto, en general los puntajes de validación obtenidos rondan los 0.4 puntos. A pesar de que las arquitecturas 4 y 5 presentaron los menores puntajes en ambos conjuntos, las diferencias entre los respectivos puntajes de entrenamiento y de validación fueron las menores. Esto indica que estos modelos presentaron una capacidad de generalización superior al resto, que mostró un mayor grado de sobreajuste.



(a) Sujeto A (b) Sujeto B

Figura 3.1: Resultados obtenidos en la evaluación de las distintas arquitecturas de MLP diseñadas. (a) Sujeto A. (b) Sujeto B.

3.1.2.2. Evaluación de la complejidad

Ambas arquitecturas, MLP_04 y MLP_05, demostraron un muy buen poder de generalización, siendo levemente superior para el primero. Sin embargo, no fue éste el único criterio tomado en cuenta en la definición de la arquitectura a entrenar y comparar con los restantes tipos de clasificadores. Se consideró, además, la complejidad de las arquitecturas; medida de acuerdo al número de parámetros a entrenar en cada una, siendo un número alto indicador de una gran complejidad y uno bajo lo contrario. El número de parámetros a entrenar fue calculado como la sumatoria de conexiones neuronales entre cada capa, dada por el producto entre la cantidad de neuronas más 1 (correspondiente al término bias) en la capa de salida, y la cantidad de neuronas en la capa subsiguiente.

Para explicar esto con mayor claridad, se toma como ejemplo la arquitectura MLP_05. La primera capa, correspondiente a la capa de entrada, cuenta con 1024 neuronas (correspondientes al número de características del vector de entrada); mientras que la capa subsecuente, primer capa oculta, posee 300. Luego, la cantidad de parámetros a entrenar en esta capa está dada por $300 \cdot (1024 + 1) = 307{,}500$. Análogamente, la capa 2 recibe $300 + 1$ (bias) conexiones neuronales y cuenta con 150 neuronas; dando un total de $150 \cdot (300 + 1) = 45{,}150$ parámetros a entrenar. Se procede de igual manera para la capa de salida y luego se suman los parámetros de cada una de ellas, dando el total de 352,952 parámetros que se muestran en la tabla 3.3.

De la comparación de las tablas 3.2 y 3.3, se puede apreciar que, mientras el modelo MLP_-04 presenta un total de 1.838.338 parámetros a entrenar, el MLP_05 sólo presenta 352.952, lo que constituye un 20 % del anterior. En consecuencia, la arquitectura MLP_05 demanda una

menor cantidad de operaciones matemáticas en su ejecución, lo que repercute sobre el tiempo de cómputo; razón por la cual fue definida como la mejor arquitectura de MLP y fue entrenada para su comparación con los demás clasificadores.

Tabla 3.2: Detalle de la complejidad de la arquitectura MLP_04

Capa	Función	Neuronas	Parámetros
1	tanh	1024	1.049.600
	Dropout	-	0
2	tanh	512	524.800
	Dropout	-	0
3	linear	256	131.328
	Dropout	-	0
4	tanh	512	131.584
	Dropout	-	0
salida	softmax	2	1.026
Total de parámetros			1.838.338

Tabla 3.3: Detalle de la complejidad de la arquitectura MLP_05

Capa	Función	Neuronas	Parámetros
1	tanh	300	307.500
	Dropout	-	0
2	tanh	150	45.150
	Dropout	-	0
salida	softmax	2	302
Total de parámetros			352.952

3.2. Clasificación binaria de potenciales P300

En esta sección, se presentan y analizan los resultados obtenidos durante la etapa de clasificación binaria, previa a la predicción de caracteres ejecutada durante la simulación online. Se evalúa la capacidad de generalización de los diferentes clasificadores, su desempeño global y la existencia de diferencias entre los resultados obtenidos por los sujetos de referencia y sus correspondientes bootstrap. En todos los casos, se comparan los puntajes F1 obtenidos, considerando mejores modelos a aquellos que mayores valores obtengan. Adicionalmente, para los SVMs, se

comparan los porcentajes de vectores de soporte empleados en la definición del hiperplano de separación.

3.2.1. Evaluación del poder de generalización y del desempeño global en la detección de eventos P300

Para cada uno de los clasificadores implementados, se presentan los resultados obtenidos en la evaluación de los conjuntos de entrenamiento y validación para el promedio de los sujetos A y B, tanto de referencia, como los generados mediante la técnica de boostrapping. Cada punto en las gráficas representa la media para cada sujeto, y las barras verticales su correspondiente desviación estándar. A partir de estos resultados, se analiza, para cada uno de los tamaños de entrenamiento considerados, la existencia de algún grado de sobreajuste de los modelos a los datos de entrenamiento, indicador del mayor o menor poder de generalización de cada uno. Además, se compara el desempeño global de los clasificadores en la detección de eventos P300, tomando como referencia los puntajes F1 obtenidos: a mayores puntajes, mayor detección precisa de estos eventos.

Como se puede apreciar en los gráficos de las figuras 3.2 y 3.3, todos los clasificadores, tanto lineales como no lineales (redes neuronales), presentaron un gran poder de generalización, obteniendo los mismos resultados para los conjuntos de entrenamiento y de validación.

Respecto del desempeño de los algoritmos, es notoria la diferencia existente entre los resultados obtenidos para los clasificadores lineales y los obtenidos para las redes neuronales. En los primeros, se observa, en todos los casos, una relación en aumento entre el puntaje F1 y el tamaño con que fue entrenado cada modelo. Es decir que, para mayores tamaños de entrenamiento, el desempeño del algoritmo mejora. Mientras que, para modelos entrenados con 3 trials, los clasificadores lineales obtuvieron puntajes de entre 0.4 y 0.5 puntos; para tamaños de entrenamiento de 12 y 15 trials, los mismos clasificadores obtuvieron puntajes de entre 0.6 y 0.7 puntos.

De comparar los clasificadores lineales entre sí, se puede apreciar que presentaron desempeños similares, aunque se observa que el SVM gaussiano obtuvo puntajes ligeramente superiores, notoriamente para el caso del sujeto A. En general, es posible apreciar que para el sujeto B se obtienen puntajes levemente superiores al A. Mientras que el puntaje máximo logrado por este último es de 0.6 puntos para el LDA y el SVM lineal, y 0.7 para el SVM gaussiano, para el sujeto B se alcanzó un puntaje máximo de aproximadamnte 0.7 puntos para todos los clasificadores lineales.

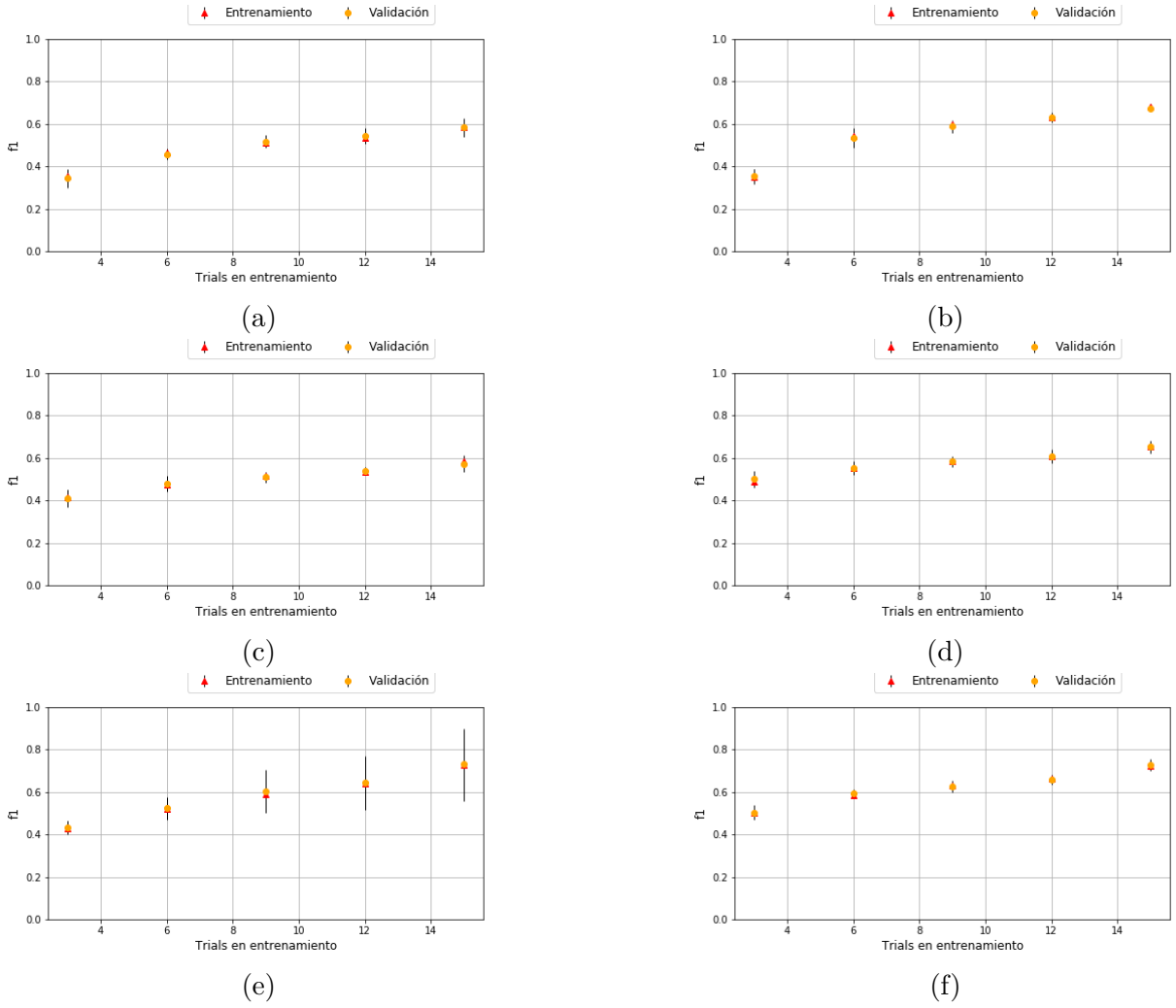


Figura 3.2: Puntajes F1 obtenidos para los clasificadores lineales en los conjuntos de entrenamiento y validación para el promedio de los sujetos A y B (referencia y bootstrap). La columna izquierda corresponde al sujeto A, y la derecha al sujeto B. La primera fila corresponde al clasificador LDA, figuras (a) y (b); la segunda fila al SVM lineal, figuras (c) y (d); y la tercera al SVM gaussiano, figuras (e) y (f).

Las redes neuronales, por su parte, presentaron comportamientos distintivos entre sí y respecto de los clasificadores lineales. En primer lugar, sólo la red CCNN demostró beneficiarse del incremento en el número de trials de entrenamiento, pasando de obtener puntajes F1 de aproximadamente 0.2 puntos (por debajo de este valor, para el sujeto A; y por encima, para el B), a valores de 0.4 y 0.6 puntos, para los sujetos A y B, respectivamente. Por su parte, el resto de las redes presentó resultados que se mantuvieron relativamente constantes respecto del número de trials usados en el entrenamiento. En comparación con los clasificadores lineales, los puntajes obtenidos por estas redes fueron menores, siendo inferiores a 0.4 puntos para el MLP, y estando comprendidos entre 0.4 y 0.6 puntos para el resto de los clasificadores, variando principalmente entre ambos sujetos. Mientras que para el sujeto A estas redes obtuvieron puntajes que rondan los 0.4 puntos; para el sujeto B lograron valores de 0.5 y 0.6 puntos, a excepción del MLP, que no presentó diferencias entre ambos sujetos.

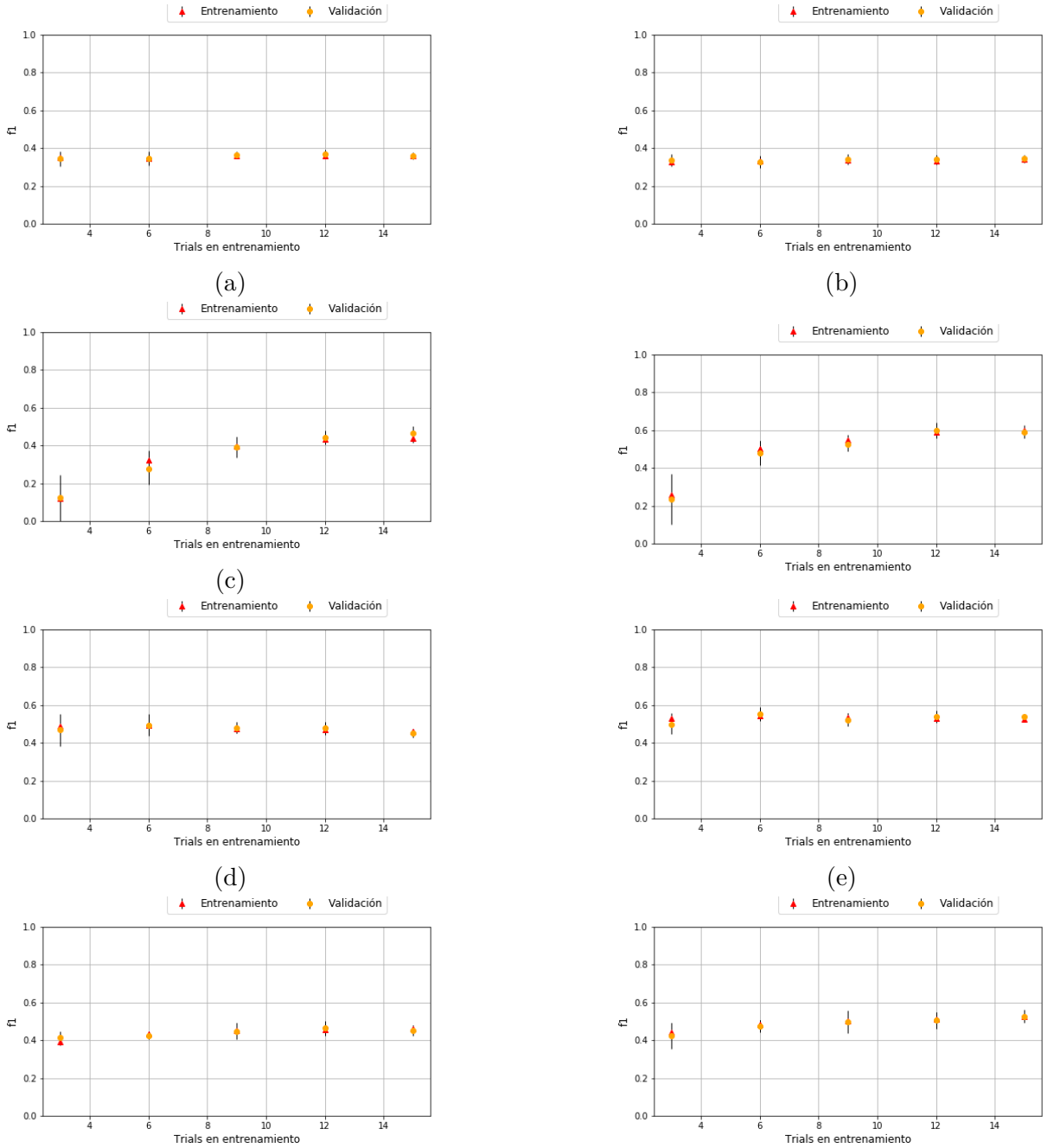


Figura 3.3: Puntajes F1 obtenidos para las redes neuronales en los conjuntos de entrenamiento y validación para el promedio de los sujetos A y B (referencia y bootstrap). La columna de la izquierda corresponde al sujeto A, y la de la derecha al sujeto B. La primer fila corresponde al MLP_05, figuras (a) y (b); la segunda a la red CCNN, figuras (c) y (d); la tercera a la red OCLNN, figuras (e) y (f) y la cuarta, a la red TSMCNN, figuras (g) y (h).

En consecuencia, de este análisis se puede concluir que, si bien ningún clasificador presentó sobreajuste, sí se puedieron observar diferencias de desempeño entre ellos. Los modelos que mejor desempeño presentaron son los lineales, que, además, se beneficiaron de un incremento en el tamaño de entrenamiento. En consecuencia, es de esperar que, en la etapa siguiente de predicción de caracteres durante la simulación online, se obtengan porcentajes de detección superiores a las redes neuronales, y que vayan en aumento conforme se incremente el tamaño

de los conjuntos empleados durante el entrenamiento.

3.2.2. Comparación del desempeño de los sujetos de referencia y boostrap

En esta sección se presentan los resultados de validación obtenidos durante la clasificación binaria para cada uno de los algoritmos evaluados para los sujetos de referencia A0 y B0, en comparación con el promedio de los resultados obtenidos para los respectivos sujetos generados a partir de la técnica de bootstrapping. Se pretende evaluar, para cada sujeto, A y B, la robustez de los diferentes modelos a partir de los distintos conjuntos de entrenamiento; no sólo variables en cuanto a la cantidad de trials empleados por carácter, sino también en cuanto a la cantidad y variedad de caracteres utilizados en el entrenamiento. Esto es porque, como se explicó en la sección 2.4, los sujetos generados a partir de la técnica de boostrapping cuentan con un menor número de caracteres en el conjunto de entrenamiento, variables entre sí.

Las figuras 3.4 y 3.5 muestran la comparación de los resultados obtenidos para los sujetos de referencia y los bootstrap, para los clasificadores lineales y no lineales (redes neuronales), respectivamente. Si bien para algunos clasificadores se puede apreciar que, para determinados tamaños de entrenamiento, existe una leve diferencia entre los puntajes obtenidos por los sujetos de referencia respecto de los bootstrap; en general, para todos los modelos, no se observan diferencias en el desempeño de los diferentes sujetos considerados. Esto indica que los clasificadores son robustos y no se ven influenciados por la cantidad y variedad de caracteres con que fueron entrenados.

Sin embargo, destaca el caso particular del SVM gaussiano que, para el sujeto A, presentó una notoria y creciente diferencia entre los puntajes obtenidos para el sujeto de referencia A0 y sus correspondientes sujetos bootstrap, conforme al incremento el número de trials usados en el entrenamiento. Esta diferencia fue más positiva para el sujeto de referencia, adoptando un máximo de 0.3 puntos para los 15 trials de entrenamiento, en que el sujeto A0 alcanzó un puntaje de clasificación perfecto (1.0 puntos), y los sujetos bootstrap lograron un puntaje de 0.7 puntos. Asimismo, es notoria la variabilidad de resultados obtenidos, en este caso, para los diferentes sujetos bootstrap, indicada por el tamaño de la barra vertical en cada punto, correspondiente a la desviación estándar. Mientras que, para el resto de los clasificadores y para el mismo SVM gaussiano en el caso del sujeto B, la variabilidad entre los sujetos bootstrap es despreciable, o resulta relevante sólo para uno o dos tamaños de entrenamiento; para el SVM gaussiano evaluado en el sujeto A, la desviación estándar se hace creciente conforme se incrementa el tamaño de entrenamiento utilizado.

De este análisis se desprende que, contrario a lo esperado, los sujetos generados mediante la técnica de bootstrapping no presentaron menor capacidad de discriminación de los eventos P300 en comparación con sus respectivos sujetos de referencia. En consecuencia, se puede afirmar que el número extra de caracteres usados en el entrenamiento de estos últimos no se traduce directamente en una mejora en la detección de los potenciales P300. Esto prueba la robustez de los modelos y resulta un dato relevante a tener en cuenta en la definición del número de caracteres a evaluar durante la etapa de calibración.

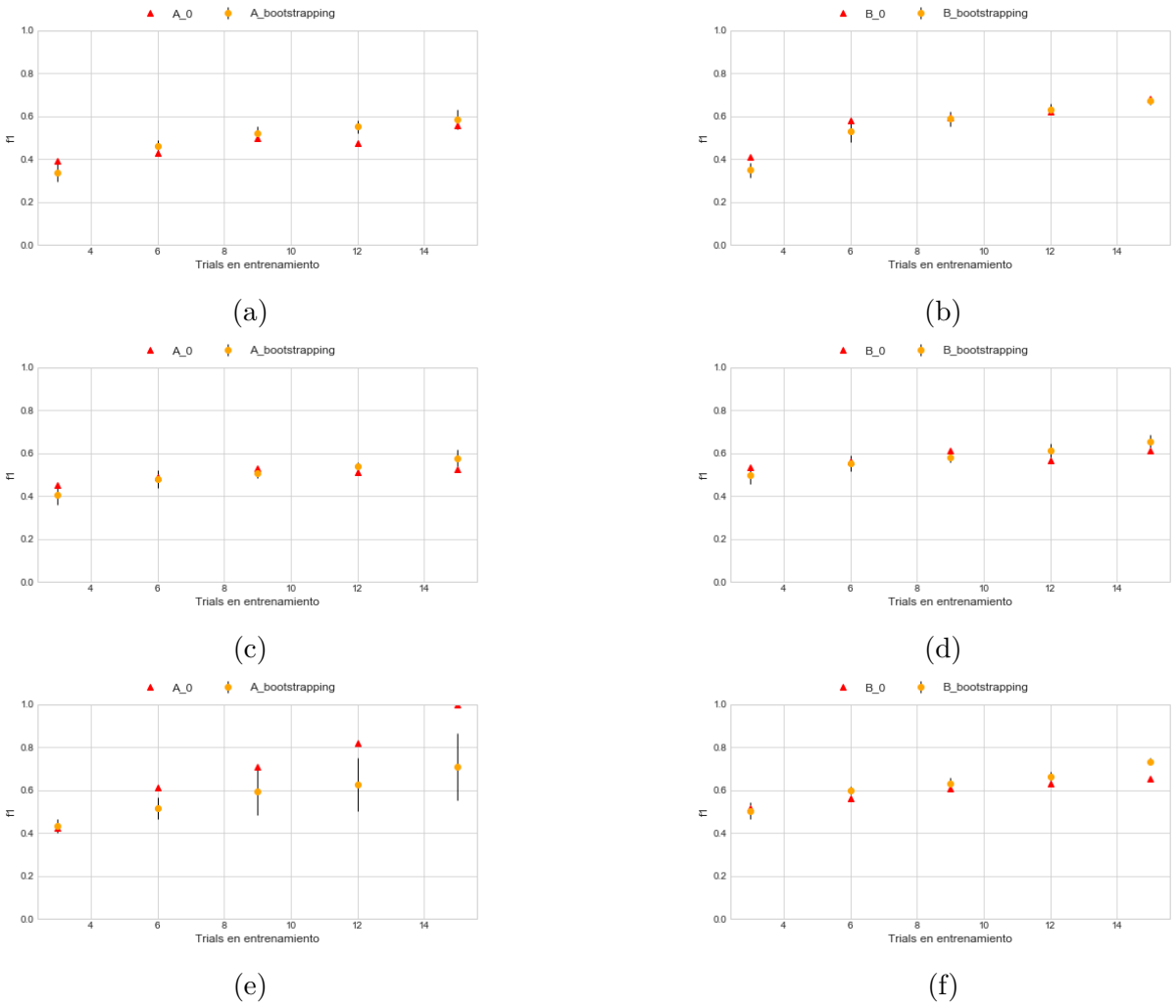


Figura 3.4: Puntajes F1 obtenidos durante la validación para el sujeto de referencia y el promedio de los boostraps, en la evaluación de los diferentes clasificadores lineales. En la columna izquierda se presentan los resultados correspondientes al sujeto A; mientras que en la derecha, los correspondientes al B. La primera fila corresponde al clasificador LDA, figuras (a) y (b); la segunda fila al SVM lineal, figuras (c) y (d); y la tercera al SVM gaussiano, figuras (e) y (f).

Figura 3.5: Puntajes F1 obtenidos durante la validación para el sujeto de referencia y el promedio de los boostraps, en la evaluación de las diferentes redes neuronales. En la columna izquierda se presentan los resultados correspondientes al sujeto A; mientras que en la derecha, los correspondientes al B. La primera fila corresponde al MLP_05, figuras (a) y (b); la segunda fila a la red CCNN, figuras (c) y (d); la tercera fila a la red OCLNN, figuras (e) y (f); y la cuarta fila a la red TSMCNN, figuras (g) y (h).

3.2.3. Comparación de SVMs Lineal y Gaussiano: Relación entre número de vectores de soporte y ejemplos de entrenamiento

El número de vectores de soporte constituye la cantidad de ejemplos del conjunto de entrenamiento que se sitúan a ambos lados del hiperplano de separación de las clases, y que son

tenidos en cuenta en el establecimiento de esta frontera de decisión. En consecuencia, resulta evidente que un número bajo de vectores de soporte indica que las dos clases pueden ser separadas correctamente [47]. En esta sección, por lo tanto, se evalúa el grado de separabilidad de las clases P300 y No-P300 en el espacio de entradas y en el espacio de características, a partir del análisis de la relación entre la cantidad de vectores de soporte definidos por el SVM lineal y el gaussiano, respectivamente, respecto del número total de ejemplos de entrenamiento. Además, se analiza la diferente proporción de vectores de soporte por cada clase, tanto para los sujetos de referencia y bootstrap entre sí, como para los sujetos A y B entre sí.

En los gráficos de la figura 3.6, se observa que existe una diferencia en el porcentaje de vectores de soporte por clase definidos por ambos clasificadores SVM. Si bien ésta se torna más pronunciada para el SVM gaussiano, notablemente para el sujeto A, en todos los casos se observa que se precisó un porcentaje mayor de ejemplos de la clase P300 respecto de la No-P300 para definir el hiperplano de decisión. Esto se explica en la medida en que, por un lado, debido al desbalance de clases presente en la base de datos, la clase P300 resulta menos representada. Por esta razón, durante el entrenamiento, se le asignó un peso mayor que a la clase No-P300, a fin de compensar la desproporción. Estos pesos, de acuerdo a lo explicado en la sección 2.7.2, asignan diferentes coeficientes de margen blando C a cada clase, mediante una relación de proporcionalidad inversa al número de instancias por clase. En términos prácticos, esto se traduce en una mayor relajación en el grado de separabilidad de la clase P300, admitiendo un número mayor de ejemplos mal clasificados que en la clase No-P300. Por otra parte, si se consideran las características distintivas del potencial P300 y de la actividad neuronal de base, plasmadas en los registros EEG, es posible intuir que este fenómeno responde también a la variabilidad en amplitud, duración y latencia que presenta la onda P300. Esta variabilidad, causada por la naturaleza del potencial y plasmada de forma gráfica en los registros EEG, se refleja en la consecuente variabilidad de los vectores de características correspondientes a la clase P300. Por ende, el clasificador necesitará un número grande de vectores de soporte que contemplen el amplio espectro de señales P300 que se puedan generar en el sujeto. En cambio, la respuesta que se obtiene ante estímulos estándar (no P300), produce una actividad cerebral similar a la de base, que presenta un menor nivel de variabilidad.

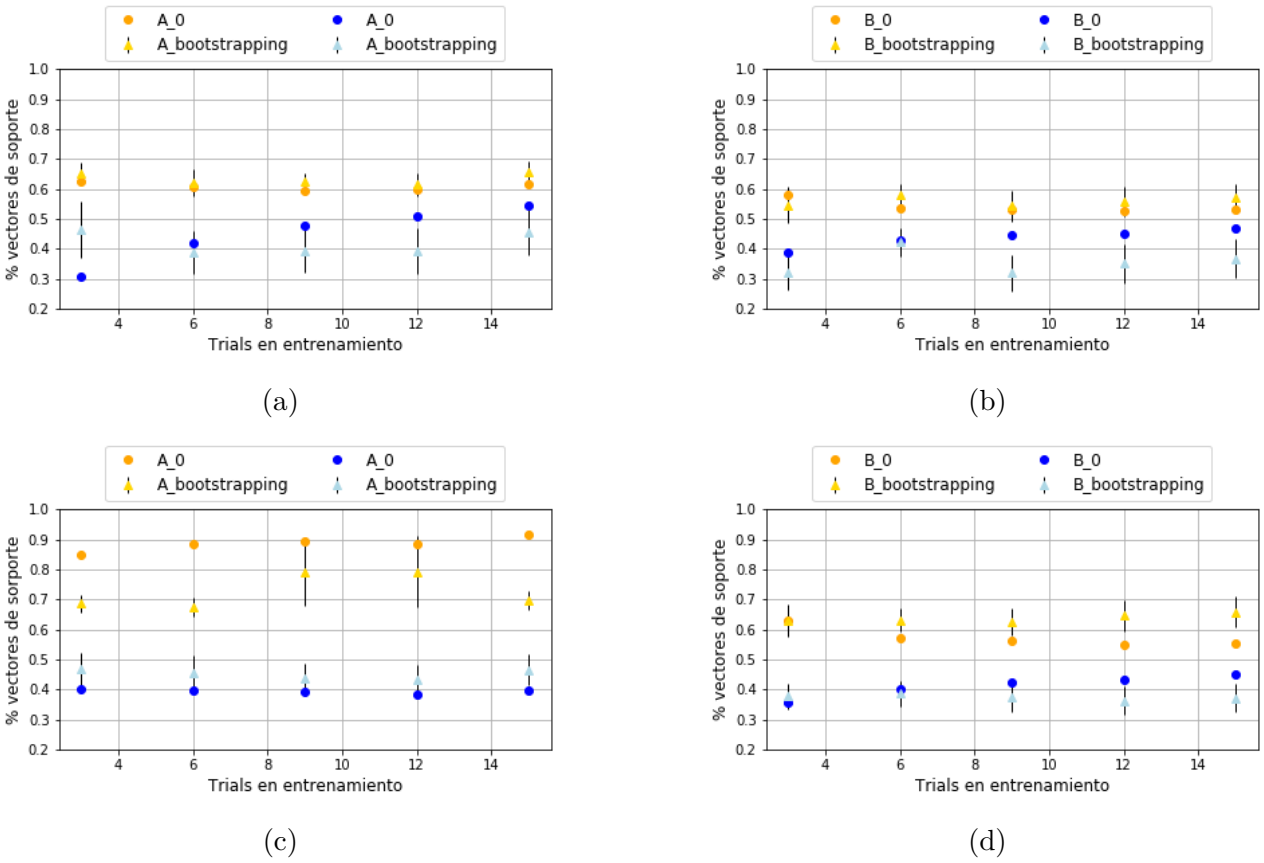
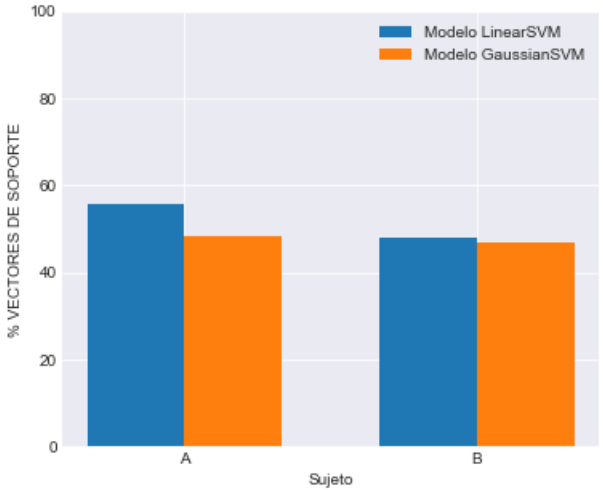
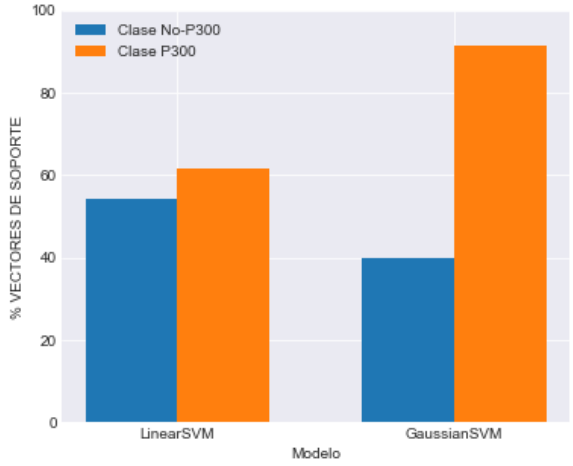


Figura 3.6: Comparación de la relación entre el número de vectores de soporte y el número de ejemplos para cada clase: P300 (en naranja), no P300 (en azul) (a) SVM lineal, sujeto A. (b) SVM lineal, sujeto B. (c) SVM gaussiano, sujeto A. (d) SVM gaussiano, sujeto B.

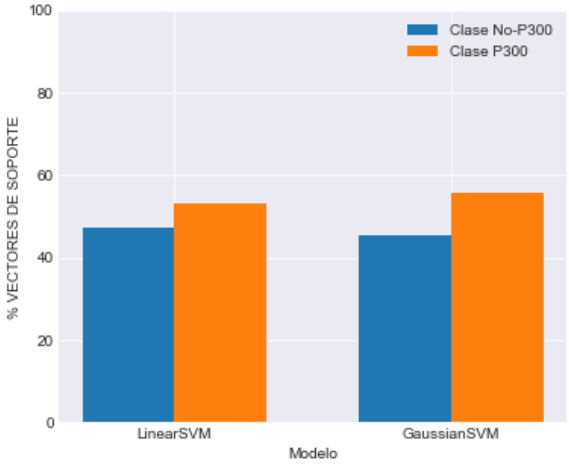
La figura 3.7a muestra una comparación del porcentaje total de vectores de soporte que cada clasificador precisa para los sujetos A y B. Se puede apreciar que, para el sujeto B, la cantidad de vectores de soporte precisada por ambos clasificadores fue prácticamente la misma; mientras que, para el A, el SVM lineal tomó cerca de un 10 % más de vectores de soporte que el gaussiano. Las figuras 3.7 b y c, por su parte, muestran, para el sujeto A y B respectivamente, una discriminación por clase del total de vectores que ambos definen. En general, se puede apreciar lo mismo que en la figura 3.6, en que la clase P300 demandó un mayor número de vectores de soporte que la No-P300. Entre ambos sujetos, se percibe que, para el sujeto B, las cantidades fueron uniformes para ambos SVM; mientras que, para el A, el SVM gaussiano necesitó un 50 % más de vectores de soporte para la clase P300 que para la No-P300.



(a)



(b)



(c)

Figura 3.7: Porcentaje de número de vectores de soporte respecto del número total de ejemplos en el conjunto de entrenamiento para los sujetos A y B de referencia. (a) Comparación del porcentaje de vectores de soporte definidos por los SVM lineal y gaussiano para los sujetos A y B. (b) y (c) Comparación del porcentaje de vectores de soporte por clase definidos por los SVM lineal y gaussiano para los sujetos A y B, respectivamente.

3.3. Simulación online

En esta etapa, se presentan y analizan los resultados obtenidos en la predicción de caracteres efectuada durante la simulación online. En primer lugar, se evalúan los diferentes tiempos de cómputo demandados por los clasificadores durante su entrenamiento. A continuación, se comparan los valores obtenidos para los sujetos de referencia y sus respectivos sujetos bootstrap; luego, se compara el desempeño obtenido para las diferentes redes neuronales para, finalmente, evaluar los resultados obtenidos para los modelos de diseño propio.

3.3.1. Evaluación del tiempo de cómputo en el entrenamiento de los clasificadores

En esta sección se hace una breve comparación de los diferentes tiempos de cómputo demandados por los algoritmos en el entrenamiento. La tabla 3.4 muestra los valores obtenidos para cada uno, en segundos, de acuerdo a los tamaños variables del conjunto de entrenamiento empleados. Los modelos fueron entrenados con un procesador Intel Core i5, de séptima generación y 2.50 GHz, con 8 GB de memoria RAM. Además, las redes neuronales fueron entrenadas con una placa de video NVIDIA GeForce 940MX. Se observa que el tiempo de cómputo se incrementa conforme lo hace el número de trials usado en el entrenamiento; pero, las mayores diferencias de tiempo requerido se observan entre los diferentes clasificadores entre sí. A medida que aumenta su complejidad, se incrementa el tiempo de entrenamiento: los valores más bajos se obtuvieron para los clasificadores lineales, en particular para el LDA; mientras que los mas altos los presentaron las redes neuronales. De éstas, la que mayor tiempo de entrenamiento demandó fue la CCNN y, la que menor, el MLP.

Tabla 3.4: Tiempo de cómputo de los clasificadores respecto del número de trials usados en el entrenamiento[1]

Modelo	Épocas de caracteres	Número de trials				
		3	6	9	12	15
LDA	1	0,008	0,01	0,013	0,016	0,021
	50	**0,4**	**0,5**	**0,65**	**0,8**	**1,05**
LinearSVM	1	0,009	0,034	0,079	0,151	0,252
	50	0,45	1,7	3,95	7,55	12,6
GaussianSVM	1	0,01	0,057	0,134	0,198	0,378
	50	0,5	2,85	6,7	9,9	18,9
MLP	1	2,932	2,416	2,486	2,537	3,658
	50	146,6	120,8	124,3	126,85	182,9
CCNN	1	5,813	3,935	4,57	4,734	7,24
	50	**290,65**	**196,75**	**228,5**	**236,7**	**362**
OCLNN	1	4,267	4,302	3,159	3,794	4,659
	50	213,35	215,1	157,95	189,7	232,95
TSMCNN	1	4,219	3,245	3,494	3,516	4,97
	50	210,95	162,25	174,7	175,8	248,5

[1]Tiempo de cómputo expresado en segundos. En negrita se resaltan los menores y mayores tiempos de cómputo obtenidos en el entrenamiento con 50 caracteres para cada uno de los números de trials usados.

3.3.2. Comparación del desempeño de los sujetos de referencia y boostrap

Las figuras 3.8 y 3.9 muestran el desempeño de los algoritmos lineales y no lineales (redes neuronales), demostrado en la predicción de caracteres durante la simulación online. Se comparan los diferentes resultados obtenidos para los sujetos de referencia A0 y B0 respecto del promedio de los sujetos bootstrap. Para estos, los puntos en las gráficas representan el promedio de los puntajes obtenidos, y las líneas verticales su correspondiente desviación estándar.

No se observa una variabilidad importante en los puntajes obtenidos para los sujetos bootstrap entre sí, en los diferentes tamaños de entrenamiento evaluados mediante los distintos clasificadores. Por otra parte, se puede apreciar que, en general, no existe una diferencia importante en el desempeño logrado por los sujetos de referencia respecto del logrado por los bootstrap; a excepción del LDA. Éste, fundamentalmente para el sujeto A, presentó resultados notablemente superiores para el sujeto de referencia en los modelos entrenados con 3, 6 y 9 trials. En la mayoría de los casos para los demás clasificadores, si bien pequeña, esta diferencia se mostró a favor de los sujetos de referencia. Es decir, estos obtuvieron un puntaje ligeramente superior al de sus respectivos sujetos bootstrap. Esta diferencia se hace más pronunciada para los modelos entrenados con un número bajo de trials (3 y 6). Para tamaños de entrenamiento superiores, la diferencia se torna prácticamente despreciable. Son excepcionales aquellos casos en que los sujetos bootstrap superaron al de referencia, y se dan para no más de uno o dos tamaños de entrenamiento en un mismo clasificador.

En general, se observa que los diferentes clasificadores se benefician del incremento en el número de trials de entrenamiento, mostrando una curva creciente para los 3, 6 y 9 trials, y alcanzando una meseta para este último tamaño de entrenamiento.

Estos resultados, en comparación con los mostrados en la sección 3.2.2 para el caso binario, también muestran que los modelos son robustos a la variabilidad de ejemplos en el conjunto de entrenamiento, y que no existe un verdadero beneficio entre entrenar los algoritmos con 50 o 60 caracteres, como es el caso de los sujetos bootstrap, y hacerlo con 85 caracteres, como los sujetos de referencia. Por otra parte, no se observó una correlación entre los resultados obtenidos durante la clasificación binaria para el SVM gaussiano y el LDA, y los obtenidos durante la predicción de caracteres. Mientras que, para el caso binario, el LDA no mostraba diferencias notorias entre los resultados obtenidos por los sujetos bootstrap y los de referencia; durante la predicción de caracteres sí demostró una diferencia apreciable. Al contrario, el SVM gaussiano mostraba una diferencia creciente entre el sujeto de referencia y los boostrap durante la clasificación binaria; pero en la simulación online no demostró entre estos una diferencia apreciable. Esto puede deberse al hecho de que, durante esta última, se suman las probabilidades de pertenencia a cada clase obtenidas para cada instancia de intensificación de las diferentes filas y columnas. En consecuencia, puede suceder que entre ellas se atenúen o acentúen las diferencias observadas durante la clasificación binaria, produciendo un efecto diferente en la predicción de caracteres.

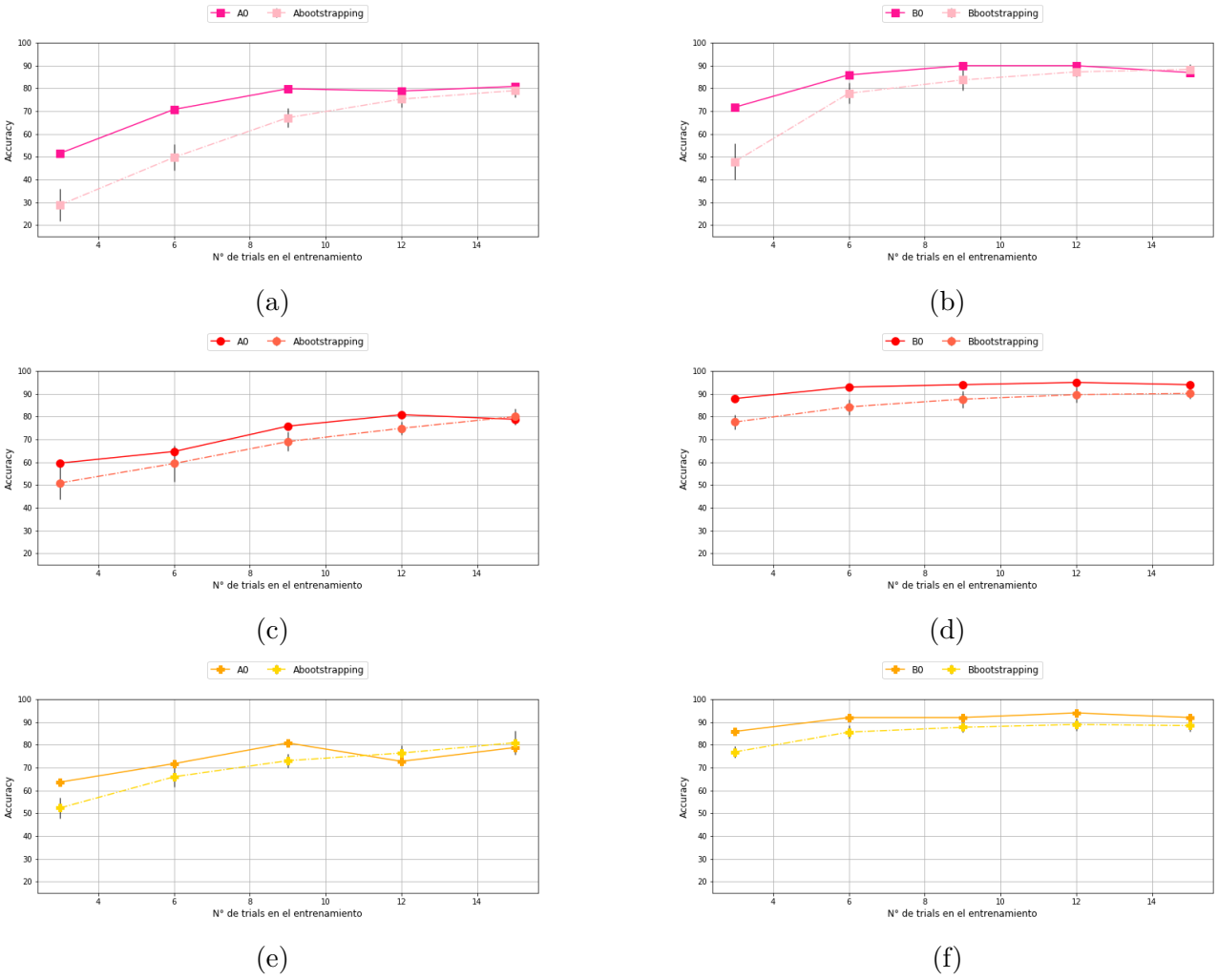


Figura 3.8: Comparación de la exactitud obtenida por los diferentes clasificadores lineales, tras 15 trials de simulación, para los sujetos de referencia A0 y B0, en comparación con sus respectivos sujetos bootstrap. La comparación se realiza para los diferentes tamaños de entrenamiento evaluados. A la izquierda se presentan los resultados correspondientes a los sujetos A (A0 y promedio de los bootstrap); a la derecha los correspondientes a los sujetos B (B0 y promedio de los bootstrap). La primera fila, figuras (a) y (b), corresponde al clasificador LDA; la segunda fila, figuras (c) y (d), corresponde al SVM lineal; y la tercera fila, figuras (e) y (f), corresponde al SVM gaussiano.

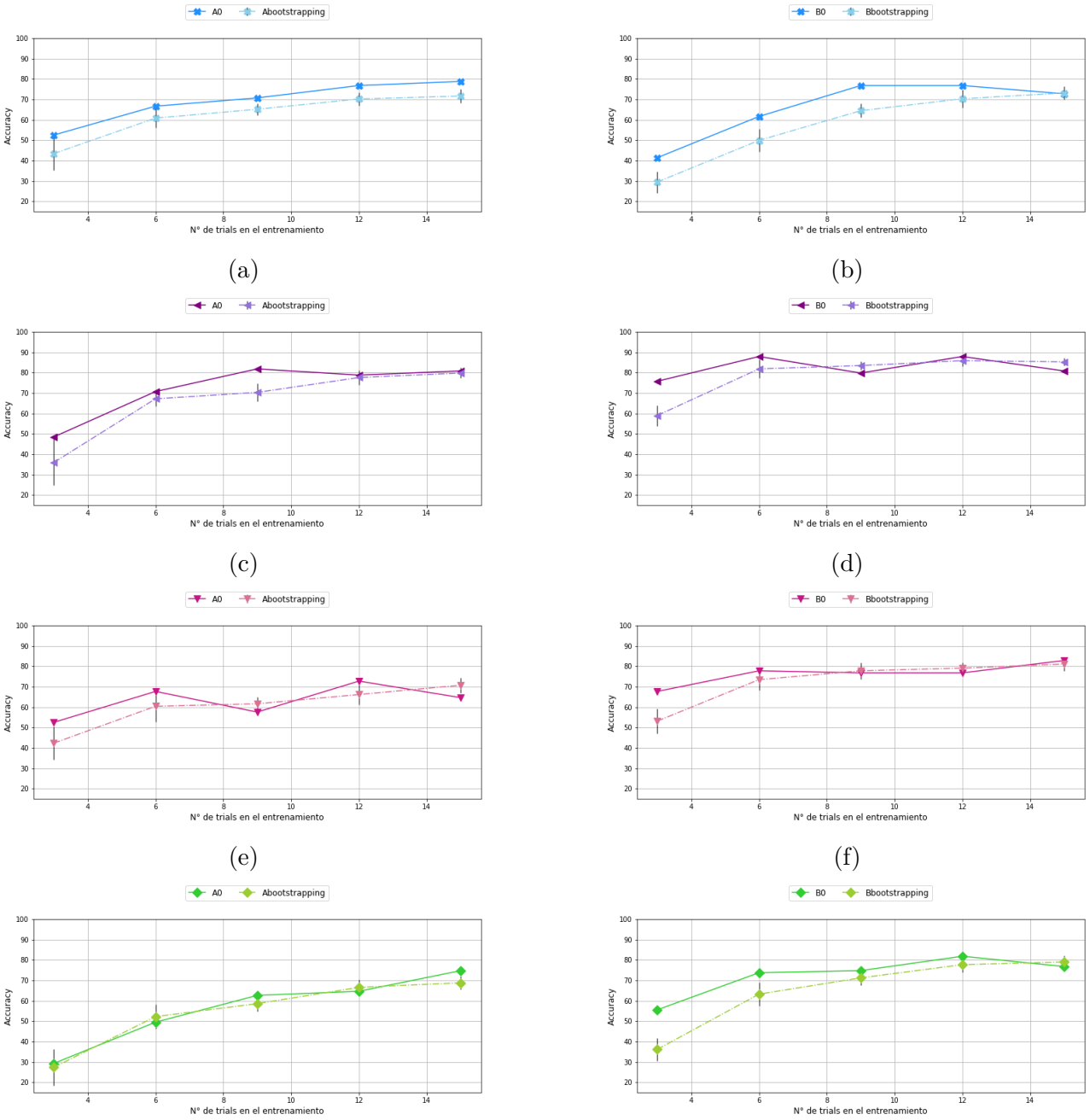


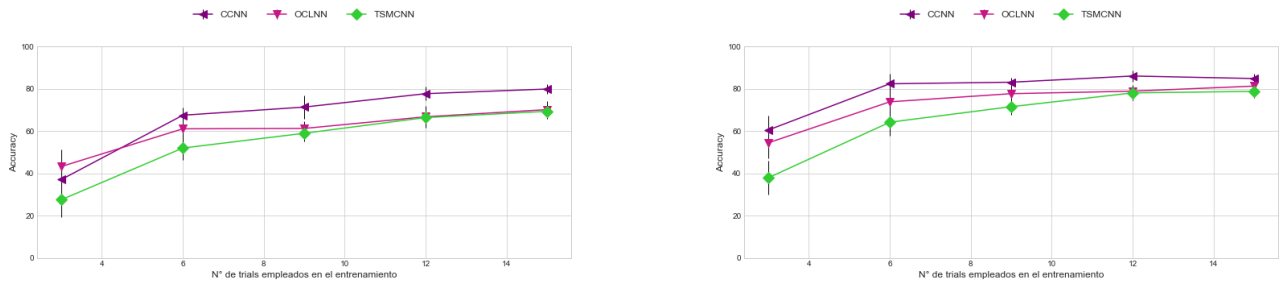
Figura 3.9: Comparación de la exactitud obtenida por las diferentes redes neuronales (clasificadores no lineales), tras 15 trials de simulación, para los sujetos de referencia A0 y B0, en comparación con sus respectivos sujetos bootstrap. La comparación se realiza para los diferentes tamaños de entrenamiento evaluados. A la izquierda se presentan los resultados correspondientes a los sujetos A (A0 y promedio de los bootstrap); a la derecha los correspondientes a los sujetos B (B0 y promedio de los bootstrap). La primera fila, figuras (a) y (b), corresponde al MLP; la segunda fila, figuras (c) y (d), corresponde a la red CCNN; la tercera fila, figuras (e) y (f), corresponde a la red OCLNN; y la cuarta fila, figuras (g) y (h), corresponde a la red TSMCNN.

3.3.3. Comparación del desempeño de las redes convolucionales

En esta sección se compara el desempeño logrado, durante la simulación online, por las diferentes redes convolucionales implementadas; dado que interesa evaluar el desempeño correspondiente a la red TSMCNN, de diseño propio, respecto del presentado por las redes implementadas a partir de la literatura. Se compara, en un primer momento, la exactitud en la predicción de caracteres respecto del tamaño de entrenamiento utilizado. Posteriormente, se evalúa también el desempeño demostrado en la predicción de caracteres, pero en relación con el número de trials utilizados durante la simulación.

3.3.3.1. Exactitud en la predicción de caracteres vs tamaño de entrenamiento

En la figura 3.10 se observa que la red convolucional que presentó el mejor desempeño es la red CCNN. Ésta alcanzó un puntaje máximo de 80 % para el sujeto A, al ser entrenado con 15 trials; y algo superior al 80 % para el sujeto B, logrado para los modelos entrenados con cantidades de trials iguales o superiores a 6. En segundo lugar, se encuentra la red OCLNN, con puntajes máximos de 75 % para el sujeto A, entrenado con 15 trials, y 80 % para el sujeto B, entrenado con 12 y 15 trials. La red TSMCNN, si bien es la que menor desempeño demostró tener, a partir de los 9 trials de entrenamiento obtuvo resultados muy cercanos, en algunos casos iguales, a la red OCLNN. Sin embargo, para modelos entrenados con menores tamaños de entrenamiento, demostró tener un poder predictivo muy pobre. En general, las tres redes se beneficiaron de un incremento en el tamaño de entrenamiento, aunque la CCNN fue la menos sensible a este factor. El incremento más notable en el desempeño de las redes se dio entre los 3 y 6 trials usados durante el entrenamiento.



(a) (b)

Figura 3.10: Comparación de los porcentajes de aciertos obtenidos durante la simulación online para las diferentes redes convolucionales. Se compara la accuracy obtenida para los modelos entrenados con 15 trials y su variación respecto de los diferentes tamaños de entrenamiento.(a) Sujeto A. (b) Sujeto B.

3.3.3.2. Exactitud en la predicción vs número de trials en la prueba

La figura 3.11 muestra los resultados obtenidos por los modelos de redes convolucionales entrenados con números de trials variables, para diferente número de trials en la simulación online. Estos resultados se presentan para el promedio de los sujetos A y B.

Resulta evidente la diferencia de desempeño de todas las redes en general para tamaños crecientes de entrenamiento. Al comparar las exactitudes máximas logradas para cada uno, se observa que, para 3 trials de entrenamiento, las redes mostraron un muy pobre desempeño. Lograron porcentajes de acierto comprendidos entre 25 % y 40 % para el sujeto A, y de 40 %

y 60 % para el sujeto B. En cambio, a partir de los 6 trials de entrenamiento, los puntajes máximos obtenidos para cada red aumentan en un 20 % para ambos sujetos.

Por otra parte, se observa que todas las redes se beneficiaron de un incremento en el número de trials usados durante la prueba: en los primeros 10 trials de simulación, se aprecia un incremento pronunciado para todos los modelos, notablemente para aquellos entrenados con mayor cantidad de datos; y, a partir de los 10 trials se establece una meseta, en la que el incremento en la exactitud de la predicción se torna más leve y prácticamente no se aprecian mejoras. En todos los casos, como se vio en el apartado anterior, la red que mejor desempeño demostró fue la CCNN, con puntajes cercanos al 80 % para el sujeto A, y por encima de éste para el sujeto B. Si bien la red TSMCNN demostró un menor desempeño para los modelos entrenados con 3 y 6 trials, a partir de los 9 trials de entrenamiento presentó un desempeño similar a la red OCLNN.

Por otra parte, en la tabla 3.5 se comparan los valores de exactitud e ITR obtenidos para las tres redes convolucionales; evaluados sobre los modelos entrenados con 15 trials y sobre el total de los trials de prueba durante la simulación, para el promedio de los sujetos A y B. Se observa que las tres redes presentaron una baja tasa de transferencia de la información, en comparación con los trabajos de referencia [9], en que, para 15 trials de prueba, los modelos entrenados con el conjunto de entrenamiento presentaron valores de 7 y 8 bpm. Más aún, la red CCNN obtuvo un puntaje de casi 6 bpm, 2 puntos menor que el obtenido en el trabajo de referencia [9], en que se implementa la misma arquitectura pero con parámetros de entrenamiento diferentes. En consecuencia, es posible que las mismas arquitecturas puedan arrojar valores superiores si se someten a un procedimiento minucioso de optimización de los parámetros correspondientes al entrenamiento de la red. A pesar de esto, cabe destacar que, entre sí, las tres redes obtuvieron un desempeño global de similar magnitud, al ser evaluadas bajo los mismos parámetros de entrenamiento, definidos como estándar. Por otra parte, retomando lo expuesto en la sección 2.7.3.3, se puede decir que, el aplicar las operaciones de convolución temporal y espacial sobre los datos originales, ya sea por separado (TSMCNN), o bien en un mismo kernel (OCLNN), no arrojó resultados superiores a los presentados por CCNN, como se esperaba. Contrariamente, fue ésta la que mejor desempeño presentó, a pesar de computar la convolución temporal sobre datos abstractos. Por lo tanto, el desempeño de las redes parece estar fuertemente ligado no sólo a su arquitectura, sino a la definición de parámetros de entrenamiento óptimos.

Tabla 3.5: Comparación de ITR y Accuracy obtenidas por las CNNs[2]

Modelo	CCNN		OCLNN		TSMCNN	
	Accuracy	ITR	Accuracy	ITR	Accuracy	ITR
Sujeto A	80,00	5,97	64,00	4,13	74,00	5,24
Sujeto B	80,00	5,97	83,00	6,22	76,00	5,48
Media	**80,00**	**5,97**	73,50	5,18	75,00	5,36

[2]Accuracy expresada en valor porcentual (%). En negrita se resaltan los máximos valores promedio obtenidos.

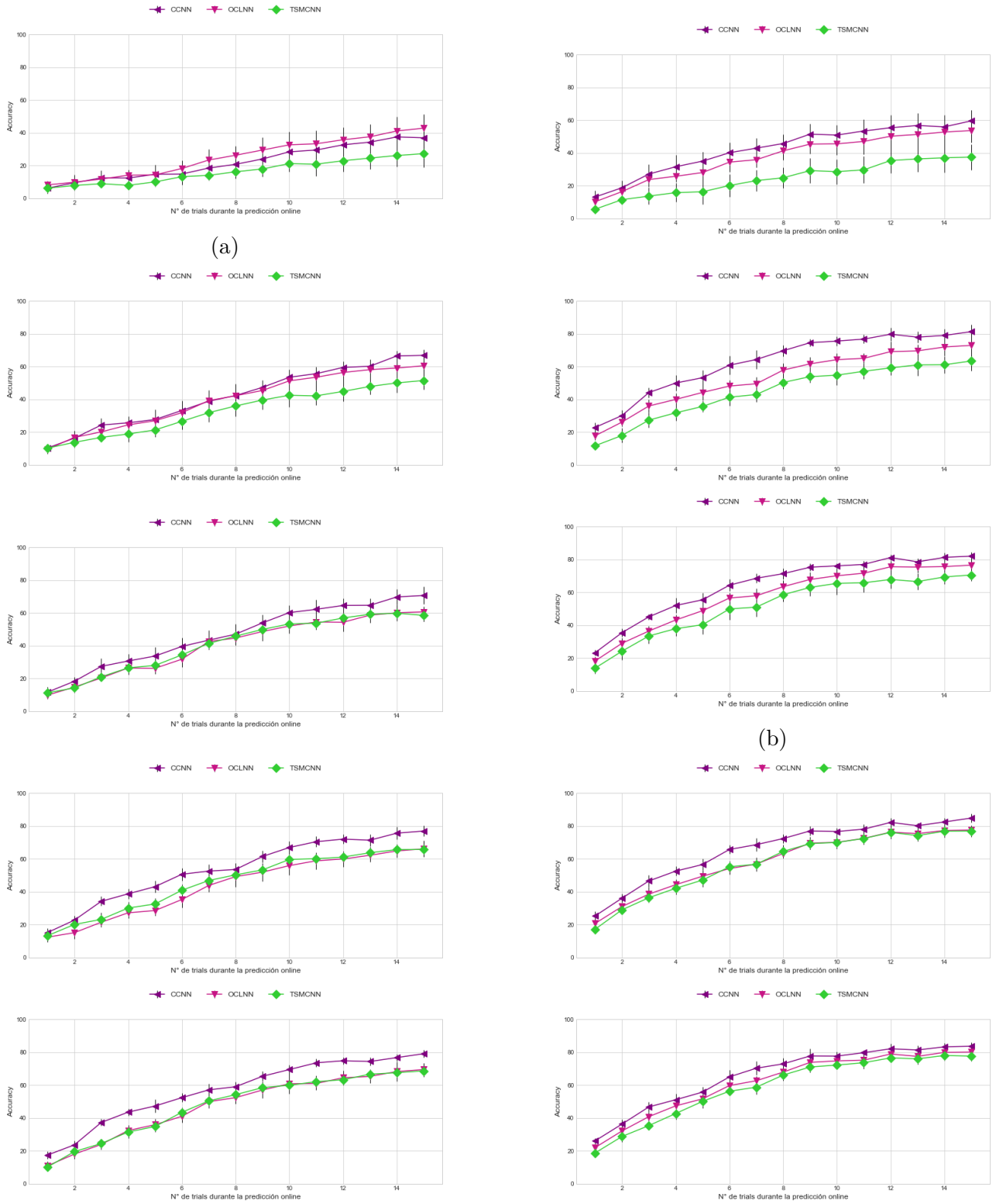


Figura 3.11: Comparación de los porcentajes de aciertos obtenidos por las diferentes redes convolucionales para sujetos entrenados con 15 trials en los diferentes trials empleados durante la simulación online. A la izquierda, resultados correspondientes al sujeto A; a la derecha, al B. De la primera a la quinta fila: modelos entrenados con 3, 6, 9, 12 y 15 trials.

3.3.4. Modelos de diseño propio

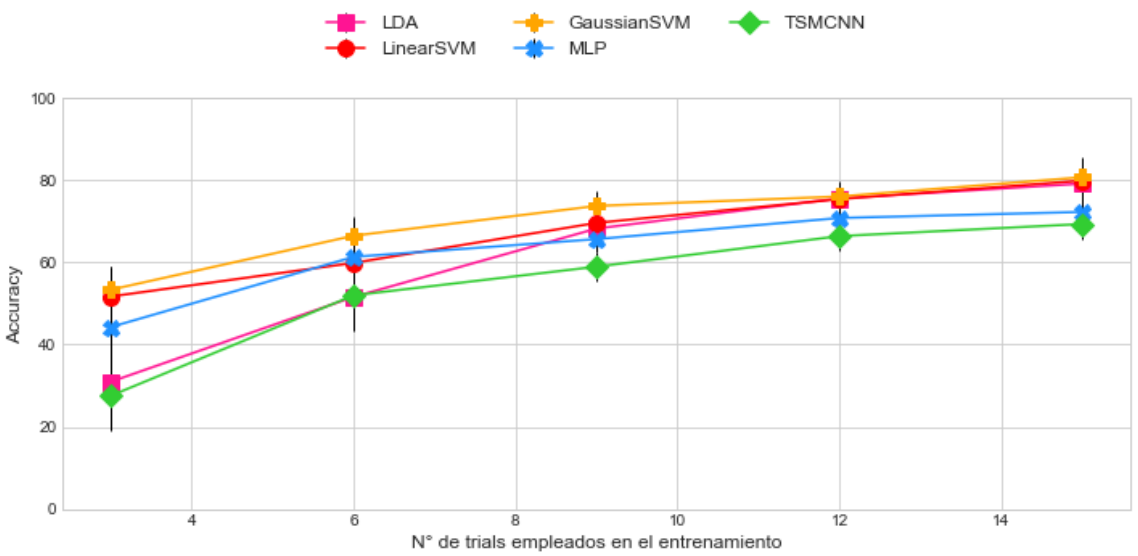
En esta sección se compara el desempeño presentado por los clasificadores online, a partir del análisis de las tasas de predicción de caracteres logradas en relación con el tamaño de entrenamiento y de prueba online, y de la tasa de transferencia de la información. El análisis está principalmente enfocado a la determinación del mejor clasificador a utilizar y de los parámetros de diseño del entrenamiento que se deban determinar para la implementación real de la interfaz; a fin de garantizar tanto una alta tasa de predicción de caracteres, como un tiempo de selección de caracteres lo más bajo posible.

3.3.4.1. Exactitud en la predicción vs tamaño de entrenamiento

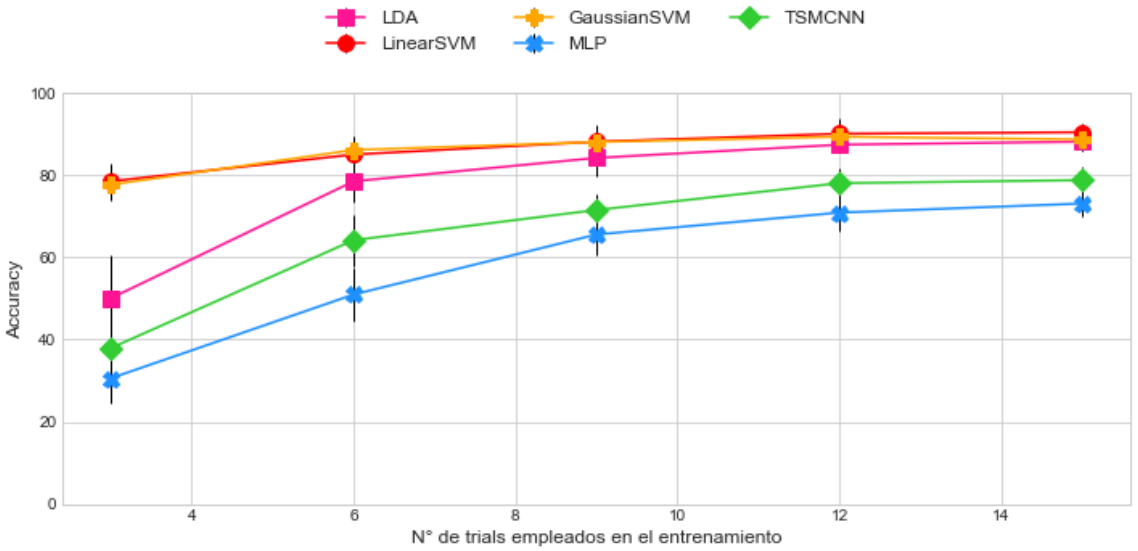
La figura 3.12 muestra, a simple vista, que los modelos que mejor desempeño presentaron en la predicción online simulada de los caracteres fueron los dos tipos de SVM, lineal y gaussiano. Estos demostraron un desempeño prácticamente equivalente para el sujeto B; mientras que, para el A, el SVM gaussiano obtuvo puntajes levemente superiores. Asimismo, es posible apreciar que, a partir de los 6 trials de entrenamiento, el clasificador LDA presentó prácticamente el mismo desempeño que ambos SVM. Los modelos no lineales, en cambio, demostraron un menor desempeño para todos los tamaños de entrenamiento, notoriamente para el sujeto B. En particular, para el sujeto A, el MLP obtuvo puntajes muy cercanos a los presentados por los clasificadores lineales; estando la red TSMCNN algo por debajo, fundamentalmente para los primeros trials de entrenamiento. En cambio, para el sujeto B, fue el MLP el que mostró el desempeño más bajo, estando la red TSMCNN levemente por encima de éste, aunque con puntajes menores en un 10 % a los logrados por los modelos lineales.

Se destaca, asimismo, la marcada diferencia que existe entre ambos sujetos. Para el sujeto A, el puntaje máximo fue de 80 %, obtenido para los clasificadores lineales entrenados con 15 trials; mientras que, para el caso del sujeto B, el máximo alcanzado fue de aproximadamente 90 %, también para los modelos lineales, pero logrado con sólo 9 trials de entrenamiento. Más aún, los SVM alcanzaron un 80 % de exactitud en los primeros tres trials de entrenamiento, en que los mismos clasificadores sólo lograron un 50 % para el sujeto A. Además, las curvas obtenidas para el sujeto A se encuentran más cercanas entre sí respecto de las del sujeto B, lo que indica que el desempeño en relación con los diferentes clasificadores varió en menor medida que para este último. Esto corrobora la influencia del sujeto o usuario de la interfaz en la clasificación de la señal, ya que factores como la atención, el estado de ánimo, la motivación pueden influir en la evocación del estímulo P300 y, por lo tanto, en su correcta clasificación.

En general, los modelos no presentaron gran variabilidad para los diferentes sujetos evaluados (de referencia y bootstrap en cada caso), siendo ésta más notoria en los modelos entrenados con 3 y 6 trials. Para ambos sujetos, entre los 12 y 15 trials no se observa una diferencia importante en el desempeño para ninguno de los clasificadores evaluados. Es probable que esto se deba a que, a partir de cierto tamaño de entrenamiento, se genere acostumbramiento en el sujeto, lo que posiblemente ocasiona una onda P300 de menor magnitud. Para el caso particular de los clasificadores lineales, en ambos sujetos se alcanza un puntaje elevado, muy cercano al máximo logrado para cada uno, con sólo 9 trials de entrenamiento.



(a)



(b)

Figura 3.12: Comparación de la exactitud promedio obtenida para los modelos de diseño propio, evaluada en los 15 trials de simulación online, de acuerdo a los diferentes tamaños de entrenamiento. (a) Sujeto A. (b) Sujeto B.

3.3.4.2. Exactitud en la predicción versus número de trials en la prueba para diferentes tamaños de entrenamiento

La figura 3.13 muestra los resultados obtenidos para los clasificadores de diseño propio, en la evaluación de su desempeño durante la predicción de caracteres online simulada, para diferentes números de trials empleados durante la simulación. Estos resultados se presentan para ambos sujetos, entrenados con distintos tamaños de entrenamiento.

Se observa que, para cada uno de los tamaños de entrenamiento evaluados, el sujeto A presentó curvas más similares entre sí, en comparación con el sujeto B. Para éste, las redes MLP y TSMCNN mostraron, en todos los casos, desempeños menores respecto del resto de los clasificadores. En cambio, para el sujeto A, a partir de los 9 trials de entrenamiento, todos los clasificadores presentaron curvas similares entre sí. Esto parece indicar que el sujeto B es más sensible al tipo de clasificador que el sujeto A. En ambos casos, los clasificadores que mejores resultados obtuvieron fueron los SVM lineal y gaussiano, que presentaron prácticamente los mismos valores para cada sujeto. A partir de los nueve trials de entrenamiento, el LDA demostró un desemeño similar a los SVMs. Por otra parte, se observa que, en general, los puntajes máximos alcanzados para el sujeto B por los diferentes clasificadores fueron superiores a los obtenidos para el A, a excepción del MLP.

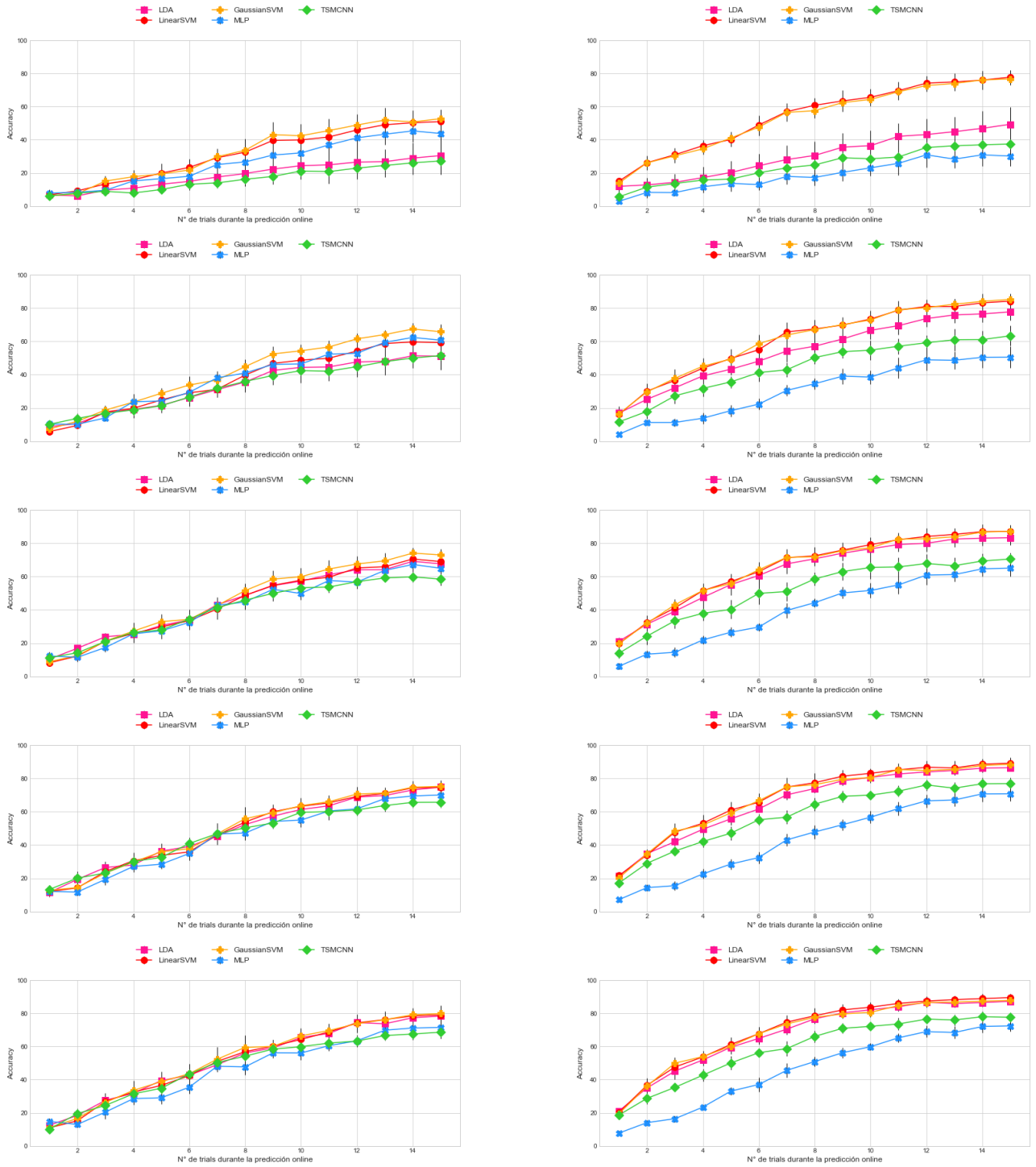


Figura 3.13: Exactitud obtenida para los clasificadores evaluados en los sujetos A y B, en los diferentes trials empleados durante la simulación online. A la izquierda, resultados correspondientes al sujeto A; a la derecha, al B. De la primera a la quinta fila: modelos entrenados con 3, 6, 9, 12 y 15 trials.

3.3.4.3. Evaluación del número de trials en entrenamiento y durante la ejecución online del deletreador

En esta sección, se profundiza el análisis en lo concerniente al número de trials utilizados tanto durante el entrenamiento, como durante la simulación online. Este estudio es fundamental para determinar la cantidad de repeticiones por carácter que resulta conveniente utilizar en la implementación real de la interfaz, para las respectivas sesiones de calibración y ejecución en línea.

Como se ha podido ver en los resultados expuestos en las secciones anteriores, los clasificadores se benefician de un número alto de trials en el entrenamiento y durante la simulación online. Esto se debe al hecho de que, para el primer caso, un mayor número de trials implica, lógicamente, una mayor cantidad de datos entregados al clasificador; lo que favorece el aprendizaje, mejorando la detección de potenciales P300 y permitiendo, luego, obtener mejores resultados en los conjuntos de datos desconocidos (el conjunto de datos de prueba y, ulteriormente, los datos que se adquieran en tiempo real durante la implementación verdadera de la interfaz). En cuanto a la cantidad de trials empleados durante la simulación online, los clasificadores ya entrenados también se benefician de la utilización de un número alto de trials, dado que provee al modelo mayor robustez en la predicción de los caracteres. Esto es así porque, de acuerdo a lo explicado en la sección 1.4, la predicción consiste en la suma de las probabilidades de existencia de un evento P300, obtenidas para cada fila y columna dentro de la matriz de caracteres. En consecuencia, una mayor cantidad de ejemplos por fila y por columna para un mismo carácter permite obtener una probabilidad más certera en su detección y, por ende, una exactitud mayor.

Es decir que, desde un punto de vista práctico, un mayor número de trials en el entrenamiento implica la obtención de un modelo con mayor poder de detección de los eventos P300; lo que se traduce en un desempeño superior en la predicción de caracteres. Ésta, a su vez, se beneficia también de un mayor número de repeticiones durante la ejecución de la prueba online. Sin embargo, no se debe dejar de tener en cuenta al usuario de la interfaz durante estas etapas, fundamentalmente durante la predicción online. Desde el punto de vista del usuario, tanto para la sesión de calibración, como para la ejecución del deletreador, un mayor número de trials se traduce en un mayor tiempo de estimulación por carácter; lo que ocasiona cansancio, fatiga y, muchas veces, frustración, ya que, además, repercute sobre la velocidad de predicción de los caracteres. Las consecuentes fatiga y desmotivación del usuario pueden, incluso, introducir 'ruido' en las señales, dado que su estado de atención se verá afectado, por lo cual la detección del P300 se tornará más complicada y menos acertada, influyendo sobre la detección de caracteres. Como esta interfaz tiene por fin constituir una herramienta alternativa de comunicación, es necesario establecer un balance entre la cantidad de aciertos obtenidos, el tiempo de entrenamiento y de uso de la interfaz, y la velocidad de deletreo, ligada a este último.

3.3.4.3.1 Análisis de la Tasa de Transferencia de Información

De lo analizado en las secciones 3.3.4.3 y 3.3.4.3, resulta que las cantidades de trials más convenientes están comprendidas entre 12 y 15 trials, para el entrenamiento; y, para la predicción online, entre 10 y 15 trials, para todos los clasificadores en general, y 9 y 15 trials para los SVM. En una etapa más profunda de análisis, y a fin de definir no sólo el o los mejores clasificadores, sino también el número de trials óptimo a utilizar durante el entrenamiento y la prueba online, se estudia la Tasa de Transferencia de Información. Ésta, de manera opuesta a la accuracy, que aumenta en relación con el número de trials, toma en consideración el tiempo necesario para el reconocimiento del carácter. Está definida por la ecuación 2.4, calculada de acuerdo a lo explicado en la sección 2.6.2.2.

La figura 3.14 muestra las diferentes curvas de ITR obtenidas por los clasificadores en función del número de trials empleados en la prueba online, para 9, 12 y 15 trials de entrenamiento. Se puede ver que, al igual que en los resultados de predicción de caracteres, los clasificadores presentaron, para el sujeto A, puntajes similares; mientras que, para el B, se observa una gran variabilidad. Es notoria, a su vez, la diferencia en puntajes máximos alcanzados por uno y otro sujeto: de 6 bpm, para el A; y de 10 bpm, para el B. Respecto del número de trials usados durante el entrenamiento, se puede apreciar que, para el sujeto A, los puntajes obtenidos no varían de manera apreciable en base a estos. En cambio, para el sujeto B se observa que, entre 9 y 12 trials de entrenamiento, existe un leve incremento en la ITR de los modelos y que, para 15 trials, este incremento es menos evidente.

En consecuencia, en lo referente al número de trials empleados durante la prueba online, se analizan los resultados obtenidos para los tamaños de entrenamiento de 12 y 15 trials. Se observa que, para el sujeto A, se produjo un leve incremento durante los 4 primeros trials, para luego establecerse una meseta. Para el sujeto B, si bien también se observa un incremento en la ITR durante los tres o cuatro primeros trials, esta tasa, luego, se mantuvo relativamente constante hasta los 8 trials de simulación online; para luego decrecer para mayor cantidad de repeticiones durante esta prueba. Además, los clasificadores que mejores ITR demostraron fueron los lineales. En consecuencia, se decidió estudiar estos tres clasificadores por separado, analizando las ITR obtenidas para tamaños de entrenamiento de 12 y 15 trials, y para el uso de 5, 10 y 15 trials durante la predicción de caracteres.

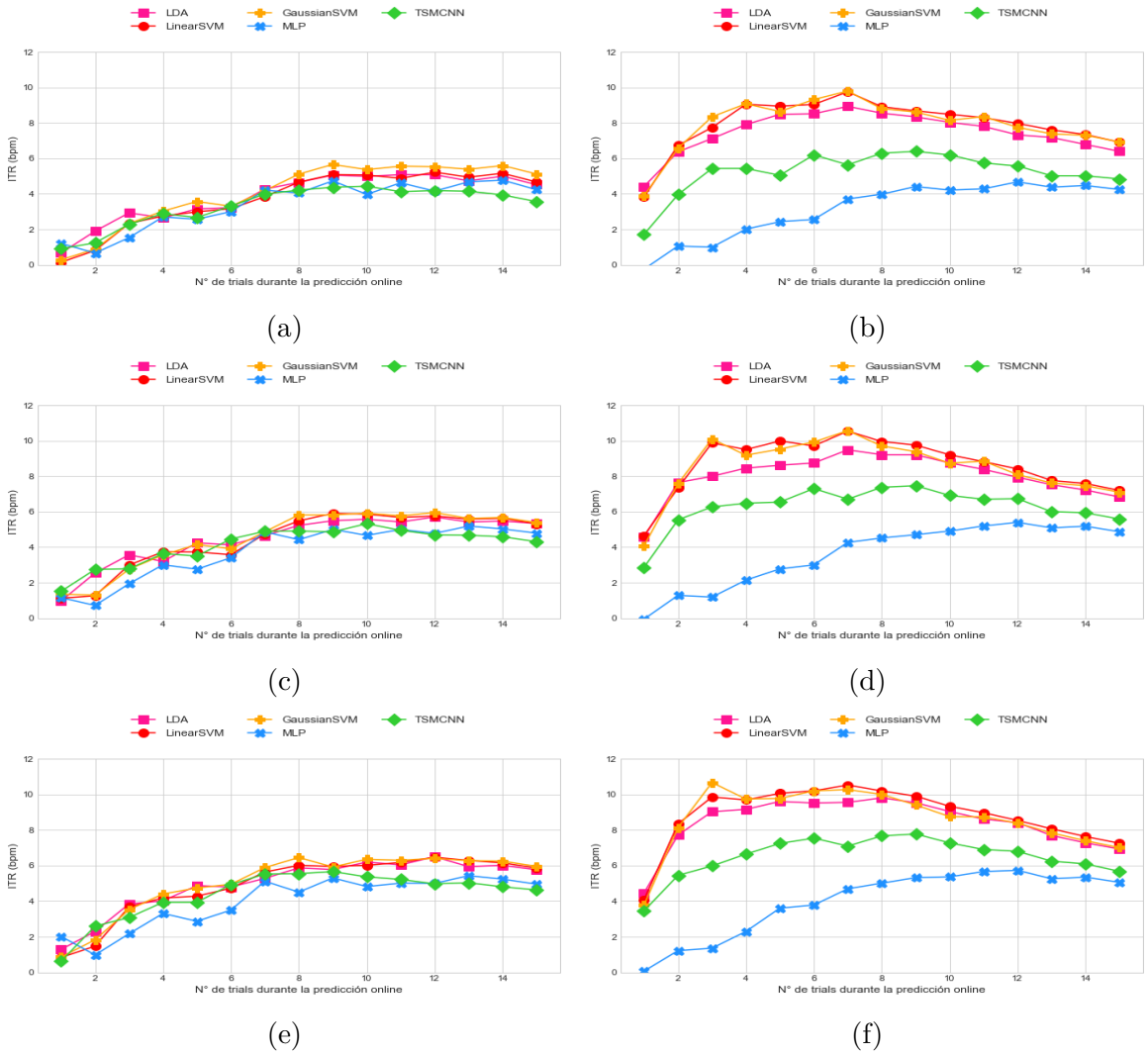


Figura 3.14: Valores de ITR en bits por minuto (bpm) obtenidos por los clasificadores para los diferentes números de trials empleados durante la simulación online. Se comparan los resultados obtenidos para el promedio de los sujetos A y B, para tamaños de entrenamiento de 9, 12 y 15 trials. A la izquierda, resultados correspondientes al sujeto A; a la derecha, al B. (a) y (b), modelos entrenados con 9 trials. (c) y (d), modelos entrenados conre 12 trials; (e) y (f), modelos entrenados con 15 trials.

La tabla 3.6 muestra los porcentajes de aciertos (Acc) y las ITR (en bits por minuto, *bpm*) para los tres clasificadores lineales, discriminando según el sujeto, el número de trials en el entrenamiento y el número de trials usados durante la simulación online. En negrita se resaltan los mejores valores de ITR obtenidos para cada clasificador.

Se puede apreciar que, en efecto, esta métrica penaliza el tiempo necesario para la predicción del carácter. Mientras que para 15 trials de prueba online se obtuvieron las mejores tasas de reconocimiento de caracteres para los tres clasificadores, superiores al 80 %; la ITR, en estos casos, penaliza el tiempo extra precisado en comparación con los 10 y 5 trials, que es de $2{,}1 \cdot 5 = 10{,}5$ y $2{,}1 \cdot 10 = 21$ segundos respectivamente. En consecuencia, si bien para 10 trials de simulación online se obtuvieron porcentajes de exactitud inferiores en aproximadamente un 10 % respecto de los obtenidos para 15 trials, las ITR fueron superiores en más de 1 punto. Por otra parte, al comparar los valores obtenidos para los modelos evaluados con 5 y 10 trials de simulación, se observa que, a pesar de que la diferencia en ITR obtenida es sólo de aproximada-

mente 0.5 puntos, la accuracy fue muy baja para 5 trials, un 20 % menor que para 10, y menor al 50 %. Por ende, no constituye un valor aceptable, a pesar de su alta ITR, y se descarta este tamaño de prueba online.

Tabla 3.6: Comparación de exactitudes e ITR de los clasificadores lineales respecto de los diferentes números de trials en entrenamiento y simulación online[3]

		LDA				LinearSVM				GaussianSVM			
Sujeto	Trials	12 trials		15 trials		12 trials		15 trials		12 trials		15 trials	
		Acc	ITR	Acc	ITR	Acc	ITR	Acc	ITR	Acc	ITR	Acc	ITR
A	5	36,36	4,24	39,45	4,87	33,73	3,73	36,55	4,28	35,91	4,15	38,82	4,74
	10	61,36	5,58	65,45	6,20	63,27	5,86	64,27	6,02	63,64	5,92	66,45	6,35
	15	74,82	5,34	78,36	5,76	74,64	5,31	79,09	5,85	75,27	5,39	79,91	5,96
B	5	55,73	8,62	59,55	9,60	61,00	9,99	61,27	10,06	59,27	9,53	60,18	9,77
	10	80,64	8,75	82,18	9,04	83,09	9,21	83,73	9,33	80,55	8,73	80,64	8,75
	15	86,55	6,83	87,27	6,93	89,18	7,20	89,55	7,25	88,45	7,10	87,82	7,01
Media	5	46,05	6,43	49,50	7,24	47,36	6,86	48,91	7,17	47,59	6,84	49,50	7,26
	10	71,00	7,16	73,82	**7,62**	73,18	7,54	74,00	**7,67**	72,09	7,33	73,55	**7,55**
	15	80,68	6,08	82,82	6,35	81,91	6,26	84,32	6,55	81,86	6,24	83,86	6,48

De acuerdo expuesto al inicio de esta sección, resulta evidente que, tomando en consideración tanto la tasa de reconocimiento de caracteres, como el tiempo demandado por cada uno, se debe establecer una solución de compromiso entre ambas, para lograr determinar el clasificador y los parámetros de diseño del experimento más convenientes a emplear en la aplicación real de la interfaz. Esto se realiza para disminuir la fatiga del usuario y aumentar la velocidad de deletreo, al tiempo que se garantiza una buena tasa de detección de caracteres (indicada por la accuracy); características deseables en una herramienta de comunicación como la constituida por esta interfaz.

Consecuentemente, analizando los resultados resaltados para los clasificadores en la tabla 3.6, se puede concluir que, si bien los tres presentaron similares tasas de transferencia de información, el SVM lineal presentó, en promedio, un valor superior al resto. Sin embargo, es prácticamente despreciable la diferencia entre la ITR de éste, de 7.67 puntos, respecto de la del LDA, de 7.62 puntos. A pesar de esto, si bien podría considerarse que el desempeño de ambos fue prácticamente equivalente, si se tiene en cuenta el proceso de optimización demandado por el SVM lineal, se puede concluir que el LDA es una opción tentadora a considerar, dado que su tasa de ITR sólo se encuentra 0.05 puntos debajo y no requiere proceso de optimización alguno.

Estos valores se dan para el uso de 15 trials durante la etapa de entrenamiento. Si se los compara con los obtenidos para los mismos modelos, pero entrenados con 12 trials, se observa que los valores de accuracy obtenidos disminuyeron en menos de un 5 %, mientras que los de ITR lo hicieron en menos de 0.5 puntos. En este caso, el SVM lineal obtuvo puntajes superiores a los otros dos clasificadores, en la medida en que su accuracy disminuyó en menos del 1 %, y

[3]Accuracy expresada en %, e ITR en bpm. En negrita se resaltan los mejores puntajes de ITR obtenidos.

continuó siendo superior en un 1 % y 2 % al LDA y SVM gaussiano, respectivamente; mientras que su ITR sólo disminuyó en 0.17 puntos y fue 0.4 y 0.2 puntos superior a los mencionados clasificadores. En consecuencia, para la elección de un tamaño de entrenamiento de 12 trials, constituye el mejor clasificador a elegir.

3.3.4.3.2 Análisis de los tiempos de selección del carácter durante la calibración e implementación online y del tiempo de cómputo de los algoritmos lineales

La tabla 3.7 muestra el tiempo de cómputo que demanda el entrenamiento de cada uno de los algoritmos; mientras que la tabla 3.8, muestra el tiempo empleado en la sesión de calibración para la adquisición de los datos. En ambas se discrimina de acuerdo al número de trials empleados en la selección del carácter y se expresa el tiempo en una y cincuenta épocas de caracteres. Se observa que, a medida que aumenta la complejidad del algoritmo clasificador, también lo hace el tiempo de cómputo demandado, por depender directamente del número de operaciones matemáticas que se deben realizar. Mientras que para el LDA se obtuvieron valores de aproximadamente 1 segundo para 12 y 15 trials de entrenamiento; el SVM gaussiano demandó un mayor tiempo de entrenamiento, de 10 y 20 segundos de acuerdo a la cantidad de trials empleados; y el SVM lineal presentó valores intermedios. Respecto del tiempo de calibración, que repercute sobre el cansancio del sujeto, se observa que un tamaño de entrenamiento de 12 trials demanda un tiempo total de 23 minutos, para entrenamientos con 50 caracteres; y, para 15 trials, este tiempo es de 28 minutos; es decir, casi un 20 % superior. Asimismo, la tabla 3.9 expresa el tiempo de selección de carácter en la prueba online para la utilización de 10 y 15 trials, y el correspondiente tiempo que demandaría deletrear una palabra de 5 letras. Se observa que este último es un 30 % menor para la implementación de una interfaz con sólo 10 trials respecto del requerido para un diseño de 15 trials. Esto constituye una importante ganancia en velocidad de deletreo, disminuyendo el cansancio del usuario y aumentando su motivación, al ser más rápida la comunicación.

Tabla 3.7: Tiempo de cómputo para 1 y 50 épocas de caracteres respecto del número de trials en entrenamiento

Modelo	Trials en entrenamiento	Épocas de caracteres 1 (s)	50 (s)
LDA	12	0,016	0,8
	15	0,021	1,05
LinearSVM	12	0,151	7,55
	15	0,252	12,6
GaussianSVM	12	0,198	9,9
	15	0,378	18,9

Tabla 3.8: Tiempo de calibración para 1 y 50 épocas de caracteres respecto del número de trials en entrenamiento

Trials en entrenamiento	Épocas de caracteres 1 (s)	50 (min)
12	27,7	23,08
15	34	28,33

Tabla 3.9: Tiempo de deletreo online para 1 y 5 caracteres respecto del número de trials empleado

Trials	Selección carácter (s)	Deletreo de 5 letras (min)
10	23,5	1,96
15	34	2,83

3.3.4.3.3 Consideraciones de diseño

El análisis expuesto en estas secciones tuvo como fin definir el clasificador y los parámetros de diseño del experimento a aplicar en la implementación futura de la interfaz. Se pudo concluir que los mejores clasificadores, para un tamaño de entrenamiento de 15 trials, fueron el LDA y el SVM lineal; mientras que, para entrenamientos de 12 trials, lo fue el SVM lineal. Si bien la elección final se verá influenciada por otros criterios de diseño de la interfaz donde el clasificador será implementado, de los resultados de este trabajo se pueden hacer las siguientes observaciones: si bien es cierto que, por un lado, el SVM lineal presentó, para modelos entrenados con 12 trials, resultados superiores al resto de los clasificadores, y muy similares a los obtenidos para 15 trials de entrenamiento; también es un hecho que, para 15 trials de entrenamiento, el LDA mostró un desempeño muy cercano al SVM lineal y, además cuenta con la ventaja de su sencillez de entrenamiento e implementación. Se observó, asimismo, que entre 12 y 15 trials de entrenamiento sólo hay $2{,}1 \cdot 3 = 6{,}3$ segundos de diferencia, que se traducen, para 50 caracteres empleados en el entrenamiento, en 6,3 s 50 = 315 s = 5,25 minutos; y que la disminución de 5 trials en la selección del carácter influye en gran medida en la velocidad de deletreo de la interfaz, requiriendo un 30 % menos de tiempo. Luego, de acuerdo a los diferentes criterios de diseño experimental que se adopten, se puede concluir que:

1. Si sólo se desea reducir la duración de la sesión de calibración, al tiempo que se obtiene una alta tasa de predicción de caracteres, será conveniente optar por el SVM lineal, con un tamaño de 12 trials por época de carácter, dando un total de 23 minutos de entrenamiento para 50 épocas de caracteres, y la utilización de 15 trials durante la prueba online.

2. Si se desea, en cambio, reducir la duración del tiempo de selección de caracteres durante el deletreo online, al tiempo que se obtiene una alta exactitud y una ITR elevada, convendrá optar por un número de 10 trials en la prueba online, para cualquiera de los tres

clasificadores. De preferencia, se utilizará el SVM lineal, por ser el que mejor desempeño mostró; o el LDA, dada su sencillez y velocidad de entrenamiento.

3. Si se desea reducir el tiempo de selección del carácter en ambas etapas, se establecerá una solución de compromiso entre ambos, optando por un tamaño de entrenamiento de 12 trials, con 10 trials durante la prueba online. El clasificador más conveniente, en este caso, es el SVM lineal.

La elección de una u otra alternativa responderá a diferentes criterios de diseño, pero en general, se puede realizar un análisis de costo-beneficio que contemple la conveniencia de una u otra alternativa. Si se tiene en cuenta que: 1) la sesión de calibración sólo se realiza al principio, en una etapa de ajuste de la interfaz; 2) puede ser dividida indistintamente en tantas sesiones como se estime necesario; se concluye que resulta conveniente utilizar el LDA como clasificador, con tamaños de entrenamiento de 15 trials. Esto es debido tanto a su sencillez como al comparable desempeño presentado con respecto al SVM lineal. Sin embargo, esto se puede afirmar si se asume que un tiempo mayor de calibración no repercute en gran medida sobre la usabilidad del sistema. Si bien esto es cierto en comparación con el tiempo de selección del carácter durante el deletreo, que hace a la herramienta de comunicación en sí; el tiempo de calibración también es de importancia en la medida en que constituye el primer contacto que establece el usuario con la interfaz, pudiendo tener efectos positivos o negativos sobre su motivación, que repercuten probablemente en la expresión del potencial P300. Además, se deberían considerar aspectos de usabilidad del sistema en la elección del clasificador, tales como la capacidad, voluntad o disponibilidad de cada sujeto de llevar a cabo varias sesiones de calibración.

Conclusiones

Durante el presente trabajo de investigación, se implementaron diversos algoritmos de clasificación de potenciales P300 en señales EEG, tanto de diseño propio como propuestos en la revisión del estado del arte. Se llevó a cabo en dos etapas, de clasificación binaria y simulación de la predicción online, para lo cual se definieron métricas apropiadas en cada caso. El análisis, en la primera etapa, estuvo enfocado en la evaluación del poder de generalización de los clasificadores en la detección de la onda P300 y su desempeño; mientras que, en la segunda etapa, fue realizado tomando siempre en consideración el mejor escenario para una implementación real de la interfaz. Es decir, no se realizó un análisis con rigurosidad científica, en que sólo se busca obtener el máximo desempeño de los algoritmos, sin tener en cuenta la aplicación real de la interfaz y los desafíos que ésta conlleva en cuanto a velocidad de la predicción y fatiga del usuario. El análisis desarrollado, en cambio, permitió definir no sólo los mejores clasificadores a implementar, sino también los parámetros de diseño del experimento, dado que esta investigación constituye una etapa previa a la implementación de una interfaz propia, que sea una herramienta de comunicación y no simplemente una fuente de datos para la investigación.

En consecuencia, se puede concluir que los objetivos planteados al inicio de esta investigación fueron satisfechos, dado que se implementaron y compararon diferentes algoritmos de clasificación dentro del paradigma del deletreador P300, evaluando su diferente desempeño y la posibilidad de reducción de los tiempos de selección de caracteres en la calibración y uso de la aplicación; lo que permitirá continuar la investigación orientada a la implementación de los mejores algoritmos en una interfaz real.

Trabajos Futuros

Si bien durante el desarrollo del trabajo se cumplieron los objetivos propuestos, surgen, al finalizar esta investigación, nuevos objetivos y propuestas a futuro, que permiten dar continuidad a la misión a largo plazo de implementar una interfaz similar a la desarrollada, pero con pictogramas. En particular, para el caso del deletreador P300, se plantean como líneas de investigación a futuro:

1. Determinación del número de caracteres mínimo a utilizar durante la etapa de calibración.
2. Evaluación de nuevas formas de disposición de los caracteres, alternativas a la matriz 6x6 empleada en este trabajo.
3. Extensión del análisis a un mayor número de sujetos, preferentemente con limitaciones motrices severas, a fin de evaluar el desempeño en la población objetivo de la interfaz, que será quien la utilice y se beneficie de ella.
4. Evaluación de la influencia de los diferentes canales de datos EEG en el desempeño de los algoritmos, y determinación de los más convenientes a emplear.
5. Realización de pruebas reales.

Con respecto a la adaptación de la presente interfaz de caracteres a una interfaz con pictogramas, se plantean las siguientes líneas de investigación:

1. Evaluación de la cantidad y disposición de los pictogramas.
2. Evaluación de las estrategias de estimulación del usuario, contemplando la utilización de agrandamiento y contraste variable de las imágenes.
3. Evaluación de la implementación de submenúes de pictogramas, clasificados de acuerdo a temas o situación del usuario, dada la gran cantidad existente.
4. Realización de pruebas reales.

Referencias

[1] Martin Abadi y col. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Brainquiry B.V. *Real-Time Filtering in BioExplorer*. URL: https://www.brainclinics.com/dynamic/media/1/documents/Onderzoeksinstituut/Filtering%20in%20BioExplorer.pdf.

[3] Imad A Basheer y Maha Hajmeer. "Artificial neural networks: fundamentals, computing, design, and application". En: *Journal of microbiological methods* 43.1 (2000), págs. 3-31.

[4] Mark F Bear, Barry W Connors y Michael A Paradiso. *Neuroscience*. Vol. 2. Lippincott Williams & Wilkins, 2007.

[5] B. Blankertz. *Bci dataset for p300 evoked potentials*. 2004. URL: http://www.bbci.de/competition/iii/desc%5C_II.pdf.

[6] Vladimir Bostanov. "BCI competition 2003-data sets Ib and IIb: feature extraction from event-related brain potentials with the continuous wavelet transform and the t-value scalogram". En: *IEEE Transactions on Biomedical engineering* 51.6 (2004), págs. 1057-1061.

[7] Laurent Bougrain, Carolina Saavedra y Radu Ranta. "Finally, what is the best filter for P300 detection?" En: *TOBI Workshop lll-Tools for Brain-Computer Interaction-2012*. 2012.

[8] Jason Brownlee. *A Gentle Introduction to the Bootstrap Method*. 2018. URL: https://iaarbook.github.io/interfaz-cerebro-computadora-BCI/.

[9] Hubert Cecotti y Axel Graser. "Convolutional neural networks for P300 detection with application to brain-computer interfaces". En: *IEEE transactions on pattern analysis and machine intelligence* 33.3 (2011), págs. 433-445.

[10] Hubert Cecotti y col. "A robust sensor-selection method for P300 brain–computer interfaces". En: *Journal of neural engineering* 8.1 (2011), pág. 016001.

[11] François Chollet y col. *Keras*. https://keras.io. 2015.

[12] Rebeca Corralejo Palacios y col. "Decoding P300 evoked potentials for Brain Computer Interfaces (BCI) aimed at assisting potential end-users at home". En: (2016).

[13] Tamer Demiralp y col. "Time–frequency analysis of single-sweep event-related potentials by means of fast wavelet transform". En: *Brain and Language* 66.1 (1999), págs. 129-145.

[14] Lawrence Ashley Farwell y Emanuel Donchin. "Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials". En: *Electroencephalography and clinical Neurophysiology* 70.6 (1988), págs. 510-523.

[15] Elliot Franz. "Development of a fast and efficient algorithm for P300 event related potential detection". Tesis doct. Temple University, 2014.

[16] William F Ganong. *Manual de fisiología médica*. 2010.

[17] P Garcia y col. "Interfaz Cerebro Computador basada en potenciales evocados visuales de estado estacionario: ensayos preliminares". En: *XVIII Congreso Argentino de Bioingenieria SABI 2011*. 2011.

[18] Ian Goodfellow y col. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[19] Galen F Hagen y col. "P3a from visual stimuli: task difficulty effects". En: *International journal of psychophysiology* 59.1 (2006), págs. 8-14.

[20] J. D. Hunter. "Matplotlib: A 2D graphics environment". En: *Computing In Science & Engineering* 9.3 (2007), págs. 90-95. DOI: 10.1109/MCSE.2007.55.

[21] Anil K Jain, Jianchang Mao y K Moidin Mohiuddin. "Artificial neural networks: A tutorial". En: *Computer* 29.3 (1996), págs. 31-44.

[22] Joanna Jarmolowska y col. "A multimenu system based on the P300 component as a time saving procedure for communication with a brain-computer interface". En: *Frontiers in neuroscience* 7 (2013), pág. 39.

[23] Eric Jones, Travis Oliphant, Pearu Peterson y col. *SciPy: Open source scientific tools for Python*. [Online; accessed Oct 2018]. 2001–. URL: http://www.scipy.org/.

[24] Matthias Kaper y col. "BCI competition 2003-data set IIb: support vector machines for the P300 speller paradigm". En: *IEEE Transactions on Biomedical Engineering* 51.6 (2004), págs. 1073-1076.

[25] Nitish Shirish Keskar y Richard Socher. "Improving generalization performance by switching from adam to sgd". En: *arXiv preprint arXiv:1712.07628* (2017).

[26] Nikhil Ketkar y col. *Deep Learning with Python*. Springer, 2017.

[27] Nizar Ali Khemri. "P300 wave detection using a commercial non-invasive EEG sensor: reliability and performance in control applications". Tesis doct. Oklahoma State University, 2006.

[28] Yiannis Kokkinos y Konstantinos G Margaritis. "Managing the computational cost of model selection and cross-validation in extreme learning machines via Cholesky, SVD, QR and eigen decompositions". En: *Neurocomputing* 295 (2018), págs. 29-45.

[29] Dean J Krusienski y col. "A comparison of classification techniques for the P300 Speller". En: *Journal of neural engineering* 3.4 (2006), pág. 299.

[30] Dean J Krusienski y col. "Toward enhanced P300 speller performance". En: *Journal of neuroscience methods* 167.1 (2008), págs. 15-21.

[31] Sourav Kundu y Samit Ari. "P300 Detection with Brain–Computer Interface Application Using PCA and Ensemble of Weighted SVMs". En: *IETE Journal of Research* 64.3 (2018), págs. 406-414.

[32] Mingfei Liu y col. "Deep learning based on Batch Normalization for P300 signal detection". En: *Neurocomputing* 275 (2018), págs. 288-297.

[33] Fabien Lotte. "Study of electroencephalographic signal processing and classification techniques towards the use of brain-computer interfaces in virtual reality applications". Tesis doct. INSA de Rennes, 2008.

[34] Steven J Luck. *An introduction to the event-related potential technique*. MIT press, 2014.

[35] Steven J Luck y Emily S Kappenman. *The Oxford handbook of event-related potential components*. Oxford university press, 2011.

[36] Joseph N Mak y Jonathan R Wolpaw. "Clinical applications of brain-computer interfaces: current state and future prospects". En: *IEEE reviews in biomedical engineering* 2 (2009), pág. 187.

[37] Ran Manor y Amir B Geva. "Convolutional neural network for multi-category rapid serial visual presentation bci". En: *Frontiers in computational neuroscience* 9 (2015), pág. 146.

[38] Federico Lecumberry Martin Patrone Ignacio Ramirez. "Interfaces Cerebro-Computadora". Tesis de mtría. Montevideo: Universidad de la Republica, mayo de 2017.

[39] Francisco José Martínez-Estudillo y col. "Modelo no lineal basado en redes neuronales de unidades producto para clasificación. Una aplicación a la determinación del riesgo en tarjetas de crédito//Non-linear model for classification based on product-unit neural networks. An application to determine credit card risk". En: (2007).

[40] P Meinicke y col. "Improving transfer rates in brain computer interface: a case study. 2002". En: *Neural Information Processing Systems*, págs. 1107-1114.

[41] Anna Caterina Merzagora y col. "Wavelet analysis for EEG feature extraction in deception detection". En: *Engineering in Medicine and Biology Society, 2006. EMBS'06. 28th Annual International Conference of the IEEE*. IEEE. 2006, págs. 2434-2437.

[42] H Mirghasemi, MB Shamsollahi y R Fazel-Rezai. "Assessment of preprocessing on classifiers used in the P300 speller paradigm". En: *Engineering in Medicine and Biology Society, 2006. EMBS'06. 28th Annual International Conference of the IEEE*. IEEE. 2006, págs. 1319-1322.

[43] Aloysio Miranda Moles y Tutor Dr Luis Carlos Silva Ayçaguer. "El método de remuestreo y su aplicación a la investigación biomédica". Tesis doct. Tesis de Especialidad en Bioestadística, Escuela Nacional de Salud Pública Carlos J. Finlay, La Habana, Cuba, 2003.

[44] Joel Monteiro. "Acquisition and processing of EEG signal for neurologic disorders associated with memory". Tesis de mtría. Sep. de 2016. DOI: 10.13140/RG.2.2.25614.77121.

[45] Andreas C Müller, Sarah Guido y col. *Introduction to machine learning with Python: a guide for data scientists*. .ºReilly Media, Inc.", 2016.

[46] Henríquez Muñoz y Claudia Nureibis. "Estudio de Técnicas de análisis y clasificación de senales EEG en el contexto de Sistemas BCI (Brain Computer Interface)". Tesis de mtría. 2014.

[47] Edgar Osuna, Robert Freund y Federico Girosi. "An improved training algorithm for support vector machines". En: *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*. IEEE. 1997, págs. 276-285.

[48] Karim G Oweiss. *Statistical signal processing for neuroscience and neurotechnology*. Academic Press, 2010. Cap. 10.

[49] F. Pedregosa y col. "Scikit-learn: Machine Learning in Python". En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.

[50] Arahal Manuel R. Pérez Valls Jesús y Teodoro Álamo Cantarero. "Herramienta MatLab para la selección de entradas y predicción neuronal de valores de bolsa". Tesis de mtría. 2012.

[51] Fernando Perez y Brian E. Granger. "IPython: a System for Interactive Scientific Computing". En: *Computing in Science and Engineering* 9.3 (mayo de 2007), págs. 21-29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org.

[52] M Pérez y J Luis. "Comunicación con computador mediante señales cerebrales. Aplicación a la tecnología de la rehabilitación". Tesis doct. Ph. D. thesis, Universidad Politécnica de Madrid, 2009.

[53] R Quian Quiroga y col. "Wavelet entropy in event-related potentials: a new method shows ordering of EEG oscillations". En: *Biological cybernetics* 84.4 (2001), págs. 291-299.

[54] Alain Rakotomamonjy y Vincent Guigue. "BCI competition III: dataset II-ensemble of SVMs for BCI P300 speller". En: *IEEE transactions on biomedical engineering* 55.3 (2008), págs. 1147-1154.

[55] Isabel C Ramírez, Carlos Barrera, Juan C Correa Juan C Correa y col. "Efecto del tamaño de muestra y el número de réplicas bootstrap". En: *Ingeniería y Competitividad* 15.1 (2013), págs. 93-101.

[56] Rajesh PN Rao. *Brain-computer interfacing: an introduction*. Cambridge University Press, 2013.

[57] *Redes Neuronales - Funcionamiento Básico*. URL: http://perso.wanadoo.es/alimanya/funcion.htm.

[58] Bertrand Rivet y col. "Impact of spatial filters during sensor selection in a visual P300 brain-computer interface". En: *Brain topography* 25.1 (2012), págs. 55-63.

[59] ADRIAN Rosebrock. "Deep Learning for Computer Vision with Python". En: *New York: Pyimageseach* (2017).

[60] Hongchang Shan, Yu Liu y Todor Stefanov. "A Simple Convolutional Neural Network for Accurate P300 Detection and Character Spelling in Brain Computer Interface." En: *IJCAI*. 2018, págs. 1604-1610.

[61] Steven Smith. *Digital signal processing: a practical guide for engineers and scientists*. Elsevier, 2013.

[62] Enrique J Carmona Suárez. "Tutorial sobre máquinas de vectores soporte (sVM)". En: *Tutorial sobre Máquinas de Vectores Soporte (SVM)* (2014).

[63] Arjon Turnip y col. "P300 detection using a multilayer neural network classifier based on adaptive feature extraction". En: *International Journal of Brain and Cognitive Sciences* 2.5 (2013), págs. 63-75.

[64] Tomas Uktveris y Vacius Jusas. "Application of convolutional neural networks to four-class motor imagery classification problem". En: *Information Technology And Control* 46.2 (2017), págs. 260-273.

[65] Bastian Venthur. "Design and implementation of a brain computer interface system". En: (2015).

[66] Bastian Venthur y Benjamin Blankertz. "Wyrm, A Pythonic Toolbox for Brain-Computer Interfacing". En: *arXiv preprint arXiv:1412.6378* (2014).

[67] Sergio René Vivar Vera y col. "Desarrollo de una BCI utilizando el potencial P300 y la diadema Mindwave®." En: *Research in Computing Science* 140 (2017), págs. 151-164.

[68] Ashia C Wilson y col. "The marginal value of adaptive gradient methods in machine learning". En: *Advances in Neural Information Processing Systems*. 2017, págs. 4148-4158.

[69] Jonathan R Wolpaw y col. “Brain-computer interface technology: a review of the first international meeting”. En: *IEEE transactions on rehabilitation engineering* 8.2 (2000), págs. 164-173.

[70] Neng Xu y col. “BCI competition 2003-data set IIb: enhancing P300 wave detection using ICA-based subspace projections for BCI applications”. En: *IEEE transactions on biomedical engineering* 51.6 (2004), págs. 1067-1072.

[71] Chenghai Yang y col. “Evaluating unsupervised and supervised image classification methods for mapping cotton root rot”. En: *Precision Agriculture* 16.2 (2015), págs. 201-215.

[72] Jieping Ye. “Least squares linear discriminant analysis”. En: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, págs. 1087-1093.

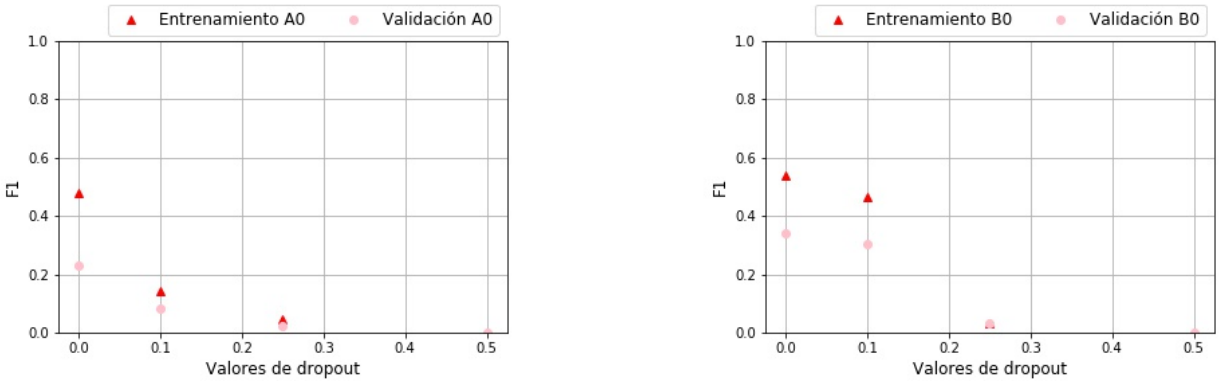
[73] Erwei Yin y col. “A speedy hybrid BCI spelling approach combining P300 and SSVEP”. En: *IEEE Transactions on Biomedical Engineering* 61.2 (2014), págs. 473-483.

[74] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. Chapman y Hall/CRC, 2012.

Apéndice A

Evaluación de diferentes valores de dropout para los MLPs

A.1. MLP_01

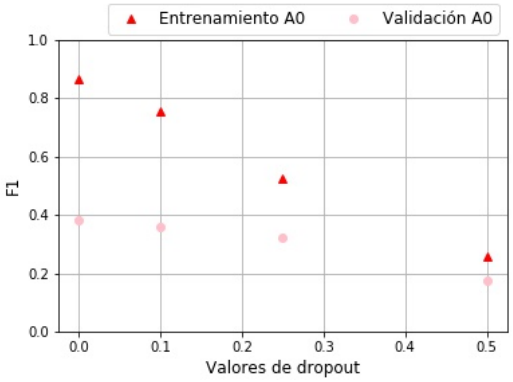


(a) Sujeto A

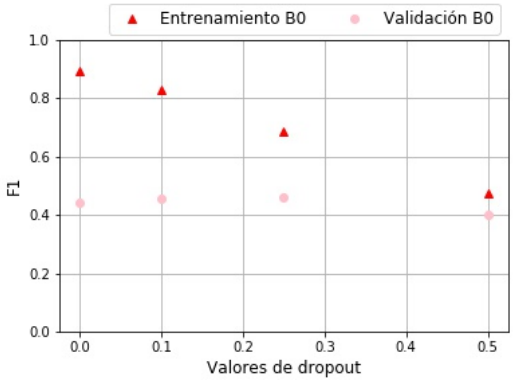
(b) Sujeto B

Figura A.1: Resultados obtenidos en la evaluación de diferentes valores de dropout para la arquitectura MLP_01. (a) Sujeto A. (b) Sujeto B.

A.2. MLP_02



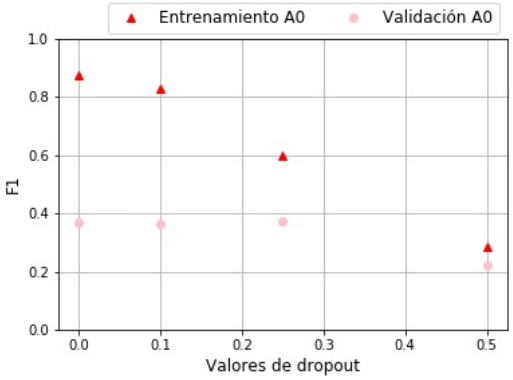
(a) Sujeto A



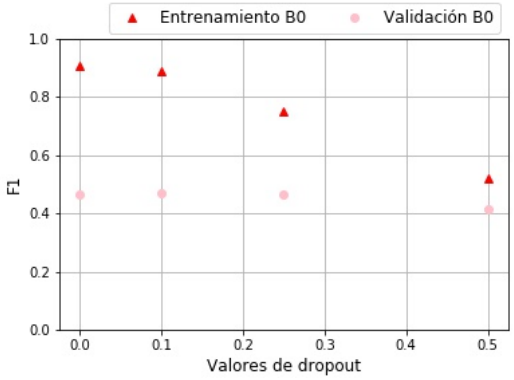
(b) Sujeto B

Figura A.2: Resultados obtenidos en la evaluación de diferentes valores de dropout para la arquitectura MLP_02. (a) Sujeto A. (b) Sujeto B.

A.3. MLP_03



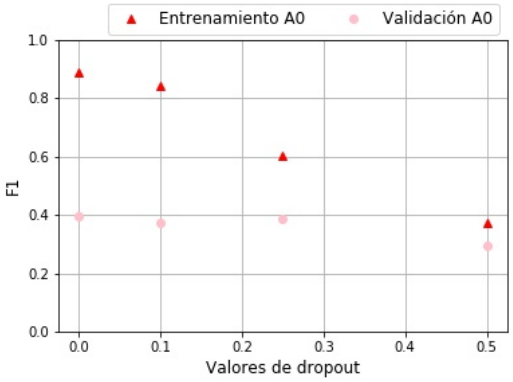
(a) Sujeto A



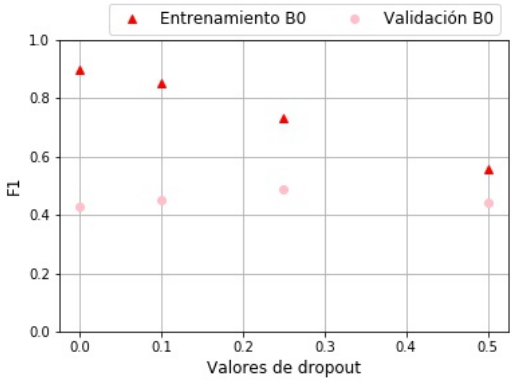
(b) Sujeto B

Figura A.3: Resultados obtenidos en la evaluación de diferentes valores de dropout para la arquitectura MLP_03. (a) Sujeto A. (b) Sujeto B.

A.4. MLP_04



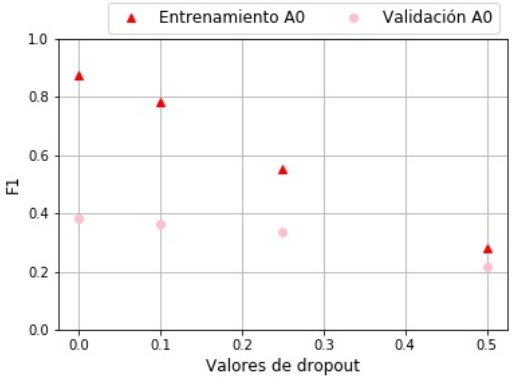
(a) Sujeto A



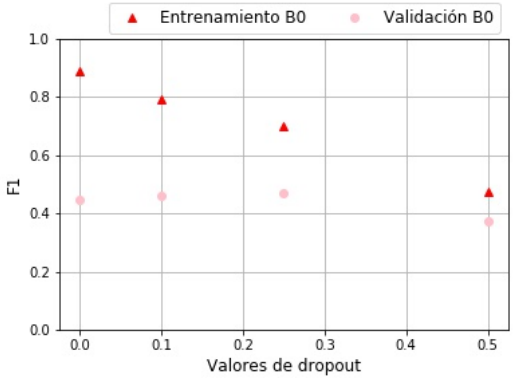
(b) Sujeto B

Figura A.4: Resultados obtenidos en la evaluación de diferentes valores de dropout para la arquitectura MLP_04. (a) Sujeto A. (b) Sujeto B.

A.5. MLP_05



(a) Sujeto A



(b) Sujeto B

Figura A.5: Resultados obtenidos en la evaluación de diferentes valores de dropout para la arquitectura MLP_05. (a) Sujeto A. (b) Sujeto B.

A.6. Comparación de las arquitecturas

Tabla A.1: Resultados de validación obtenidos para las diferentes arquitecturas construidas.

Modelo	Sujeto	recall	precision	F1	acc	TP	TN	FN	FP
MLP_01	A_0	0,79	0,24	0,37	0,53	624	1798	166	2002
	B_0	0,78	0,25	0,38	0,58	580	2079	163	1768
MLP_02	A_0	0,75	0,25	0,38	0,60	558	2218	182	1632
	B_0	0,82	0,32	0,46	0,68	622	2486	133	1349
	B_0	0,84	0,33	0,47	0,69	639	2518	121	1312
MLP_04	A_0	0,74	0,28	0,40	0,65	538	2448	188	1416
	B_0	0,76	0,35	0,48	0,72	581	2731	181	1097
MLP_05	A_0	0,82	0,23	0,36	0,51	639	1687	142	2122
	B_0	0,87	0,24	0,38	0,53	664	1749	98	2079
MLP_06	A_0	0,85	0,23	0,36	0,48	667	1559	122	2242
	B_0	0,86	0,25	0,38	0,56	629	1935	105	1921

Apéndice B

Código fuente

Código B.1: Cargado de la base de datos y generación de sujetos bootstrap

```
from __future__ import division

import sys
import time
import logging

import numpy as np
from scipy.io import loadmat
from os import path

import wyrm.processing as proc
from wyrm import io
from wyrm.types import Data
import pickle

from sklearn.externals import joblib
from os import listdir
from os.path import isfile, join

# replay the experiment in real time?
REALTIME = False

# TODO: Change file paths
TRAIN_DATA_A = 'BCI_Comp_III_Wads_2004/data/Subject_A_Train.mat'
TEST_DATA_A = 'BCI_Comp_III_Wads_2004/data/Subject_A_Test.mat'

TRAIN_DATA_B = 'BCI_Comp_III_Wads_2004/data/Subject_B_Train.mat'
TEST_DATA_B = 'BCI_Comp_III_Wads_2004/data/Subject_B_Test.mat'


CHANNEL_DATA = 'BCI_Comp_III_Wads_2004/data/eloc64.txt'

TRUE_LABELS_A = 'WQXPLZCOMRKO97YFZDEZ1DPI9NNVGRQDJCUVRMEUOOOJD2UF\
YPOO6J7LDGYEGOA5VHNEHBTXOO1TDOILUEE5BFAEEXAW_K4R3MRU'
TRUE_LABELS_B = 'MERMIROOMUHJPXJOHUVLEORZP3GLOO7AUFDKEFTWEOOALZOP9R\
OCGZET1Y19EWX65QUYU7NAK_4YCJDVDNGQXODBEV2B5EFDIDNR'

STIMULUS_CODE = {
    # cols from left to right
    1 : "agmsy5",
```

```
    2 : "bhntz6",
    3 : "ciou17",
    4 : "djpv28",
    5 : "ekqw39",
    6 : "flrx4_",
    # rows from top to bottom
    7 : "abcdef",
    8 : "ghijkl",
    9 : "mnopqr",
    10: "stuvwx",
    11: "yz1234",
    12: "56789_"
}

MARKER_DEF_TRAIN = {'target': ['target'], 'nontarget': ['nontarget']}
MARKER_DEF_TEST = {i : [i] for i in STIMULUS_CODE.values()}

SEG_IVAL = [0, 800]

def load_bci_data(filename, ch_ival):
    ### ADAPTED FROM Wyrms load_bcicomp3_ds2
    """Load the BCI Competition III Data Set 2.
    This method loads the data set and converts it into Wyrm's ``Data``
    format. Before you use it, you have to download the data set in
    Matlab format and unpack it. The directory with the extracted files
    must contain the ``Subject_*.mat``- and the ``eloc64.txt`` files.
    .. note::
        If you need the true labels of the test sets, you'll have to
        download them separately from
        http://bbci.de/competition/iii/results/index.html#labels
    Parameters
    ----------
    filename : str
        The path to the matlab file to load
    Returns
    -------
    cnt : continuous `Data` object
    Examples
    --------
    >>> dat = load_bcicomp3_ds2('/home/foo/data/Subject_A_Train.mat')
    """
    STIMULUS_CODE = {
        0 : "blankMatrix",
        # cols from left to right
        1 : "agmsy5",
        2 : "bhntz6",
        3 : "ciou17",
        4 : "djpv28",
        5 : "ekqw39",
        6 : "flrx4_",
        # rows from top to bottom
        7 : "abcdef",
        8 : "ghijkl",
        9 : "mnopqr",
        10: "stuvwx",
        11: "yz1234",
        12: "56789_"
        }
```

```
    # load the matlab data
    data_mat = loadmat(filename)
    # load the channel names (the same for all datasets
    eloc_file = path.sep.join([path.dirname(filename), 'eloc64.txt'])
    with open(eloc_file) as fh:
        data = fh.read()
    channels = []
    for line in data.splitlines():
        if line:
            chan = line.split()[-1]
            chan = chan.replace('.', '')
            channels.append(chan)
    # fix the channel names, some letters have the wrong capitalization
    for i, s in enumerate(channels):
        s2 = s.upper()
        s2 = s2.replace('Z', 'z')
        s2 = s2.replace('FP', 'Fp')
        channels[i] = s2
    # The signal is recorded with 64 channels, bandpass filtered
    # 0.1-60Hz and digitized at 240Hz. The format is Character Epoch x
    # Samples x Channels
    data = data_mat['Signal'][ch_ival[0]:ch_ival[1],:,:]
    data = data.astype('double')

    # For each sample: 1 if a row/colum was flashed, 0 otherwise
    flashing = data_mat['Flashing'][ch_ival[0]:ch_ival[1],:]
    flashing = flashing.reshape(-1)

    tmp = []
    for i, _ in enumerate(flashing):
        if i == 0:
            tmp.append(flashing[i])
            continue
        if flashing[i] == flashing[i-1] == 1:
            tmp.append(0)
            continue
        tmp.append(flashing[i])
    flashing = np.array(tmp)
    # For each sample: 0 when no row/colum was intensified,
    # 1..6 for intensified columns, 7..12 for intensified rows
    stimulus_code = data_mat['StimulusCode'][ch_ival[0]:ch_ival[1],:].reshape(-1)
#     print('stim_code: ', stimulus_code.shape)
    stimulus_code = stimulus_code[flashing == 1]
    # 0 if no row/col was intensified or the intensified did not contain
    # the target character, 1 otherwise

    fs = 240
    data = data.reshape(-1, 64)
    timeaxis = np.linspace(0, data.shape[0] / fs * 1000,
     data.shape[0], endpoint=False)
    dat = Data(data=data, axes=[timeaxis, channels],
    names=['time', 'channel'], units=['ms', '#'])
    dat.fs = fs


    try:
        stimulus_type = data_mat['StimulusType'][ch_ival[0]:ch_ival[1],:].reshape(-1)
```

```
        target_mask = np.logical_and((flashing == 1),
         (stimulus_type == 1)) if len(stimulus_type) > 0 else []
        nontarget_mask = np.logical_and((flashing == 1),
        (stimulus_type == 0)) if len(stimulus_type) > 0 else []
        targets = [[i, 'target'] for i in timeaxis[target_mask]]
        nontargets = [[i, 'nontarget'] for i in timeaxis[nontarget_mask]]

        # The target characters
        target_chars = data_mat.get('TargetChar')[0][ch_ival[0]:ch_ival[1]]
    except KeyError:
        targets = []
        nontargets = []
        pass

    # preparing the markers
    dat.stimulus_code = stimulus_code[:]
    stim = []
    flashing = (flashing == 1)
    flashing = [[i, 'flashing'] for i in timeaxis[flashing]]
    for i,_ in enumerate(flashing):
        stim.append([flashing[i][0], STIMULUS_CODE[stimulus_code[i]]])
    stimulus_code = stim


    markers = flashing[:]
    markers.extend(targets)
    markers.extend(nontargets)
    markers.extend(stimulus_code)
    markers.sort()
    dat.markers = markers[:]

    return dat

def load_boostrapped_data(file, ch_ival):
    """ Adapted from load_bcicomp3_ds2
    Auxiliary function for generating bootstrapping
    subjects.
    """
    STIMULUS_CODE = {
        0 : "blankMatrix",
        # cols from left to right
        1 : "agmsy5",
        2 : "bhntz6",
        3 : "ciou17",
        4 : "djpv28",
        5 : "ekqw39",
        6 : "flrx4_",
        # rows from top to bottom
        7 : "abcdef",
        8 : "ghijkl",
        9 : "mnopqr",
        10: "stuvwx",
        11: "yz1234",
        12: "56789_"
        }

    # load the matlab data
    data_mat = loadmat(filename)
```

```
# load the channel names (the same for all datasets
eloc_file = path.sep.join([path.dirname(filename), 'eloc64.txt'])
with open(eloc_file) as fh:
    data = fh.read()
channels = []
for line in data.splitlines():
    if line:
        chan = line.split()[-1]
        chan = chan.replace('.', '')
        channels.append(chan)
# fix the channel names, some letters have the wrong capitalization
for i, s in enumerate(channels):
    s2 = s.upper()
    s2 = s2.replace('Z', 'z')
    s2 = s2.replace('FP', 'Fp')
    channels[i] = s2
# The signal is recorded with 64 channels, bandpass filtered
# 0.1-60Hz and digitized at 240Hz. The format is Character Epoch x
# Samples x Channels
data = data_mat['Signal'][ch_ival[0]:ch_ival[1],:,:]
data = data.astype('double')

# For each sample: 1 if a row/colum was flashed, 0 otherwise
flashing = data_mat['Flashing'][ch_ival[0]:ch_ival[1],:]
flashing = flashing.reshape(-1)
##Creates an array where only the initial intensifications of each series appear
tmp = []
for i, _ in enumerate(flashing):
    if i == 0:
        tmp.append(flashing[i])
        continue
    if flashing[i] == flashing[i-1] == 1:
        tmp.append(0)
        continue
    tmp.append(flashing[i])
flashing = np.array(tmp)
# For each sample: 0 when no row/colum was intensified,
# 1..6 for intensified columns, 7..12 for intensified rows
stimulus_code = data_mat['StimulusCode'][ch_ival[0]:ch_ival[1],:].reshape(-1)
stimulus_code = stimulus_code[flashing == 1]
# 0 if no row/col was intensified or the intensified did not contain
# the target character, 1 otherwise

fs = 240
data = data.reshape(-1, 64)
timeaxis = np.linspace(0, data.shape[0] / fs * 1000,
data.shape[0], endpoint=False)
dat = Data(data=data, axes=[timeaxis, channels],
names=['time', 'channel'], units=['ms', '#'])
dat.fs = fs

try:
    stimulus_type = data_mat['StimulusType'][ch_ival[0]:ch_ival[1],:].reshape(-1)
    target_mask = np.logical_and((flashing == 1),
    (stimulus_type == 1)) if len(stimulus_type) > 0 else []
    nontarget_mask = np.logical_and((flashing == 1),
    (stimulus_type == 0)) if len(stimulus_type) > 0 else []
    targets = [[i, 'target'] for i in timeaxis[target_mask]]
```

```
        nontargets = [[i, 'nontarget'] for i in timeaxis[nontarget_mask]]

        # The target characters
        target_chars = data_mat.get('TargetChar')[0][ch_ival[0]:ch_ival[1]]
    except KeyError:
        targets = []
        nontargets = []
        pass

    # preparing the markers
    dat.stimulus_code = stimulus_code[:]
    stim = []
    flashing = (flashing == 1)
    flashing = [[i, 'flashing'] for i in timeaxis[flashing]]
    for i,_ in enumerate(flashing):
        stim.append([flashing[i][0], STIMULUS_CODE[stimulus_code[i]]])
    stimulus_code = stim


    markers = flashing[:]
    markers.extend(targets)
    markers.extend(nontargets)
    markers.extend(stimulus_code)
    markers.sort()
    dat.markers = markers[:]

    return dat

def get_bootstrap_indexes(dat, n_ch, subject): # generate the indexes
    n_samples = np.arange(n_ch)
    indexes= {}
    for i in range(10): #number of subjects to generate
        #generate random pool of samples of same length as original dataset
        x = np.random.choice(n_samples.ravel(), size=len(n_samples),
         replace=True)
        #get its unique values indexes
        _, idx = np.unique(x, return_index=True)
        s = subject+'_'+str(i+1)
        indexes[s] = idx
    for idx in indexes.keys():
        print('Number of samples for', idx,':\t ',indexes[idx].shape)
    return indexes
def bootstrap_data(dat, idx_array):

    """Loads the data and applies bootstrapping technique
    to generate 10 new subjects.
    """
    data = dat.data.reshape(85,-1)[idx_array]
    data = data.reshape(-1,64)
    timeaxis = dat.axes[0].reshape(85,-1)[idx_array].reshape(-1)
    markers = [[i,j] for i,j in dat.markers if i in timeaxis]
    axes = dat.axes[:]
    axes[0] = timeaxis
    d = dat.copy(data=data, axes=axes)
    d.markers = markers
    dat.indexes = idx_array
    return d
```

```
# benchmark subjuects
train_A = load_bci_data(TRAIN_DATA_A, ch_ival=[0,None])
train_B = load_bci_data(TRAIN_DATA_B, ch_ival=[0,None])

# bootstrapping subjects
indexes_A = get_bootstrap_indexes(train_A, 85, 'A')
SubjectsData_A = {'A_0': train_A}
print()
for i in range(len(indexes_A)):
    print('Loading subject number ', i+1)
    subject = 'A_'+str(i+1)
    SubjectsData_A[subject] = bootstrap_data(train_A, indexes_A[subject])
with open('Data/train_data_subjs_A.pkl', 'wb') as f:
    for s in SubjectsData_A.keys():
        pickle.dump(SubjectsData_A[s] , f)
indexes_B = get_bootstrap_indexes(train_B, 85, 'B')
SubjectsData_B = {'B_0': train_B}
print()
for i in range(len(indexes_B)):
    print('Loading subject number ', i+1)
    subject = 'B_'+str(i+1)
    SubjectsData_B[subject] = bootstrap_data(train_B, indexes_B[subject])

# Test data
test_A = load_bci_data(TEST_DATA_A, ch_ival=[None,None])
test_B = load_bci_data(TEST_DATA_B, ch_ival=[None,None])
#labels A and B
labels_A = TRUE_LABELS_A
labels_B = TRUE_LABELS_B
test_A.labels = TRUE_LABELS_A
test_B.labels = TRUE_LABELS_B

with open('Data/test_data_subjs_AB.pkl', 'wb') as f:
    pickle.dump([test_A, test_B], f)
```

Código B.2: Optimizacioón SVM Gausiano

```
import scipy.io as spio
import pickle
from sklearn.externals import joblib
import numpy as np
import scipy as sp

from os import path
from scipy.io import loadmat
from wyrm.types import Data
from wyrm import processing as proc

from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler

def preprocessing_simple(dat, MRK_DEF, *args, **kwargs):
    """Simple preprocessing that reaches 97% accuracy.
    """
    fs_n = dat.fs / 2
```

```
    b, a = proc.signal.butter(5, [10 / fs_n], btype='low')
    dat = proc.filtfilt(dat, b, a)

    dat = proc.subsample(dat, 20)
    epo = proc.segment_dat(dat, MRK_DEF, SEG_IVAL)
    fv = proc.create_feature_vectors(epo)
    return fv, epo

def load_grid_gaussianSVM():
    clf = SVC(kernel = 'rbf', probability=True)
    scoring = ['f1', 'precision', 'recall', 'roc_auc', 'accuracy']
    C_range = np.linspace(start=1000, stop=10000, num=4, endpoint = True)
    gamma_range = np.linspace(start=0.001, stop=0.1, num=3, endpoint=True)
    param_grid = dict(C = C_range, gamma = gamma_range)
    grid = GridSearchCV(clf, param_grid = param_grid, scoring = scoring,
     cv = 3, refit = 'f1', verbose = 42, n_jobs=-1)
    return grid

def load_grid_linearSVM():
    clf = SVC(kernel = 'linear', probability=True)
    scoring = ['f1', 'precision', 'recall', 'roc_auc', 'accuracy']
    C_range = np.linspace(start=1000, stop=10000, num=4, endpoint = True)
    param_grid = dict(C = C_range)
    grid = GridSearchCV(clf, param_grid = param_grid, scoring = scoring,
    cv = 3, refit = 'f1', verbose = 42, n_jobs=-1)
    return grid

def perform_gridSearchCV(X, y):
    grid = load_grid_gaussianSVM()
    grid.fit(X, y)
    results = grid.cv_results_
    best_params = grid.best_params_
    best_score = grid.best_score_
    print("The best parameters are %s with a score of %0.2f"
      %(best_params, best_score))
    return grid.best_estimator_, best_params, best_score

def get_metrics(clf, X_valid, y_valid):
    metrics_dict = {}
    pred_class = clf.predict(X_valid)
    metrics_dict['recall'] = metrics.recall_score(y_valid, pred_class)
    metrics_dict['precision'] = metrics.precision_score(y_valid, pred_class)
    metrics_dict['f1'] = metrics.f1_score(y_valid, pred_class)
    metrics_dict['acc'] = metrics.accuracy_score(y_valid, pred_class)
    tn, fp, fn, tp = metrics.confusion_matrix(y_valid, pred_class).ravel()
    metrics_dict['tp'] = tp
    metrics_dict['tn'] = tn
    metrics_dict['fp'] = fp
    metrics_dict['fn'] = fn
    return metrics_dict

def prepare_data(data, subject):
    fv_train, epo = preprocessing_simple(data, MARKER_DEF_TRAIN, SEG_IVAL)
    scaler = StandardScaler()
    X = scaler.fit_transform(fv_train.data) #traininig samples
    y = fv_train.axes[0] #class labels
    X_train, X_valid, y_train, y_valid = train_test_split(X, y,
     test_size = 0.3, train_size = 0.7)
```

```
#     filename = 'Scalers SVM/SVMScaler_'+subject+'.pkl'
#     joblib.dump(scaler, filename)
    return X_train, X_valid, y_train, y_valid

SubjectsData_A = {}
filename = '../Data/train_data_subjs_A.pkl'
with open(filename, "rb") as f:
    for i in range(11):
        subject = 'A_'+str(i)
        SubjectsData_A[subject] = pickle.load(f)
SubjectsData_B = {}
filename = '../Data/train_data_subjs_B.pkl'
with open(filename, "rb") as f:
    for i in range(11):
        subject = 'B_'+str(i)
        SubjectsData_B[subject] = pickle.load(f)

# preprocessing and feature extraction
data = SubjectsData_A['A_0']
fv_train, epo = preprocessing_simple(data, MARKER_DEF_TRAIN, SEG_IVAL)
X = fv_train.data #traininig samples
y = fv_train.axes[0] #class labels
X_train, X_valid, y_train, y_valid = train_test_split(X, y,
test_size = 0.3, train_size = 0.7)

C_range = np.linspace(start=1000, stop=10000, num=4, endpoint = True)
gamma_range = np.linspace(start=0.001, stop=0.1, num=3, endpoint=True)


subjects_dict = {'A': SubjectsData_A, 'B': SubjectsData_B}

for subjects in subjects_dict.keys():
    resultsFile ='GaussianSVM_'+subjects+'subjects.csv'
    columnNames = 'Subject,C,gamma,cvScore,recall,precision,\
    f1,acc,tp,tn,fp,fn,n_sv,n_alpha,alpha=C'
    with open(resultsFile, 'w') as f:
        f.write(columnNames)
    conf_matrices = {}
    for subject in subjects_dict[subjects].keys():
        data = subjects_dict[subjects][subject]
        X_train, X_valid, y_train, y_valid = prepare_data(data, subject)

        clf , best_params, best_score = perform_gridSearchCV(X_train, y_train)
        gamma = best_params['gamma']
        C = best_params['C']
        name_params = 'GaussianSVM_C='+str(C)+'-g='+str(gamma)
        filename = 'Trained_Classifiers/'+name_params+'_'+subject+'.pkl'
        joblib.dump(clf, filename)

        # Support vectors and lagrange multipliers
        support_vectors = clf.support_vectors_
        n_sv = clf.n_support_
        alpha = clf.dual_coef_
        n_alpha = alpha[0].size
        alpha_eqC = np.count_nonzero(abs(alpha[0]) == C)

        filename = 'Trained_Classifiers/Support_Vectors_and_Lagrange\
        Multipliers/GaussianSV_LM_'+subject+'.pkl'
```

```
        joblib.dump([support_vectors, alpha], filename)
        metrics_, cm = get_metrics(clf, X_valid, y_valid)
        conf_matrices[subject] = cm
        resData = '\n'+subject+','+str(C)+','+str(gamma)+','+str(best_score)
        for m in metrics_.keys():
            resData += ','+str(metrics_[m])
        resData += ','+str(n_sv)+','+str(n_alpha)+','+str(alpha_eqC)
            with open(resultsFile, 'a') as f:
                f.write(resData)
```

Código B.3: Entrenamiento LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn import metrics
from wyrm.types import Data
from wyrm import processing as proc
from scipy.io import loadmat
from wyrm.types import Data
import pickle
from sklearn.externals import joblib
from os import path
import numpy as np
import scipy as sp

def crop_nTrials(dat, trial, n_ch):
    n_s = 504 #number of samples per trial
    samples_per_ntrials = {'3': 3*n_s, '6': 6*n_s, '9': 9*n_s, '12': 12*n_s}
    n = samples_per_ntrials[str(trial)]
    data = dat.data.reshape(n_ch,-1,64)[:,:n,:].reshape(-1,64)
    timeaxis = dat.axes[0].reshape(n_ch,-1)[:,:n].reshape(-1)
    markers = [[i,j] for i,j in dat.markers if i in timeaxis]
    axes = dat.axes[:]
    axes[0] = timeaxis
    dat = dat.copy(data=data, axes=axes)
    dat.markers = markers
    return dat

def train(dat):
    # Low-pass filtering
    fs_n = dat.fs / 2
    b, a = proc.signal.butter(5, [10 / fs_n], btype='low')
    dat = proc.lfilter(dat, b, a)

    # Subsampling
    dat = proc.subsample(dat, 20)

    # Epoch segmentation
    epo = proc.segment_dat(dat, MARKER_DEF_TRAIN, SEG_IVAL)

    # Feature extraction
    fv = proc.create_feature_vectors(epo)
    X = fv.data
    y = fv.axes[0]

    # Splitting training set into training and validation set with a 9:1 ratio
    X_train, X_valid, y_train, y_valid = train_test_split(X, y,
    test_size = 0.1, train_size = 0.9)
```

```
    # Training the classifier
    clf = LinearDiscriminantAnalysis(solver='eigen')
    clf.fit(X_train,y_train)
    return clf, X_train, y_train, X_valid, y_valid

SUBJECTS = {'A': SubjectsData_A, 'B': SubjectsData_B}
trials = [3, 6, 9, 12, 15]
for s in SUBJECTS.keys():
    subject_dict = SUBJECTS[s]
    # Open validation results file
    valResultsFile = 'Trained_Classifiers/LDA/ValResultsLDA_'+s+'subjects.csv'
    columnNames = 'Subject,Trial,recall,precision,f1,acc,tp,tn,fp,fn'
    with open(valResultsFile,'w') as f:
        f.write(columnNames)

    # Open training results file
    trainResultsFile = 'Trained_Classifiers/LDA/TrainResultsLDA_'+s+'subjects.csv'
    with open(trainResultsFile, 'w') as f:
        f.write(columnNames)
    for subject in subject_dict.keys():
        print('\n###############################################################################')
        print('                              Subject ', subject)
        print('###############################################################################')

        #prepare data
        dat = subject_dict[subject]
        n_ch = dat.data.reshape(-1,7794,64).shape[0]
        for t in trials:
            if t==15:
                data = dat
            else:
                data = crop_nTrials(dat, t, n_ch)

            clf, X_train, y_train, X_valid, y_valid = train(data)
            filename = 'Trained_Classifiers/LDA/'+subject+'_'+str(t)+\
            'Trials_LDAclf.joblib'
            joblib.dump(clf, filename)


            # get and save validation metrics
            metrics_ = get_metrics(clf, X_valid, y_valid)
            resData = '\n'+subject+','+str(t)
            for m in metrics_.keys():
                resData += ','+str(metrics_[m])
            with open(valResultsFile, 'a') as f:
                f.write(resData)

            # get and save training metrics
            metrics_ = get_metrics(clf, X_train, y_train)
            resData = '\n'+subject+','+str(t)
            for m in metrics_.keys():
                resData += ','+str(metrics_[m])
            with open(trainResultsFile, 'a') as f:
                f.write(resData)
```

Código B.4: Arquitectura redes convolucionales

```
import tensorflow as tf
from keras.utils import to_categorical
from keras.models import model_from_json
import keras
from keras.initializers import glorot_uniform
from keras.layers import Activation
from keras import backend as K
from keras.utils.generic_utils import get_custom_objects
from tensorflow import keras
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation, Flatten, Input
from keras.layers.merge import concatenate
from keras.layers.convolutional import Conv2D
from keras.wrappers.scikit_learn import KerasClassifier


class CustomSigmoid(Activation):

    def __init__(self, activation, **kwargs):
        super(CustomSigmoid, self).__init__(activation, **kwargs)
        self.__name__ = 'customSigmoid'


def customSigmoid(x):
    return (K.tanh(2/3*x) * 1.7159)

get_custom_objects().update({'customSigmoid': CustomSigmoid(customSigmoid)})

class CCNN:
    @staticmethod
    def build(height, width, depth):

        model = Sequential()
        inputShape = (height, width, depth)

        # Layer 1: CONV => Custom_Sigmoid
        model.add(Conv2D(10, (1,64), input_shape=inputShape, name='L1'))
        model.add(Activation('customSigmoid'))

        # Layer 2: CONV => Custom_Sigmoid
        model.add(Conv2D(50, (13,1), name='L2'))
        model.add(Activation('customSigmoid'))

        # Layer 3: FC => Sigmoid
        model.add(Flatten())
        model.add(Dense(100, activation = 'sigmoid', name='L3'))

        # Output layer: softmax classifier
        model.add(Dense(2, activation = 'softmax', name='OutputLayer'))

        # return the constructed network architecture
        return model

class OCLNN:
    @staticmethod
    def build(height, width, depth):
        model = Sequential()
```

```
        inputShape = (height, width, depth)

        # Layer 1: CONV => Custom_Sigmoid
        #N = Ts * Fs = 800 ms * 120 Hz = 96
        N = 96
        model.add(Conv2D(16, (int(N/15),64), strides=int(N/15),
        input_shape=inputShape, activation='relu', name='relu'))
        model.add(Dropout(0.25, name='dropout'))


        # Output layer: softmax classifier
        model.add(Flatten(name='flatten'))
        model.add(Dense(2, activation = 'softmax', name='output'))

        # return the constructed network architecture
        return model

class TSMCNN(): #Time space merged CNN
    @staticmethod
    def build(height, width, depth):
        N = 96
        # dropout = 0.1
        inputShape = (height, width, depth)
        visible = Input(shape=inputShape)
        # First feature extractor
        conv1 = Conv2D(16, kernel_size=(int(N/12),1), strides=int(N/15),
        activation='relu', name='TempRelu')(visible)
        flat1 = Flatten(name='flatten01')(conv1)
        # dropout1 = Dropout(dropout, name='dropout01')
        # Second feature extractor
        conv2 = Conv2D(16, kernel_size=(1,64), activation='relu',
         name='SpaceRelu')(visible)
        flat2 = Flatten(name='flatten02')(conv2)
        # dropout2 = Dropout(dropout, name='dropout02')
        # merge feature extractors
        merge = concatenate([flat1, flat2], name='merge')
        output = Dense(2, activation = 'softmax', name='output')(merge)
        model = Model(inputs=visible, outputs=output)
        return model

class MLP_SGD:
    @staticmethod
    def build(input_dim):
        dropout = 0.5
        model = Sequential()
        model.add(Dense(300, input_dim = input_dim, activation = 'tanh', name='tanh01'))
        model.add(Dropout(dropout, name='dropout01'))
        model.add(Dense(150, activation = 'tanh', name='tanh02'))
        model.add(Dropout(dropout, name='dropout02'))
        model.add(Dense(2, activation = 'softmax', name='output'))

        # return the constructed network architecture
        return model
```

Código B.5: Entrenamiento de redes convolucionales

```
from keras.optimizers import SGD
from keras.utils import to_categorical
```

```
from keras.models import model_from_json

#TODO: include model architecture on the file
def get_metrics(model, X_valid, y_valid, Nnet=False):
    metrics_dict = {}
    pred_class = model.predict(X_valid)
    if Nnet:
        pred_class = pred_class.argmax(axis=1)
    metrics_dict['recall'] = metrics.recall_score(y_valid, pred_class)
    metrics_dict['precision'] = metrics.precision_score(y_valid, pred_class)
    metrics_dict['f1'] = metrics.f1_score(y_valid, pred_class)
    metrics_dict['acc'] = metrics.accuracy_score(y_valid, pred_class)
    tn, fp, fn, tp = metrics.confusion_matrix(y_valid, pred_class).ravel()
    metrics_dict['tp'] = tp
    metrics_dict['tn'] = tn
    metrics_dict['fp'] = fp
    metrics_dict['fn'] = fn
    return metrics_dict

EPOCHS = 100
BATCH_SIZE = 128

def train(dat):
    fv, mean, std = preprocessing_ConvNets(dat, MARKER_DEF_TRAIN)
    filename = 'Trained_Nnets/'+mdl+'/Preprocessing_Parameters/'+subject+\
    '_'+str(t)+'Mean_Std.joblib'
    joblib.dump([mean, std], filename)
    X = np.expand_dims(fv.data, -1)
    y = fv.axes[0]

    X_train, X_valid, y_train, y_valid = train_test_split(X, y,
     test_size = 0.1, train_size = 0.9)

    #Creating the architecture (layers)
    # n_features = X_train.shape[1]
    n_P300 = X_train[y_train == 1].shape[0]
    n_nonP300 = X_train[y_train == 0].shape[0]

    model = models[mdl].build(width=64, height=96, depth=1)
    model.compile(optimizer='sgd',
          loss='binary_crossentropy',
          metrics=['accuracy', precision, recall, f1],
          weighted_metrics = [f1])

    H = model.fit(X_train, to_categorical(y_train),
                epochs = EPOCHS, batch_size = BATCH_SIZE,
                validation_split = 0.2,
                # callbacks = [csv_logger],
                class_weight = {1: n_nonP300/n_P300*0.4, 0: n_P300/n_nonP300})

    return model, X_train, X_valid, y_train, y_valid

def preprocessing_ConvNets(dat, MRK_DEF, mean=None, std=None, *args, **kwargs):
    dat = proc.subsample(dat, 120)

    fs_n = dat.fs / 2
    b, a = proc.signal.butter(5, [20 / fs_n], btype='low')
    dat = proc.lfilter(dat, b, a)
```

```
    b, a = proc.signal.butter(5, [0.1 / fs_n], btype='high')
    dat = proc.lfilter(dat, b, a)

    epo = proc.segment_dat(dat, MRK_DEF, [0, 800])

    # get mean and standard deviation among the channels axis
    if mean is None and std is None:
        mean = np.mean(epo.data.reshape(-1,64), axis=0)
        std = np.std(epo.data.reshape(-1,64), axis=0)

    # Normalization
    Iij = (epo.data - mean)/std
    fv = epo.copy(data=Iij, fs=120)

    return fv, mean, std

SUBJECTS = {}
SubjectsNames = []
for i in range(11):
    SubjectsNames.append('A_'+str(i))
SUBJECTS['A'] = SubjectsNames
SubjectsNames = []
for i in range(11):
    SubjectsNames.append('B_'+str(i))
SUBJECTS['B'] = SubjectsNames

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config)

trials = [3, 6, 9, 12, 15]
# models = {'TSMCNN': TSMCNN}
models = {'CCNN': CCNN, 'OCLNN': OCLNN, 'TSMCNN': TSMCNN}
columnNames = 'Subject,Trial,recall,precision,f1,acc,tp,tn,fp,fn'
for mdl in models.keys():
    for s in SUBJECTS:
        # Open validation results file
        valResultsFile = 'Trained_Nnets/'+mdl+'/'+s+'s_valResults'+mdl+'.csv'
        trainResultsFile = 'Trained_Nnets/'+mdl+'/'+s+'s_trainResults'+mdl+'.csv'
        with open(valResultsFile,'w') as f:
            f.write(columnNames)
        with open(trainResultsFile, 'w') as f:
            f.write(columnNames)
        for subject in SUBJECTS[s]:
            print('\n#####################################################################')
            print('_____________________________Subject_', subject)
            print('#####################################################################')
            #load and prepare data
            filename = '../Data/'+subject+'_trainData.pkl'
            with open(filename, "rb") as f:
                dat = pickle.load(f)
            n_ch = dat.data.reshape(-1,7794,64).shape[0]
            for t in trials:
                if t==15:
                    data = dat
                else:
                    data = crop_nTrials(dat, t, n_ch)
```

```
with tf.device('/GPU:0'):
    model, X_train, X_valid, y_train, y_valid = train(data)
# SAVE MODEL
filename = 'Trained Nnets/'+mdl+'/'+subject+'_'+str(t)+\
'Trials_'+mdl+'.json'
weights_filename = 'Trained Nnets/'+mdl+'/'+subject+'_'+str(t)+\
'_Trials_'+mdl+'_weights.h5'
# serialize model to JSON
model_json = model.to_json()
with open(filename, "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights(weights_filename)
print("Saved model to disk")

# get and save validation metrics
metrics_ = get_metrics(model, X_valid, y_valid, Nnet=True)
resData = '\n'+subject+','+str(t)
for m in metrics_.keys():
    resData += ','+str(metrics_[m])
with open(valResultsFile, 'a') as f:
    f.write(resData)

# get and save training metrics
metrics_ = get_metrics(model, X_train, y_train, Nnet=True)
resData = '\n'+subject+','+str(t)
for m in metrics_.keys():
    resData += ','+str(metrics_[m])
with open(trainResultsFile, 'a') as f:
    f.write(resData)
```

Código B.6: Simulación online de SVM Gausiano

```
from __future__ import division

import sys
import time
import logging
import numpy as np
from scipy.io import loadmat
from os import path
# change this to the path of mushu if you don't have it in your
# PYTHONPATH already
sys.path.append('../../mushu')
sys.path.append('../')

import libmushu
from libmushu.amplifier import Amplifier
from libmushu.driver.replayamp import ReplayAmp
from wyrm.types import RingBuffer
import wyrm.processing as proc
from wyrm import io
from wyrm.types import Data, BlockBuffer, RingBuffer
from pandas import read_csv

#Classifiers
from sklearn import metrics
```

```
from sklearn.externals import joblib
from os import listdir
from os.path import isfile, join
import pickle

STIMULUS_CODE = {
    # cols from left to right
    1 : "agmsy5",
    2 : "bhntz6",
    3 : "ciou17",
    4 : "djpv28",
    5 : "ekqw39",
    6 : "flrx4_",
    # rows from top to bottom
    7 : "abcdef",
    8 : "ghijkl",
    9 : "mnopqr",
    10: "stuvwx",
    11: "yz1234",
    12: "56789_"
}

MARKER_DEF_TRAIN = {'target': ['target'], 'nontarget': ['nontarget']}
MARKER_DEF_TEST = {i : [i] for i in STIMULUS_CODE.values()}

SEG_IVAL = [0, 800]


class RReeplayAmp(Amplifier):
    """ Adapted from ReplayAmp, which was not working"""

    def configure(self, data, marker, channels, fs, realtime=True, blocksize_ms=None, bloc
        """

        Parameters
        ----------
        data
        marker
        channels
        fs
        realtime
        blocksize_ms : float
            blocksize in milliseconds
        blocksize_samples : int
            blocksize in samples

        Raises
        ------
        TypeError :
            if ``blocksize_ms`` and ``blocksize_s`` are given.

        """
        if [blocksize_ms, blocksize_samples].count(None) != 1:
            raise TypeError("blocksize_ms and blocksize_samples are mutually exclusive.")

        self.data = data
        # slow python
```

```
        self.marker = marker
        # fast numpy
        self.marker_ts = np.array([ts for ts, s in marker]) #marker time_sample
        self.marker_s = np.array([s for ts, s in marker]) #marker sample
        self.channels = channels
        self.fs = fs
        self.realtime = realtime

        if blocksize_ms:
            samples = fs * (blocksize_ms / 1000)
            if not samples.is_integer():
                raise ValueError("Resulting blocksize is not integer, please fix the block
            self.samples = samples
        if blocksize_samples:
            self.samples = blocksize_samples

    def start(self):
        self.last_sample_time = time.time()
        self.pos = 0

    def stop(self):
        pass

    def get_data(self):
        """

        Returns
        -------
        chunk, markers: Markers is time in ms since relative to the
        first sample of that block.

        """
        if self.realtime:
            elapsed = time.time() - self.last_sample_time
            blocks = (self.fs * elapsed) // self.samples
            samples = int(blocks * self.samples)

        else:
            samples = self.samples

        elapsed = samples / self.fs
        self.last_sample_time += elapsed
        # data
        chunk = self.data[self.pos:int(self.pos+samples)]

        # markers
        # fast numpy version
        mask = self.marker_ts < (elapsed * 1000)
        markers = list(zip(self.marker_ts[mask], self.marker_s[mask]))
        self.marker_ts = self.marker_ts[~mask]
        self.marker_s = self.marker_s[~mask]
        self.marker_ts -= elapsed * 1000

        self.pos += samples
        return chunk, markers

    def get_channels(self):
        return self.channels
```

```
    def get_sampling_frequency(self):
        return self.fs

    @staticmethod
    def is_available():
        return True

def online_character_experiment(amp, cfy, scaler, subject, trial):
    amp_fs = amp.get_sampling_frequency()
    amp_channels = amp.get_channels()

    # csv results file
    chcolumnNames = 'Character,Trial 1,Trial 2,Trial 3,Trial 4,Trial 5, Trial 6,\
    Trial 7,Trial 8,Trial 9,Trial 10,Trial 11,Trial 12,Trial 13,Trial 14,Trial 15,Trial 16
    chResultsFile = 'Results/Character Results/GaussianSVM/'+subject+'_'+str(trial)+\
    'Trials_CharacterResults_GaussianSVM.csv'
    with open(chResultsFile, 'w') as f:
        f.write(chcolumnNames)

    #buf = BlockBuffer(4)
    rb = RingBuffer(5000)

    fn = amp_fs / 2
    b_low, a_low = proc.signal.butter(5, [10 / fn], btype='low')
#       b_high, a_high = proc.signal.butter(5, [.4 / fn], btype='high')

    zi_low = proc.lfilter_zi(b_low, a_low, len(amp_channels))
#       zi_high = proc.lfilter_zi(b_high, a_high, len(amp_channels))

    amp.start()
    markers_processed = 0
    current_letter_idx = 0
    current_letter = TRUE_LABELS[current_letter_idx].lower()

    letter_prob = {i : 0 for i in 'abcdefghijklmnopqrstuvwxyz123456789_'}
    endresult = []

    t0 = time.time()
    data, markers = amp.get_data()

    trial_results = []
    while True:
        t0 = time.time()

        # get fresh data from the amp
        data, markers = amp.get_data()
        if len(data) == 0:
            continue

        # we should rather wait for a specific end-of-experiment marker
        if len(data) == 0:
            break

        # convert to cnt
        cnt = io.convert_mushu_data(data, markers, amp_fs, amp_channels)
```

```
        # low-pass and subsample
        cnt, zi_low = proc.lfilter(cnt, b_low, a_low, zi=zi_low)
#         cnt, zi_high = proc.lfilter(cnt, b_high, a_high, zi=zi_high)

        cnt = proc.subsample(cnt, 20)

        newsamples = cnt.data.shape[0]

        # enter the ringbuffer
        rb.append(cnt)
        cnt = rb.get()

        # segment
        epo = proc.segment_dat(cnt, MARKER_DEF_TEST, SEG_IVAL,
        newsamples=newsamples)
        if not epo:
            continue

        fv = proc.create_feature_vectors(epo)
#         logger.debug('Markers processed: ')
#         logger.debug(markers_processed)
# #         logger.debug(markers_processed)
        X = scaler.transform(fv.data)
        clf_out = cfy.predict_proba(X)[:,1]
#         clf_out = proc.lda_apply(fv, cfy)
        markers = [fv.class_names[cls_idx] for cls_idx in fv.axes[0]]
        result = zip(markers, clf_out)

        for s, score in result: #letters in one row/column
            if (markers_processed%12 == 0):
                current_pred_letter = sorted(letter_prob.items(),
                key=lambda x: x[1])[-1][0]
                if (current_pred_letter.upper() == TRUE_LABELS[current_letter_idx]):
                    trial_results.append('1')
                else:
                    trial_results.append('0')

            if markers_processed == 180: # 12 trials x 15 repetitions = 180
                endresult.append(sorted(letter_prob.items(), key=lambda x: x[1])[-1][0])
                letter_prob = {i : 0 for i in 'abcdefghijklmnopqrstuvwxyz123456789_'}
                markers_processed = 0
                current_letter_idx += 1
                current_letter = TRUE_LABELS[current_letter_idx].lower()
                n_ch = len(endresult)

                with open(chResultsFile,'a') as f:
                    f.write('\n'+str(n_ch)+','+','.join(trial_results))

                trial_results = []
            for letter in s:
                letter_prob[letter] += score
            markers_processed += 1

        if len(endresult) == len(TRUE_LABELS)-1:
            break

    print(''.join(endresult))
    print(TRUE_LABELS)
```

```
    acc = np.count_nonzero(np.array(endresult) ==
    np.array(list(TRUE_LABELS.lower()[:len(endresult)]))) / len(endresult)
    print ("Accuracy:", acc * 100)

    amp.stop()
    return acc

    # test data
filename = 'Data/test_data_subjs_AB.pkl'
test_data = {}
with open(filename, "rb") as f:
    test_data['A'], test_data['B'] = pickle.load(f)

#### Getting Classifiers Filenames
mypath = 'Train_Subjects/Trained_Classifiers/GaussianSVM/'
files = [f for f in listdir(mypath) if isfile(join(mypath, f))]

SUBJECTS = {}
SubjectsNames = []
for i in range(3,11):
    SubjectsNames.append('A_'+str(i))
SUBJECTS['A'] = SubjectsNames
SubjectsNames = []
for i in range(11):
    SubjectsNames.append('B_'+str(i))
SUBJECTS['B'] = SubjectsNames

trials = [3,6,9,12,15]
clf_files = {}
for s in SUBJECTS.keys():
    for subject in SUBJECTS[s]:
        clf = {}
        for t in trials:
            for fname in files:
                if subject+".pkl" in fname and str(t)+'Trials_' in fname:
                    clf[t] = fname
        clf_files[subject] = clf
#bbuffer = BlockBuffer(12)
amp = RReeplayAmp()
for s in SUBJECTS.keys(): # A or B
    cnt = test_data[s]
    TRUE_LABELS = cnt.labels
    for subject in SUBJECTS[s]:
        print('\n##########################################################################')
        print('                                  SUBJECT: ', subject)
        print('##########################################################################\n')

        chcolumnNames = 'Training_Trials,_Accuracy'
        AccVsTrialsFile = 'Results/AccVsTrials/GaussianSVM/'+subject+\
        '_AccVsTrialsResults_GaussianSVM.csv'
        with open(AccVsTrialsFile, 'w') as f:
            f.write(chcolumnNames)
        for t in trials:

            print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
            print('Number_of_trainig_trials:_', t)

            # loas scaler
```

```
            filename = 'Train_Subjects/Trained_Classifiers/GaussianSVM/Scalers/'\
            +str(t)+'Trials_SVMScaler_'+subject+'.pkl'
            scaler = joblib.load(filename)
            #load classifier
            filename = join(mypath, clf_files[subject][t])
            clf = joblib.load(filename)

            #configure amplifier
            amp.configure(data=cnt.data, marker=cnt.markers, channels=cnt.axes[-1],
            fs=cnt.fs, realtime=False, blocksize_samples=12)
            acc = online_character_experiment(amp, clf, scaler, subject, trial=t)
            with open(AccVsTrialsFile, 'a') as f:
                f.write('\n'+str(t)+','+str(acc))
```

Código B.7: Simulación online de Redes Convolucionales

```
from __future__ import division

import sys
import time
import logging
import numpy as np
from scipy.io import loadmat
from os import path
# change this to the path of mushu if you don't have it in your
# PYTHONPATH already
sys.path.append('../../mushu')
sys.path.append('../')

import libmushu
from libmushu.amplifier import Amplifier
from libmushu.driver.replayamp import ReplayAmp
from wyrm.types import RingBuffer
import wyrm.processing as proc
from wyrm import io
from wyrm.types import Data, BlockBuffer, RingBuffer
from pandas import read_csv

#Classifiers
from sklearn import metrics

from sklearn.externals import joblib
from os import listdir
from os.path import isfile, join
import pickle

STIMULUS_CODE = {
    # cols from left to right
    1 : "agmsy5",
    2 : "bhntz6",
    3 : "ciou17",
    4 : "djpv28",
    5 : "ekqw39",
    6 : "flrx4_",
    # rows from top to bottom
    7 : "abcdef",
    8 : "ghijkl",
    9 : "mnopqr",
```

```
    10: "stuvwx",
    11: "yz1234",
    12: "56789_"
}

MARKER_DEF_TRAIN = {'target': ['target'], 'nontarget': ['nontarget']}
MARKER_DEF_TEST = {i : [i] for i in STIMULUS_CODE.values()}

SEG_IVAL = [0, 800]


class RReeplayAmp(Amplifier):
    """ Adapted from ReplayAmp, which was not working"""

    def configure(self, data, marker, channels, fs, realtime=True, blocksize_ms=None, bloc
        """

        Parameters
        ----------
        data
        marker
        channels
        fs
        realtime
        blocksize_ms : float
            blocksize in milliseconds
        blocksize_samples : int
            blocksize in samples

        Raises
        ------
        TypeError :
            if ``blocksize_ms`` and ``blocksize_s`` are given.

        """
        if [blocksize_ms, blocksize_samples].count(None) != 1:
            raise TypeError("blocksize_ms and blocksize_samples are mutually exclusive.")

        self.data = data
        # slow python
        self.marker = marker
        # fast numpy
        self.marker_ts = np.array([ts for ts, s in marker]) #marker time_sample
        self.marker_s = np.array([s for ts, s in marker]) #marker sample
        self.channels = channels
        self.fs = fs
        self.realtime = realtime

        if blocksize_ms:
            samples = fs * (blocksize_ms / 1000)
            if not samples.is_integer():
                raise ValueError("Resulting blocksize is not integer, please fix the block
            self.samples = samples
        if blocksize_samples:
            self.samples = blocksize_samples

    def start(self):
        self.last_sample_time = time.time()
```

```
        self.pos = 0

    def stop(self):
        pass

    def get_data(self):
        """

        Returns
        -------
        chunk, markers: Markers is time in ms since relative to the
        first sample of that block.

        """
        if self.realtime:
            elapsed = time.time() - self.last_sample_time
            blocks = (self.fs * elapsed) // self.samples
            samples = int(blocks * self.samples)

        else:
            samples = self.samples

        elapsed = samples / self.fs
        self.last_sample_time += elapsed
        # data
        chunk = self.data[self.pos:int(self.pos+samples)]

        # markers
        # fast numpy version
        mask = self.marker_ts < (elapsed * 1000)
        markers = list(zip(self.marker_ts[mask], self.marker_s[mask]))
        self.marker_ts = self.marker_ts[~mask]
        self.marker_s = self.marker_s[~mask]
        self.marker_ts -= elapsed * 1000

        self.pos += samples
        return chunk, markers

    def get_channels(self):
        return self.channels

    def get_sampling_frequency(self):
        return self.fs

    @staticmethod
    def is_available():
        return True

def online_character_experiment(amp, cfy, scaler, subject, trial):
    amp_fs = amp.get_sampling_frequency()
    amp_channels = amp.get_channels()

    # csv results file
    chcolumnNames = 'Character,Trial 1,Trial 2,Trial 3,Trial 4,Trial 5, Trial 6,\
    Trial 7,Trial 8,Trial 9,Trial 10,Trial 11,Trial 12,Trial 13,Trial 14,Trial 15,Trial 16
    chResultsFile = 'Results/Character Results/GaussianSVM/'+subject+'_'+str(trial)+\
    'Trials_CharacterResults_GaussianSVM.csv'
    with open(chResultsFile, 'w') as f:
```

```
        f.write(chcolumnNames)

    #buf = BlockBuffer(4)
    rb = RingBuffer(5000)

    fn = amp_fs / 2
    b_low, a_low = proc.signal.butter(5, [10 / fn], btype='low')
#       b_high, a_high = proc.signal.butter(5, [.4 / fn], btype='high')

    zi_low = proc.lfilter_zi(b_low, a_low, len(amp_channels))
#       zi_high = proc.lfilter_zi(b_high, a_high, len(amp_channels))

    amp.start()
    markers_processed = 0
    current_letter_idx = 0
    current_letter = TRUE_LABELS[current_letter_idx].lower()

    letter_prob = {i : 0 for i in 'abcdefghijklmnopqrstuvwxyz123456789_'}
    endresult = []

    t0 = time.time()
    data, markers = amp.get_data()

    trial_results = []
    while True:
        t0 = time.time()

        # get fresh data from the amp
        data, markers = amp.get_data()
        if len(data) == 0:
            continue

        # we should rather wait for a specific end-of-experiment marker
        if len(data) == 0:
            break

        # convert to cnt
        cnt = io.convert_mushu_data(data, markers, amp_fs, amp_channels)


        # low-pass and subsample
        cnt, zi_low = proc.lfilter(cnt, b_low, a_low, zi=zi_low)
#           cnt, zi_high = proc.lfilter(cnt, b_high, a_high, zi=zi_high)

        cnt = proc.subsample(cnt, 20)

        newsamples = cnt.data.shape[0]

        # enter the ringbuffer
        rb.append(cnt)
        cnt = rb.get()

        # segment
        epo = proc.segment_dat(cnt, MARKER_DEF_TEST, SEG_IVAL,
        newsamples=newsamples)
        if not epo:
            continue
```

```
        fv = proc.create_feature_vectors(epo)
#         logger.debug('Markers processed: ')
#         logger.debug(markers_processed)
# #         logger.debug(markers_processed)
        X = scaler.transform(fv.data)
        clf_out = cfy.predict_proba(X)[:,1]
#         clf_out = proc.lda_apply(fv, cfy)
        markers = [fv.class_names[cls_idx] for cls_idx in fv.axes[0]]
        result = zip(markers, clf_out)

        for s, score in result: #letters in one row/column
            if (markers_processed%12 == 0):
                current_pred_letter = sorted(letter_prob.items(),
                key=lambda x: x[1])[-1][0]
                if (current_pred_letter.upper() == TRUE_LABELS[current_letter_idx]):
                    trial_results.append('1')
                else:
                    trial_results.append('0')

            if markers_processed == 180: # 12 trials x 15 repetitions = 180
                endresult.append(sorted(letter_prob.items(), key=lambda x: x[1])[-1][0])
                letter_prob = {i : 0 for i in 'abcdefghijklmnopqrstuvwxyz123456789_'}
                markers_processed = 0
                current_letter_idx += 1
                current_letter = TRUE_LABELS[current_letter_idx].lower()
                n_ch = len(endresult)

                with open(chResultsFile,'a') as f:
                    f.write('\n'+str(n_ch)+','+','.join(trial_results))

                trial_results = []
            for letter in s:
                letter_prob[letter] += score
            markers_processed += 1

        if len(endresult) == len(TRUE_LABELS)-1:
            break

    print(''.join(endresult))
    print(TRUE_LABELS)
    acc = np.count_nonzero(np.array(endresult) ==
    np.array(list(TRUE_LABELS.lower()[:len(endresult)]))) / len(endresult)
    print ("Accuracy:", acc * 100)

    amp.stop()
    return acc

    # test data
filename = 'Data/test_data_subjs_AB.pkl'
test_data = {}
with open(filename, "rb") as f:
    test_data['A'], test_data['B'] = pickle.load(f)

#### Getting Classifiers Filenames
mypath = 'Train_Subjects/Trained_Classifiers/GaussianSVM/'
files = [f for f in listdir(mypath) if isfile(join(mypath, f))]

SUBJECTS = {}
```

```
SubjectsNames = []
for i in range(3,11):
    SubjectsNames.append('A_'+str(i))
SUBJECTS['A'] = SubjectsNames
SubjectsNames = []
for i in range(11):
    SubjectsNames.append('B_'+str(i))
SUBJECTS['B'] = SubjectsNames

trials = [3,6,9,12,15]
clf_files = {}
for s in SUBJECTS.keys():
    for subject in SUBJECTS[s]:
        clf = {}
        for t in trials:
            for fname in files:
                if subject+".pkl" in fname and str(t)+'Trials_' in fname:
                    clf[t] = fname
        clf_files[subject] = clf
#bbuffer = BlockBuffer(12)
amp = RReeplayAmp()
for s in SUBJECTS.keys(): # A or B
    cnt = test_data[s]
    TRUE_LABELS = cnt.labels
    for subject in SUBJECTS[s]:
        print('\n##########################################################################')
        print('                                  SUBJECT: ', subject)
        print('##########################################################################\n')

        chcolumnNames = 'Training Trials, Accuracy'
        AccVsTrialsFile = 'Results/AccVsTrials/GaussianSVM/'+subject+\
        '_AccVsTrialsResults_GaussianSVM.csv'
        with open(AccVsTrialsFile, 'w') as f:
            f.write(chcolumnNames)
        for t in trials:

            print('\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
            print('Number of trainig trials: ', t)

            # loas scaler
            filename = 'Train Subjects/Trained Classifiers/GaussianSVM/Scalers/'\
            +str(t)+'Trials_SVMScaler_'+subject+'.pkl'
            scaler = joblib.load(filename)
            #load classifier
            filename = join(mypath, clf_files[subject][t])
            clf = joblib.load(filename)

            #configure amplifier
            amp.configure(data=cnt.data, marker=cnt.markers, channels=cnt.axes[-1],
            fs=cnt.fs, realtime=False, blocksize_samples=12)
            acc = online_character_experiment(amp, clf, scaler, subject, trial=t)
            with open(AccVsTrialsFile, 'a') as f:
                f.write('\n'+str(t)+','+str(acc))
```