Proyecto Fin de Carrera
Grado Ingeniería Electrónica, Robótica y Mecatrónica (GIERM)

Machine Learning para MI-BCI orientada al procesado de las señales EEG en tiempo real

Autor: Isabel Ortiz Ramírez

Tutor: Sergio A. Cruces Álvarez

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

Sevilla, 2021

Proyecto Fin de Carrera
Grado Ingeniería Electrónica, Robótica y Mecatrónica (GIERM)

Machine Learning para MI-BCI orientada al procesado de las señales EEG en tiempo real

Autor:

Isabel Ortiz Ramírez

Tutor:

Sergio A. Cruces Álvarez

Catedrático de Universidad

Dpto. de Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Carrera: Machine Learning para MI-BCI orientada al procesado de las señales EEG en tiempo real

Autor: Isabel Ortiz Ramírez
Tutor: Sergio A. Cruces Álvarez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia
A los profesores que me han ayudado

Agradecimientos

Especial gratitud a mi tutor, Sergio A. Cruces Álvarez, sin sus explicaciones se me habría dificultado muchísimo la realización del trabajo. A pesar de que yo tardé mucho en terminar el TFG, siempre estuvo pendiente de si necesitaba ayuda, incluso en verano, y en dedicarme horas a explicarme lo que no entendía, gracias por ser paciente conmigo. Me ha ayudado inconmensurablemente a saber sobre la realización y programación de las interesantes interfaces cerebro-ordenador.

También mis agradecimientos a Antonio Márquez Tristán por su ayuda con el código para la interfaz en MATLAB. El código que realicé lo hice a partir de uno proporcionado por él.

Mi eterna gratitud a todas las personas que han facilitado investigaciones y documentación sobre los BCI y el Machine Learning, los cuales me han ayudado a desarrollar este trabajo.

Además, tengo que dar gracias a todos los profesores que me han ayudado durante mi realización del grado.

Finalmente, agradezco a mi familia y especialmente a mi padre y a mi madre por contribuir a mi formación académica e intentar ayudarme cuando lo necesito.

Resumen

Una interfaz cerebro-ordenador (Brain Computer Interface (BCI)) es un sistema que lee señales emitidas en el cerebro, las interpreta y las convierte en ordenes según como se haya identificado la señal producida. Es una forma de comunicarse el cerebro con un ordenador sin necesidad de interactuar físicamente con él. Esto tiene multitud de usos, pero es especialmente útil para ayudar a personas con discapacidad. A medida que pasan los años cada vez hay más avances en estos sistemas, es una tecnología que sigue desarrollándose, pues todavía no se conocen todos los misterios del cerebro humano, aunque ha habido grandes avances en los últimos años.

En lo que respecta a este trabajo, primero se hace una introducción con posibles aplicaciones de estos sistemas y un resumen teórico sobre las diferentes señales cerebrales más relevantes para sistemas BCI. Hay diferentes tipos de BCI según si son invasivos o no y según la señal cerebral que interese identificar. La BCI en la que se centra este trabajo es la Interfaz Cerebro Ordenador de Movimiento Imaginario (MI-BCI), la cual consiste en que una persona imagina realizar un movimiento (como mover mano izquierda o derecha) y se identifica que movimiento ha pensado según las características de la señal registrada. Cuando uno se imagina realizar un movimiento, se produce una señal cerebral similar a la producida cuando el movimiento es realmente realizado.

Para registrar la señal cerebral querida se usa el electroencefalograma (EEG), el cual mide la actividad electromagnética. Se colocan electrodos en el cuero cabelludo. Al usar esto hay que tener en cuenta que la señal registrada presentará ruido o artefactos, la señal estará mezclada con otras señales que no son las de interés y llegará a varios electrodos. Las señales presentarán baja resolución espacial, baja SNR y se verán afectadas por interferencias de otras señales de la cabeza.

El MI-BCI se desarrolla en tres etapas: preprocesamiento de la señal de entrada, extracción de patrones de características y clasificación. La primera etapa consiste en adaptar o transformar los datos de la señal para facilitar el proceso de las dos etapas siguientes. La segunda etapa consiste en seleccionar y obtener numéricamente las características de cada señal que permitan diferenciarse unas de otras con mayor facilidad. Existen diferentes métodos, pero en este trabajo se usa el Common Spatial Pattern (CSP), el cual genera filtros espaciales que minimizan la variación de una clase y maximizan la variación de otra clase simultáneamente. La tercera etapa consiste en predecir el movimiento del sujeto según las características obtenidas en la etapa anterior. Existen muchos métodos de clasificación, también conocidos como métodos de Machine Learning, pero el más utilizado en estos sistemas es el conocido como Análisis de Discriminantes Lineales (LDA). Se explicarán y se probarán también otros tipos de clasificadores, aparte del ya mencionado, para comparar resultados. Se busca un clasificador con alta probabilidad de acierto y que sea sencillo de implementar, de manera que no se demore demasiado la clasificación.

El algoritmo de clasificación debe ser entrenado utilizando varios patrones, los cuales se conocen la clase a la que pertenecen. Este trabajo se realizó con el objetivo de que el entrenamiento fuera en tiempo real (online), es decir, a la vez que se van obteniendo patrones de cada clase se va realizando el entrenamiento del algoritmo. Antes de realizar el algoritmo en tiempo real se probó a hacerlo offline, ya se tienen todos los patrones almacenados para el entrenamiento. El entrenamiento puede ser supervisado (se conoce la clase de los patrones) o no supervisado (no se conoce la clase de los patrones).

Una vez entrenado el algoritmo se debe comprobar su eficiencia, por ello se realiza una prueba de testeo con otros patrones, distintos al de entrenamiento, del mismo usuario.

El conjunto de datos que se analiza es el MI-BCI dataset 2a de la BCI Competition IV, son datos disponibles públicamente.

Abstract

A Brain Computer Interface (BCI) is a system that reads signals emitted in the brain, interprets them and converts them into commands according to how it has identified the signal produced. It is a way of communicating the brain with a computer without the need to physically interact with it. This has a multitude of uses, but is especially useful for helping people with disabilities. As the years go by there are more and more advances in these systems, it is a technology that continues to develop, as all the mysteries of the human brain are still not known, although there have been great advances in recent years.

As far as this work is concerned, first an introduction is made with possible applications of these systems and a theoretical summary about the different brain signals most relevant for BCI systems. There are different types of BCI depending on whether they are invasive or not and depending on the brain signal to be identified. The BCI on which this work focuses is the Imaginary Movement Brain-Computer Interface (MI-BCI), which consists of a person imagining a movement (such as moving a left or right hand) and identifying which movement he/she has thought of according to the characteristics of the recorded signal. When one imagines performing a movement, a brain signal is produced similar to the one produced when the movement is actually performed.

The electroencephalogram (EEG), which measures electromagnetic activity, is used to record the brain signal. Electrodes are placed on the scalp. When using this it is necessary to take into account that the recorded signal will present noise or artifacts, the signal will be mixed with other signals that are not the ones of interest and will reach several electrodes. The signals will present low spatial resolution, low SNR and will be affected by interference from other signals from the head.

MI-BCI is developed in three stages: input signal preprocessing, feature pattern extraction and classification. The first stage consists of adapting or transforming the signal data to facilitate the processing of the next two stages. The second stage consists of selecting and numerically obtaining the characteristics of each signal that allow them to be more easily differentiated from each other. There are different methods, but this work uses the Common Spatial Pattern (CSP), which generates spatial filters that minimize the variation of one class and maximize the variation of another class simultaneously. The third stage consists of predicting the subject's movement according to the features obtained in the previous stage. There are many classification methods, also known as Machine Learning methods, but the most widely used in these systems is the one known as Linear Discriminant Analysis (LDA). Other types of classifiers, apart from the one already mentioned, will also be explained and tested to compare results. We are looking for a classifier with a high probability of success and that is simple to implement, so that the classification does not take too long.

The classification algorithm must be trained using several patterns, which are known to which class they belong. This work was carried out with the objective of training in real time (online), in other words, while obtaining patterns of each class, the training of the algorithm is carried out at the same time. Before performing the algorithm in real time, it was tested offline, and all the patterns are already stored for training. The training can be supervised (the class of the patterns is known) or unsupervised (the class of the patterns is not known).

Once the algorithm has been trained, its efficiency must be checked, so a test is performed with other patterns, different from the training one, from the same user.

The data set analyzed is the MI-BCI dataset 2a of the BCI Competition IV, which is publicly available data.

*** Translated with the help of www.DeepL.com/Translator (free version) ***

Índice

Notación

k , C_k	Índice que marca una clase del conjunto de clases posibles, la clase k
C_i	Una clase distinta a la que indica k
k	Cada valor discreto de la señal en FFT
t	Tiempo
τ	Índice que marca un ensayo (***trial***). Factor de traslación en el WT
$\mathbf{X}_\tau$	Matriz de registros de EEG correspondiente al ensayo τ
$s_j(t)$, $\mathbf{s}_j$, $\mathbf{S}_\tau$	Conjunto de fuentes localizadas en distintas partes de la corteza cerebral
s	Factor de escala en WT
S	Umbral en decisión trees
N_s	Número de fuentes a usar
x_n	Valor discreto de la señal en FFT
$x_i(t)$, x_i	Valores de señal registrada en cada electrodo (valores observados de la muestra), escalares de matriz $\mathbf{X}_\tau$
$k(x,\mathrm{z})$	Función Kernel. En el Kernel la ' x ' y ' z ' son características de una clase y otra.
$\mathbf{x}_{columna\_c}$, $\mathbf{x}$	Vector que es una columna de $\mathbf{X}_\tau$, cada columna es una muestra. Vector de variables aleatorias
$\mathbf{x}_{fila\_i}$	Vector que es una fila de $\mathbf{X}_\tau$, cada fila son valores que toma un canal en diferentes tiempos
$x(t)$	Componentes de $\mathbf{x}$
N_x	Número de electrodos, número de filas de matriz $\mathbf{X}_\tau$
L, T	Número de muestras en un ensayo, número de columnas de $\mathbf{X}_\tau$, la longitud del registro en muestras.
$n(t)$, $\mathbf{n}$, $\mathbf{N}_\tau$	Ruido en las fuentes
N_τ	Número de ensayos
a_{ji}	Coeficiente que modela la atenuación de la fuente j-ésima hasta el electrodo i-ésimo
$\mathbf{A}$	Matriz con coeficientes a_{ji}
a_i	Parámetros de AR
i	Índice de número de canal EEG, índice de filas de matriz $\mathbf{X}_\tau$, índice filas de $\mathbf{A}$
j	Índice número de fuente, índice de columnas de $\mathbf{A}$. En decision tree es el número de nodo.

c, j	Índice número de columnas matriz $\mathbf{X}_\tau$
r_k	Ruido blanco de media cero.
r_k	En la parte de clasificación es la raíz cuadrada de la distancia de Mahalanobis.
r_k^2	Distancia de Mahalanobis
x_k	Coeficientes obtenidos de AR
$\psi_{s,\tau}(t)$	Función wavelet madre
$W_f(s,\tau)$	Base de WT
$f(t)$	Función con respecto al tiempo
$\mathbf{C}_{X_\tau}^{(0)}$, $\mathbf{C}_{muestral}$	Matriz de covarianza muestreada
$\mathbf{C}_{x_i x_j}$	Matriz covarianza estadística de dos variables aleatorias
$\mathbf{C}_{X_\tau}^{(1)}$	Matriz covarianza muestreada refinada
E	Esperanza estadística
cov	Calculo covarianza
n	Número de filas y columnas matriz de covarianza
$\langle x \rangle$	Media muestral de variable aleatoria x.
$\hat{\Sigma}_{f\|c_x}$, $\mathbf{V}_k$	Matriz de covarianza de características de cada clase
$\|\ \|$	Determinante de lo que haya dentro.
$\hat{\Sigma}_{x\|c_x}^{(1)}$	Matriz de covarianza media de las señales de entrada
$\hat{\Sigma}_f$	Matriz de Covarianza media de los patrones, de todas las clases, teniendo en cuenta la probabilidad a priori de cada clase
p	Número de filtros en reducción de dimensión. Número de variables clasificadoras, número de características de un ensayo.
p	La dimensionalidad de cada observación
p_i	En decisión tree es la probabilidad de éxito. En regresión logística p_i es la probabilidad del suceso, éxito.
p_{jk}	Proporción del nodo j que pertenece a la clase k.
q	Probabilidad de fracaso.
$\mathbf{C}_{Y_\tau}$	Matriz de covarianza del ensayo de después de aplicar filtrado espacial a las observaciones.
$\mathbf{W}$	Matriz de filtro espacial en CSP
$\mathbf{w}$	Vectores de filtrado espacial, en CSP. En Regresión logística, este símbolo corresponde a los coeficientes por los que se multiplica las entradas.
$\mathbf{w}$	En SVM es un vector perpendicular al hiperplano
$\mathbf{Y}_\tau$	Matriz resultante de aplicar filtrado espacial a las señales EEG en un ensayo
Y_i	En regresión logística es una distribución binomial $\sim B(p_i, n_i)$
$y(t)$	Variables obtenidas al aplicar filtrado a las componentes de $\mathbf{X}_\tau$
y	En el kernel es la respuesta prevista. En regresión logística es la salida de la neurona.
y_i	En decisión trees es el valor real de la respuesta a predecir

$\hat{y}_{R_j}$	En decisión trees es la media de la variable respuesta en la región R_j .
$J(w)$	Cociente de Rayleigh
J	En decisión trees indica el número de regiones que minimiza el RSS. En Regresión logística es la función de coste.
$\mathbf{R}_{\mathbf{Xs}}$	Matriz de covarianza de la señal de interés
$\mathbf{R}_{\mathbf{X}i}$	Matriz covarianza de la no señal de interés.
$x_{\mathbf{s}}(t)$	Señal EEG de interés
$x_i(t)$	Señal EEG de no interés
$\mathbf{V}$	Matriz de filtro espacial completa
$\mathbf{D}$	Matriz diagonal, sería la evaluación de la función $J(w)$ para cada uno de los filtros espaciales
λ_k	Autovalor
λ	Parámetro de ajuste en RDA, que determina si las covarianzas deben estimarse de forma independiente
γ	Parámetro de RDA para convertir covarianza
γ	Parámetro que controla el comportamiento del kernel
$\mathbf{I}$	Matriz identidad
$\mathbf{A}$, $\mathbf{B}$	Matriz genérica para ejemplo
$\mathbf{f}_\tau$	Las características de un ensayo
f_k	En clasificador Naive Bayes, es la manera en que se denomina a una parte de la función de decisión
σ^2 , var	Varianza
$\hat{\sigma}^2$	Promedio de las varianzas muestrales
μ	Media
π_k	Fracción del conjunto de datos(patrones) en la clase k
$\hat{\pi}_k$	La probabilidad previa de que una observación pertenezca a la clase k
a_p	Coeficientes de la función f_{fisher}
f_{fisher}	Una función que será combinación lineal de p variables clasificadoras
$F(\alpha)$	Función cuadrática en términos de proyección α
α	El filtro óptimo para proyectar (vector de proyección)
α_i	En SVM son multiplicadores de Lagrange
N_{c_k}	Número de patrones de la clase k
N_c	Número de clases
$p(c_k)$	Probabilidad a priori de la clase k
β	Umbral que te permite saber a partir de qué valor se asigna a una clase u otra
log, ln	Logaritmo natural
$k_{asignada}$	La clase que se le asigna al ensayo
k_τ	La clase real de un ensayo
$sign$	Signo, -1 o +1.

$\hat{d}_k$	Función de decisión, depende del patrón de características
k_{means}	cantidad de patrones cercanos que se tienen en cuenta para clasificar con clasificador KNN
$\mathbf{f}_j$	Patrón para clasificar actual en KNN. En decisión trees es el predictor a encontrar en cada iteración.
$\mathbf{f}_s$	Patrones de entrenamiento
d_E	Distancia euclidiana
R_j	Nodos terminales en los árboles de decisión
χ^2	Estadístico chi-square
$\sigma(z)$	Función sigmoide
e	Número e, número de Euler
z	Valor numérico real
logit	Inversa de la sigmoide
SNR	Signal-to-noise ratio
<	Menor o igual
>	Mayor o igual

1 INTRODUCCIÓN

"Sin duda no hay progreso."

Charles Robert Darwin

Una interfaz es el medio por el cual una persona se comunica con una máquina. Un ejemplo de método tradicional de comunicarse con una máquina es usando un teclado, un ratón, un micrófono o una pantalla. La persona piensa en lo que quiere que haga la máquina y realiza una interacción física[1]. Esto sería más eficiente si se saltase el paso de la interacción física. Entra entonces en escena las interfaces cerebro-ordenador.

1.1 Motivación y aplicaciones de las interfaces cerebro-ordenador

"Los sistemas de Interfaz Cerebro-Computadora (BCI), son una tecnología en auge y pleno desarrollo, que pretende traducir y convertir los pensamientos del usuario, a través de señales cerebrales, en acciones y comandos, lo que permitiría a los usuarios comunicarse, manejar el entorno y otros dispositivos que le facilitarán el desempeño en su vida diaria" [1]. A parte de dar comandos, tal vez, en un futuro también se pueda recibir información directamente al cerebro, pero esto todavía está en desarrollo. Un ejemplo de desarrollo de BCI es la empresa Neuralink de Elon Musk, la cual está actualmente desarrollando un dispositivo que permitirá "*tratar pacientes que sufran de discapacidades causadas por desórdenes neurológicos mediante estimulación cerebral directa. Además, busca lograr una simbiosis total con la inteligencia artificial*" [2].

"Esta tecnología ha atraído a muchos investigadores durante la última década con el motivo de desarrollar una BCI eficiente, robusta y fiable, lo cual ha supuesto que cada uno de los diferentes grupos de investigación creara su propia manera de proceder respecto a la tecnología de sus dispositivos. A pesar de las diferencias entre los dispositivos todos siguen el mismo principio de funcionamiento básico: la medición de la actividad cerebral mediante sensores, procesado de la señal adquirida para obtener sus características de interés y, por último, interaccionar con el entorno de la forma deseada por el usuario" [3].

[1] Manual o por voz

Se recomienda leer el documento [4], ya que trata sobre la misma temática que este trabajo. También se recomienda leer [5]- [6]- [7]- [8]- [9]- [10]- [11]- [12]- [13]- [14]- [15]- [16]- [17]- [18]- [19]- [20].

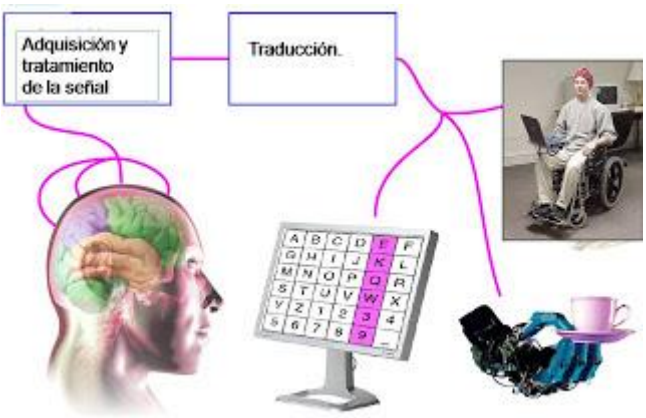


Figura 1-1. Ejemplos control motor artificial con BCI. Con las señales cerebrales amplificadas e identificadas las órdenes que se quiere, pudiéndose conseguir que el usuario controle aplicaciones informáticas, prótesis o exoesqueletos robóticos o el movimiento de una silla de ruedas. Fuente: [24].

Los BCI tienen multitud de aplicaciones. A continuación, se mencionarán las más relevantes:

- Discapacidad:

Un ejemplo de uso BCI es para movimiento de una silla de ruedas. Se desarrolló en Europa un sistema BCI en el que una persona con parálisis de piernas podía establecer órdenes de movimiento a una silla de ruedas en un entorno virtual con solo pensar en mover sus piernas. "*El sistema BCI utilizado en este caso se basaba en la detección del ritmo beta, de forma que en presencia de actividad en esa banda espectral el avatar que representa al usuario se desplaza hacia delante y en ausencia permanece parado*" [21]. Un ejemplo de esta aplicación se encuentra en el artículo [22].

Figura 1-2. Los sistemas BCI son de gran utilidad para personas discapacitadas. Fuente: [23].

En la Figura 1-2 se ve un ejemplo de uso del BCI, para personas discapacitadas. *"La empresa Emotiv Systems lanzó, a principios de 2009, una interfaz neuronal comercial llamada EPOC, capaz de detectar los pensamientos del usuario y traducirlos a comandos comprensibles por un programa de ordenador"* [21]. No solo serviría para mover una silla de ruedas, sino que también se podría manejar el ordenador sin tener que usar las manos o la voz. Estos dispositivos permiten la comunicación con el ambiente mediante la elección sucesiva de símbolos, normalmente letras del alfabeto. El que se considera el primer dispositivo realizado de este tipo es el deletreador P300. Un ejemplo de esta utilización es presentar al usuario letras organizadas en una matriz. Cada una de las filas y columnas de la matriz es iluminada de manera aleatoria. "*Cuando el usuario se fija en una letra, la infrecuente iluminación de ésta (dos veces por cada ciclo de 12 iluminaciones) hace que se produzca un potencial P300 cada vez que se ilumina, el cual es registrado por el sistema BCI*" [21]. Este concepto de la potencia se explicará en el apartado 2.

Los sistemas BCI serán de gran ayuda a las personas con movilidad reducida. Los sistemas de reconocimiento de voz también han sido de gran ayuda en este ámbito, pero hay que tener en cuenta que hay personas que no tienen la capacidad de comunicarse con la voz.

- Prótesis y órtesis

Cuando se vuelve a conectar un miembro del que se ha perdido la capacidad de controlar o que ha sido amputado, se puede hacer uso de una neuroprótesis [24]. Básicamente es el control de un motor artificial. "*El desarrollo de neuroprótesis depende de la tasa de transferencia de información que el BCI tenga*" [21]. Si se usan registros de dentro de la corteza cerebral como señales de control, método invasivo, la tasa de transferencia de información es elevada y se pueden controlar piernas o brazos robóticos con más facilidad que con un BCI basado en EEG, el cual solo podría realizar tareas de control sencillas [21]. Un ejemplo de uso para prótesis de una mano (Figura 1-3) se encuentra en el artículo [25], presenta "*un proyecto de investigación que tiene como objetivo la manipulación de una prótesis de mano en un ambiente virtual de simulación utilizando una interfaz BCI*" [21], se puede apreciar en la Figura 1-6.

Figura 1-3. RSL Steeper, BeBionic. Fuente: [25].

Un ejemplo de órtesis[2] es el de la compañía Neurolutions, para la rehabilitación de ictus, el cual se ve en Figura 1-4.

Figura 1-4. El dispositivo consta de un casco que registra señal de EEG y de un exoesqueleto con forma de guante que se sitúa sobre la mano del paciente que se va a rehabilitar. El casco graba señal de EEG del lado del cerebro no dañado por el ictus, y lo envía a un guante robótico que asiste al usuario en los movimientos de la mano. Mediante una Tablet se controla la sesión de rehabilitación, y se guía al paciente en los ejercicios a realizar. Fuente: [238].

Figura 1-5. Ejemplo órtesis. Fuente: [24].

Figura 1-6. Prótesis de mano en una simulación virtual utilizando BCI. Fuente: [25].

[2] También se le denomina exoesqueleto

➢ Crear canales sensoriales artificiales.

Actualmente se está desarrollando sistema BCI para quienes han perdido la audición o la vista.

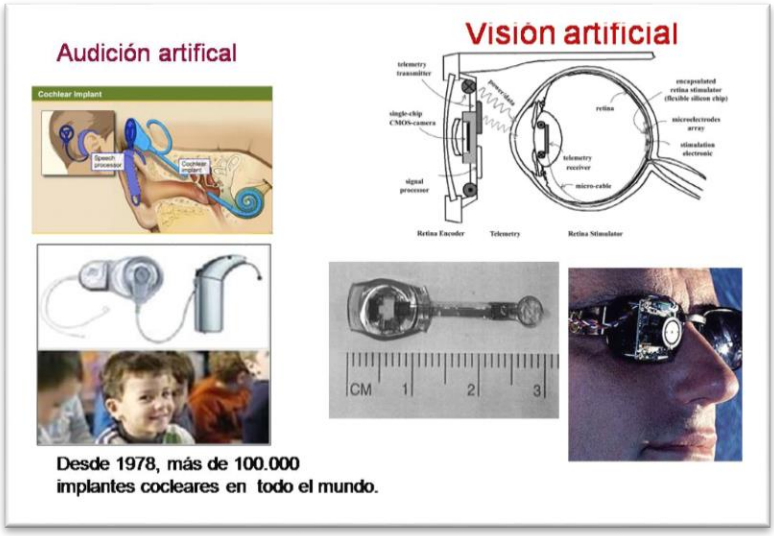


Figura 1-7. Esto es un ejemplo de canal sensorial artificial, en el cual "*el sistema nervioso central interactúa con un chip electrónico*" situado en los implantes cocleares. Además, están siendo investigados los chips para visión artificial. Fuente: [24].

➢ Rehabilitación neurológica.

Se está investigando su uso para regular desordenes neuronales como el Parkinson, pérdidas de memoria o epilepsia. Otra utilidad es para saber el estado de pacientes en coma (Figura 1-8).

Figura 1-8. mindBEAGLE es una interfaz cerebro-computadora para evaluar la conciencia y la percepción de los pacientes en coma / encerrados. Es móvil y, en ocasiones, ofrece comunicación con los pacientes. Fuente: [24].

➢ Arte y ocio

También se puede usar para conceptos artísticos, como para un traje que cambie con el pensamiento (Figura 1-9), unas orejas de gato que cambian de posición según tus emociones (Figura 1-10), un selector de música de acuerdo con las ondas cerebrales del usuario (Figura 1-11) o para controlar un juguete.

Figura 1-9. El BCI de g.Pangolin utiliza rejillas de electrodos de 16 canales que se montan con una tira adhesiva en la piel humana después de que se llena con gel de electrodo conductor. Esto fue utilizado con fines artísticos para controlar un vestido interactivo. Fuente: [26]- [27].

Figura 1-10. Neurowear: nekomimi. Fuente: [237]

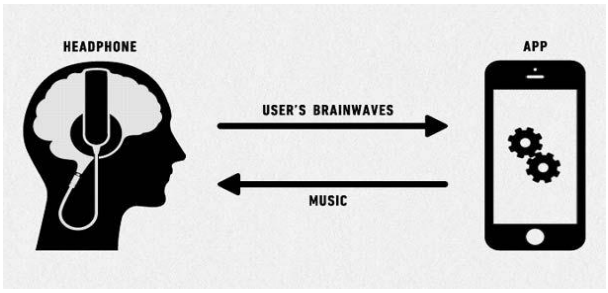


Figura 1-11. Mico. Fuente: [237]

➢ Domótica y teleoperación.

Un ejemplo de este uso es el proyecto de utilización de BCI como herramienta de entrenamiento cognitivo, el cual ayuda a prevenir los efectos del envejecimiento, propuesto por el Grupo de Ingeniería Biomédica de la Universidad de Valladolid. Desarrollando una aplicación BCI que pueda instaurar el control de aparatos domóticos y electrónicos en el hogar [28]. El control de los diferentes aparatos de la vivienda ayudará sobre todo a personas de avanzada edad con dificultades de movimiento y a personas discapacitadas. Un ejemplo sería "*usando un conjunto de diodos LED. Cada diodo representaría una opción diferente y se iluminarían a distinta frecuencia, de forma que al fijar la vista en un determinado LED se produce el potencial SSVEP con la misma frecuencia con la que éste se ilumina*" [21]. También se podría usar el BCI para la teleoperación de robots, como se ve en documento [29], en donde explica cómo se usa BCI para controlar un dron usando MATLAB.

➢ Videojuegos y realidad virtual

Los sistemas BCI serán un gran avance en el campo de los videojuegos y la realidad virtual. Los videojuegos basados en BCI se pueden catalogar en tres grupos: multimodal; multiusuario; y modal y monousuario. Los de tipo multimodal permiten recibir órdenes usando las señales EEG y algún control físico, por ejemplo, un Joystick[3]. Los de tipo multiusuario permiten la acción mutua y relacionarse con otros jugadores. Por último, el tipo modal y monousuario solo posibilitan la participación de un solo jugador y las órdenes son solo recibidas a través de las señales cerebrales EEG. Algunos ejemplos de videojuegos que han sido adaptados a BCI son World of Warcraft y Tetris [30].

[3] Palanca de mando

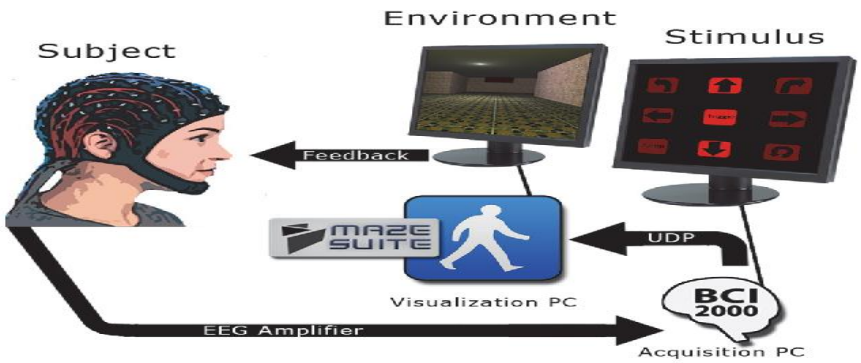


Figura 1-12. Un EEG-BCI basado en P300 para control de navegación espacial. Fuente: [31].

Figura 1-14. BCI sirve para realidad virtual. Fuente [32].

Figura 1-13. Ejemplo BCI en videojuegos. Fuente: [24].

1.2 Objetivos

El objetivo principal del trabajo es realizar la programación en MATLAB de una interfaz cerebro-ordenador para clasificar las señales cerebrales correspondientes a imaginar el movimiento de la mano derecha o izquierda. En concreto, se realizará la programación de la obtención de características con CSP y la clasificación inicialmente con LDA, luego se probarán otros clasificadores. Se realizará la comprobación de resultados de probabilidad de acierto en entrenamiento online y testeo, así como del tiempo máximo que se puede tardar en clasificar. Se quiere verificar qué clasificador es mejor para el MI-BCI. Además, se realizarán diferentes pruebas para comprobar cómo afectan a los resultados cambiar algunas condiciones y parámetros en la clasificación.

1.3 Organización de la memoria

La memoria de este trabajo se estructura de la siguiente forma:

En el capítulo 1 se motivan las diferentes aplicaciones de las interfaces cerebro-ordenador y se describen la estructura de la memoria y los objetivos del trabajo.

En el capítulo 2 se realiza una breve compresión teórica sobre qué consiste el funcionamiento de un sistema BCI, además de los tipos de sistemas que hay según la señal cerebral a detectar y la interfaz física (hardware) utilizada para obtenerla. En el primer apartado se explica de forma simple como se producen las señales cerebrales y cuales son de interés para los BCI. En el segundo apartado aparece una explicación sobre el hardware usado en estos sistemas; las categorías en que se puede clasificar los BCI; una resumida explicación sobre el BCI que se va a desarrollar en este trabajo (MI-BCI); y cómo es el procedimiento por el cual se adquirió la base de datos a utilizar; además de comentar algunos datos de interés sobre el software que se usó para programar la interfaz (MATLAB).

Los capítulos 3, 4 y 5 corresponden a la explicación de las tres etapas necesarias a realizar en un BCI: preprocesamiento, obtención de características y clasificación.

El capítulo 3 explica el modelado de la señal de entrada (EEG) y en qué consiste el preprocesamiento.

En el capítulo 4 se explican diferentes formas que existen de obtener características de la señal orientadas a sistemas BCI. El método CSP se explica con más detalle que los otros, pues es el único que se programó en la interfaz realizada en este proyecto.

En el capítulo 5 se explican diferentes métodos para realizar el aprendizaje automático (Machine Learning), lo cual permite asignar una clase a la señal de entrada según sus características.

En el capítulo 6 se muestran los resultados experimentales de este trabajo. Se hicieron pruebas con diferentes tipos de clasificadores: LDA, QDA, Naive Bayes, KNN, SVM, Decision Trees, Logistic Regression y Ensemble Classifiers. Se pueden programar de tres formas en MATLAB: escribiendo el código paso por paso cada una de las operaciones a realizar; utilizando las funciones especiales de MATLAB, en las que ya viene internamente programado las operaciones y se le puede especificar algunas variaciones del algoritmo poniéndolas como entrada en la función; y, por último, utilizando una aplicación de MATLAB, que permite realizar el entrenamiento y comprobar su funcionamiento con validación cruzada. En el primer apartado muestra los resultados con un solo usuario en una sesión y en el segundo apartado con un conjunto de usuarios, siendo dos sesiones por usuario. Se ha organizado la memoria de forma que cada prueba realizada sea un subapartado del clasificador que corresponda, los cuales se muestran a su vez como subapartados de los dos apartados principales ya comentados (un usuario y varios usuarios).

En el capítulo 7 se hace una explicación de la estructura de la programación utilizada y qué archivo hay que ejecutar según el clasificador que se quiera usar, además se comenta en cada uno cuántos ensayos se hizo que esperaran antes de empezar a entrenar el clasificador. El LDA realizado paso por paso, sin usar funciones especiales de MATLAB, se encuentra explicado de manera más exhaustiva, ya que era el principal clasificador que se iba a usar en el proyecto.

En el capítulo 8 se expresan unas conclusiones sobre este trabajo y posibles líneas futuras a realizar.

1.4 Conclusiones

En este capítulo se ha hecho una breve definición del concepto de interfaz y se ha mostrado lo que motiva al desarrollo de estos sistemas, poniendo de ejemplo diferentes aplicaciones: ayudar a personas con discapacidad; para uso en prótesis o exoesqueletos; crear canales sensoriales artificiales; rehabilitación neurológica; arte y ocio; domótica y teleoperación; videojuegos y realidad virtual.

También se ha planteado los objetivos principales del trabajo y se ha explicado cómo se estructura la memoria de éste.

2 MARCO TEÓRICO Y PLANTEAMIENTO

"La vida, la naturaleza, la humanidad solo son bellas palabras cuando son transformadas por un cerebro creador."

Edmond Jaloux

Se repasará los conceptos necesarios sobre el cerebro humano y los tipos de BCI. Además, se hará el planteamiento de cómo se obtienen los datos de las señales cerebrales en el caso del sistema MI-BCI.

2.1. Señales del cerebro humano

Para entender los sistema BCI de imaginación motora en el que se centra este trabajo, es importante tener nociones básicas de cómo funciona el cerebro humano y las diferentes señales importantes que intervienen.

2.1.1 Resumen del cerebro humano

El cerebro humano puede verse como un sistema distribuido en el que cada parte del él se encarga de determinadas funciones y, dependiendo del sistema BCI que se vaya a utilizar, se leerán las señales de distintas partes de éste [33].

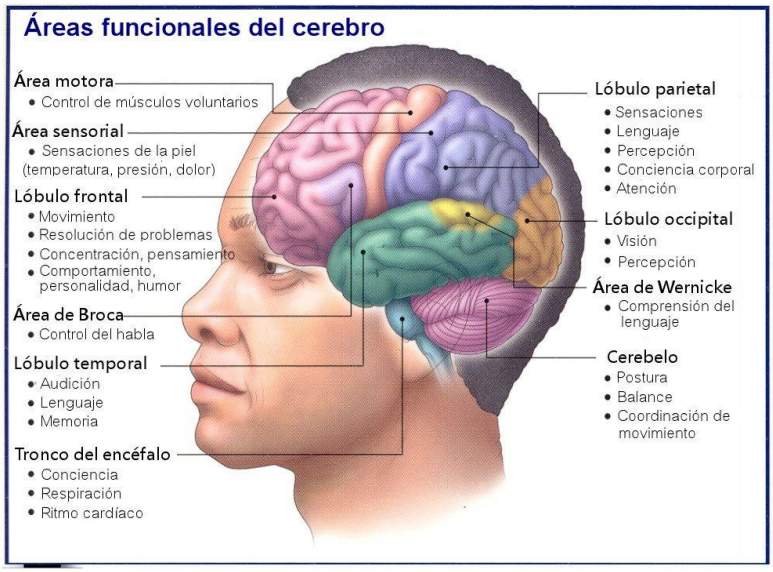


Figura 2-1. Áreas del cerebro. Fuente: [34].

El hemisferio derecho dirige el lado izquierdo del cuerpo y está asociado con el pensamiento matemático y científico, el procesamiento secuencial de la información y alberga las capacidades de escribir y hablar. El hemisferio izquierdo dirige el lado derecho del cuerpo y se asocia con la imaginación, la creatividad, la orientación espacial, las emociones, la intuición y la multitarea [33].

A su vez, el cerebro se divide en diferentes áreas según su función. En la Figura 2-1 se puede apreciar donde se encuentran y qué se encarga de gestionar cada parte. Con respecto al BCI, las zonas más relevantes serían:

- El córtex motor es la parte que controla los movimientos motores. Ésta es la parte que se monitoriza en los sistemas MI-BCI.
- El lóbulo frontal es relevante en los sistemas BCI de deletreo P300 porque es una de las áreas donde se genera la onda P300.
- El lóbulo occipital es de interés para los sistemas BCI basados en el paradigma SSVEP.

Figura 2-2. Señales del cerebro. Fuente: [35].

En el cerebro se producen diferentes señales eléctricas. En la Figura 2-2 se pueden apreciar algunas de las diferentes ondas del cerebro con sus correspondientes frecuencias. "*Las ondulaciones de los trazos eléctricos se denomina Ondas Cerebrales. Las ondas registradas se deben a los cambios de polarización que ocurren en las terminaciones sinápticas corticales excitatorias e inhibitorias de las neuronas"* [36]. Para comprender mejor esto hay que tener claro el concepto de neurona.

La unidad básica que forma el sistema nervioso son las células llamadas neuronas. Como se ve en Figura 2-3, este tipo de célula está formada por el soma, que recibe señales de un conjunto de protuberancias conocidas como dendritas. Los axones son largas fibras que transmiten señales eléctricas, denominadas potenciales de acción, y funcionan de forma que da lugar a una comunicación binaria. El axón se extiende desde el soma y es terminado por las terminales o botones sinápticos, que transmiten la señal al siguiente elemento, generalmente la dendrita de otra neurona. Una conexión sináptica puede excitar o inhibir la actividad en una dendrita de la siguiente neurona. Si se alcanza un umbral, la neurona objetivo liberará un impulso de origen electroquímico a través de su axón, transmitiendo la señal a otras neuronas conectadas. "*En el cerebro hay miles de millones de neuronas*" [37], y aunque no es posible ni práctico observar la actividad individual de cada célula, existen técnicas para observar los potenciales evocados que producen a un nivel más global [38]- [33].

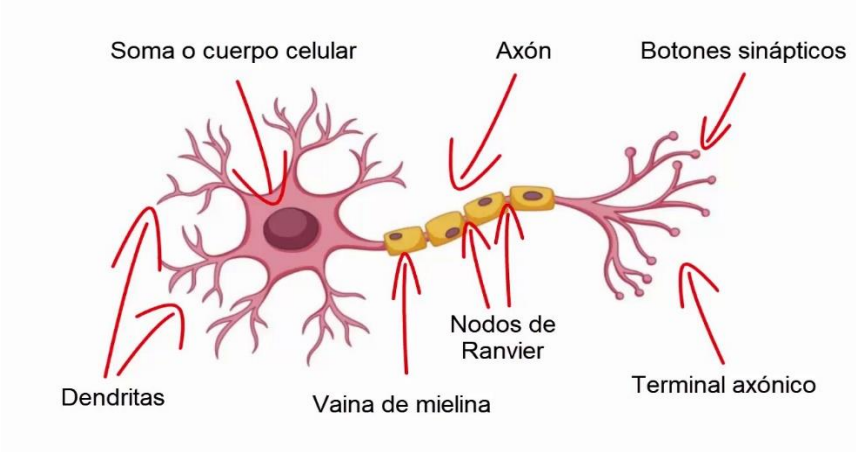


Figura 2-3. Neurona. Fuente: [39].

2.1.2 Señales Encefalográficas (EEG)

"Las señales eléctricas producidas por el cerebro son generadas por la diferencia de potencial en la membrana celular de las neuronas y este proceso es la base del funcionamiento de nuestro sistema nervioso. El registro de estas bioseñales es lo que se conoce como Electroencefalograma (EEG) y los ritmos de la actividad neuronal constituyen un lenguaje de comunicación propio de las neuronas" [25]. El EEG es el método más fácil para registrar y experimentar con las señales eléctricas procedentes de la actividad del cerebro humano.

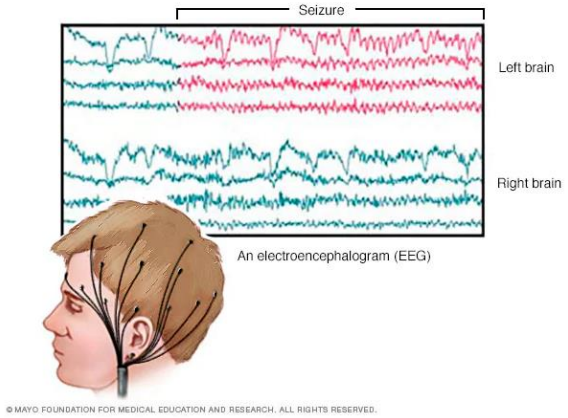


Figura 2-4. "*El electroencefalograma muestra resultados que permiten ver los cambios en la actividad cerebral*". Fuente: [40].

El método del EEG, obtención de un registro encefalográfico[4], consiste en colocar electrodos sobre el cuero cabelludo del usuario para medir el campo electromagnético producido por el cerebro, es decir, el que se genera a través de la actividad de las neuronas. En otras palabras, para registrar la actividad eléctrica (voltaje) generada por el cerebro. Los millones de potenciales de acción que se disparan constantemente producen un campo electromagnético que puede medirse [33]- [36]. "*El ancho de banda del EEG va de DC-100 Hz, con el mayor porcentaje de potencia*[5] *entre 0.5 y 60 Hz. Las amplitudes sobre el cráneo oscilan entre 2 y 100 µV*" [41].

[4] Encéfalo: "Parte central del sistema nervioso de los vertebrados, encerrada y protegida en la cavidad craneal y formada por el cerebro, el cerebelo y el bulbo raquídeo" [260].
[5] "El potencial en reposo de una neurona es de entre -60 y -70 mV" [41].

El objetivo de las mediciones es localizar que parte del cerebro está activa. Las señales electrofisiológicas son ruidosas y la resolución espacial es baja. Si no se colocan los electrodos dentro de la cabeza, es más difícil de percibir de donde viene la señal. *"Los potenciales de acción que registran en el EEG, aparecen en forma de ondas con una frecuencia que van de 1 a 100 ciclos por segundo, y con una amplitud que oscila desde 0 a 300 microvoltios"* [36]. Las señales son débiles en la escala de microvoltios, se les debe amplificar. "*La actividad eléctrica promedio del encéfalo es casi la centésima parte de la actividad cardíaca. Por esta razón se requieren de amplificadores de gran estabilidad y sensibilidad, para obtener un registro sin distorsión de la actividad encefálica"* [36] . En resumen, es bastante menor la magnitud de la actividad eléctrica registrada en el EEG (µV), en comparación con la actividad eléctrica producida por una única neurona (mV). Esto es consecuencia de que la señal es filtrada y atenuada al pasar por las diferentes capas de tejido desde las neuronas hasta los electrodos de registro [41].

Además, las señales de EEG suelen estar contaminadas por artefactos que disminuyen la calidad de la señal. La contaminación por artefactos se refiere a la producida por los movimientos de la persona, como parpadeo, respirar o tragar [33].

Los electrodos de tipo secos, con gel o con solución salina son los más comunes. El gel sirve para adaptar la impedancia entre el cuero cabelludo y el electrodo. Los electrodos secos tienen un rendimiento tan bueno como los húmedos (con gel) con respecto a la relación SNR (señal/ruido), pero son más sensibles al movimiento [33].

Un problema para tener en cuenta es que dependiendo del usuario y la sesión los resultados variarán. Cada persona es distinta y, además, es complicado colocar los electrodos en la misma posición. Por otra parte, debido a los movimientos o secado del gel, la señal no será estacionaria dentro de la misma sesión [33].

Con respecto a las señales concretas que se quieren medir, se eligen según el tipo de sistema BCI a realizar. Hay varios tipos de sistemas BCI, los cuales se fundamentan en diferentes procesos fisiológicos del cerebro. Los investigadores tratan de encontrar patrones en la electrofisiología del cerebro que puedan ser medidos y al mismo tiempo puedan ser activados voluntariamente por la intención del usuario. La decisión de elegir un determinado tipo de sistema para controlar un dispositivo puede depender del propio dispositivo o de las limitaciones del usuario [38]- [33].

En el cerebro humano se han observado algunas señales características que en circunstancias particulares pueden utilizarse potencialmente en un BCI. Próximamente se mencionarán algunos de los paradigmas de BCI más usados [38]- [33].

2.1.3 P300

Hay BCI basado en potenciales evocados P300, se basa en la actividad cerebral que se produce cuando una persona no sabe cuándo va a ocurrir un evento, pero lo está esperando activamente [33].

Aproximadamente 300 ms después de haberse producido ese estímulo visual o auditivo poco usual, el momento en que ocurre es aleatorio, se produce el potencial P300, el cual es un pico de amplitud que aparece en el EEG. "*Habitualmente, se presenta al usuario una tanda de estímulos de los que solo unos pocos tienen relación con la intención del usuario. De esta forma, los estímulos de interés, al ser infrecuentes y estar mezclados con otros estímulos mucho más comunes, provocan la aparición de un potencial P300 en la actividad cerebral del usuario. Dicho potencial se observa principalmente en las zonas central y parietal del córtex cerebral"* [28].

Figura 2-5. Ejemplo sistema deletreo P300. Fuente: [42].

Un ejemplo de su uso es el deletreador P300, el cual consiste en presentar un tablero o pantalla con el alfabeto o comandos en forma de tabla. Se resaltan de forma aleatoria directamente las celdas o señalizando filas y columnas. El método de indicar los elementos de la tabla se puede hacer de varias maneras [38].

El usuario se concentrará en un carácter o comando, mientras se produce el resaltado aleatorio. Cuando se resalta la posición que contiene la letra o comando en la que el usuario se está concentrando, se dispara la onda P300 después de 300ms. El proceso se repite unas cuantas veces antes de tomar una decisión para compensar las imprecisiones [38].

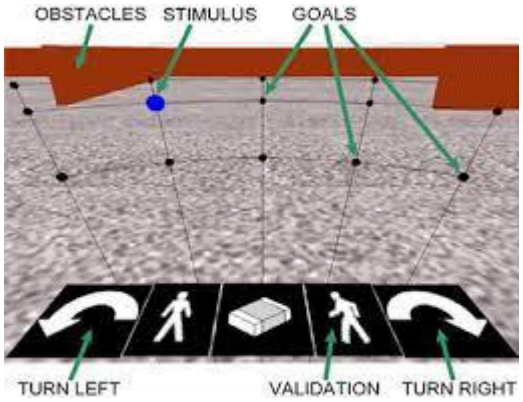


Figura 2-6. Ejemplo P300 con comandos. Fuente: [43].

Las limitaciones de velocidad del sistema son un problema, sin embargo, el principal problema sigue siendo la precisión, ya que el número de decisiones incorrectas puede ser muy alto. A pesar de ello, es de gran utilidad para algunas personas discapacitadas.

Se puede consultar el desarrollo de un BCI P300 en el artículo [44]. También puede resultar de interés consultar los artículos [45]- [43].

2.1.4 SSEVP

Hay "*BCI basadas en potenciales evocados visuales de estado estable (SSVEP*)" [28] . El SSVEP se refiere a un fenómeno electrofisiológico que se *"detectan en el EEG registrado sobre la zona visual del córtex cerebral tras haberse aplicado un estímulo visual*" [28] a la persona (por ejemplo, LEDs o regiones en una pantalla). Estos potenciales se hacen inalterables si la tasa de presencia del estímulo está por encima de 6 repeticiones por segundo (6 Hz). Una persona al enfocar su mirada en "*una imagen que parpadea a una frecuencia determinada, se puede*

detectar esa frecuencia analizando el espectro de la señal EEG, ya que aumenta la amplitud del SSVEP en la frecuencia de la imagen parpadeante y en su segundo y tercer armónico" [28] .

En otras palabras, cuando una persona mira una luz parpadeante en el rango de 1-100Hz, esa misma frecuencia puede ser observada y detectada en ondas electromagnéticas en la corteza visual [33]. Un ejemplo de su uso sería poner varias luces en distintos lugares y parpadeando a diferentes frecuencias, permitiendo así saber hacia dónde está mirando el usuario. El principal beneficio de este procedimiento es que no precisa un entrenamiento, solo que el usuario mire las luces [38]. Sin embargo, hay que tener en cuenta que la frecuencia espacial de un estímulo estructurado está relacionada con rasgos individuales como la agudeza visual o la edad. También existe una diferencia significativa en la magnitud de las SSVEP entre la estimulación del parpadeo del centro (fóvea centralis) frente a la periferia del campo visual. Las condiciones ambientales también influyen en el rendimiento del BCI, por ejemplo, el brillo y la frecuencia de la pantalla, la distancia a la pantalla, entre otras. Las magnitudes de SSVEP están moduladas por los estados de atención de los sujetos [46]. Por otra parte, la eficiencia del BCI puede depender crucialmente de la selección de la frecuencia de parpadeo. En el artículo [46] cuenta que se puede crear un algoritmo adaptativo en bucle cerrado para averiguar las frecuencias más favorables para cada persona. Igualmente se recomienda consultar el documento [47].

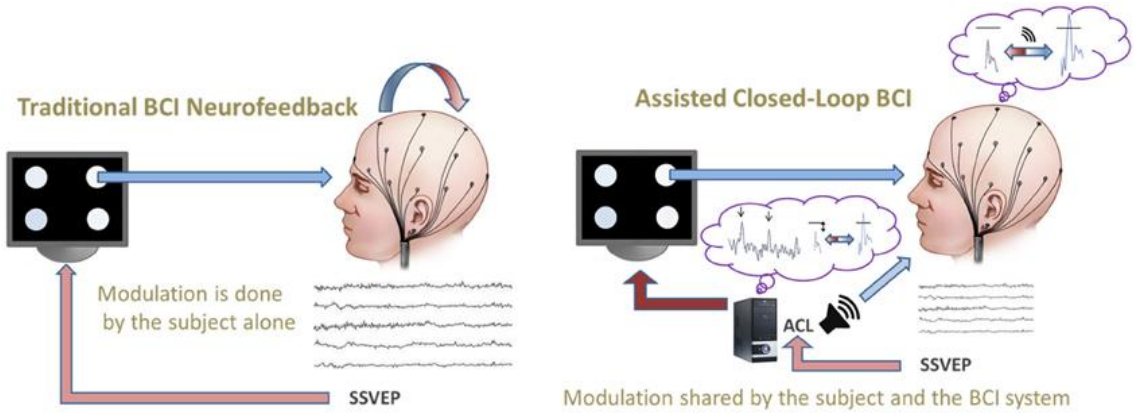


Figura 2-7. "*Comparación de un neurofeedback BCI tradicional (izquierda) frente al paradigma de bucle cerrado asistido (derecha) que informa tanto al sujeto (sobre su actividad cerebral en relación con el objetivo del BCI) como al sistema (sobre las especificidades del sujeto). En este ejemplo, el bucle cerrado asistido proporciona información en tiempo real al sistema sobre las frecuencias de parpadeo más eficaces y al sujeto sobre la distancia real al umbral predefinido mediante una retroalimentación auditiva continua*". Fuente: [46] .

2.1.5 N200

Hay BCI de potencial evocado en movimiento (mVEP-BCI). El N200 es un potencial evocado que se produce cuando el usuario mira un objeto en movimiento, se captan en el EEG grabado sobre la zona visual del córtex cerebral y aparece 200ms después del estímulo aproximadamente. El ejemplo principal de uso es el deletreador N200, es igual que el N300, pero las celdas se resaltan con un marcador en movimiento [38]. Se puede consultar más información del P300 y N200 en [48].

2.1.6 Imaginación motora

Los BCI basados en imaginación motora (o imágenes mentales motoras) consisten en pensar realizar un movimiento (como mover mano derecha o izquierda) u otras tareas mentales. Este BCI se le denomina Interfaz Cerebro-Ordenador de Imaginación Motora (MI-BCI).

El córtex motor controla los movimientos voluntarios del cuerpo y cada parte de la corteza motora controla una parte diferente del cuerpo (Figura 2-8), siendo difícil establecer los límites exactos entre las diferentes partes. Se sabe que el córtex motor que se sitúa en el hemisferio izquierdo dirige la parte derecha del cuerpo, y el izquierdo la derecha.

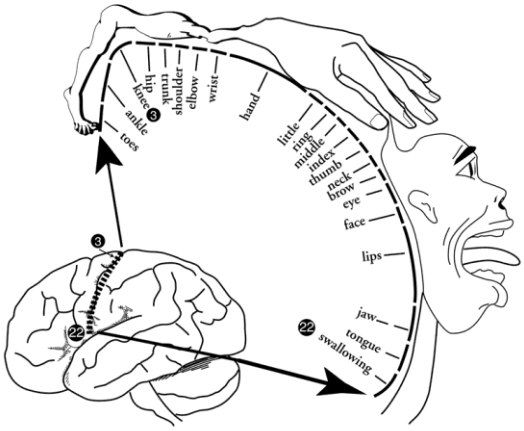


Figura 2-8. Córtex motor. Fuente: [49].

La creación de imágenes mentales motoras es una destreza cognitiva[6], la cual consiste en imaginar la realización de una actividad motora, en ausencia de la activación real del mismo. "*La capacidad de generar imágenes mentales es mayor en los sujetos con mayores niveles de actividad física que en los sujetos sedentarios*" [50]. "*Este tipo de tareas mentales producen cambios en la amplitud de los ritmos sensoriomotores µ (8-12 Hz) y ß (16-24 Hz)"* [28], registrados sobre el córtex somatosensorial y motor, es ahí donde se producen. Estos ritmos presentan algunas diferencias cuando la realización del movimiento es real o imaginaria. Lo que hay que tener más presente es que, durante el proceso de imaginación motora, ocurren dos eventos que resultan de gran interés. Cuando alguien realiza un movimiento motor o lo imagina, estos ritmos se debilitan (pudiendo desaparecer), produciéndose el llamado **Evento Relacionado con la Desincronización (ERD)**. El fenómeno contrario, asociado al aumento de potencia, se le llama **Evento Relacionado con la Sincronización (ERS)**. Estos dos fenómenos son los que permiten ayudar a diferenciar entre un movimiento y otro [33]. En otras palabras, se producen dos fenómenos cuando se realiza un movimiento imaginario: El primero es un ERD en el área correspondiente al movimiento y el segundo es un ERS en la ubicación de la corteza relacionada con la otra clase, no hay actividad motora en esa región [38]. Se puede consultar un estudio sobre estos fenómenos en el artículo [51]. También puede resultar de interés consultar [52].

[6] *"Se conoce como habilidades o capacidades cognitivas a las aptitudes del ser humano relacionados con el procesamiento de la información, es decir, los que implican el uso de la memoria, la atención, la percepción, la creatividad y el pensamiento abstracto o analógico"* [261].

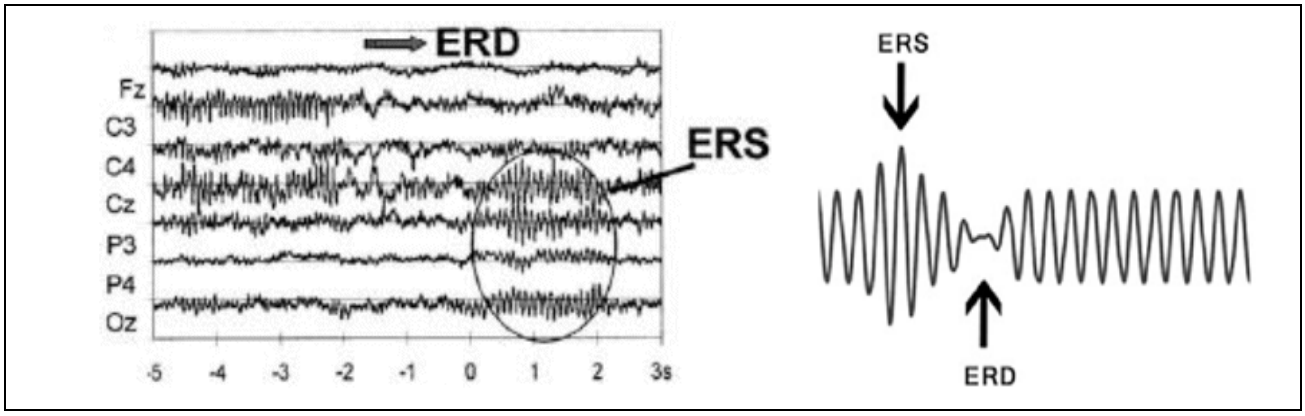


Figura 2-9. Ejemplo de EEG durante el movimiento del dedo derecho [53]. "*Muestra el comportamiento de señales cerebrales en distintos electrodos al realizarse una actividad motora".* Puede apreciarse que las señales provenientes de cada electrodo están desincronizadas (ERD) antes del tiempo 0 s., *"a partir del tiempo 0 s. la actividad de las señales de los electrodos entra en un periodo de sincronización (ERS) de forma que las mismas actúan de manera similar en cuanto a frecuencia. A partir de los 2 s. aproximadamente, las señales de todos los electrodos nuevamente entran en un estado de desincronización*" [54]. Fuente: [54]- [55].

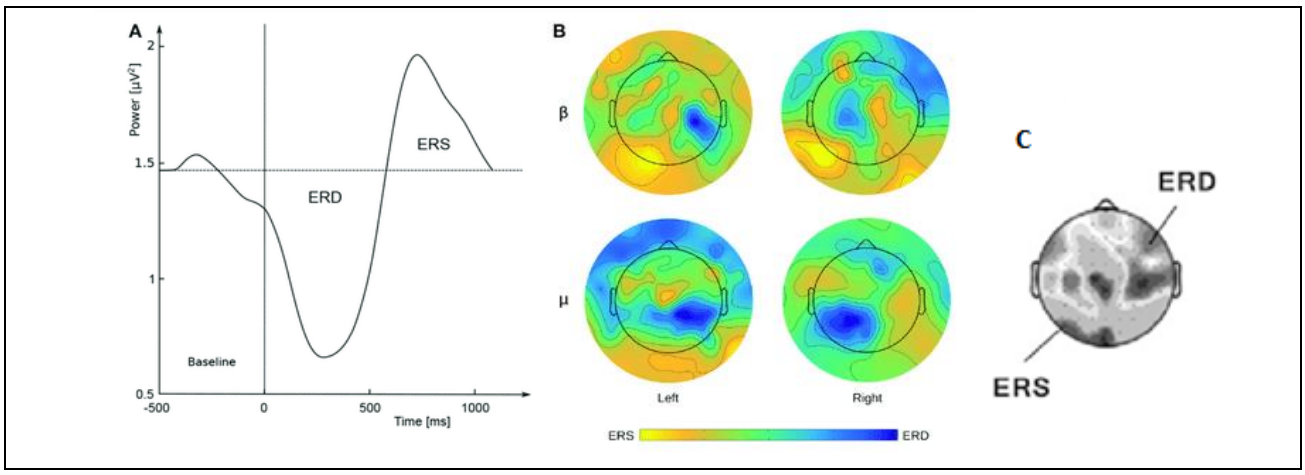


Figura 2-10. A) Ejemplo de transcurso de tiempo ERD/ERS, se representa la potencia con respecto al tiempo. *"La línea vertical indica el inicio de la imaginación del movimiento*. B) *Mapas de distribución de la intensidad de la señal (μV^2) en el cuero cabelludo (μ = 8-12 Hz; β = 18-30 Hz) durante el movimiento imaginario de la mano izquierda o derecha"* [56]. C) *"Mapa topográfico de los componentes del ERD/ERS en una imaginación motora del movimiento de una mano*" [55]. Fuente: [56] - [55].

El objetivo final es dar órdenes al ordenador a través de la imaginación de los movimientos motores, de forma que el movimiento de cada parte del cuerpo se asocia a un comando diferente [33]. La dificultad está en cómo diferenciar los movimientos de las imágenes mentales, pues son señales débiles con ruido. Este trabajo se centrará en este tipo de BCI.

2.1.7 Resumen de señales del cerebro humano

"Para entender estos sistema es importante tener nociones básicas de cómo funciona el cerebro humano y las diferentes señales importantes que intervienen."

Cada parte del cerebro se encarga de unas funciones distintas. Dependiendo del sistema BCI que se quiera hacer se leerán las señales cerebrales de una zona u otra. En el caso de imaginación motora, interesa leer las señales que se producen en el córtex motor.

En el cerebro humano se han observado algunas señales características que en circunstancias particulares pueden utilizarse potencialmente en un BCI. Se trata de encontrar patrones en la que puedan ser medidos y al mismo tiempo puedan ser activados voluntariamente por la intención del usuario. Destacan los paradigmas de P300, SSEVP, N200 y la imaginación motora.

Se buscan variaciones en las señales registradas. En el caso de imaginación motora, se buscan los eventos llamados **ERS** y **ERD** en la señal EEG**.** Cuando alguien realiza un movimiento motor o lo imagina, los ritmos sensoriomotores se debilitan, se produce ERS. El fenómeno contrario es ERD.

"Las ondas que se crean en el cerebro son debidas a la actividad eléctrica que se produce en las neuronas. Aunque no es práctico observar la actividad individual de cada una, existen técnicas para observar su actividad a nivel global. Para el MI-BCI, "*se registrará estas bioseñales con lo que se conoce como Electroencefalograma (EEG)".* Consiste en colocar electrodos sobre el cuero cabelludo del usuario para medir el campo electromagnético producido. El objetivo de las mediciones es localizar que parte del cerebro está activa."

El principal inconveniente es que la magnitud de la actividad eléctrica registrada en el EEG es bastante menor en comparación con la actividad generada por una única neurona, se debe a que la señal es filtrada y atenuada al pasar por las diferentes capas de tejido desde las neuronas hasta los electrodos de registro. Deberá ser amplificada.

Además, "hay que tener presente que la señal tendrá artefactos y que dependiendo del usuario y la sesión los resultados variarán debido a que la colocación de los electrodos puede variar. Por otra parte, debido a los movimientos o secado del gel, la señal no será estacionaria dentro de la misma sesión."

2.2 BCI

El objetivo de una interfaz cerebro-ordenador es monitorear la actividad cerebral y hacer una predicción sobre la intención del usuario, el cual estará pensando en la realización de una acción entre un conjunto de posibles comandos. Físicamente posee tres partes principales: los sensores, el procesamiento de la señal y la aplicación [1].

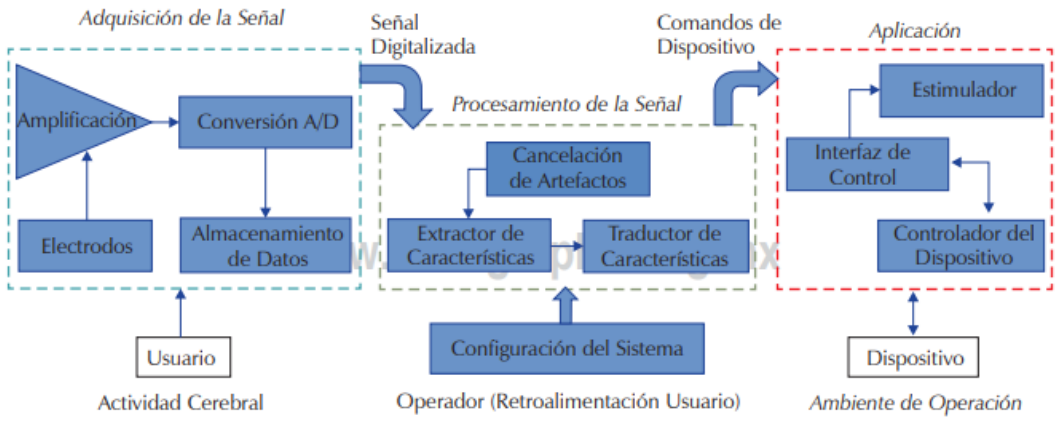


Figura 2-11. Elementos de un sistema interfaz cerebro-ordenador. Fuente: [21].

La interfaz debe poder clasificar las señales del cerebro, asignando un comando (un movimiento en caso de MI-BCI) según la señal recibida. Este problema se resuelve realizando tres pasos: preprocesamiento de datos, obtención de características y entrenamiento del clasificador.

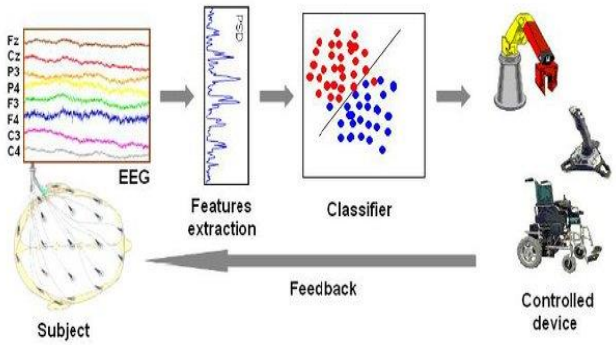


Figura 2-12. Este es el proceso habitual de una BCI para poder controlar dispositivos electrónicos. Fuente: [57].

2.2.1 Interfaces físicas y tipos de sistemas BCI

La actividad de las neuronas es medida por las interfaces cerebro-ordenador, para obtener la señal que será procesada y convertida en comandos. Se pueden diferenciar tres tipos de dispositivos según el procedimiento de obtención de la señal [3]:

- **Dispositivos invasivos**: Un implante neural, es un dispositivo tecnológico que se conecta directamente al cerebro, por tanto, se requiere una intervención quirúrgica. Normalmente "*se coloca en la superficie del cerebro o conectado a la corteza cerebral. El sensor puede penetrar la corteza cerebral de forma que mide la actividad eléctrica de neuronas individuales o puede colocarse en la superficie del córtex para medir la actividad eléctrica de grupos de neuronas. Consecuentemente, la señal obtenida es muy nítida*" [58]. El objetivo principal de estos implantes es reemplazar la parte del cerebro que ha dejado de funcionar correctamente. Un ejemplo de productos de este tipo son los dispositivos BrainGate.

- **Dispositivos parcialmente invasivos**: Cuando la implantación de electrodos es sobre la superficie cerebral se le llama semi-invasivos o parcialmente invasivos [1]. "*Son grabadores externos que detectan señales de dispositivos implantados superficialmente. Un ejemplo es la electrocorticografía (ECoG), que registra la actividad del cerebro a través de una rejilla de electrodos que se incrusta quirúrgicamente*" [59].

- **Dispositivos no invasivos**: "*La actividad eléctrica se mide en la superficie del cuero cabelludo. La señal obtenida es la superposición de todas las neuronas del cerebro (no de neuronas individuales o grupos localizados) y tiene una resolución más pobre debido a que el cráneo del usuario debilita y distorsiona las señales generadas por las neuronas"* [58]. Varias empresas como NeuroSky y Emotiv desarrollan interfaces no invasivas basadas en señales EEG [24].

"Existen diferentes métodos para registrar la actividad cerebral: EEG, electrocorticografía (ECoG), magnetoencefalografía (MEG), tomografía por emisión de positrones (Positron Emission Tomography, PET) o imágenes de resonancia magnética funcional (functional Magnetic Resonance Imaging, fMRI). La ECoG es una técnica invasiva, es decir, requiere de una intervención para la colocación de electrodos en la superficie cortical. Por su parte, las técnicas MEG, PET y fMRI requieren de instalaciones y equipos de alto coste. Por ello, el método más empleado para el registro de la actividad cerebral en sistemas BCI es el EEG, ya que se trata de una técnica sencilla, no invasiva, portátil y de bajo coste" [28].

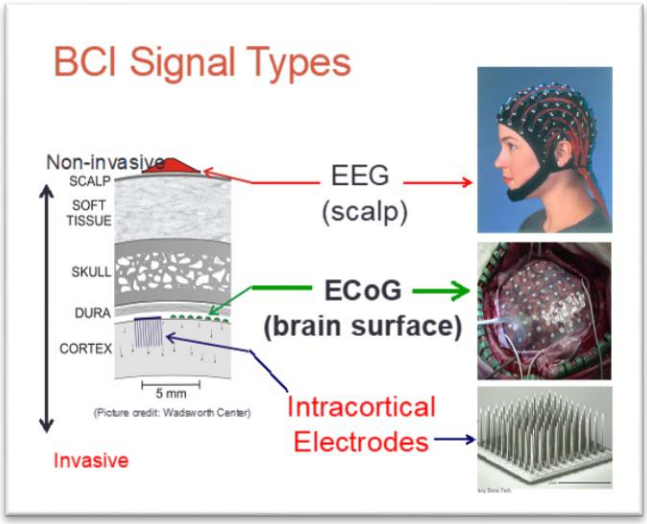


Figura 2-13. Si el sistema BCI es no invasivo tiene que detectar la señal después de atravesar el cráneo y la piel. Fuente: [60].

Según la naturaleza de la señal a medir, los sistemas BCI se pueden clasificar en dos categorías:

- Endógenos: "*Dependen de la capacidad del usuario para controlar su actividad electrofisiológica, como puede ser la amplitud del EEG en una banda de frecuencia específica sobre un área concreta del córtex cerebral y requieren un periodo de entrenamiento*" [28]. Un ejemplo son los basados en imaginación motora (ritmos sensoriomotores) o en potenciales corticales lentos (Slow Cortical Potentials, SCP). Los SCP son cambios lentos de voltaje generados sobre el córtex cerebral. "*Los SCP negativos se asocian de manera habitual con el movimiento*" y otras funciones que implican un arousal[7] [28]. Como se explicó con anterioridad, las imaginación motora se basa en imaginar un movimiento, "*la imaginación de tareas mentales concretas produce cambios en la amplitud de los ritmos μ (8-12 Hz) y ß (16-24 Hz) sobre la zona somatosensorial y motora del córtex cerebral*" [61].

- Exógenos: "*Dependen de la actividad electrofisiológica evocada por estímulos externos y no necesitan de una etapa intensiva de entrenamiento"*. Un ejemplo son los basados en potenciales evocados visuales de estado estable (Steady State Visual Evoked Potentials, SSVEP) o en potenciales evocados P300 [61]- [28].

[7] Activación cortical. *"La excitabilidad cortical es una medida de la respuesta de la corteza a la estimulación"* [262].

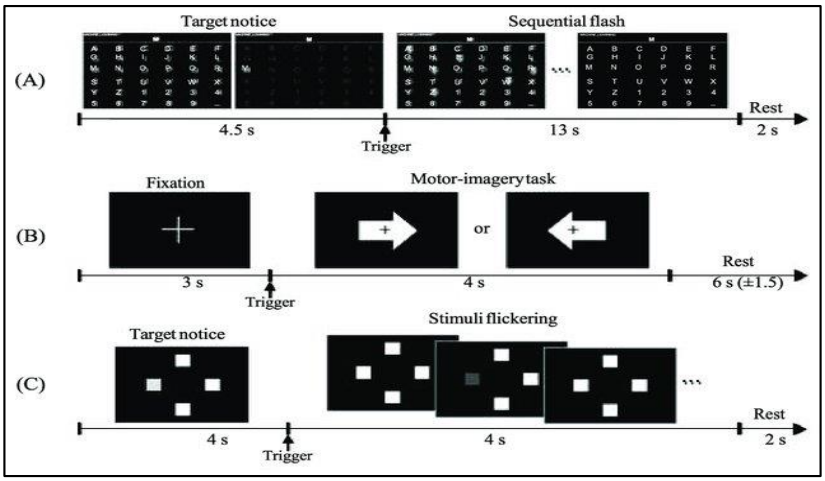


Figura 2-14. Diseños experimentales para tres paradigmas BCI: Los paradigmas de deletreo 6×6 ERP (A), MI de clase binaria (B) y SSVEP de cuatro frecuencias objetivo (C). Se realizan de forma secuencial a la hora de recoger datos de los sensores del casco. Fuente: [62].

En conclusión, en este trabajo se trata de un MI-BCI no invasivo basado en imaginación motora. La interfaz física usada para este trabajo es el EEG, debido a su bajo coste relativo, la portabilidad y ser una técnica no invasiva.

2.2.2 MI-BCI

El sistema MI-BCI consistirá en lo siguiente:

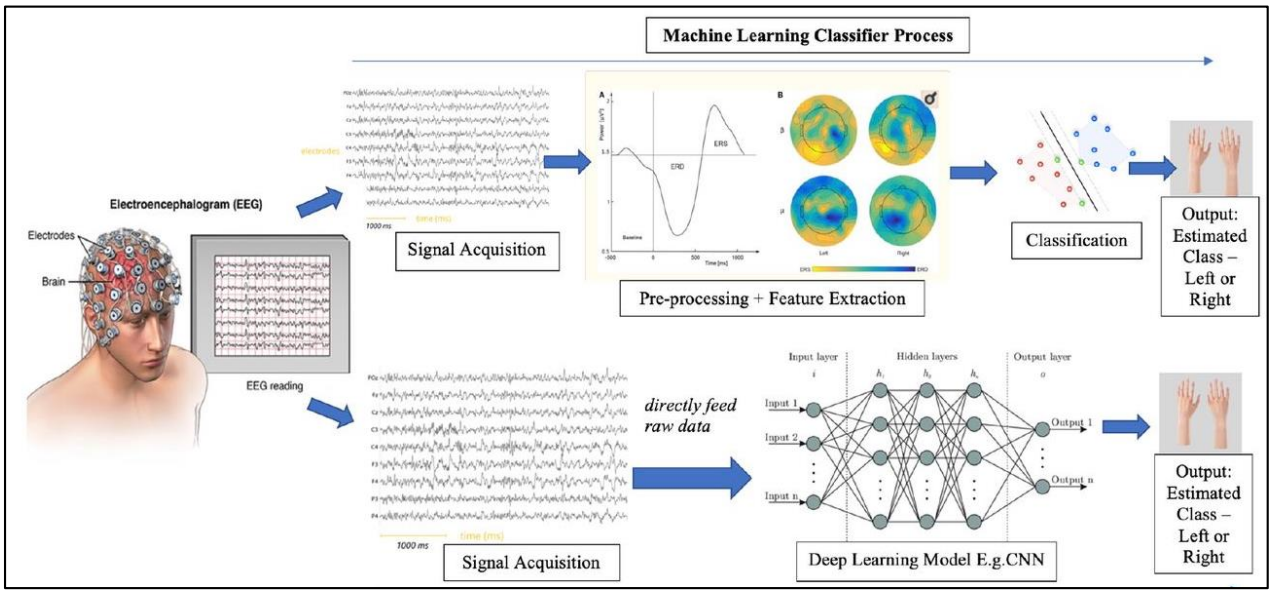


Figura 2-15. Esquema de proceso Machine Learning en MI-BCI, probándose dos diferentes clasificadores. Fuente: [19].

Para utilizar el sistema MI-BCI se sitúa en la cabeza del usuario un casco o diadema con electrodos, las señales que leen los sensores se pasan a un amplificador. Ejemplos de cascos se encuentran en la Figura 2-16 y la Figura 2-17. Un ejemplo de amplificador de señal para BCI es el de la Figura 2-18.

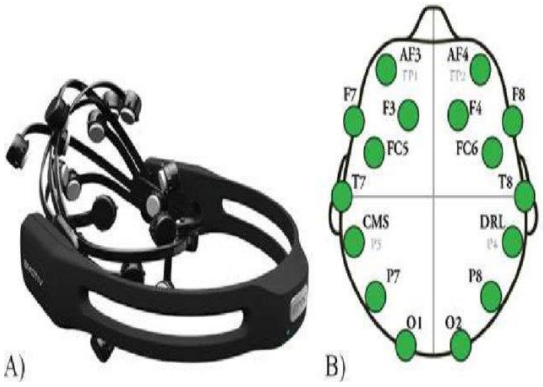


Figura 2-17. (a) Emotiv EEG system. (b) Electrodes topography. Fuente: [63].

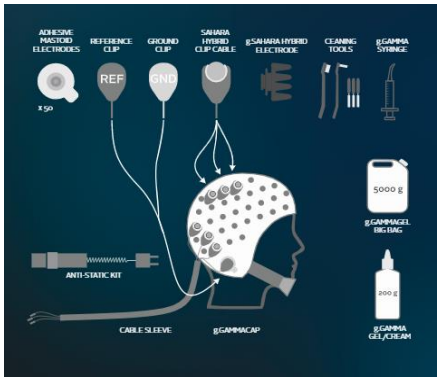


Figura 2-16. Ejemplo casco BCI. Fuente: [64].

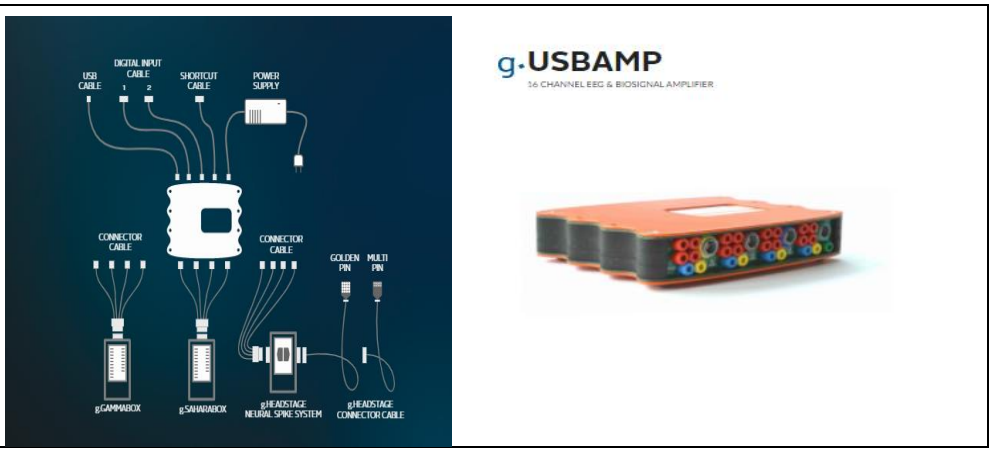


Figura 2-18. Amplificador señal BCI. Fuente: [64].

El usuario estaría sentado en una posición cómoda en la que pueda estar quieto durante el periodo de tiempo en que se utiliza el sistema. El casco y la posición de los sensores no pueden cambiar dentro de una sesión porque, como se ha dicho anteriormente, estos sistemas se basan en información espacial y pueden verse seriamente afectados por pequeñas variaciones en la posición o el comportamiento de los sensores.

En este proyecto el dispositivo de salida del sistema MI-BCI será una pantalla. A continuación, se explicará en qué consistió el proceso de obtención de datos. Cabe mencionar que existen otros paradigmas y variantes al sistema realizado.

El proceso se basa en hacer varios ensayos con un usuario, para entrenar el algoritmo (clasificador) que identificará qué movimiento se ha hecho, para aprender a discriminar entre las órdenes. Cada vez que un usuario haga uso del sistema se llamará sesión, habiendo una fase de entrenamiento en cada sesión, componiéndose cada sesión de varios ensayos. Se mostrará por pantalla al usuario indicaciones de si debe pensar en mover mano derecha o izquierda, haciéndose una pausa entre cada indicación para que el usuario pueda dejar de pensar en el movimiento. Cada vez que el usuario realiza uno de los movimientos indicados, es lo que se denomina ensayo. Cada ensayo que se graba se guardará, además, con la etiqueta de la acción que se le indicaba al usuario que hiciese, es decir, la clase real que se corresponde en cada ensayo.

Estas indicaciones suelen ser flechas para indicar la mano y una cruz para indicar cuándo se realiza una pausa, siendo el tiempo que está la flecha entre unos 3 o 6 segundos normalmente. Un ejemplo de esto se muestra en la Figura 2-19.

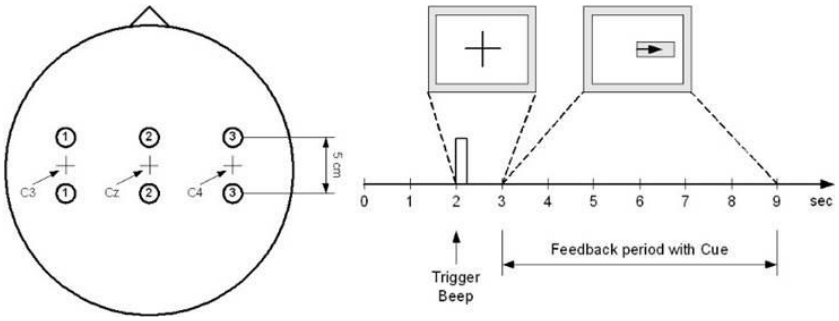


Figura 2-19. Ejemplo de obtención de señal de mano derecha con electrodos en un ensayo. Fuente: [65].

Si se hace en tiempo real la clasificación, se le puede indicar a la vez al usuario como se está estimando la clase del movimiento que está haciendo, para poder así saber si está pensando correctamente. Si no es en tiempo real, lo que se hace es guardar la sesión del usuario y luego se entrena el clasificador con el conjunto de todos los ensayos.

Para la comparación entre diferentes algoritmos, pues hay diferentes formas de extraer características y realizar clasificación, una posible medida usada es el porcentaje de aciertos. Se puede comprobar qué algoritmo ha estimado correctamente más veces. Otra medida interesante es el tiempo de entrenamiento y el tiempo en obtener resultados, la complejidad del algoritmo.

Una vez grabados y etiquetados los ensayos de una sesión, queda la cuestión de cuáles de los datos se van a usar como entrenamiento y cuáles para testeo, y así comprobar la eficacia del algoritmo. Una posibilidad sería que, hasta un determinado número de ensayos realizados, de manera sucesiva, sean de entrenamiento, y los que vengan después sean de testeo.

Si no es en tiempo real (offline) se usa la Validación Cruzada (Cross Validation). "*Es una técnica que se utiliza para seleccionar la configuración correcta de los parámetros de un algoritmo y consiste en separar el conjunto de entrenamiento en conjuntos más pequeños para entrenar y probar el algoritmo utilizando las diferentes configuraciones de los parámetros y seleccionar la configuración que mejor funcione sobre el conjunto de entrenamiento*" [33]. "*Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones*" [66].

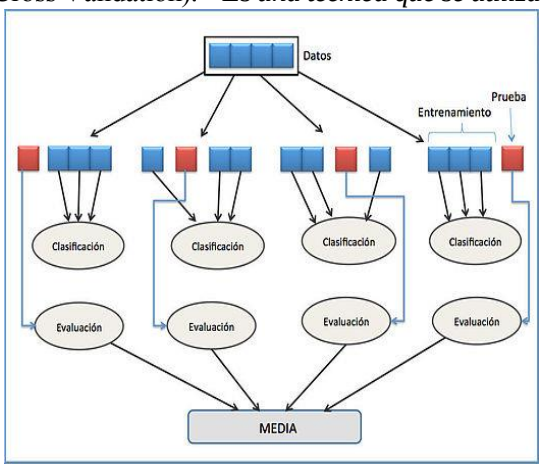


Figura 2-20. Esquema k-fold cross validation, con k=4 y un solo clasificador. Fuente: [66].

En lo que respecta a este trabajo, en la programación, lo que se hizo fue imitar como sería extraer los datos y clasificar en tiempo real. Se iban cogiendo de uno en uno los ensayos de los datos de competición en cada sesión, acumulándose a los que se consideran los datos de entrenamiento, hasta llegar a un determinado número (por ejemplo, 100). Los siguientes datos de la sesión serían de testeo.

Para probar los algoritmos de obtención de características y clasificación de las señales EEG, se utilizó un conjunto de datos que se utilizan en muchos trabajos de investigación, el conjunto de datos 2a de la competición BCI IV [67], proporcionado por Institute for Knowledge Discovery, de Graz University of Technology. En este trabajo no se llegó a usar un casco, solo se trabajó con los datos de competición. Estas son grabaciones de las señales obtenidas de varias sesiones y usuarios. Además, el Departamento de Teoría de la Señal y Comunicaciones de la Universidad de Sevilla se encargó del proceso de extraer los ensayos, filtrar y convertir los datos a un formato más conveniente.

Al usar este conjunto datos no es necesario preocuparse por el preprocesamiento de la señal, es decir, no hay que preocuparse por la calidad de los datos. En resumidas cuentas, se trata de un conjunto de grabaciones de EEG en las que los ensayos siguen un esquema, el cual se explicará a continuación.

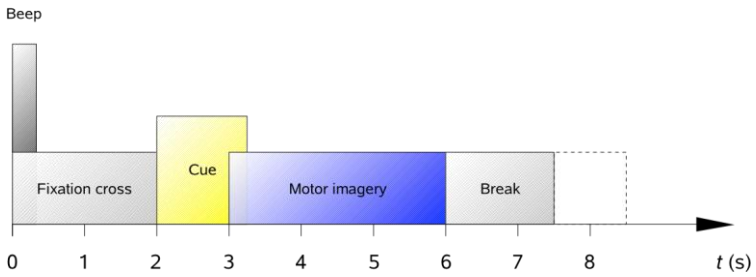


Figura 2-21. Esquema temporal de un ensayo con respecto al tiempo. Fuente: [67].

Como se explica en el documento [67], el conjunto de datos de EEG es de 9 sujetos. Había 4 tipos diferentes de labores de imaginación motora, la imaginación del "*movimiento de la mano izquierda (clase 1), la mano derecha (clase 2), ambos pies (clase 3) y la lengua (clase 4)*". En este trabajo solo es de interés las clases 1 y 2.

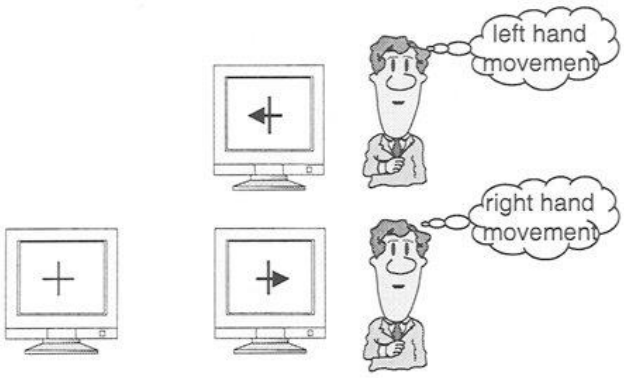


Figura 2-22. Ejemplo paradigma experimental para la recogida de datos de EEG durante la imaginería motora. Fuente: [68].

Se realizó una grabación de unos 5 minutos al principio de cada sesión, para estimar la influencia del electrooculograma (EOG), es decir, para reducir la influencia del movimiento ocular en la señal de EEG. Esto se debe a que, "*en condiciones habituales existe una diferencia de potencial de aproximadamente de 0,4 a 5 mV entre la córnea y la membrana de Bruch situada en la parte posterior del ojo*" [69]. Esta señal no es útil para el objetivo de este proyecto, pero hay sistemas que aprovechan esta señal, un ejemplo es el artículo [70].

Se pidió a los sujetos que miraran una pantalla negra con una cruz de fijación durante dos minutos, luego durante otro minuto permanecieron con los ojos cerrados, y finalmente se registró un minuto de movimiento ocular [38].

Como se ilustra en Figura 2-21, cada ensayo comienza con un pitido para llamar la atención del usuario, aparece una cruz en la pantalla y después de dos segundos aparece una flecha, apuntando a la izquierda, a la derecha, hacia abajo o hacia arriba. Ésta indica al usuario qué movimiento debe pensar. Si solo fuese clases 1 y 2, entonces sería flecha apuntando izquierda o derecha. La flecha desaparece a los 1,25 segundos y aparece una cruz. Los sujetos debían estar realizando la tarea de imaginación motora hasta que la cruz desapareciese. Tras un total de cuatro segundos realizando la tarea de pensar, el usuario descansaba un tiempo. Acabado el tiempo de descanso se procedía con el siguiente ensayo.

Se grabaron dos sesiones en días diferentes para cada sujeto. El primer día se grabó el conjunto destinado a entrenamiento y el segundo el conjunto destinado a testeo. Aunque se pueden tratar de forma independiente, es decir, se podrían usar ambas como entrenamiento, ya que están todas etiquetadas. Habrá 288 ensayos por usuario disponibles. En otras palabras, se grabaron 500 muestras por ensayo, 144 ensayos por sesión, siendo dos sesiones por usuario y habiendo 9 usuarios.

Con respecto al casco usado, se utilizaron 22 electrodos de Ag/AgCl (con distancias entre electrodos de 3,5 cm) colocados en el cuero cabelludo para registrar el EEG. Todas las señales se registraron de forma monopolar, con el mastoides izquierdo como referencia y el derecho como tierra. Además de los 22 canales de EEG, se registraron 3 canales monopolares de EOG. Todas las señales, EEG y EOG, se muestrearon a 250 Hz y se filtraron con paso de banda entre 0,5 Hz y 100 Hz. Se utilizó un filtro notch[8] para eliminar la interferencia de la línea eléctrica de 50Hz, el ruido de línea. Se fijó la sensibilidad del amplificador a 100µV para las señales EEG y de 1mV para las señales EOG.

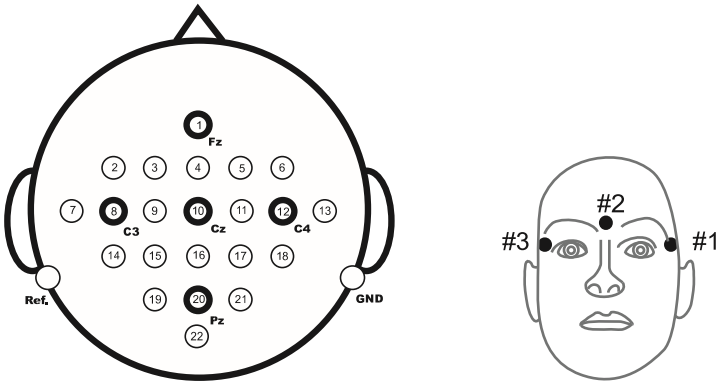


Figura 2-23. Izquierda: Montaje de electrodos correspondiente al sistema internacional 10-20. Derecha: Montaje de electrodos de los tres canales monopolares de EOG. Fuente: [67].

Los canales EOG son para el procesamiento de artefactos, este concepto se mencionó brevemente en el apartado 2.1.2. Se denomina artefacto al error de datos causado por el instrumento de medición, el cual puede provocar una mala interpretación o resultados erróneos [71]. En otras palabras, toda actividad que es notada por los sensores y que no proviene del cerebro se denomina artefacto[9]. Hay que tener en cuenta que no solo existen los artefactos debido al movimiento de los ojos. Existen artefactos fisiológicos y extra-fisiológicos. Los fisiológicos son la actividad muscular, el artefacto ginocinético (lengua), movimientos oculares, Electrocardiograma[10] (ECG), respiración y piel. Los extra-fisiológicos son los electrodos[11], corriente alterna y movimientos en el ambiente. Para profundizar en este tema, es recomendable leer el informe [72].

[8] *"Filtro elimina banda, es un filtro electrónico que no permite el paso de señales cuyas frecuencias se encuentran comprendidas entre las frecuencias de corte superior e inferior"* [247].
[9] Ruido.
[10] Pulsos cardiacos.
[11] Por ejemplo, estallido del electrodo.

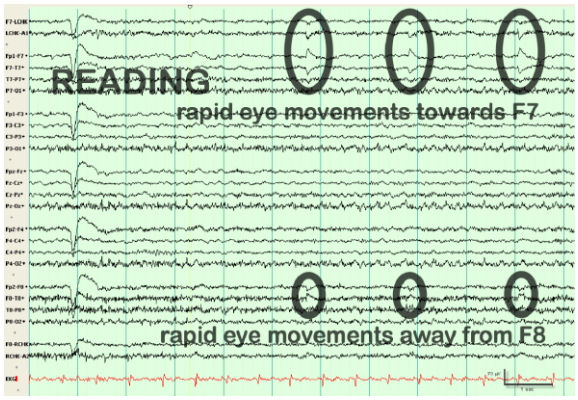


Figura 2-24. Artefacto movimiento de los ojos. Fuente: [73].

2.2.3 Software

La programación de la interfaz se realizó en MATLAB versión R2018b, "*es una plataforma de programación y cálculo numérico utilizada por millones de ingenieros y científicos para analizar datos, desarrollar algoritmos y crear modelos"* [74].

Figura 2-25. g.NEEDacces. Fuente: [75].

Con respecto a los sistemas BCI, se quiere hacer mención como información interesante, si se hace uso de la tecnología de **g.tec**, saber que existe la posibilidad de usar **g.NEEDaccess – MATLAB API** para leer los datos. Primero hay que decir que, **g.NEEDaccess** es un servicio de servidor que facilita la adquisición de datos de forma sencilla e independiente de la plataforma desde dispositivos a través de una red, y al hacerlo, alivia la carga de trabajo del usuario en gran medida. Permite a los usuarios adquirir fácilmente datos de dispositivos **g.tec** sin tener que ocuparse de los conceptos de bajo nivel de la adquisición de datos. El servidor se encarga de la adquisición y el preprocesamiento de los datos para que el usuario reciba datos listos para analizar [75].

El **g.NEEDaccess – MATLAB API** es un **MATLAB toolbox** que se conecta a los controladores de dispositivos **g.tec** de nivel inferior para permitir a los usuarios leer datos de bioseñales como EEG, ECoG, EMG[12], EOG y ECG dentro del entorno MATLAB. El entorno MATLAB permite usuarios configurar fácilmente su propia adquisición y análisis de señales. La API para MATLAB contiene comandos que dan acceso completo al amplificador. Hay comandos para leer los datos, ajustar los filtros de paso de banda y de muesca, cambiar la frecuencia de muestreo del amplificador, definir las derivaciones bipolares y calibrar el sistema [75].

2.2.4 Resumen de BCI

Se diferencian tres tipos de dispositivos según el método de obtención de la señal:

- Invasivos: Es un implante que se conecta a la corteza cerebral. "*Consecuentemente, la señal obtenida es muy nítida".*
- Parcialmente invasivos: La implantación de electrodos es sobre la superficie cerebral
- No invasivos: "*Se mide en la superficie del cuero cabelludo".*

Los BCI se pueden clasificar en dos grupos según la señal a medir:

[12] Electromiografía: *"es una prueba de diagnóstico para analizar la salud de los músculos y las células nerviosas (neuronas) que los controlan mediante el análisis de la actividad eléctrica en los músculos"* [249].

- Endógenos: "*Dependen de la capacidad del usuario para controlar su actividad"* cerebral, por lo que se necesita un periodo de entrenamiento.
- Exógenos: Dependen de estímulos externos.

Este trabajo trata un BCI endógeno no invasivo. Se usa el EEG debido a su bajo coste, la portabilidad y por ser no invasivo.

El proceso del MI-BCI consiste en lo siguiente: Se le pone al usuario electrodos que recogen la señal y la envían a un amplificador, el cual aplicará un procesado a la señal y la enviará al ordenador donde se llevará a cabo las etapas del proceso de Machine Learning.

Durante una sesión, se le va indicando al usuario cuando debe imaginar mover la mano derecha o la izquierda, haciéndose una pausa entre cada indicación. En una sesión se realizan varios ensayos. Cada ensayo que se graba estará etiquetado con su clase real.

Se utilizó para los experimentos la base de datos 2a de la competición BCI IV proporcionada por Institute for Knowledge Discovery. El Departamento de Teoría de la Señal y Comunicaciones de la Universidad de Sevilla se encargó de realizarles el preprocesado a las señales, es decir, no hay que preocuparse por la calidad de los datos. Habiendo utilizado un casco de 22 electrodos, se grabaron 500 muestras por ensayo, 144 ensayos por sesión, siendo dos sesiones por usuario, habiendo 9 usuarios.

En lo que respecta a este trabajo, en la programación, lo que se hizo fue imitar como sería extraer los datos y clasificar en tiempo real.

2.4 Conclusiones

El cerebro humano se puede dividir en distintas zonas, encargándose cada zona de una tarea concreta. La actividad de las neuronas que hay en el cerebro, produce señales eléctricas que pueden ser medidas. Para detectar las actividades de imaginación motora habrá que leer la señal eléctrica de la zona denominada córtex motor. No consiste en medir la actividad de una neurona, sino de un conjunto de ellas. El EEG consiste en medir el campo electromagnético producido por las señales eléctricas del cerebro, colocando electrodos en el cuero cabelludo del usuario. Son señales débiles y hay ruido, si se colocan dentro de la cabeza será más fácil saber de qué parte proceden las señales, pero entonces habría que realizar una intervención quirúrgica.

Dependiendo del BCI que se quiera realizar, interesaran unas señales u otras. Algunas de las señales más interesantes a identificar para un BCI son las de P300, SSEVP, N200 e imaginación motora.

Hay que buscar variaciones en las señales. En el caso de imaginación motora, interesan los fenómenos de **ERD** y **ERS**. Cuando alguien realiza un movimiento motor o lo imagina, los ritmos sensoriomotores se debilitan, se produce ERS. El fenómeno contrario es ERD.

Los sistemas BCI tienen las siguientes etapas: adquisición de la señal, preprocesamiento, extracción de características, clasificación y aplicación. Se pueden clasificar en invasivo o no invasivos. A su vez también se clasifican en endógenos y exógenos.

El sistema BCI de este trabajo es un MI-BCI no invasivo, esto es un sistema endógeno basado en imaginación motora. Se detectará si el sujeto piensa en mover mano izquierda o derecha.

Para probar los algoritmos de extracción de características y clasificación de las señales EEG, se utilizó un conjunto de datos que se utilizan en muchos trabajos de investigación, el conjunto de datos 2a de la competición BCI IV [67], proporcionado por Institute for Knowledge Discovery, de Graz University of Technology. En este trabajo no se llegó a usar un casco, solo se trabajó con los datos de competición. Estos son grabaciones de las

señales obtenidas de varias sesiones y usuarios. La Universidad de Sevilla proporcionó los datos ya preprocesados. Les aplicó un filtrado y extrajeron los intervalos de tiempo relevante de cada sesión

El conjunto de datos de EEG es de 9 usuarios, habiendo 2 sesiones por usuario y 144 ensayos por sesión. Había cuatro tipos diferentes de labores de imaginación motora, la imaginación del movimiento de la mano izquierda (clase 1), la mano derecha (clase 2), ambos pies (clase 3) y la lengua (clase 4). En este trabajo solo es de interés las clases 1 y 2. Se utilizaron 22 electrodos para captar las señales EEG y hay 288 ensayos por usuario disponibles.

En lo que respecta a este trabajo, en la programación en MATLAB, lo que se hizo fue imitar como sería extraer los datos y clasificar en tiempo real. Se iban cogiendo de uno en uno los ensayos de los datos de competición, acumulándose a los que se consideran los datos de entrenamiento, hasta llegar a un determinado número. Los siguientes serían para la validación.

3 PREPROCESAMIENTO DE LOS DATOS

"Los errores por usar datos inadecuados son mucho menores que los que se cometen por no usar ningún tipo de datos."

Charles Babbage

La primera etapa del algoritmo de un proceso de aprendizaje automático es el preprocesamiento de la señal, en este caso la señal de EEG. Consiste en operar sobre las señales para eliminar información que pueda dificultar el proceso de obtención de características y clasificación, es decir, para eliminar información que pueda entorpecer el proceso en una etapa posterior. Esta etapa depende del tipo de señal que se analice.

Otra acción relacionada con el preprocesamiento de datos es que, se podría eliminar del conjunto aquellos datos que estén corrompidos o sean no relevantes para el estudio a realizar.

3.1 Modelado de señal EEG

Es importante comprender el modelo de las observaciones, ya que esto puede resultar de ayuda en el momento de decidir cuáles son las características más importantes para extraer o qué procesos conservarán la información más relevante para la clasificación [38].

Como ya se mencionó, el casco BCI consiste en una serie de sensores que miden la actividad electromagnética en el cuero cabelludo. Produciéndose esta actividad electromagnética en la superficie debido a la actividad de las neuronas, transmitiéndose entre ellas señales eléctricas, por tanto, generando campos electromagnéticos. Sin embargo, solo se podrá medir una actividad global del cerebro y no una parte concreta. Por consecuencia, señales eléctricas procedentes de las diferentes partes del cerebro se mezclan, se alteran al atravesar el cuero cabelludo y se llenan de artefactos[13].

Como se mencionó en la sección 2.1.6, se buscan los fenómenos **ERS** y **ERD** que se detectan en la señal EEG. Se puede estimar su localización, pero encontrar estas ondas en cada ensayo no es una solución fiable. Hay que buscar las variaciones que se producen, sin buscarlas expresamente [33]. A continuación, se explica de manera matemática como se obtendrían los datos finales de las señales con las que se trabajaría.

[13] Ruido

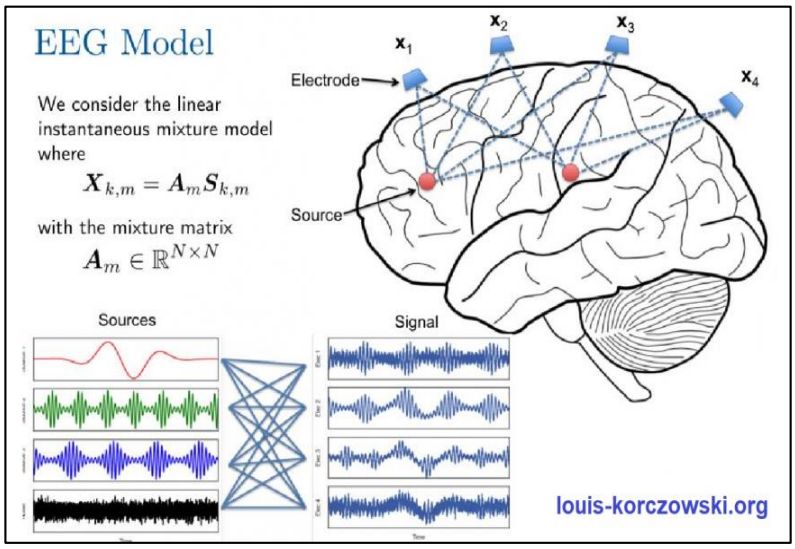


Figura 3-1. Las observaciones X registradas por electrodos no invasivos (EEG) son en realidad una mezcla de fuentes neuronales reales S (en rojo). Fuente: [76].

Como se ha comentado, la actividad cerebral registrada con un electroencefalograma puede entenderse como producida principalmente por un conjunto de fuentes localizadas en distintas partes de la corteza, estas se denotarán como $s_j(t)$ [77]. El número de fuentes a usar se denotará con N_s . Cada una se atenúa debido a la propagación dentro de la cabeza del usuario antes de llegar a un electrodo y ser registrado como parte de la señal $x_i(t)$,dónde i representa el número del canal EEG, desde 1 hasta N_x. Se podría considerar un vector $\mathbf{x}_{columna\_c} = [x_1(t),...,x_{N_x}(t)]^T$ cuyo tamaño es el número de canales EEG, habiendo un vector para cada tiempo t. Cada canal es una combinación lineal de las fuentes más ruido aleatorio, $n(t)$. La ecuación que representa esto sería:

$$x_i(\mathrm{t}) = \sum_{j=1}^{N_s} a_{ji} s_j(\mathrm{t}) + \mathrm{n}(\mathrm{t}) \qquad (3.1)$$

Dónde a_{ji} es un coeficiente que modela la atenuación de la fuente j-ésima hasta el electrodo i-ésimo. Aunque este valor puede variar debido a los movimientos del casco EEG o del gel secado, se asume aproximadamente constante.

Los sensores proporcionan un flujo continuo de datos. Las señales EEG se muestrean al ser procesadas por la computadora, entonces un único canal se podría representar como un vector $\mathbf{x}_{fila\_i} = [x_i(t_0), x_i(t_0 + T_s),..., x_i(t_0 + (L-1)T_s)] \in \mathbb{R}^{1xL}$, siendo T_s el periodo de muestreo, t_0 el momento de inicio del registro y L el número de muestras registradas. Del mismo modo, las fuentes activas se pueden representar cada una como un vector $\mathbf{s}_j \in \mathbb{R}^{1xL}$. Resultando entonces la ecuación (3.1) con vectores:

$$\mathbf{x}_{fila\_i} = \sum_{j=1}^{N_s} a_{ji} \mathbf{s}_j + \mathbf{n} \qquad (3.2)$$

Considerando que $\mathbf{a}_i = [a_1,..., a_{N_s i}] \in \mathbb{R}^{1xN_s}$ y siendo $\mathbf{S} \in \mathbb{R}^{N_s xL}$ la concatenación por filas de las diferentes fuentes, esta combinación lineal se puede escribir así:

$$\mathbf{x}_{fila\_i} = \mathbf{a}_i\mathbf{S} + \mathbf{n} \qquad (3.3)$$

Se tienen N_x señales de EEG que se pueden concatenar por filas en la matriz, similar a lo que se hizo con las fuentes, obteniéndose la señal EEG muestreada completa. Representándose las observaciones de EEG en forma de matriz como combinación de señales activas del cerebro [38]. Las $\mathbf{x}_{fila\_i}$ ayudan a formar una matriz.

$$\mathbf{X}_\tau = \mathbf{A}\mathbf{S}_\tau + \mathbf{N}_\tau \quad \in \mathbb{R}^{N_x xL} \qquad (3.4)$$

Siendo formada la matriz $\mathbf{A}$ por los coeficientes a_{ji}, $\mathbf{A} \in \mathbb{R}^{N_x xN_s}$. Aunque el subíndice ji era apropiado para representar la atenuación debida a la propagación desde la fuente j al electrodo i, en este caso el término a_{ji} se situará en la fila i y la columna j de $\mathbf{A}$. La $\mathbf{N}_\tau$ contiene los ruidos $\mathbf{n}$ para un ensayo τ. La expresión (3.4) sólo es posible porque el proceso de propagación de las señales puede considerarse instantáneo y, por tanto, no hay retardo entre que una fuente se registre en diferentes electrodos [38]. La $\mathbf{X}_\tau$ es una mezcla de las fuentes más un ruido, siendo la dimensión de la matriz el número de canales por el número de muestras.

3.2 Preprocesamiento

Las señales de EEG utilizadas en el trabajo se proporcionaron preprocesadas. Como ya se mencionó en el apartado 2.2.2, el proceso de extraer los ensayos, filtrar y convertir los datos a un formato más conveniente, lo llevó a cabo el Departamento de Teoría de la Señal y Comunicaciones de la Universidad de Sevilla. A continuación, se comenta como venían originalmente las señales y qué tratamientos se le hicieron.

Después de la grabación, las señales se filtraron paso banda entre 0,5 Hz y 100 Hz. Además, un filtro de muesca (notch) eliminó la interferencia de la señal de línea de 50Hz.

Las sesiones tenían datos de varios ensayos de manera continua, una sesión contiene todas las muestras de los ensayos. Hay eventos que indican cuándo empieza y cuándo termina un ensayo. Se dividieron en trozos, siendo cada trozo uno de los ensayos.

La ventana seleccionada comienza 0,5 segundos después de que comience la imaginación motora. Los dos segundos que se consideraron los más relevantes para la detección de imaginación motora, 500 muestras, se guardaron en una matriz separada para cada ensayo. Se descartan los primeros 0,5 segundos del ensayo porque el ERD comienza aproximadamente 0,5 segundos después de que se le indique al usuario que realice la imaginación motora [33].

Las variaciones de los datos de los sensores se miden con respecto a un sistema de referencia dado. Además, se filtran los datos con un paso de banda entre 8Hz-32Hz antes de ser almacenados. Esta es la banda en la que se estima que se pueden encontrar la mayoría de las señales SMR, es decir, las relacionadas con el ERD y ERS. Se consideran que las señales EEG obtenidas por los canales están entre 8-30Hz. También hay que restar la media estimada de cada ensayo. Muchas veces el amplificador tiene una opción que te permite hacer eso automáticamente.

Un paso de preprocesamiento común para muchos problemas de Machine Learning[14] es normalizar los datos. Las normalizaciones habituales incluyen el desplazamiento en la media[15] o el desplazamiento y escalado de los datos en un rango [78].

[14] Aprendizaje automático
[15] La media de los datos desplazados sea 0

Se recopila cada observación final preprocesada en vectores que serán las columnas de una matriz de datos final, habiendo una matriz por ensayo. A esta matriz la denominaremos a partir de ahora $\mathbf{X}_t$, sería la de la ecuación (3.4) con datos ya preprocesados. Cada observación sería $\mathbf{x}_{columna\_c} = [x_1(t), x_2(t), ..., x_{Nx}(t)]^T$. Como ya se mencionó, cada columna de esta matriz corresponde a una muestra (un tiempo) y cada fila los valores de un canal en cada una de esas muestras.

El preprocesamiento estaría terminado. En los sistemas MI-BCI se suele incluir también en el preprocesamiento el cálculo de las covarianzas[16] de los ensayos, ya que es lo que se necesita, para disminuir el trabajo de la siguiente etapa. El tema de las covarianzas se explica más detenidamente en el apartado 4.4. En lo que respecta a los artefactos, hay que tener en cuenta que pueden producir cambios muy importantes en las covarianzas de los ensayos, que a su vez afectan al funcionamiento de los algoritmos de extracción de características y clasificación. La eliminación de artefactos es importante tenerlo en cuenta en cualquier sistema BCI. La mayoría de los algoritmos de eliminación de artefactos son técnicas de preprocesamiento que se aplican antes de que puedan afectar al sistema. Una vez detectada una muestra artefacto se puede borrar, se puede intentar reconstruir la señal o se puede ignorar y utilizar técnicas de protección para disminuir el efecto[17] [33].

3.3 Conclusiones

El preprocesamiento consiste en operar sobre las señales EEG de entrada para eliminar información que pueda dificultar el proceso de extracción de características y clasificación. Para ello hay que comprender el modelo de las observaciones.

Hay que buscar las variaciones que en las ondas se producen. Para el caso de MI-BCI, hay que buscar la variación de **Evento Relacionado con la Desincronización (ERD)** y el **Evento Relacionado con la Sincronización (ERS)**, sin buscarla explícitamente.

Al medirse una actividad electromagnética global de las neuronas, las señales eléctricas procedentes de diferentes partes se mezclan, además de presentar ruido. Hay que tener en cuenta que los artefactos pueden producir cambios muy importantes en las covarianzas de los ensayos, que a su vez afectan al funcionamiento de los algoritmos de extracción de características y clasificación, como se verá más adelante.

Viéndolo matemáticamente, las observaciones registradas por los electrodos son en realidad una mezcla de fuentes neuronales reales. Cada canal es una combinación lineal de las fuentes más ruido aleatorio.

Las señales EEG registradas por los electrodos en un ensayo, se pueden modelar como una matriz en la que cada fila corresponde a cada valor obtenido en uno de los canales, mientras cada columna es el tiempo en que se tomó la muestra del valor de ese canal, es decir, su dimensión es el número de canales por el número de muestras. En el capítulo 4 se explicará que a esta matriz se le realiza una reducción de dimensionalidad y, una vez reducida, se obtiene a partir de ellas las características de un ensayo, habiendo una matriz por cada ensayo.

Los datos de las señales EEG venían preprocesados, es decir, filtrados y preparados para su uso.

---

[16] *"La covarianza es un valor que indica el grado de variación conjunta de dos variables aleatorias respecto a sus medias"* [271]. *"La varianza es un caso particular de la covarianza cuando dos variables aleatorias son idénticas"* [271]. Para saber más sobre la matriz de covarianza y la matriz de correlación, se puede consultar la referencia [270]. *"Si las medidas de correlación utilizadas son coeficientes de producto-momento, la matriz de correlación es la misma que la matriz de covarianza de las variables aleatorias estandarizadas"* [272].

[17] Ejemplo: normalización

4 OBTENCIÓN DE CARACTERÍSTICAS

"El cerebro humano es una máquina increíble para reconocer patrones."

Jeff Bezos

El objetivo es reducir la señal que se desea clasificar a un conjunto limitado de características relevantes, las cuales la describen lo mejor posible.

Existen varios métodos para la selección de características relevantes en señales EEG para MI-BCI. Se pueden dividir en las siguientes grupos: "*los métodos que sustraen información temporal de la señal; métodos que extraen información frecuencial; métodos híbridos, basados en representaciones tiempo-frecuencia, que aprovechan tanto la información temporal y frecuencial; y métodos que aprovechan la información espacial para realizar un filtrado espacial antes de la extracción de características basadas en la información temporal y/o frecuencial"* [79].

Algunos métodos concretos que destacan "*son el análisis en el dominio del tiempo y frecuencia, filtros adaptados, potencia espectral, Common Spatial Patterns (CSP), Transformada Rápida de Fourier (FFT), Modelos Autorregresivos (AR), Transformada Discreta de Wavelet (DWT), Transformada Discreta Wavelet Packet (DWPT), Entropía, Transformada de Hilbert, Redes Neuronales y difusas, y combinaciones entre ellas"* [79]. Con respecto a filtros espaciales destacan CSP, Common Average Referencing (CAR), Principal Component Analysis (PCA), Surface Laplacian (SL) e Independent Component Analysis (ICA) [80]- [79]. El más utilizado en MI-BCI es el procedimiento Common Spatial Patterns (CSP).

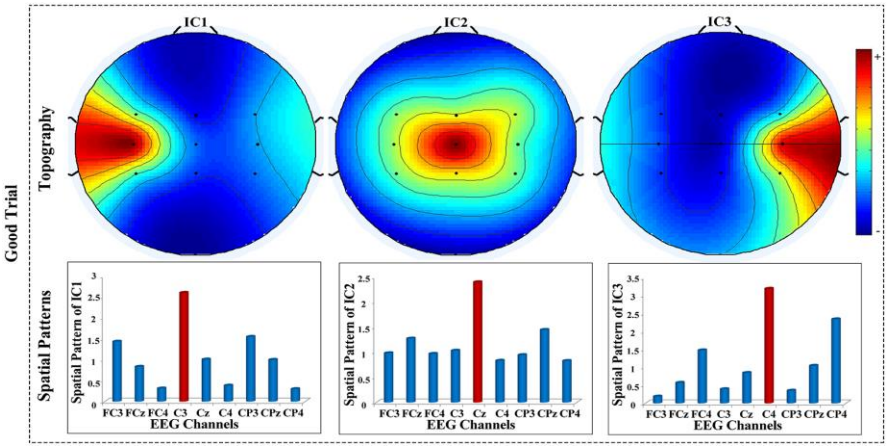


Figura 4-1. "*Plantillas de mapas topográficos MRICs*[18] *y patrones espaciales de un buen ensayo. El análisis de componentes independientes (ICA) como método de filtrado espacial puede separar los componentes independientes relacionados con el motor (MRICs) de las señales de electroencefalograma multicanal (EEG).*" Fuente: [81].

A continuación, se explican algunos de estos métodos relevantes de obtención de características en BCI, haciéndose una explicación más detallada en el caso de CSP.

4.1 Fast fourier transform (FFT)

Es una técnica en el dominio de la frecuencia. "*La transformada rápida de Fourier, conocida por la abreviatura FFT (del inglés Fast Fourier Transform) es un algoritmo eficiente que permite calcular la transformada de Fourier discreta (DFT) y su inversa*" [82]. El primer método de obtención de características utilizado para MI-BCI se basó en este método [83]. Se aplica para estimar la potencia en las bandas de frecuencia elegidas en los espectros generados por FFT, es decir, "*se utiliza para indagar la distribución de la amplitud de espectro en EEG y extraer el pico del espectro para reflejar las diferentes tareas del cerebro"* [79].

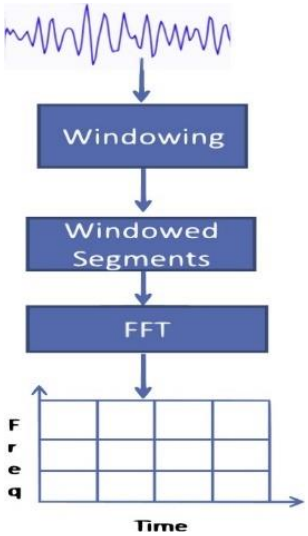


Figura 4-2.Proceso de transformada de Fourier a corto plazo. Fuente: [83].

[18] Separate motor-related independent components

"*Mediante el uso de la FFT, la señal de EEG puede mapearse desde el dominio del tiempo al dominio de la frecuencia. El espectro de frecuencia de la señal es reconocido por descomponer la señal en su correspondiente sinusoidal de diferentes frecuencias*" [79]. El análisis de Fourier descompone la señal en sus componentes de frecuencia y determina su intensidad relativa.

La FFT para una señal discreta x_n, donde k es cada valor discreto de la señal:

$$X(k) = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \qquad con \quad k = 0,1,2,...,N-1 \qquad (4.1)$$

Un ejemplo de BCI usando FFT sería utilizar un chip de adquisición de electroencefalografía (EEG) para extraer SSVEP de las señales de EEG y transformarlas mediante el uso de FFT en el dominio de frecuencia. Luego, estos SSVEP se pueden convertir en comandos para controlar varios dispositivos. Se puede consultar este ejemplo en [84].

Las principales ventajas de FFT son:

- "*La FFT puede ser calculada de forma relativamente rápida o alrededor de un tiempo real*" [79].

Las principales desventajas de FFT son:

- Los espectros obtenidos a partir de la FFT en etapas cortas tienen una resolución pobre cuando se comparan con un modelo autorregresivo. Aunque la resolución de la FFT puede mejorarse aplicando una función de ventana, como la ventana de Hanning, sigue teniendo una resolución pobre [83]. Esto se puede apreciar en la Figura 4-4.

- La FFT no tiene en cuenta la información temporal, por lo que no es capaz de analizar las señales de EEG no estacionarias[19]. Para representar la señal no estacionaria se podría usar la transformada de Fourier de corta duración (Short Time Fourier Transform, STFT). En la STFT, la señal se divide en pequeñas tramas superpuestas sobre las que se aplica la FFT colocando una función de ventana en el eje temporal. Cuando se aplica la función de ventana de tiempo fijo a la STFT, se produce una resolución de tiempo-frecuencia fija que limita el uso de la STFT [83].

- "*Las frecuencias utilizadas para descomponer una señal son una función de la frecuencia de muestreo de la señal y del número de frecuencias deseadas. Sin modificar estos dos parámetros, estas frecuencias no son seleccionables; una onda sinusoidal simple cuya frecuencia no cae en una de las frecuencias de la transformada producirá un espectro con dispersión de energía para muchas frecuencias*" [79].

- Es muy sensible al ruido [79].

4.2 Modelos Autorregresivos (AR)

Es una técnica en el dominio del tiempo. El modelo autorregresivo es el enfoque paramétrico que estima la densidad del espectro de potencia (PSD) de la señal. Normalmente, se prefieren las etapas cortas a las más largas para el análisis, con el fin de caracterizar los rápidos cambios que se producen en la señal de EEG [83].

[19] Una onda estacionaria permanece confinada en un espacio y tiene puntos (nodos) que permanecen inmóviles, estacionarios [235]

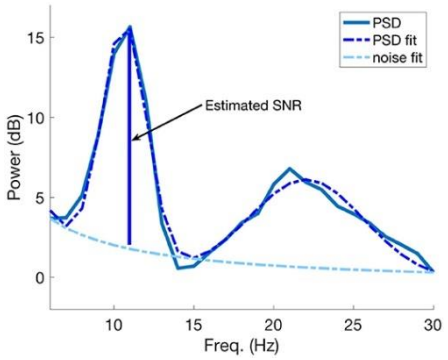


Figura 4-3. "*Un ejemplo de estimación de SNR utilizando el modelo PSD. La estimación de SNR coincide con la diferencia máxima entre el pico de PSD ajustado mayor y la curva de ruido estimada en el valor de frecuencia correspondiente del pico.*"
Fuente: [17].

"*Tienen en cuenta la correlación de las observaciones actuales frente a sus antecesoras. Se utilizan los coeficientes del modelo como características en la discriminación de los patrones de pensamiento. El modelo AR de orden* ***n*** *es descrito por la ecuación (4.2), donde* a_i *son los parámetros AR,* x_i *son los valores observados de la muestra,* r_k *es un término de error que representa un ruido blanco de media cero*" [79]. Los coeficientes son estimados a partir de la muestra actual y de un número finito de muestras anteriores.

$$x_k = a_1 x_{k-1} + \ldots + a_p x_{k-p} + r_k \qquad (4.2)$$

La validez de la estimación espectral depende de la selección del orden adecuado del modelo, donde el orden del modelo determina aproximadamente el número de picos espectrales que deben capturarse. Si el orden del modelo es demasiado bajo, el AR produce un espectro suave, mientras que si es demasiado alto el espectro tiene picos espurios. El orden del modelo para el EEG oscila entre 3 y 20 [83].

"*Los modelos autorregresivos adaptativos (AAR) y modelos autorregresivos Multivariables (MVAR), son técnicas más recientes, en los cuales sus parámetros* a_i *cambian con el tiempo para adaptarse a la variación del espectro de EEG*" [79].

Las principales ventajas de AR son:

- "*Posee ventajas respecto a FFT en su resolución de frecuencia y buenas estimaciones espectrales a partir de segmentos cortos de la señal EEG*" [79]- [85].

Las principales desventajas de AR son:

- "*No existen directrices claras sobre cómo elegir los parámetros de la estimación espectral*" [79].
- "*Es importante determinar el orden óptimo del modelo AR ya que un orden de modelo demasiado bajo tiende a suavizar el espectro, y muy alto tiende a introducir picos espurios*" [79].

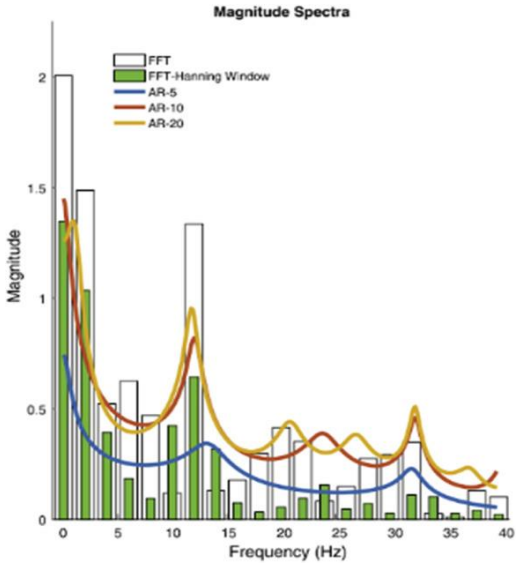


Figura 4-4. Comparación de los espectros generados por la FFT y el modelo AR. Fuente: [83].

4.3 Wavelet transform (WT)

La transformada Wavelet es la técnica de obtención de características en el dominio del tiempo-frecuencia. La WT de una función $f(t)$ descompone una señal en forma de funciones, que forman una función base y son denominados "**wavelets**", (4.3). Se utiliza para representar la función mediante un número infinito de wavelets donde cada wavelet tiene características específicas de tiempo-frecuencia [83]- [79].

La base mencionada es un conjunto de wavelets en escala y trasladadas de una función wavelet conocida como wavelet madre, (4.4), en que s es el factor de escala y τ el de traslación.

$$W_f(s,\tau) = \int f(t)\psi_{s,\tau}(t)dt \qquad (4.3)$$

$$\psi_{s,\tau}(t) = \frac{1}{s}\psi\left(\frac{t-\tau}{s}\right) \qquad (4.4)$$

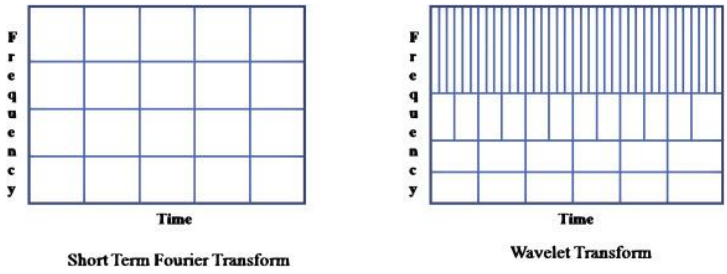


Figura 4-5.Comparación de la resolución obtenida por la STFT y la WT. Fuente: [83].

Una derivación del WT es el **Continuous Wavelet Transform** (CWT), se puede ver un ejemplo de su uso en el artículo de la referencia [86], en el cual se explica en lo siguiente: La CWT y la **Transformada de Fourier** (FT) tienen métodos similares. La FT obtiene coeficientes de correlación entre la señal original y una señal sinusoidal. Del mismo modo, la CWT obtiene coeficientes de correlación entre una wavelet madre y la original. Sin embargo, a diferencia de la FT, donde la señal se descompone en un dominio de frecuencia, la CWT asigna

la señal a un dominio de tiempo-frecuencia controlando la forma de la wavelet madre. En este caso, la forma de la wavelet madre se controla mediante parámetros de escalado y desplazamiento.

Se puede consultar ejemplos de cómo usar la **transformada wavelet diádica discreta** (DDWT) y la **transformada wavelet packet** (WPT), que son otras derivaciones del WT, en el artículo [87]. También puede resultar de interés el artículo [65].

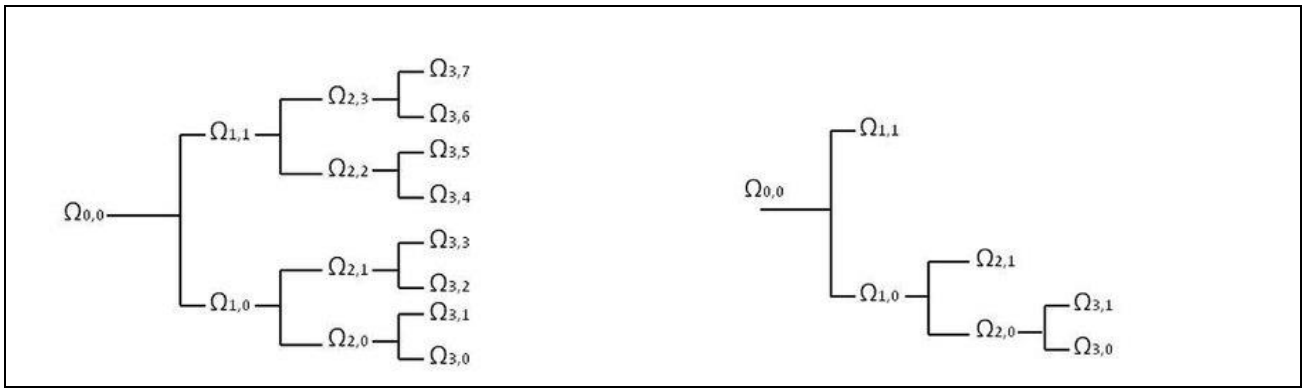


Figura 4-6. "*Descomposición del vector $\Omega \in \mathbb{R}N$ hasta el nivel de descomposición 3. Izquierda: mediante **transformada wavelet packet**. WPT es una versión generalizada de DDWT. Derecha: mediante **descomposición multiresolución***." Fuente: [87].

Las principales ventajas de WT son:

- Conveniente para el tratamiento de señales no estacionarias. La FFT y el modelo AR sólo detectan las características espectrales de las señales y no obtienen un buen rendimiento con las señales de EEG no estacionarias. La transformada Wavelet combina la información de frecuencia y la información del dominio del tiempo, lo que proporciona un mejor rendimiento en comparación con la FFT o el AR [83] .

- "*Puede proporcionar una descomposición de las señales en tiempo-frecuencia multi-nivel. Esto permite el uso simultáneo de intervalos largos de tiempo para la información de baja frecuencia e intervalos cortos de tiempo para información de alta frecuencia*" [79]. La WT utiliza una ventana de tamaño variable, de forma que las frecuencias altas se evalúan en la ventana más corta y las frecuencias bajas en la ventana más larga, como se ve en la Figura 4-5, la WT tiene un mejor rendimiento en la resolución temporal de las frecuencias altas en comparación con la STFT [83].

Las principales desventajas de WT son:

- "*Uno de los inconvenientes de La transformada wavelet es el incremento de los requisitos de memoria debido a su algoritmo basado en la convolución de filtros. La adecuada selección de una wavelet madre y el número de niveles de descomposición son muy importantes en el análisis de señales de EEG para encontrar niveles de clasificación promisorios*" [79].

4.4 Common Spatial Patterns (CSP)

Aunque existen múltiples métodos para seleccionar características destacables, reduciendo dimensionalidad, en las señales de EEG, el más utilizado es el conocido como Common Spatial Pattern (Patrones Espaciales Comunes).

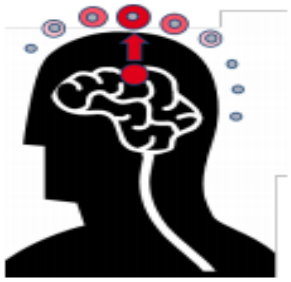


Figura 4-7. Debido al efecto de difuminación, la señal de origen subyacente se reparte entre varios canales. El filtrado espacial ayuda a recuperar esta señal de origen. Fuente: [60].

El CSP genera filtros espaciales que minimizan la variación de una clase y maximizan la variación de otra clase a la vez. Es uno de los métodos de extracción de características más eficaces en lo que se refiere a la clasificación de tareas de imaginación motora binaria. En la interfaz cerebro-ordenador, el objetivo del filtrado espacial utilizado por el algoritmo CSP, es calcular las características cuyas variaciones son óptimas para discriminar dos clases de mediciones de EEG. Un esquema de ejemplo de uso del CSP se muestra en la Figura 4-8.

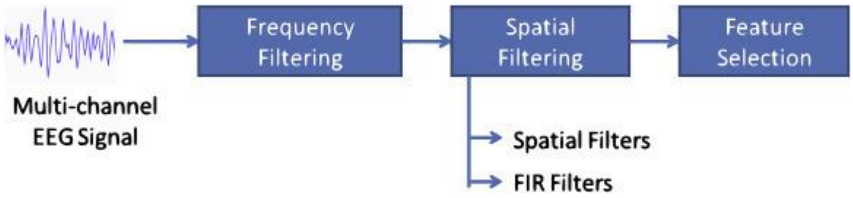


Figura 4-8. Proceso de Common Spatial Pattern. Fuente: [83].

El algoritmo del CSP se basa en el método de Análisis de Componentes Principales[20] (PCA). "*Es una técnica utilizada para describir un conjunto de datos en relación con variables no correlacionadas (…). Los componentes se ordenan por la cantidad de varianza original que representan, por lo que la técnica es bastante útil para reducir la dimensionalidad de una agrupación de datos*" [88]. Este método se emplea principalmente en análisis exploratorio de datos[21] y para crear modelos de clasificación, predecir características. El procedimiento se resume en realizar "*el cálculo de la descomposición en autovalores de la matriz de covarianza, normalmente tras centrar los datos en la media de cada característica*" [88].

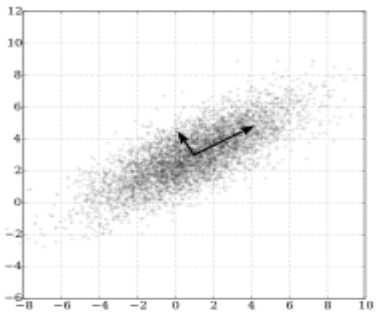


Figura 4-9. PCA de un conjunto de datos bidimensional, "*una distribución normal multivariante centrada en (1,3) con desviación estándar 3 en la dirección aproximada (0,866, 0,5) y desviación estándar 1 en la dirección perpendicular a la anterior*." Fuente: [88].

El PCA busca la "*proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados*" [88], es decir, se basa en encontrar una transformación lineal del espacio de características original

[20] Principal Component Analysis.
[21] "Es el tratamiento estadístico al que se someten las muestras recogidas durante un proceso de investigación en cualquier campo científico" [263].

que establece "*un nuevo sistema de coordenadas para el conjunto original de datos, en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje, la segunda varianza más grande es el segundo eje, y así sucesivamente*" [88]. "*Ésta convierte un conjunto de observaciones de variables posiblemente correlacionadas en un conjunto de valores de variables sin correlación lineal llamadas componentes principales*" [88]. Los primeros componentes principales describen la mayor parte de la varianza de los datos[22]. "*Estos componentes de bajo orden a veces contienen el aspecto de más interés de la información, y los demás componentes se pueden ignorar"* [88]. Por tanto, al estar los componentes más relevantes primero, las últimas dimensiones pueden contener poca información y ser descartadas, reduciendo la cantidad de características a utilizar para el proceso de clasificación [88].

El PCA es útil, pero si posteriormente se usan los datos proyectados en un problema de clasificación, al no hacer uso de las etiquetas de clase de los datos, se estará potencialmente formando representaciones de dimensiones inferiores que serán subóptimas, en términos de qué tan bien separadas estarán las clases [89]. Si se quiere implementar PCA en MATLAB se puede consultar [90].

En este trabajo se dispone de dos clases de señales EEG muestreadas: mano izquierda y derecha. Casi todos los sistemas BCI de imaginación motora (MI-BCI) resumen la mayor parte de la información relevante sobre las mediciones en dos tipos de matrices de covarianza: las matrices de covarianza de las observaciones filtradas (empleadas para la reducción de la dimensionalidad) y las matrices de covarianza de las características (que se requieren para la clasificación con LDA) [77].

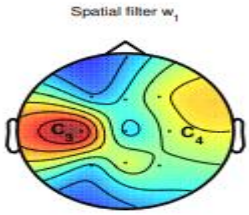


Figura 4-10. Ejemplo de filtrado espacial con CSP en la cabeza de una persona. Se puede ver que "*el máximo peso del filtro espacial es para los electrodos C3, que cubre la zona dedicada al movimiento de la derecha*". Fuente: [91].

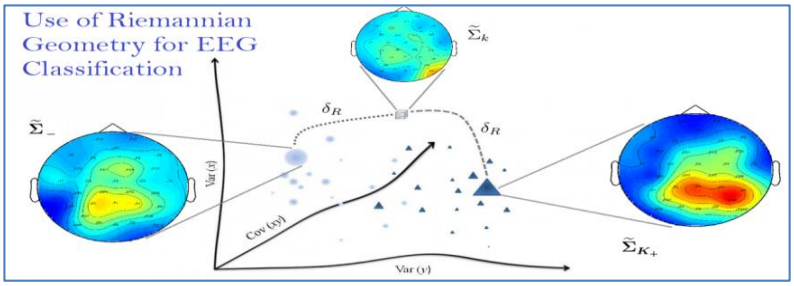


Figura 4-11. El CSP se basa en el cálculo de las covarianzas medias de cada clase. En esto también se basa la clasificación usando geometría de Rieman. Todas las estadísticas de segundo orden están encapsuladas en las matrices de covarianza de la señal EEG. "*La geometría de Riemann es el estudio de las variedades diferenciales con métricas de Riemann; es decir de una aplicación que, a cada punto de la variedad, le asigna una forma cuadrática definida positiva en su espacio tangente, aplicación que varía suavemente de un punto a otro*" [92]. Aunque no se va a profundizar en su explicación, se puede consultar unos estudio sobre el uso de la geometría de Riemann para BCI en los artículos: [93]- [94]. Una mejora de CSP clásico es el uso combinado de CSP y Rieman, el

[22] Más cuanto más correlacionadas estuvieran las variables originales

cual se halla en: [91]. En él se muestra como "*el filtrado espacial CSP y la extracción de la varianza Log se resume en un cálculo de una distancia de Riemann en el espacio de las matrices de covarianza*". Fuente: [76].

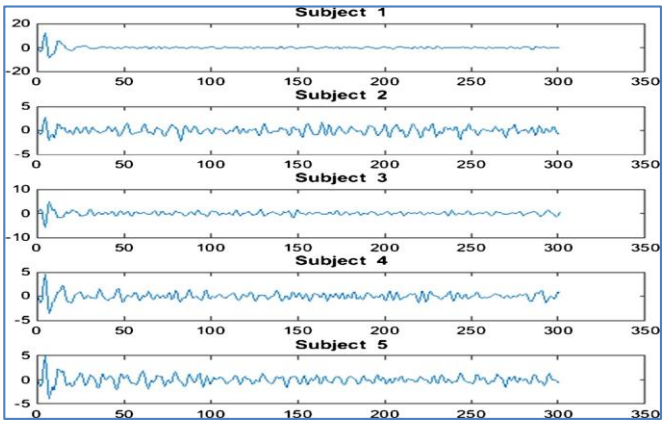


Figura 4-12. Ejemplo de la señal de salida de CSP para diferentes sujetos. Fuente: [95].

En CSP se intenta seleccionar los subespacios *p-dimensional* de las observaciones que conservan la mayor parte del poder de discriminación. Es un método diseñado para el caso de tener dos clases.

En el capítulo 3 se explicó cómo se deducía el modelo matricial de las observaciones. Cada uno de los ensayos es una señal EEG muestreada. En otras palabras, cada muestra es un vector formado por observaciones, registradas por cada canal, quedando como resultado una matriz formada por vectores de observaciones para cada ensayo. Por comodidad para el lector, se recordará a continuación la expresión del modelo de observaciones:

$$\mathbf{X}_\tau = \mathbf{A}\mathbf{S}_\tau + \mathbf{N}_\tau \quad (4.5)$$

Donde X_τ , $N_\tau \in R^{N_x \times T}$; $S_\tau \in R^{N_s \times TS}$, denominándose T a la longitud del registro en muestras y N_x a el número de canales. Siendo la τ el número del ensayo. Se han agrupado todos los tiempos de un ensayo en una matriz X_τ y se han hecho varias sesiones, por tanto, hay varias matrices X_τ .

La información más relevante no está en las señales de EEG en sí mismas, sino en su matriz de covarianza muestreada. Como las observaciones ya han sido centradas, la matriz de covarianza espacial del EEG del ensayo τ viene dada por:

$$\mathbf{C}_{X_\tau}^{(0)} = \frac{1}{T}\mathbf{X}_\tau \mathbf{X}_\tau^T \quad (4.6)$$

Acumulas productos y divides por T, es como hacer una media temporal. Se hace de todos los sensores a la vez, por eso queda una matriz de resultado, siendo de dimensiones $\mathbf{C}_{X_\tau}^{(0)} \in \mathbb{R}^{N_x x N_x}$. Hay que calcular una covarianza por cada ensayo.

Básicamente hay que calcular correlación con matriz $\mathbf{X}_\tau$. Hacer una correlación temporal entre el vector y el vector traspuesto. "*En probabilidad y estadística, la correlación indica la fuerza y la dirección de una relación lineal y proporcionalidad entre dos variables estadísticas. Se considera que dos variables cuantitativas están correlacionadas cuando los valores de una de ellas varían sistemáticamente con respecto a los valores homónimos de la otra: si tenemos dos variables P y B, existe correlación entre ellas si al disminuir los valores de P lo hacen también los de B y viceversa*" [96].

Lo que se está calculando es la esperanza del vector correlacionado con él mismo traspuesto. Para calcular esa esperanza se usa el estimador temporal de la esperanza, el cual se explicará a continuación.

Si las entradas en un vector columna $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ son variables aleatorias, cada una con varianza finita y valor esperado, entonces la matriz de covarianza $\mathbf{C}_{x_i x_j}$ es la matriz cuya (i, j) entrada es la covarianza entre la variable x_i, x_j [97] :

$$\mathbf{C}_{x_i x_j} = \operatorname{cov}\left[x_i, x_j\right] = E\left[\left(x_i - E\left[x_i\right]\right)\left(x_j - E\left[x_j\right]\right)\right] \qquad (4.7)$$

Esta es la covarianza estadística de dos variables aleatorias (x_i, x_j). Se trataría a los sensores como variables aleatorias, la covarianza mide el parecido entre sensores.

Cuando se hace para todos los sensores, y no solo sensor i y sensor j, se tendría todas las correlaciones a la vez así:

$$\mathbf{C}_{xx} = \begin{pmatrix} E\left[(x_1 - E[x_1])(x_1 - E[x_1])\right] & E\left[(x_1 - E[x_1])(x_2 - E[x_2])\right] & \dots & E\left[(x_1 - E[x_1])(x_j - E[x_j])\right] \\ \vdots & \ddots & & \vdots \\ E\left[(x_i - E[x_i])(x_1 - E[x_1])\right] & \cdots & & E\left[(x_i - E[x_i])(X_j - E[x_j])\right] \end{pmatrix} \quad (4.8)$$

Siendo E la esperanza estadística, valor esperado (media) de su argumento.

Cuando $i = j$ queda que $\operatorname{cov}\left[x_i, x_i\right] = \operatorname{var}[x_i, x_i]$, quedando la matriz definida como:

$$\mathbf{C}_{xx} = \begin{pmatrix} Var(x_1) & Cov(x_1, x_2) & \dots & Cov(x_1, x_n) \\ \vdots & \ddots & & \vdots \\ Cov(x_n, x_1) & \cdots & & Var(x_n) \end{pmatrix} \quad (4.9)$$

En el preprocesamiento, a todas las señales de los sensores se les restó la media, por eso $E[x_n] = 0$. Todas nuestras señales tienen aproximadamente de media 0, por eso en este trabajo no se le resta la media en el cálculo. Por tanto, la covarianza del vector es la esperanza del vector por el vector traspuesto.

Sin embargo, la esperanza estadística no se puede aplicar porque no se conoce la densidad de probabilidad, es una covarianza teórica. Se cambia la esperanza estadística por la media muestral. La matriz de covarianza muestral es así:

$$\mathbf{C}_{muestral} = \frac{1}{T-1}\sum_{j=1}^{T}(x_j - \langle x \rangle)(x_j - \langle x \rangle)^T \quad (4.10)$$

Se puede dividir por T o T-1, son dos estimadores posibles. Los valores esperados necesarios en la fórmula de covarianza se estiman utilizando la media muestral.

$$\langle x \rangle = \frac{1}{T}\sum_{j=1}^{T} x_j \quad (4.11)$$

Donde x_j es la j-ésima observación del vector de datos aleatorios. Se sabe que en este trabajo es $\langle x \rangle = 0$.

Como se mencionó anteriormente (capítulo 3), en la práctica, los vectores se adquieren experimentalmente poniendo en cada columna los valores de cada sensor en un tiempo de muestreo. Siendo N_x el número de sensores y T el número de tiempos de muestreo.

$$\mathbf{X}_\tau = \begin{bmatrix} x_1(t_1) & x_1(t_2) & \cdots & x_1(t_T) \\ x_2(t_1) & x_2(t_2) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_x}(t_1) & \cdots & \cdots & x_{N_x}(t_T) \end{bmatrix} \quad (4.12)$$

Por tanto, al final queda uno partido de número de tiempos muestral por el sumatorio para todos los tiempos de $\mathbf{X}_\tau \mathbf{X}_\tau^T$, es decir, la ecuación (4.6). Pudiéndose expresar de la siguiente manera:

$$\mathbf{C}_{X_\tau}{}^{(0)} = \frac{1}{T} \begin{bmatrix} x_1(t_1) & x_1(t_2) & \cdots & x_1(t_T) \\ x_2(t_1) & x_2(t_2) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_x}(t_1) & \cdots & \cdots & x_{N_x}(t_T) \end{bmatrix} \begin{bmatrix} x_1(t_1) & x_1(t_2) & \cdots & x_1(t_T) \\ x_2(t_1) & x_2(t_2) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_x}(t_1) & \cdots & \cdots & x_{N_x}(t_T) \end{bmatrix}^T \quad (4.13)$$

Para más información sobre este desarrollo se puede consultar la referencia [98].

Volviendo al cálculo de CSP, se tiene que calcular la covarianza de cada ensayo, $\mathbf{C}_{X_\tau}$. Se puede calcular por dos métodos. La notación usada es $\mathbf{C}_{X_\tau}^{(i)}$, siendo i la indicación del método. El método estándar sería el de la ecuación (4.6), y luego existe una iteración refinada mejorada partiendo que se tiene (4.6), sería:

$$\mathbf{C}_{X_\tau}^{(1)} = \frac{\mathbf{C}_{X_\tau}^{(0)}}{Tr\left\{\mathbf{C}_{X_\tau}^{(0)}\right\} / N_x} \quad (4.14)$$

Los ensayos pueden tener una potencia desigual, la implementación de (4.14) normaliza las matrices de covarianza del EEG. Si se calculan las covarianzas de muestras que tienen potencias distintas, una mucho más débil que la otra, la fuerte va a dominar a la débil. Esto es debido a que la covarianza es un producto de valores, el producto más grande domina al más pequeño. Por ejemplo, podemos tener un ensayo con una o dos muestras muy grandes en comparación con todas las demás. Cuando se calcula la covarianza muestral con formula (4.6), esas dos muestras grandes son las que van a dominar el resultado. Entonces la forma de cálculo con (4.6) no es muy buena. Por otra parte, con (4.14), antes de calcular la covarianza normalizas las muestras en potencia, para que todas tengan aproximadamente la misma ganancia. En otras palabras, no se deja que unas muestras sean mucho más fuertes que otras. Se quiere ver parecido entre los sensores, pero si en un tiempo dado los sensores tienen mucha amplitud, esos tiempos van a dominar mucho en la covarianza. Si yo hago un proceso previo a través del cual normalizo en amplitud los sensores va a ser más correcto el resultado. Hay momentos en que el amplificador te puede dar más respuesta y en otros menos, por diferentes motivos la respuesta puede ser más fuerte y recibirse mejor.

En una matriz de covarianza los elementos diagonales son la varianza de cada sensor, el primer elemento diagonal es varianza del primer sensor, el segundo elemento la del segundo sensor y así sucesivamente. Hacer la traza de una matriz de covarianza es sumar los elementos diagonales. En la traza de (4.14) se suma la potencia de todos los sensores de las observaciones, y eso es la potencia del vector de observaciones. La matriz covarianza normalizada ($\mathbf{C}_{X_\tau}^{(1)}$) tiene potencia uno, la suma de sus elementos diagonales es uno.

En resumen, en la fórmula (4.14) se calcula la covarianza clásica (4.6) y luego se normaliza en potencia. La traza es suma de las diagonales de la matriz, que viene a ser la potencia de las observaciones. Darse cuenta de que el 1/T se iría por estar en numerador y denominador de la división.

A continuación, se hace una media aritmética de las covarianzas de los ensayos. La ecuación (4.15) es la media de las covarianzas de todos los ensayos, haciéndose promedio de los ensayos de una misma clase, en este caso K=1,2. Se suman las covarianzas de cada clase, y se divide por el número que haya de cada clase respectivamente.

$$\hat{\Sigma}^{(1)}_{x|c_x} = \frac{1}{N_{c_k}} \sum_{\tau:c(\tau)=c_k} \mathbf{C}^{(1)}_{X_\tau} \qquad k = 1,...,K \qquad (4.15)$$

Se obtiene covarianzas de una de las clases (Por ejemplo: mano derecha) y hallo su valor medio, eso se le designa $\hat{\Sigma}^{(1)}_{x|c_x}$, (sigma) covarianza media. Hay que hacer esto porque cada ensayo te da una covarianza ligeramente distinta, las $\mathbf{C}^{(1)}_{X_\tau}$ oscilan en torno a su correspondiente $\hat{\Sigma}^{(1)}_{x|c_x}$. Lo mismo se haría para la otra clase. El objetivo es que los ensayos de mano izquierda estén en torno a una zona y los de derecha en torno a otra zona. Se puede decir que se está filtrando el ruido haciendo una covarianza media, el ruido se atenúa con el promediado. Los ensayos tienen mucho ruido, y como se usará después para extraer características, no se quiere que sea tan ruidoso. Para elegir el subespacio en que reducimos dimensión es mejor obtener datos mucho más precisos. Además, al hacer promediado a una señal persistente[23], ésta se refuerza, se obtiene mejor resultado. La señal es persistente y el ruido es incorrelado[24], por tanto, el ruido no se refuerza, se atenúa más al hacer media.

Resumiendo, se ha hecho el preprocesamiento, se ha obtenido covarianza de cada ensayo y luego se obtienen covarianzas medias de la clase 1 y la clase 2. Después se procede a reducir la dimensión.

Como se mencionó anteriormente, debido al efecto de emborronamiento del cráneo y el cerebro, la señal de fuente subyacente se distribuye en varios canales. El filtrado espacial ayuda a recuperar esta señal de fuente. La reducción de dimensión o filtrado espacial se realiza para conseguir una buena clasificación, pues son muchos sensores. Es mejor ver las dimensiones más relevantes y clasificar en ese espacio reducido, para trabajar con menos variables en la clasificación. Se trabajará con una matriz de covarianza de menor dimensión, para ello se multiplica la original por una matriz que reduce su dimensión. La covarianza de la matriz de observaciones filtrada espacialmente es:

$$\mathbf{C}_{Y_\tau} = \mathbf{W}^T \mathbf{C}^{(0)}_{X_\tau} \mathbf{W} \qquad (4.16)$$

Siendo $\mathbf{W}$ la matriz de filtro espacial, la cual se forma con la concatenación de los filtros espaciales $\mathbf{w}$, los vectores se ordenan según sus valores propios asociados. Los vectores propios seleccionados, $\mathbf{w} = [w_1,...,w_p]$, se agrupan en la matriz de filtros espaciales, siendo $p < N_x$, utilizándose para realizar la reducción de la dimensionalidad de las observaciones. Se denota como p al número de filtros.

La cuestión es cómo calcular la $\mathbf{W}$ para aplicar el filtrado espacial, es decir, para sustituirla en (4.16). Se profundizará sobre el concepto de filtro espacial y cómo obtenerlo a continuación.

[23] "*Que sigue durando o se mantiene constante por largo tiempo*" [267].
[24] Cada ruido es distinto. La variables incorreladas o de "*correlación nula se da cuando no hay dependencia de ningún tipo entre las variables*". La nube de puntos tiene una forma redondeada [248].

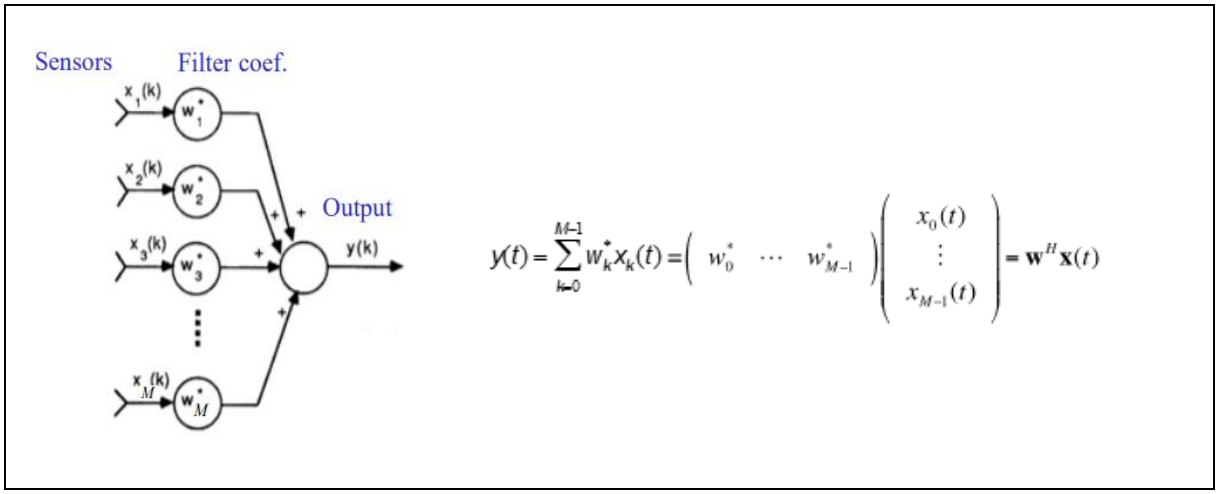


Figura 4-13. Filtrado de sensores referido al tiempo, combina la información de los múltiples canales en una sola señal. Intenta extraer una de las fuentes, busca deshacer la mezcla de señales. El valor de los sensores es multiplicado por unos coeficientes *w,* éstos forman un vector **w**. Se multiplica el vector de filtrado espacial (**w**) traspuesto por el vector de las observaciones (**x**). En el caso de ser varias muestras de tiempo, habrá una matriz de muestras **X** y una matriz de filtros **W**. Otro detalle para tener presente, en este caso las señales son reales, el conjugado no hace falta. La $\mathbf{w}^H$ sería equivalente a poner $\mathbf{w}^T$, la traspuesta. Fuente: [99].

Básicamente el filtro espacial consiste en que unos parámetros multiplican los datos de los sensores, como se muestra en Figura 4-13. Usar una pequeña cantidad de nuevos canales definidos como una combinación lineal de los originales. Se denomina filtro espacial porque la combinación sólo se realiza a lo largo de la dimensión espacial, correspondiente a los diferentes canales del EEG, y no de la dimensión temporal.

La señal resultante de aplicar un filtrado espacial a los sensores, a la señales EEG, sería la matriz $\mathbf{Y}_\tau$, formada por las variables $y(t)$. La $\mathbf{Y}_\tau$ es el conjunto de $y(t)$ en forma de matriz, formándose una matriz por cada ensayo.

$$\mathbf{Y}_\tau = \mathbf{W}^T\mathbf{X}_\tau \in \mathbb{R}^{pxT} \quad (4.17)$$

Hay algo que no hay que confundir, $\mathbf{Y}_\tau$ de (4.17) es el espacio reducido que va a tener la información relevante, pero no son las características. Las $y(t)$ dependen del tiempo, se les calcula su potencia y dependiendo de si es débil o fuerte esa potencia, eso sí es una característica.

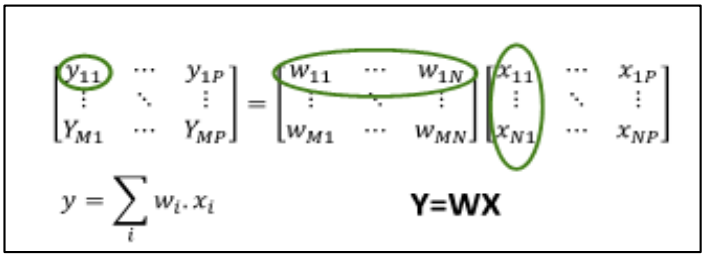


Figura 4-14. Este es un ejemplo de cómo se calcularía la matriz **Y**, cuya dimensión es M canales filtrados espacialmente por P muestras. "*Cada fila de **X** son P muestras consecutivas de uno de los N canales. Cada fila de **W** es un conjunto de pesos de N canales que constituyen un filtro espacial concreto. Cada canal filtrado espacialmente de **Y** es una suma ponderada de todos los canales de **X**, donde los pesos de los canales están definidos por **W***". Fuente: [99].

Con respecto la obtención de la matriz $\mathbf{W}$, se partirá de la función (4.18) para su obtención, la cual depende de los vectores $\mathbf{w}$ que conforman las columnas de la matriz $\mathbf{W}$:

$$J(w) = \frac{\mathbf{w}^T \hat{\Sigma}^{(1)}_{x|c_1} \mathbf{w}}{\mathbf{w}^T \hat{\Sigma}^{(1)}_{x|c_2} \mathbf{w}} \qquad \mathbf{w} \in \mathbb{R}^{N_x} \quad (4.18)$$

La función $J(w)$ se basa en el cociente de **Rayleigh** [100], es un cociente de varianzas[25] (σ^2), siendo el numerador la varianza de la clase 1 y el denominador la de la clase 2. Los puntos críticos de ese cociente, que maximizan y minimizan la función, son los filtros espaciales que se quieren obtener. Se puede interpretar como una relación señal/ruido (SNR), para reducir dimensión habría que maximizar una SNR en filtrado espacial. Se va a hacer una breve explicación sobre esto a partir de las señales originales ($x(t)$) y la Figura 4-13 [99].

Las observaciones tienen dos componentes, que son las dos clases mencionadas. Una será la señal de interés y la otra se considerará ruido.

$$x(t) = \underset{\text{signals of interest}}{x_s(t)} + \underset{\text{interference and noise}}{x_i(t)} \qquad (4.19)$$

Donde $x_s(t) = \sum_{i=1}^{p} a_j s_j(t)$, esto es lo explicado en el capítulo 3. Por otra parte, se define lo siguiente:

$$\mathbf{R}_{\mathbf{Xs}} = E[x_s(t) x^H{}_s(t)] \qquad (4.20)$$

$$\mathbf{R}_{\mathbf{X}i} = E[x_i(t) x^H{}_i(t)] \qquad (4.21)$$

Asumiendo que la media de las señales es cero, se deduce que la varianza (potencia) de la salida del filtro espacial, cuyo resultado es un número, es $E\left[|y(t)|^2\right] = E\left[y(t) y^H(t)\right] = \mathbf{w}^H E\left[\mathbf{x}(t)\,\mathbf{x}^H(t)\right]\mathbf{w} = \mathbf{w}^H \mathbf{R}_{\mathbf{X}} \mathbf{w}$.

Sabiendo que $y(t) = y_s(t) + y_i(t) = \mathbf{w}^H \mathbf{x}_s(t) + \mathbf{w}^H \mathbf{x}_i(t)$, se obtiene lo siguiente:

$$J(w) = SNR(w) = \frac{E[|y_s(t)|^2]}{E[|y_i(t)|^2]} = \frac{\mathbf{w}^H \mathbf{R}_{\mathbf{X}s} \mathbf{w}}{\mathbf{w}^H \mathbf{R}_{\mathbf{X}i} \mathbf{w}} \qquad (4.22)$$

En lugar de usar cada señal EEG ($x(t)$), se usan las matrices de covarianza de un ensayo, se ve que queda lo mismo que se expresó en la función (4.18). En lugar de trabajar con los ensayos, que son matrices grandes, se trabajará con las matrices de covarianza de los ensayos, pues se pueden calcular al principio de la etapa de reducción de la dimensionalidad y permite una normalización de la potencia de las fuentes, que como se explicó anteriormente, provoca una mejor estimación de las matrices de covarianza media de las clases y mejores filtros espaciales [38]- [77].

Una $\mathbf{w}$ que sería un buen filtro, sería la que haga que se tenga mucha potencia de las fuentes que dan la señal deseada, la asociada a la clase que se quiere caracterizar. El cociente sería la potencia de la que me interesa frente a la que no me interesa, es decir, la varianza de clase 1 entre la de la clase 2 o viceversa. El máximo local de una es el mínimo de la otra. La función (4.18) y su inversa comparten puntos críticos.

[25] *"Cuando se realiza un movimiento perteneciente a cierta clase, la varianza de la señal procedente de esa zona del córtex debería ser mínima debido a la ERD y la varianza de la señal procedente de la región opuesta debería ser máxima como resultado de la ERS. La relación de varianzas refleja estas dos condiciones. Tanto los máximos como los mínimos corresponden a una diferencia máxima de varianza"* [38].

Las covarianzas medias son conocidas de los datos de entrenamiento, calculándose con (4.15), así que la función (4.18) solo depende de $\mathbf{w}$. En conclusión, para obtener cada $\mathbf{w}$ de la matriz $\mathbf{W}$, hay que hallar el máximo de una función cuadrática, es un problema de autovalores, que en MATLAB se resuelve con el comando ***eig***:

$$[\mathbf{V},\mathbf{D}] = eig(\hat{\Sigma}^{(1)}_{x|c_1}, \hat{\Sigma}^{(1)}_{x|c_2}) \qquad (4.23)$$

Este comando permite obtener los principales vectores propios generalizados (autovectores) y sus autovalores [101]. Dado una matriz $\mathbf{A}$, un vector propio generalizado es un vector que satisface ciertos criterios que son más relajados que los de un vector propio (ordinario) [102]. Los autovalores serían los máximos de la función $f(\mathbf{w}) = \mathbf{w}^T\mathbf{A}\mathbf{w}$, es decir, $\lambda_k = \max \frac{\mathbf{w}^T\mathbf{A}\mathbf{w}}{\mathbf{w}^T\mathbf{w}}$. El autovalor de un vector propio "*es el factor de escala por el que ha sido multiplicado*" [103]. Son autovalores generalizados en este trabajo. Los autovectores generalizados de una matriz $\mathbf{A}$ asociados a λ_k "*son (excluyendo al vector nulo), los vectores de los espacios nulos de las matrices*" $(\mathbf{A}-\lambda\mathbf{I})^k$, k = 1, 2, 3, ... [104].

En (4.23) la $\mathbf{V}$ es la matriz de filtros[26] completa, en las columnas de $\mathbf{V}$ están las direcciones, es decir, están los filtros espaciales. La $\mathbf{D}$ es una matriz diagonal, sería la evaluación de la función $J(w)$ para cada uno de los filtros espaciales, las SNR obtenidas. En cada elemento de la diagonal están los $J(w)$ máximos, siendo el primer elemento el primer valor que da un máximo total, el segundo elemento el segundo máximo, y así sucesivamente.

Otra manera de calcular el máximo sería, a partir de (4.18) , hallar el gradiente teniendo en cuenta que las matrices de covarianza son simétricas. El valor óptimo del filtro espacial será un máximo o un mínimo. Para averiguarlo se iguala este gradiente a cero, para localizar los puntos críticos. Tratándose, como ya se mencionó, de un problema de valores propios generalizado.

Hay tantas soluciones linealmente independientes como la dimensión de las matrices originales, es decir, tantas soluciones como canales EEG. Dicho esto, no todas son igual de relevantes. Los vectores propios asociados a los valores propios extremos contienen más información [38].

Se ha conseguido cambiar el espacio de los sensores a uno más pequeño que tiene la información relevante. Por cada filtro $\mathbf{w}$, habrá una característica. Se coge más de un filtro, es importante tanto el $\mathbf{w}$ que hace máximo cociente como el que lo hace mínimo (la mano contraria). En otras palabras, hay que buscar una dirección que maximice el cociente, pero no solo el máximo es importante, también el mínimo. Tanto los máximos como los mínimos corresponden a una diferencia máxima de varianza [38]. Por ejemplo, se podría buscar cuatro direcciones que minimicen $J(w)$ y cuatro que la maximicen. La dirección que maximiza es una de las direcciones a coger, pero una vez obtenida la descuentas y hallas una segunda dirección que lo vuelve a maximizar, y una tercera y así hasta todas las dimensiones. Como se mencionó anteriormente, las direcciones de en medio no tienen tanto efecto, se descartan. Se quiere que haya más de una característica para que clasifique mejor, por eso se busca otra $\mathbf{w}$ que dé el máximo de $J(w)$, pero en una dirección ortogonal a la que ya se ha extraído. Se miran las direcciones ortogonales a esa para ver dónde hay otro máximo. En definitiva, solo se cogen los primeros máximos y mínimos relevantes, esas son las direcciones que se utilizan para lograr obtener las características.

El algoritmo de obtención de las características en MATLAB, partiendo de lo ya obtenido, sería:

$$[\sim, ind] = sort(diag(\mathbf{D})) \qquad (4.24)$$

$$\mathbf{V} = \mathbf{V}(:, ind) \quad (4.25)$$

[26] Lo que se llamaba $\mathbf{W}$ anteriormente.

$$\mathbf{W} = [\mathbf{V}(:,1:p/2), \mathbf{V}(:, N_x - p/2 + 1 : N_x)] \quad (4.26)$$

$$\mathbf{C}_{Y_\tau} = \mathbf{W}^T \mathbf{C}_{X_\tau}^{(0)} \mathbf{W}, \qquad \tau = 1, ..., N_\tau \quad (4.27)$$

$$\mathbf{f}_\tau = \log(diag(\mathbf{C}_{Y_\tau}) / sum(diag(\mathbf{C}_{Y_\tau}))) \quad (4.28)$$

Para asegurarse que en MATLAB aparecen ordenados los elementos de la diagonal de la matriz $\mathbf{D}$, (4.23), se usa comando ***sort***[27], como se ve en (4.24), y luego en (4.25) se ordenan columnas de $\mathbf{V}$, (4.23). Así se asegura que el primer máximo esté en la primera columna, el segundo máximo en la segunda, y así sucesivamente.

Como ya se aludió, el número de columnas de $\mathbf{V}$ es el mismo al número de sensores, se cogen los extremos como se ve en (4.26). En este trabajo se usan 8 filtros, las cuatro primeras columnas y las cuatro últimas, las que se considerarían cuatro máximos y cuatro mínimos. La letra ***p*** en (4.26) es el número de características que se van a usar, es decir, el número de filtros a usar.

La ecuación (4.27) sería equivalente a hacer covarianza de $\mathbf{Y}_\tau$, se recuerda que $\mathbf{Y}_\tau$ es la salida de los filtros. La covarianza al ser filtrada es de dimensión $p \, x \, p$. Se ha reducido la dimensión, la covarianza sin reducir era de dimensión $N_x \, x \, N_x$**.**

Con el filtrado espacial obtengo la varianza de la entrada y el logaritmo de la varianza va a ser la característica. Las características se obtienen cogiendo los elementos diagonales de la covarianza de $\mathbf{Y}_\tau$ (las varianzas de cada observación filtrada espacialmente) y normalizando por la suma de todas ellas, como se aprecia en (4.28). En lugar de trabajar con la varianza[28] de las señales directamente, el procedimiento estándar es utilizar el logaritmo natural de estos valores como características. Se quiere que las características sean lo más parecida posible a una Gaussiana. Como las varianzas son positivas[29], se aplica el logaritmo para tener valores positivos y negativos, siendo así más Gaussiano. La varianza de las señales como variable aleatoria no se distribuye normalmente, pero su logaritmo sí se aproxima mejor a una distribución normal. Darse cuenta de que el logaritmo es una función monótonamente crecientes, es decir, x>y implica log (x)>log (y) [105]- [106].

Finalmente, se tiene un vector de características por cada ensayo, $\mathbf{f}_\tau = [f_1, f_2, ..., f_p]^T$.

[27] Este comando ordena los elementos de un vector en orden ascendente. En caso de ser una matriz tratará las columnas como vectores y ordenará cada columna [266].
[28] La varianza "*representa la potencia de la fluctuación*" [268]. "*Potencia de la componente alterna*" [269].
[29] Una potencia siempre es positiva.

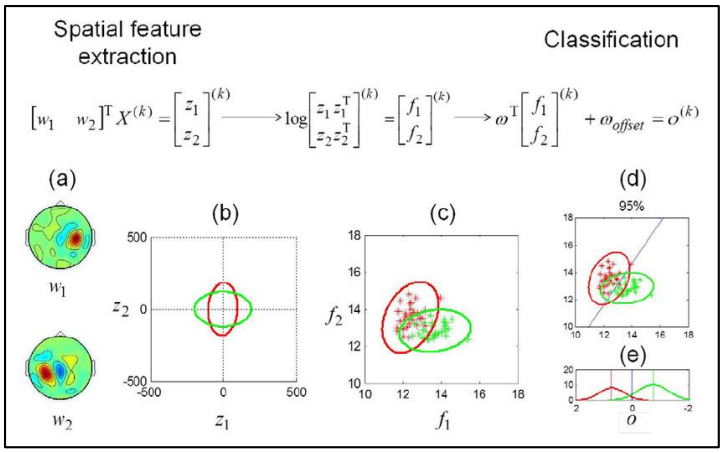


Figura 4-15. Ejemplo de uso del Common Spatial Patterns (CSP) convencional. Se muestran datos de 2 características y el modelo de análisis discriminante lineal de Fisher (FLDA): *(a) "Filtros CSP"; (b) "Dos conjuntos de datos después de la rotación por CSP para maximizar la relación de las varianzas a lo largo de los dos ejes"* [107]. Se deduce cómo es la varianza de la señal proyectada. *"El verde indica la MI del lado izquierdo y el rojo indica la MI del lado derecho; (c) logaritmo de la varianza de la característica proyectada; (d) la línea de discriminación de FLDA; y (e) distribuciones de los resultados del clasificador para dos clases*." Fuente: [108].

Las principales ventajas de CSP son:

- Permite obtener altos rendimientos de clasificación [60].
- Es eficiente computacionalmente y sencillo de aplicar. Uno de los enfoques más populares y eficientes [60].
- Resuelve el problema de que se mezclasen las señales, siendo una de varianza máxima y otra de varianza mínima, rastreando de qué sensor provenían.
- "*Tiene la capacidad de proyectar señales EEG procedentes de varios canales en un subespacio donde se destacan las diferencias entre las clases y se minimizan las similitudes*" [79].
- *"Hay un método alternativo llamado CSP dependiente del tiempo (TDCSP) que optimiza los filtros CSP y refleja de forma efectiva los cambios de la distribución espacial discriminativa en el tiempo*" [79].

Las principales desventajas de CSP son:

- No es robusto al ruido y a las no estacionariedades [60].
- Propenso al sobreajuste [60].
- Requiere muchos ejemplos de entrenamiento, lo que lleva a largos tiempos de calibración [109].
- CSP no puede utilizarse si hay más de dos clases, pero existen variantes del CSP que son multiclase [77].
- "*El CSP se utiliza habitualmente para analizar los patrones espaciales de las señales de EEG multicanal MI. Sin embargo, el CSP depende considerablemente de la banda de frecuencia de las señales de EEG."* Para hacer frente a este problema, existen variantes del algoritmo, son versiones ampliadas del CSP. Por

ejemplo, el CSP de banco de filtros (FBCSP) o el CSP regularizado de banco de filtros (FBRCSP) [86]. También existe regularizado CSP (RCSP), se puede consultar una explicación detallada de este método en el documento [110].

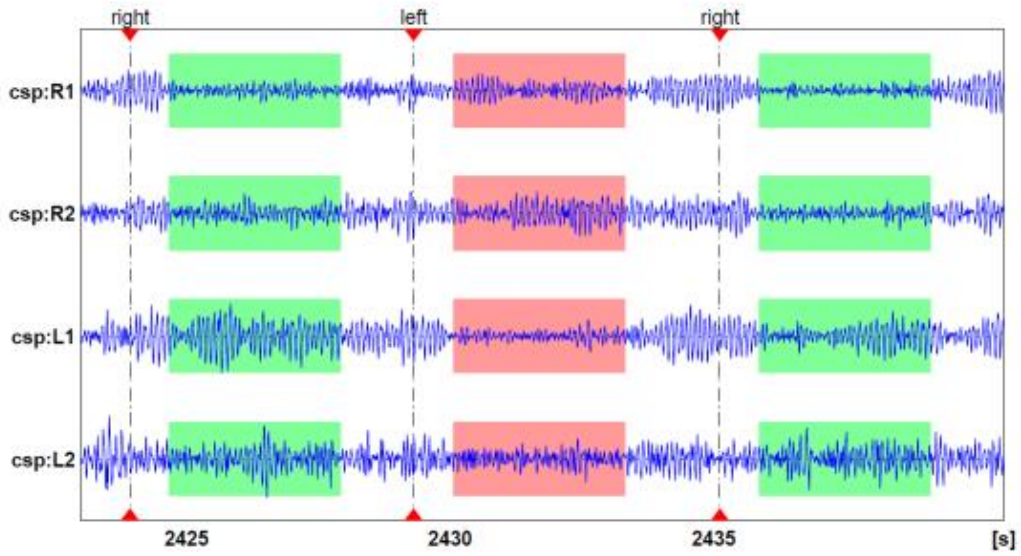


Figura 4-16. Ejemplo de señales de EEG filtradas con cuatro CSP diferentes, durante la imaginación motora de mano izquierda y derecha. Fuente: [109].

4.4.1 Filter bank common spatial pattern (FBCSP)

Es una variación del filtro espacial CSP. *"El rendimiento del filtro espacial CSP depende de la banda de frecuencia operativa del EEG. El CSP se suele utilizar un rango de frecuencias amplio o se selecciona manualmente un rango de frecuencias específico para el sujeto. El FBCSP realiza una selección autónoma de las características clave de discriminación temporal-espacial del EEG"* [111]. Consiste en filtrar las mediciones de las señales EEG con paso de banda en múltiples bandas de frecuencia utilizando un banco de filtros. Luego, para cada banda de frecuencia, se optimizan los filtros espaciales mediante el algoritmo clásico CSP. Las características CSP se extraen de cada una de estas bandas. A continuación, entre los múltiples filtros espaciales obtenidos, se seleccionan las mejores características resultantes mediante algoritmos de selección de características[30], es decir, seleccionan automáticamente pares de bandas de frecuencias discriminativas y sus correspondientes características CSP [112]- [111].

[30] Normalmente una selección de características basada en la información mutua.

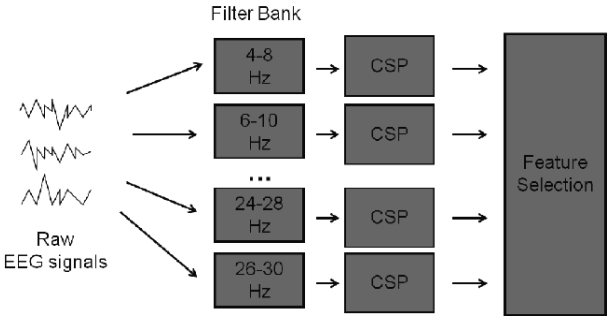


Figura 4-17. Principio de los patrones espaciales comunes del banco de filtros (FBCSP): 1) filtrado de paso de banda de las señales EEG en múltiples bandas de frecuencia utilizando un banco de filtros; 2) optimización del filtro espacial CSP para cada banda; 3) selección de los filtros más relevantes (tanto espaciales como espectrales) utilizando la selección de características en las características resultantes. Fuente: [112].

En el documento [111] se ve un ejemplo de su uso en BCI. En él se ve que se establecen cuatro etapas de procesamiento de las mediciones de EEG: filtros de paso de banda múltiple utilizando filtros de fase cero de Chebyshev de tipo II, filtrado espacial utilizando el algoritmo CSP, selección de características de CSP y clasificación de las características de CSP seleccionadas. Otro documento que se puede consultar es [113].

4.5 Conclusiones

El objetivo es extraer de la señal un conjunto de características relevantes que la describen. Existen varios métodos, algunos de los que destacan en BCI son los siguientes:

- Fast fourier transform (FFT): Es en el dominio de la frecuencia. "*Mediante el uso de la FFT, la señal EEG puede mapearse desde el dominio del tiempo al de la frecuencia"*. Descompone la señal en sus componentes de frecuencia y determina su intensidad relativa.
- Modelos Autorregresivos (AR): Es en el dominio del tiempo. Estima la densidad del espectro de potencia (PSD) de la señal.
- Wavelet transform (WT): Es en el dominio del tiempo-frecuencia. Descompone una señal en forma de funciones, que forman una función base y son denominados "**wavelets**".
- Filter bank common spatial pattern (FBCSP): Es una variación del filtro espacial CSP. Consiste en filtrar las mediciones EEG con paso de banda en múltiples bandas de frecuencia utilizando un banco de filtros. Para cada banda se obtienen los filtros espaciales mediante CSP y se selecciona el mejor filtro.
- Common Spatial Patterns (CSP): El método más comúnmente usado es el CSP. Genera filtros espaciales que minimizan la variación de una clase y maximizan la variación de otra clase simultáneamente. Resolviendo el problema de que se mezclen las señales, pues rastrea de qué sensores provienen.
  En los MI-BCI se puede resumir la mayor parte de la información relevante sobre las mediciones en dos tipos de matrices de covarianza: las matrices de covarianza de las observaciones filtradas (empleadas para la reducción de la dimensionalidad) y las matrices de covarianza de las características (que se requieren para la clasificación en LDA).

  A partir de la matriz de observación del ensayo, se calcula la matriz de covarianza muestral. Después, se le aplica a esta covarianza una normalización de las muestras en potencia (método refinado), para hacerla menos sensibles al ruido. Se obtendrá una matriz de covarianza por cada ensayo.

  Sabiendo esto, se obtienen las covarianzas medias de cada clase, se hace una media aritmética de las covarianzas de los ensayos.

Después se procede a reducir la dimensión. Se trabajará con una matriz de covarianza de menor dimensión, para ello se multiplica la original por la matriz que contiene los filtros espaciales **w**.

La obtención de los filtros se basa en el cociente de **Rayleigh**, siendo el numerador la varianza de la clase 1 y el denominador la de la clase 2. Se puede interpretar como una SNR. Las observaciones tienen dos componentes, las dos clases mencionadas. Una será la señal de interés y la otra se considerará ruido.

Los puntos críticos de ese cociente que maximizan y minimizan la función son los filtros espaciales que se quieren obtener, hallar máximos y mínimos de una función cuadrática, es un problema de autovalores.

Una vez obtenida la matriz de filtros espaciales, se halla la covarianza de las observaciones filtradas espacialmente. Las características de cada ensayo son el logaritmo natural de las varianzas (potencia) normalizadas de la señal de salida del filtro espacial. Se aplica logaritmo para que las características sean lo más parecido posible a una Gaussiana.

5 MACHINE LEARNING

"Machine learning is a core, transformative way by which we're re-thinking how we're doing everything."

Sundar Pichai

Una vez obtenidas las características del ensayo se procede con el aprendizaje automático, también conocido como Machine Learning. El aprendizaje automático se trata fundamentalmente de extraer el valor de grandes conjuntos de datos. A menudo, la motivación es producir un algoritmo que pueda imitar o mejorar el desempeño humano [89].

Básicamente consiste en algoritmos que permiten predecir de qué clase es un elemento conociendo las características de éste. Por ejemplo, si se quisiese clasificar frutas en una imagen, sus características podrían ser el color, diámetro, área, centro de masa y bounding box[31]. Una vez determinada las características hay que predecir el tipo de fruta que se está estudiando, según su parecido con las clases de frutas que puede haber. Las frutas del mismo tipo tendrán características parecidas, pero lo más probable es que no sean iguales. Además, puede haber una característica que sea muy parecida a la de otro tipo, por ejemplo, los plátanos y limones son amarillos, pero no tienen la misma forma. Hay que saber elegir las características y el método de clasificación adecuado según la circunstancia. Por si interesa, un ejemplo de clasificación de imágenes se encuentra en el documento [114].

Hay dos etapas en la clasificación: Lo primero es el entrenamiento del clasificador y lo segundo la validación de éste. En el entrenamiento del clasificador, a partir de la información relevante de un conjunto de muestras, se configuran los parámetros internos que regirán el comportamiento del algoritmo. En la validación se usan muestras distintas a las del entrenamiento y se le asigna a cada muestra una clase en función de los rasgos que comparten con las muestras de entrenamiento.

[31] Cuadrado que circunscribe el objeto

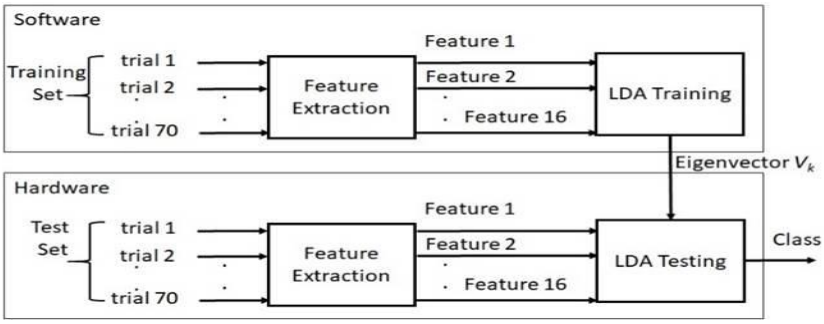


Figura 5-1. Ejemplo de proceso de un clasificador, primero de entrena y luego se valida, en este caso con un LDA (siendo V_k un vector propio que proporcionó la mejor separabilidad entre clases). Fuente: [65].

Existen diferentes clasificadores, los más usados en BCI son ANN, kNN, LDA, SVM, LR, entre otros. No se puede establecer la calidad general de un método de clasificación sobre otro, porque depende del conjunto de datos de cada caso [80].

Los tipos de Machine Learning son:

- Supervisados: En este tipo se conoce la clase a la que pertenecen los patrones[32] de entrenamiento, las muestras de entrenamiento están etiquetadas. Aprenden funciones que relacionan los datos de entrada y salida. Si la variable de salida es categórica es un problema de clasificación, si la salida es un valor de un espacio continuo es un problema de regresión. Ejemplos de supervisados: Análisis discriminante (LDA, QDA y RDA), SVM, Naive Bayes, KNN, Decission Trees, LR, Nearest Archetype, Gradiente descendente, Riemannian manifold.

- No supervisados: Las muestras de entrenamiento no están etiquetadas. El objetivo es aumentar el conocimiento estructural de los datos (clustering, reducción de dimensionalidad o aprendizaje topológico). Estos clasificadores tienden a ser menos precisos que los supervisados. Ejemplos de no supervisados: K-means, Gaussian EM. Dentro de los no supervisados se encuentra el Aprendizaje de la Representación (Deep Learning) [115].

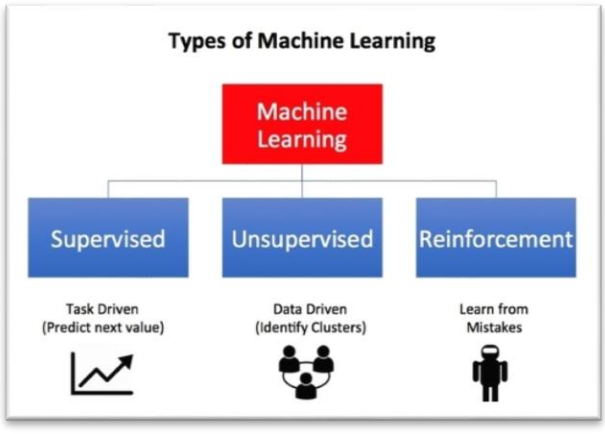


Figura 5-2. Tipos Machine Learning. Fuente: [115].

El algoritmo de clasificación es tan importante como el método de extracción de características. Los métodos no lineales poseen un mejor rendimiento de clasificación en comparación con los métodos lineales; sin embargo, son más complicados de implementar. Los clasificadores lineales, como el LDA o el SVM, son más adecuados

[32] Un patrón es un conjunto de características

para la implementación de sistemas BCI en tiempo real debido a su bajo coste computacional, lo que ha convencido a los investigadores para utilizarlos en la clasificación de señales biomédicas [65].

Se explicarán algunos clasificadores supervisados. Exponiendo de manera más exhaustiva el clasificador LDA, ya que da buenos resultados en la clasificación de dos clases y es una buena elección para diseñar sistemas BCI online con una respuesta rápida, en el código de MATLAB es el que principalmente se usó.

5.1 Análisis discriminante

5.1.1 Linear Discriminant Analysis (LDA)

Normalmente, para aprender qué tipo de vector de características corresponde a qué clase[33], los clasificadores intentan modelar qué área del espacio de características está cubierta por los vectores de características de entrenamiento de cada clase (clasificador generativo) o intentan modelar el límite entre las áreas cubiertas por los vectores de características de entrenamiento de cada clase (clasificador discriminante). En el caso de BCI, los clasificadores más utilizados son los clasificadores discriminantes y, en particular, los clasificadores de análisis discriminante lineal (LDA) [112]. El LDA es particularmente popular porque sirve tanto como clasificador como técnica de reducción de dimensionalidad que está supervisada [116]- [117].

Puede interpretarse desde dos perspectivas. La primera interpretación, más procedimental, es debida a Fisher; permite comprender mejor cómo el LDA realiza la reducción de la dimensionalidad. La segunda interpretación es probabilística; es útil para entender los supuestos del LDA [116].

El LDA se aplica generalmente para clasificar patrones en dos clases, como se ve en Figura 5-4, aunque es posible extender el método a varias clases, poniendo de ejemplo la Figura 5-3. El LDA para varias clases es conocido también como **Canonical Variates**, el cual se puede consultar en el libro [89].

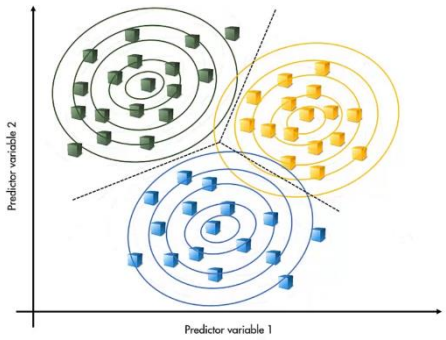


Figura 5-3. Caso de clasificación de más de dos clases con LDA. Similar al clasificador de Bayes, el análisis discriminante funciona asumiendo que las observaciones de cada clase de predicción pueden modelarse con una distribución de probabilidad normal. Sin embargo, no se asume la independencia de cada predictor. Fuente: [78].

"*Es un sistema que define una función de discriminación lineal y representa un hiperplano*[34] *en el espacio de características con el objetivo de separar las clases*" [118]- [80]. En un caso bidimensional sería una recta en el plano y en unidimensional sería un punto. El vector de características pertenecerá a una clase u otra según el lado del hiperplano donde esté. Al ser un hiperplano la función discriminante, hace que sean de gran sencillez su implementación, pero implica a su vez que sea poco general su utilización.

[33] Cada clase es un estado mental

[34] Un hiperplano *"es una extensión del concepto de plano. Es un análogo de muchas dimensiones al plano (de dos dimensiones) en el espacio tridimensional"* [243].

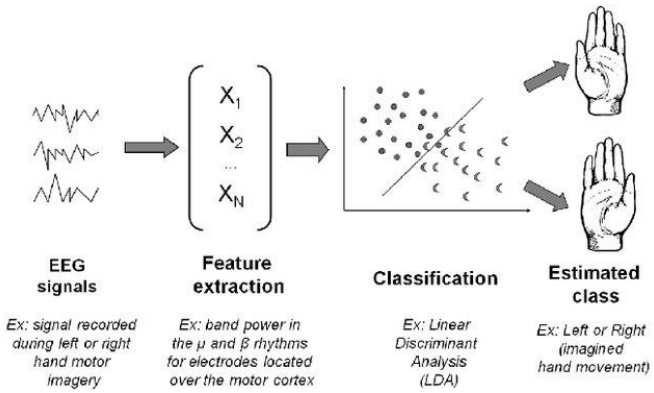


Figura 5-4.Una estructura clásica de procesamiento de señales EEG para BCI basado en imaginación motora, es decir, un MI-BCI reconociendo movimientos imaginados a partir de señales EEG. Fuente: [112].

Es la opción más recomendada para realizar el sistema MI-BCI online, que se quiere diseñar en este trabajo. Es sencillo y da una precisión aceptable. Al no tener alto coste computacional, es rápido. El problema principal que puede tener es que puede dar clasificaciones erróneas por presencia de valores atípicos o ruido fuerte. Por otra parte, hay que tener en cuenta que si hay muy pocos datos de entrenamiento la probabilidad de acierto será baja. Esto pasa en general con todos los clasificadores. A mayores datos de entrenamiento, mejor puede ser el clasificador. En el caso del MI-BCI online, al principio se tendrá una probabilidad de acierto de aproximadamente del 50%, prácticamente aleatorio, pues el clasificador se iniciará cuando tenga un dato de cada clase, siendo dos clases las posibles.

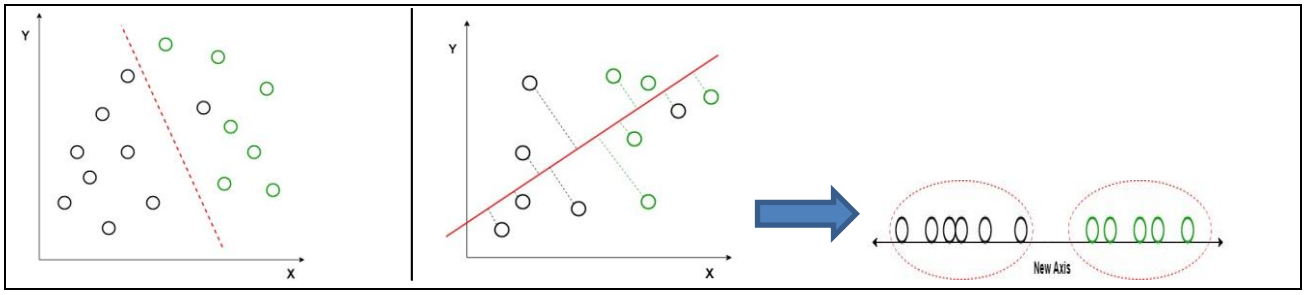


Figura 5-5. En este ejemplo, no hay una recta que separe bien ambas clases, hay que usar una proyección. Se tiene que buscar una recta sobre la que proyectar de forma adecuada. Finalmente, se reduce el gráfico de dos dimensiones a uno en una dimensión para maximizar la separabilidad entre las clases. Se separa a partir de un punto intermedio las clases, en lugar de una línea en el plano. El LDA es un clasificador lineal en el que se realiza una proyección. Fuente: [119].

Como se verá en la Figura 5-6, y como se vio en la Figura 5-5, se simplifica la clasificación al encontrar una dirección en el espacio de características que permita clasificar correctamente las dos clases, es decir, se buscará la mejor recta sobre la que proyectar los patrones en el espacio de dimensión-p.

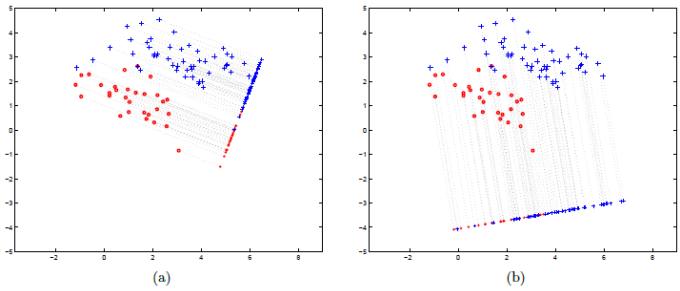


Figura 5-6. Las cruces grandes azules representan datos de la clase 1 y los círculos grandes rojos de la clase 2. Se proyectarán en una dirección todos los patrones. Sus proyecciones en 1 dimensión están representadas por sus contrapartes pequeñas. (a): Análisis discriminante lineal de Fisher: Aquí hay poca superposición de clases en las proyecciones. (b): Reducción de dimensión no supervisada usando Análisis de Componentes Principales. Hay una superposición de clases considerable en la proyección. Tanto el LDA como el PCA son métodos de reducción de dimensionalidad, pero LDA está supervisada. Tanto en (a) como en (b) la proyección unidimensional es la distancia a lo largo de la línea, medida desde un punto fijo elegido arbitrariamente en la línea. Fuente: [89].

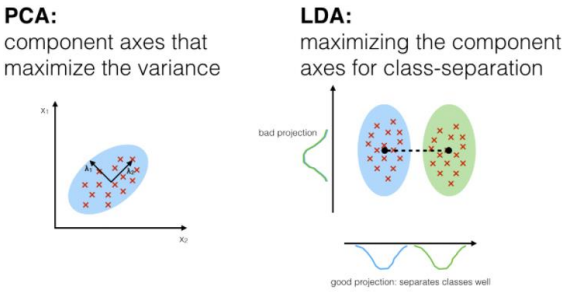


Figura 5-7. Comparando realización de LDA y PCA. Fuente: [120].

Hay dos propiedades que se desea que se cumplan a la hora de realizar la clasificación:

- **Dispersión interclase alta**: Una dispersión externa alta, es que la media de cada conjunto de patrones de cada clase (las nubes de puntos rojos y azules en Figura 5-6) tienen que estar lo más separadas posibles.

- **Dispersión intraclase baja**: Una dispersión interna baja, los patrones de la misma clase estén lo más cercanos posible a su media. La varianza en cada conjunto de patrones sea pequeña.

Estas propiedades dependen de las características que se usen. También es deseable que las características sean lo más independientes posibles, correlación baja, para facilitar la clasificación.

Como se vio en Figura 5-6, se busca una proyección tal que las distribuciones proyectadas tengan una superposición mínima. Esto se puede lograr si las medias gaussianas proyectadas están separadas al máximo, esto es que $(\mu_1 - \mu_2)^2$ es grande. Pero no es tan sencillo, ya que si las varianzas σ_1^2, σ_2^2 son grandes, podría haber una gran superposición en las clases.

Por tanto, la función objetivo sería:

$$\frac{(\mu_1 - \mu_2)^2}{\pi_1\sigma_1^2 + \pi_1\sigma_2^2} \quad (5.1)$$

Donde π_k representa la fracción del conjunto de datos (patrones) en la clase k [89].

Se puede obtener un compromiso entre dispersión interclase alta y dispersión intraclase baja. Existen varias técnicas para calcular la función discriminante que permitirá asignar a cada muestra su clase. Se basa este problema en el método de Fisher[35]. El método LDA también se conoce como Fisher-LDA, pues es una generalización del discriminante lineal propuesto por Fisher.

Se trata de crear, a partir de p variables clasificadoras una función que será combinación lineal de dichas variables, por ejemplo $f_{fisher} = a_1 f_1 + ... + a_p f_p$. Se persigue que los valores de la función se diferencien todo lo posible entre una clase y otra, y sean parecidos para los de la misma clase. Habrá que hallar los valores de los coeficientes que permitan esa condición. Sería reducir las p variables independientes a una única dimensión. Para hallar los coeficientes sería maximizar la variación de f_{fisher} entre clases y minimizar la variación dentro de cada clase, es decir, maximizar el ratio $\frac{\text{variación interclase}}{\text{variación intraclase}}$. Las variaciones intraclase e interclase se calculan tomando en cuenta las correspondientes sumas de los cuadrados de las desviaciones de las muestras con respecto las medias[36]. "*La solución resultante indica que los coeficientes desconocidos son las coordenadas de un autovector asociado al mayor autovalor de cierta matriz cuyos elementos dependen únicamente de los valores observados de las variables independientes*" [121].

La función (5.1) se puede expresar como una función cuadrática en términos de proyección α:

$$F(\alpha) = \frac{\alpha^T \mathbf{A} \alpha}{\alpha^T \mathbf{B} \alpha} \quad (5.2)$$

Siendo α el filtro óptimo para proyectar las características. Se puede apreciar que es semejante al planteamiento explicado en el CSP, maximizar un cociente de Rayleigh, el cociente de dos funciones cuadráticas semidefinidas positivas. Se podría decir que se está aplicando una reduciendo de dimensión a las características. Se obtendría la proyección óptima resolviendo un problema de autovalores, pero en este caso la matriz de arriba tiene rango 1, la solución es un único vector.

En resumen, lo que se ha hecho es hallar una dimensión en la que se separan muy bien las clases. Una vez se proyectan las características de dimensión-p, estarán en dimensión-1, una recta. Si los patrones proyectados están por encima de un umbral (un punto de esa recta) son de una clase, y si no, son de la otra clase.

Una vez expuesto el concepto, se explicará los cálculos que hay que realizar en el proceso del LDA. Primero se halla la media de cada conjunto de características (patrones) de cada clase:

$$\mu_k = \frac{1}{N_{C_k}} \sum_{\tau : c(\tau) = c_k} \mathbf{f}_\tau \qquad k = 1, 2 \qquad (5.3)$$

Denominando la k como cada clase correspondiente.

Se halla variación de los patrones de cada clase:

$$\hat{\Sigma}_{f|c_x} = \frac{1}{N_{C_k}} \sum_{\tau : c(\tau) = c_k} (\mathbf{f}_\tau - \mu_k)(\mathbf{f}_\tau - \mu_k)^T \qquad k = 1, 2 \qquad (5.4)$$

[35] *"En el reconocimiento de patrones, el discriminante lineal de Fisher es un método discriminante lineal cuya intención es proyectar puntos de datos en un espacio p-dimensional en un espacio c-1 dimensional, de modo que en este espacio se encuentren diferentes tipos de puntos muestrales. Las proyecciones de la imagen deben estar separadas tanto como sea posible, y los similares deben ser lo más compactos posible "* [251].

[36] Centroides de cada clase

Básicamente, es calcular la matriz de covarianza de cada clase, se obtendrían entonces dos matrices de dimensión pxp.

Luego se halla la probabilidad a priori de la clase:

$$p(c_k) = {N_{c_k}}\Big/{(N_{c_1} + N_{c_2})} \qquad k = 1,2 \qquad (5.5)$$

La covarianza media (sigma) de los patrones:

$$\hat{\Sigma}_f = p(c_1)\hat{\Sigma}_{f|c_1} + p(c_2)\hat{\Sigma}_{f|c_2} \qquad (5.6)$$

Es la covarianza de la clase 1 ponderada con la probabilidad de que se dé la clase 1, más la covarianza de la clase 2 por la probabilidad que se dé clase 2. Normalmente, en los proyectos con MI-BCI, se asumen clases equiprobables. En este trabajo, para simplificar el cálculo y reducir coste computacional, se probó a estimar la covarianza media de los patrones como la covarianza de la matriz formada por los vectores de características de entrenamiento, $\hat{\Sigma}_f = \text{cov}([\mathbf{f}_1 \quad \dots \quad \mathbf{f}_{N_\tau}])$. Obteniéndose un clasificador semejante al que ofrecen las rutinas internas de MATLAB para LDA.

El filtro óptimo para proyectar es:

$$\alpha = \hat{\Sigma}_f^{-1}(\mu_1 - \mu_2) \qquad (5.7)$$

Se halla el umbral, el punto medio, el cual te permite saber a partir de qué valor se establece que un patrón sea de una clase o de otra, dependiendo de si es mayor o menor a este valor:

$$\beta = \frac{1}{2}(\mu_1 - \mu_2) - \frac{\log p(c_1) - \log p(c_2)}{(\mu_1 - \mu_2)^T \hat{\Sigma}_f^{-1}(\mu_1 - \mu_2)}(\mu_1 - \mu_2) \qquad (5.8)$$

Los patrones se separan en torno a un punto sobre la línea real en que se han proyectado. Si las probabilidades de las clases son iguales entonces β está centrado, el segundo término es cero.

Una vez creado un modelo de clasificador a partir de los datos, se puede usar para clasificar nuevas observaciones.

Obtenido el α y β óptimos para la clasificación, se implementa el clasificador con los datos de testeo, calculando sus características y determinando la clase que se le asigna a cada patrón, $k_{asignada}$, siendo de clase 1 o 2. Se establece en qué región del espacio de predicción se encuentra cada patrón de validación. La $k_{asignada}$ se calcula del siguiente modo:

$$k_{asignada} = 1.5 - 0.5 sign(\alpha^T(\mathbf{f}_\tau - \beta)) \qquad (5.9)$$

La función de MATLAB $sign()$ da el signo de lo que halla entre paréntesis, es decir, da 1 o -1. La parte $\alpha^T(\mathbf{f}_\tau - \beta)$ se puede interpretar como el margen de error que se está teniendo al realizar la clasificación, si se conoce la clase real del patrón. El margen se usa en el código de MATLAB para indicarle al usuario si está pensando de manera correcta o no, al entrenar el clasificador en tiempo real.

En lo que respecta al proceso de validación del clasificador en el MI-BCI; primero se hallaría la covarianza de las observaciones y se filtraría espacialmente, (4.27), usando los datos de testeo. Luego se calcularían las

características usando (4.28). Finalmente, se evaluaría el vector de características obtenido en (5.9), prediciendo qué clase es.

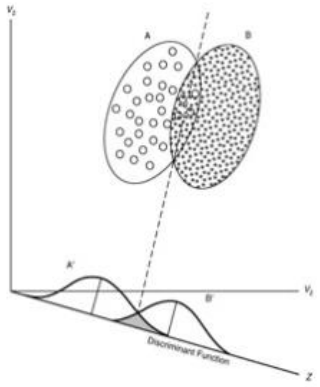


Figura 5-8. Cuando se trata de más de una variable de predicción, el clasificador LDA asume que las observaciones de la k-ésima clase se extraen de una distribución gaussiana multivariante N(μ_k,Σ). Además, se presupone que tienen distintas medias y matrices de covarianza aproximadamente iguales. En caso de no cumplirse las premisas, el algoritmo no funcionará tan bien como se desea. Fuente: [123].

Otra posible manera de implementar el LDA, sería considerando que "*una observación se clasifica en un grupo si la distancia al cuadrado (distancia de Mahalanobis[37]) de la observación hasta el centro del grupo (media) es el mínimo"* [122]. *"Hay una parte única de la fórmula de la distancia al cuadrado para cada grupo que se denomina función discriminante lineal para ese grupo. Para cualquier observación, el grupo con la distancia al cuadrado más pequeña tiene la función discriminante lineal más grande y la observación se clasifica entonces en ese grupo"* [122].

El clasificador asigna una observación a una clase cuya puntuación (función) discriminante es mayor, pudiéndose expresar esta del siguiente modo [123]:

$$\hat{d}_k(f) = \mathbf{f}^T \hat{\Sigma}^{-1}_{f|c_x} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k \hat{\Sigma}^{-1}_{f|c_x} - \hat{\mu}_k + \log(p(c_k)) \qquad (5.10)$$

Por otra parte, existen variantes y mejoras del LDA, como Regularized Fisher LDA [124]- [83].

Para saber más sobre el análisis discriminante se puede consultar el documento [125].

5.1.1.1 Resumen del clasificador LDA

En el caso del clasificador LDA, se busca un hiperplano que permita separar las dos clases. Se simplifica la clasificación buscando una recta sobre la que proyectar los patrones, en el espacio de características, de forma que queden los de distinta clase lo más separados posible y lo de las misma clase lo más cercanos posible; permitiendo así clasificar correctamente las dos clases. Una vez realizada esta proyección, se establece un umbral, un punto en la recta, que establezca la separación de las clases. Al clasificar una nueva muestra, también se proyecta sobre esta recta, y luego se determina el tipo de muestra de acuerdo con la posición del punto proyectado [117]. Los valores que sean mayor de este umbral se considerarán de una clase y los menores de la otra clase.

Para ello hay que calcular la media, la covarianza y la probabilidad a priori de cada clase. Con ello se puede hallar la covarianza media de los patrones (de todas las clases), y así hallar el filtro óptimo para proyectar, luego se hallaría el umbral de separación.

[37] *"Se diferencia de la distancia euclídea en que tiene en cuenta la correlación entre las variables aleatorias"* [155].

5.1.2 Quadratic Discriminant Analysis (QDA)

Es una versión generalizada del LDA, pero con una superficie de decisión cuadrática para separar las medidas de dos o más clases, como se ve en la Figura 5-9. Esto permite que se disponga de cierta curvatura para ajustarse mejor a escenarios que se alejan moderadamente de la linealidad. *"Se encuentra en un punto medio entre el método no paramétrico K-NN y los métodos lineales LDA y Regresión Logística"* [126]. *"QDA es más adecuado cuando hay un número limitado de observaciones ya que, al hacer ciertas asunciones sobre los límites de decisión, no se corre el riesgo de sobreajuste"* [126]. Una desventaja de QDA es que no puede utilizarse como técnica de reducción de la dimensionalidad, mientras que LDA sí.

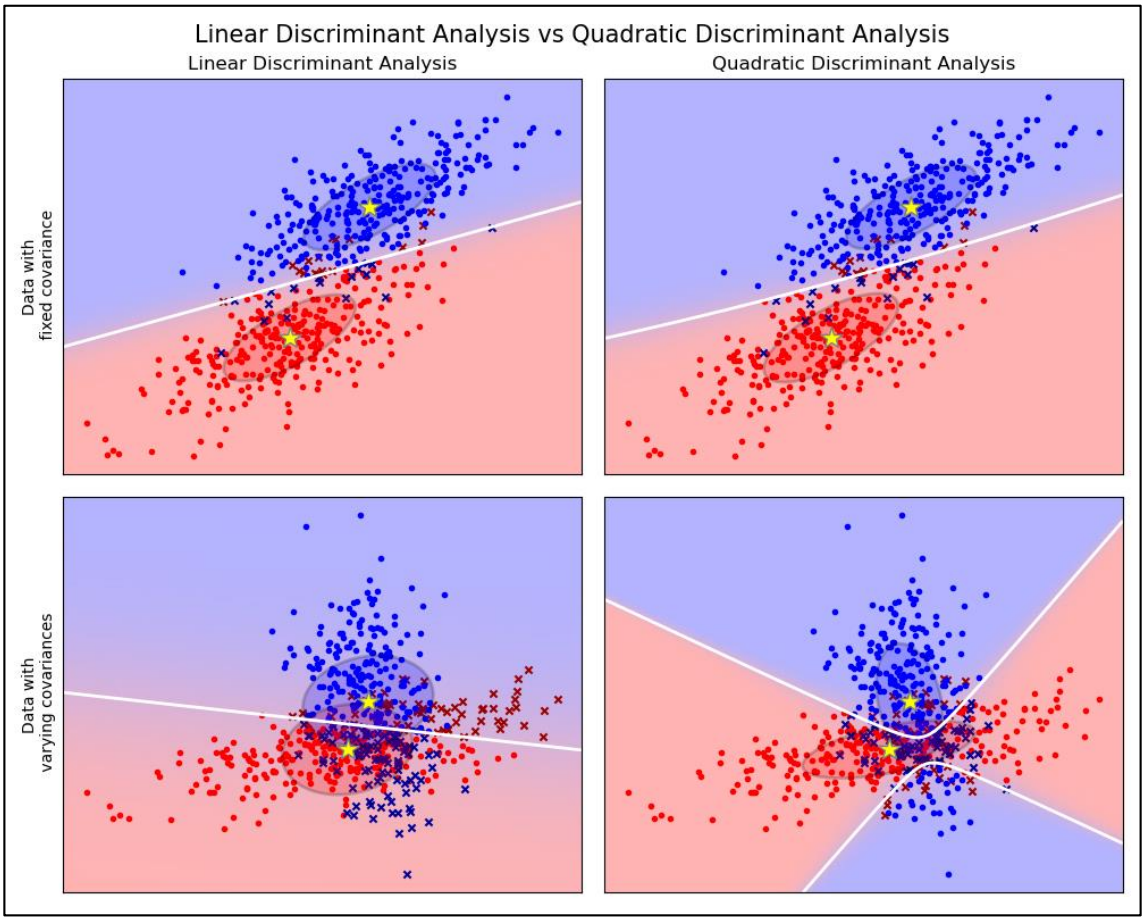


Figura 5-9. Comparación LDA y QDA. El gráfico muestra los límites de decisión para el análisis discriminante lineal y el análisis discriminante cuadrático. En la fila superior se tienen datos con misma covarianza y en la inferior covarianzas distintas. La fila inferior demuestra que el análisis discriminante lineal solo puede aprender límites lineales, mientras que el análisis discriminante cuadrático puede aprender límites cuadráticos y, por lo tanto, es más flexible. Fuente: [127].

Tanto LDA como QDA suponen que las observaciones de cada clase se extraen de una distribución gaussiana [123]. Sin embargo, el análisis discriminante lineal presupone que las matrices de covarianzas dentro de cada grupo deben ser aproximadamente iguales [128], mientras que el cuadrático no parte de este supuesto. Hay que tener en cuenta que, "*si se considera que la distancia de Mahalanobis es una manera adecuada de medir la distancia de una observación a un grupo, entonces no se necesita hacer supuestos sobre la distribución subyacente de los datos"* [122].

Tanto en el lineal como en el cuadrático, una observación se podría clasificar en el grupo que posee la distancia al cuadrado menor. No obstante, la distancia al cuadrado se puede simplificar en una función lineal en el LDA, pero no en el caso cuadrático.

En QDA se estima la covarianza para cada clase. Siendo la función de discriminación cuadrática la siguiente:

$$d_k(f) = -\frac{1}{2}\mathbf{f}^T\hat{\Sigma}_{f|c_x}^{-1}\mathbf{f} + \mathbf{f}^T\hat{\Sigma}_{f|c_x}^{-1}\hat{\mu}_k - \frac{1}{2}\hat{\mu}_k{}^T\hat{\Sigma}_{f|c_x}^{-1}\hat{\mu}_k - \frac{1}{2}\log\left|\hat{\Sigma}_{f|c_x}^{-1}\right| + \log(\hat{\pi}_k) \qquad (5.11)$$

Llamando $\hat{\pi}_k$ a la probabilidad previa de que una observación corresponda a la k-ésima clase, $\hat{\Sigma}_{f|c_x}$ la covarianza de cada clase y μ_k la media de cada clase. El clasificador asigna una observación a la clase para la $d_k(x)$ más grande [123] - [129].

Un ejemplo de uso de este clasificador se puede consultar en el documento [130], donde se puede apreciar una comparación de uso del LDA y QDA aplicados en BCI.

"Si en el modelo QDA se supone que las matrices de covarianza son diagonales, entonces se considera que las entradas son condicionalmente independientes en cada clase, y el clasificador resultante es equivalente al clasificador Gaussiano Naive-Bayes" [127].

Para saber más sobre el análisis discriminante cuadrático se puede consultar [131].

5.1.2.1 Resumen del clasificador QDA

En el QDA se pretende separar las clases utilizando curvas en lugar de rectas, utilizándose una función de decisión cuadrática. Se mide la distancia de un patrón al centro de cada clase, asignándole la clase de la que a menor distancia esté, en LDA se podría realizar también aplicando este concepto; considerando que LDA presupondría matrices de covarianza iguales para cada clase y QDA no. La medida de distancia que se suele usar es la distancia de Mahalanobis. En LDA la distancia se simplifica en una función lineal, mientras que el QDA no.

5.1.3 Regularized Discriminant Analysis (RDA)

El análisis discriminante regularizado (RDA) es un compromiso entre el LDA y el QDA. La regularización es el proceso de encontrar un pequeño conjunto de predictores que produzcan un modelo predictivo eficaz. El LDA es sencillo en los casos en que el número de observaciones[38] (N_τ) es mayor que la dimensionalidad de cada observación (p), pero si esto no sucede hay dos preocupaciones principales. Primero, la estimación de la matriz de covarianza de la muestra es singular y no se puede invertir. Aunque podemos usar la inversa generalizada en su lugar, la estimación será muy inestable debido a la falta de observaciones. En segundo lugar, la alta dimensionalidad hace que la operación de matriz directa sea formidable. Para solucionarlo se le hace una serie de cambios [132].

Para resolver el problema de la singularidad, en lugar de utilizar $\hat{\Sigma}_{f|c_x}$ directamente, se usa:

$$\tilde{\Sigma}_{f|c_x}(\lambda) = \lambda\hat{\Sigma}_{f|c_x} + (1-\lambda)I_p \quad (5.12)$$

[38] El número de observaciones es el número de patrones, el número de conjuntos de características o de ensayos.

Se sustituye $\hat{\Sigma}_{f|c_x}$ por $\tilde{\Sigma}_{f|c_x}(\lambda)$ en la función discriminante explicada en apartados anteriores. El $\lambda \in [0,1]$ es un parámetro de ajuste que determina si las covarianzas deben estimarse de forma independiente (λ =1), el operador RDA realiza LDA, o deben agruparse (λ =0), el operador RDA realiza QDA [116]- [133].

Hay otras formas de regularización. Se puede reducir $\hat{\Sigma}_{f|c_x}$ a una varianza agrupada $\hat{\Sigma}$, matriz covarianza regularizada. Luego, puede reducirse hacia la covarianza diagonal (covarianza proporcional a la identidad)) requiriendo:

$$\tilde{\Sigma}_{f|c_x}(\lambda) = \lambda\hat{\Sigma}_{f|c_x} + (1-\lambda)\hat{\Sigma} \quad (5.13)$$

$$\hat{\Sigma}(\gamma) = \gamma\hat{\Sigma} + (1-\gamma)\hat{\sigma}^2 I \quad (5.14)$$

Donde γ=1 conduce a la covarianza agrupada y γ=0 conduce a la covarianza proporcional a la identidad. Siendo $\hat{\sigma}^2$ el promedio ponderado de las varianzas de las muestras para cada una de las clases K (centroides). La sustitución de $\hat{\Sigma}_{f|c_x}$ por $\hat{\Sigma}(\lambda,\gamma)$ conduce a una noción más general de covarianza [116]. Hay dos parámetros, λ y γ, que controlan la regularización [134].

Un ejemplo de su uso en MATLAB se puede ver en [135] y para una explicación teórica más detallada se puede consultar el artículo [132]. No se llegó a implementar este clasificador en MATLAB para los experimentos, sería una posible línea de futuro de este trabajo su implementación para comprobar su eficiencia.

5.2 Support Vector Machine (SVM)

"*El SVM es un método kernel estándar en clasificación*" [136]. Este clasificador crea un hiperplano o conjunto de hiperplanos para separar los elementos de cada clase. La principal diferencia con LDA es que persigue maximizar la distancia entre las muestras de entrenamiento más cercanas y el hiperplano [118]. Además, realiza la clasificación sin generar funciones de densidad de probabilidad de las clases, como sí hacen los clasificadores Naive-Bayes o el de Análisis Discriminante.

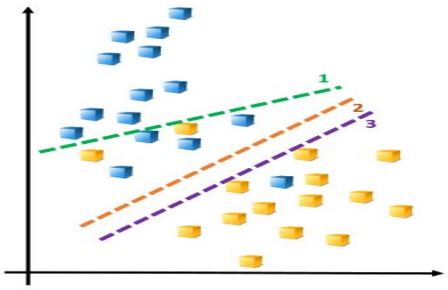


Figura 5-10. Un algoritmo de Support Vector Machine (SVM) clasifica los datos al encontrar el "mejor" hiperplano que separa todos los puntos de datos. Fuente: [78].

Funcionan en problemas de clasificación binaria[39]. Se puede adaptar para usarla para varias clases, usando varias SVM, pero no es aconsejable para esos casos, ya que sería lento.

En un espacio *p*-dimensional, un hiperplano es definido como un subespacio plano y afín[40] de dimensiones *p*−1. En el caso de dos dimensiones, el hiperplano es una recta tal que así: $a_0 + a_1 f_1 + a_2 f_2 = 0$. Dados a_0, a_1 y a_2, todos los puntos [f_1, f_2] que cumplen la igualdad, pertenecen al hiperplano. Esto se puede generalizar fácilmente a p-dimensiones

[39] Dos clases
[40] "El término afín significa que el subespacio no tiene por qué pasar por el origen" [137].

y comprobar si el vector $\mathbf{f}_\tau = [f_1, f_2, ..., f_p]^T$ cumple la igualdad correspondiente. Cuando no cumple la igualdad significa que el punto en cuestión cae a un lado o al otro del hiperplano. Se determina la clase de un punto $\mathbf{f}$ con el signo de una ecuación [137].

La clase asignada como respuesta tiene dos posibles valores de la función de decisión, se identificarán como +1 y -1. Teniendo en cuenta esto, las dos condiciones de pertenecer a una clase u otra, cuando no cumple igualdad a cero, pueden simplificarse. La función de decisión sería:

$$\hat{d}_k(f^*) = a_0 + a_1 f_1^* + ... + a_p f_p^* \quad (5.15)$$

Además, la magnitud de (5.15) da información de cómo de cerca está la observación del hiperplano, averiguando así la confianza del clasificador [137]. Este concepto es semejante al aplicado en LDA, mencionando lo del margen de error, véase ecuación (5.9).

Para las muestras perfectamente separables linealmente, el número de posibles hiperplanos es infinito (Figura 5-14). La solución a este problema es usar el llamado **hiperplano óptimo de separación**, siendo éste el que se encuentra más alejado de todos los patrones de entrenamiento.

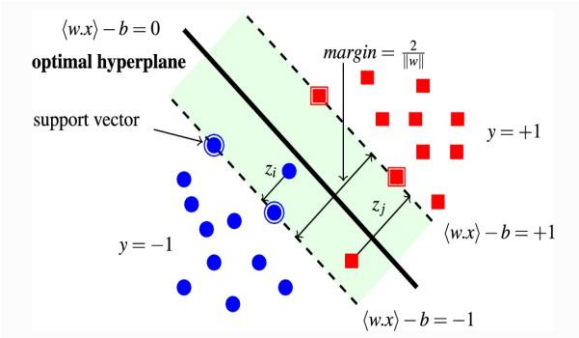


Figura 5-11. Un ejemplo de SVM, en el cual se están permitiendo errores. Fuente: [138].

Como se ve en Figura 5-11, consiste en determinar la distancia perpendicular de cada muestra al hiperplano que se esté estudiando. La menor de estas distancias es lo que se conoce como **margen** en la mayoría de la documentación, la distancia menor entre las muestras de una clase y la otra. El denominado **maximal margin hyperplane** es el hiperplano que consigue que esa distancia mínima entre el hiperplano y los patrones sea lo más grande posible. Esto es solo una idea teórica que no se realiza de manera práctica, pues habría infinitos hiperplanos para medir distancia. La forma realista de resolver el problema es con métodos de optimización [137].

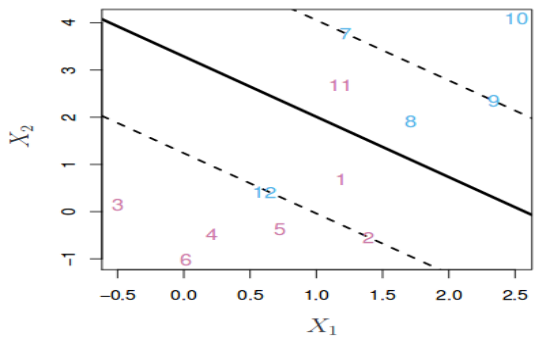


Figura 5-12. Ejemplo SVM permitiendo errores. Siendo los rojos de una clase y los azules de otra clase. Fuente: [137].

Normalmente los patrones no son separables linealmente de manera perfecta. *"Se puede extender el concepto de **maximal margin hyperplane** para obtener un hiperplano que casi separe las clases, pero permitiendo que*

cometa unos pocos errores. A este tipo de hiperplano se le conoce como ***Support Vector Classifier*****"** [137]. "*Se permite que ciertas observaciones estén en el lado incorrecto del margen o incluso del hiperplano"* [137].

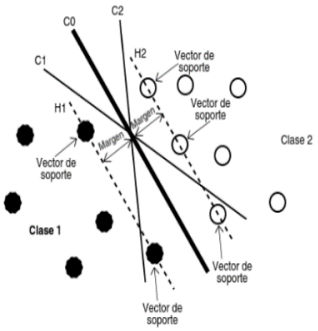


Figura 5-15 Hiperplano óptimo (CO) que maximiza el margen entre los hiperplanos H1 y H2. Fuente: [140]

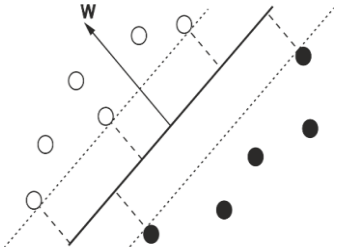


Figura 5-13. Un clasificador lineal, separando con una recta dos clases. El límite de decisión es la línea gruesa, el SVM es parecido al LDA. Fuente: [118].

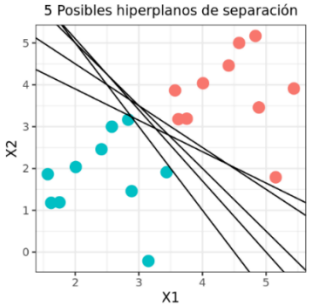


Figura 5-14. Varios hiperplanos posibles. Fuente: [137] .

Se muestra en la Figura 5-15 el problema de separación de dos clases perfectamente separables linealmente. En esa figura, H1 y H2 definen dos hiperplanos, $H2 = \mathbf{w}^T\mathbf{x}_i + b = +1 = y_2$ y $H1 = \mathbf{w}^T\mathbf{x}_i + b = -1 = y_1$, siendo **w** un vector ortogonal al hiperplano C0 y siendo +1 y -1 las etiquetas de cada clase, es decir, la y_i [139]. Quedando entonces como condición de clasificación:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad (5.16)$$

Los patrones más cercanos de cada clase están en estos dos hiperplanos (H1 y H2), son los que se denomina **vectores de soporte**[41]. El hiperplano óptimo es el cual "*maximiza el margen y minimiza el riesgo empírico*[42]" [140]. El margen se calcula a partir de la distancia de los dos hiperplanos. Desarrollándolo matemáticamente, quedará que el margen solo depende de las componentes del vector **w**:

$$margen = margen_1 + margen_2 = 2\frac{1}{\|\mathbf{w}\|} \qquad (5.17)$$

Maximizar el margen será equivalente a minimizar $\frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\mathbf{w}^T\mathbf{w}$. Añadiéndole a esa minimización la condición de clasificación (5.16), se transforma el problema de clasificación en uno de optimización cuadrática estándar, utilizando multiplicadores de Lagrange[43] [139]. Después de realizar cálculos se acaba obteniendo la función del clasificador como:

$$\mathrm{d}(\mathbf{x}) = \mathrm{sign}(\sum_i y_i\alpha_i\mathbf{x}^T\mathbf{x}_i + b) \quad (5.18)$$

[41] "*El hiperplano de separación es definido como el vector entre los puntos, de las 2 clases, más cercanos, a los que se les llama vector soporte*" [244].

[42] Es el error medio medido sobre el conjunto de evaluación [241].

[43] "*En los problemas de optimización, el método de los multiplicadores de Lagrange es un procedimiento para encontrar los máximos y mínimos de funciones de múltiples variables sujetas a restricciones*" [259].

La solución del SVM se puede expresar a partir del producto escalar entre vectores. Siendo α_i los multiplicadores de Lagrange. Sale una solución "*no paramétrica, lo que quiere decir que no habría que ajustar ningún parámetro durante el entrenamiento*" [139]. No obstante, "*cuando los datos no son linealmente separables se tendrá que introducir tolerancia a errores de clasificación, va a ser necesario ajustar algunos parámetros*" [139].

Al depender el hiperplano solo de una pequeña proporción de patrones, habrá robustez frente a patrones bastante alejados del hiperplano. Sólo la modificación de los vectores de soporte modifica la solución del SVM [137].

Una posibilidad para tratar datos de manera no lineal sería usando suavizado de "*condición del margen e introducir tolerancia a errores de clasificación*" [140]. Para ello habría que utilizar variables de holgura. La otra solución más utilizada es el **truco del Kernel**, el cual se detallará proximamente.

En resumen, el SVM consistirá en "*mapear los datos en un espacio de p-dimensión y encontrar un hiperplano de separación con el máximo margen de acuerdo con* ***el teorema de Cover*****"** [141]. Este teorema afirma que un conjunto de datos de entrenamiento que no se puede separar linealmente, probablemente se podrá convertir en otro conjunto de entrenamiento que sea separable linealmente si se proyecta en un espacio que tiene dimensión mayor, mediante una transformación no lineal [141]. A este proceso es lo que se conoce también como **truco del Kernel**, "*permite que algoritmos lineales se puedan aplicar a problemas no lineales*" [140]. En otras palabras, es posible crear un SVM con límite de decisión no lineal mediante una función kernel. Usar un SVM no lineal conduce a un límite de decisión más flexible en el espacio de datos, lo que puede aumentar la precisión de la clasificación.

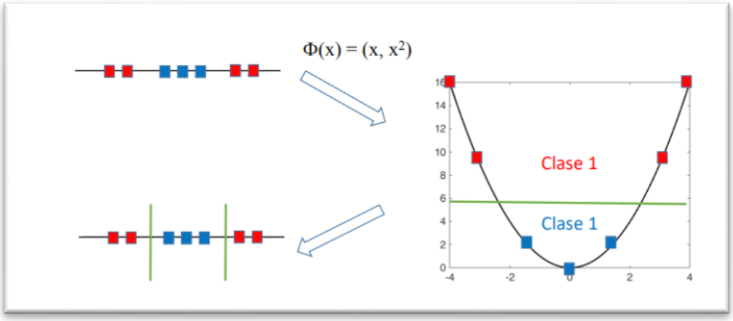


Figura 5-16. Ejemplo de utilización de expansión de la dimensionalidad para clasificación, truco del Kernel Fuente: [136].

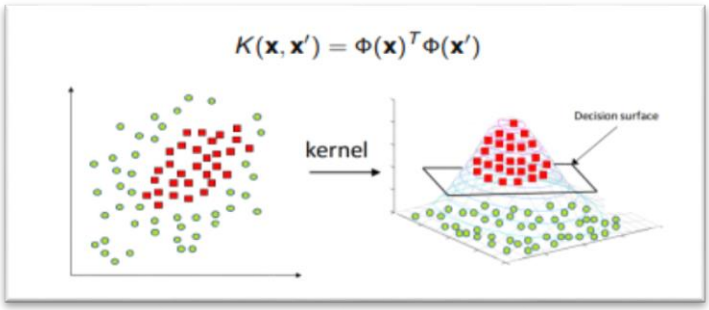


Figura 5-17. Ejemplo del método kernel. "*Habitualmente no es necesario conocer explícitamente el mapping Φ(x), basta con conocer la función núcleo o kernel asociado. Los métodos kernel obtienen una solución lineal en el espacio de características (que se convierte en una solución no lineal en el espacio de entrada)"*. Fuente: [136].

"*Un kernel es una función que devuelve el resultado del producto escalar entre dos vectores realizado en un nuevo espacio dimensional distinto al espacio original en el que se encuentran los vectores*" [137], se define matemáticamente como $k(x,\mathrm{z}) = \varphi(\mathbf{x})^T \varphi(\mathbf{z})$. Hay varios tipos de kernel, como el Sigmoid, el Polinomial-homogénea, el Perceptrón o la Función de base radial Gaussiana. El que se suele utilizar en el campo de las BCI es el gaussiano[44] [118]:

$$k(x,\mathrm{z}) = \exp(-\gamma \left\| x - \mathrm{z} \right\|^2) \qquad (5.19)$$

El γ "*controla el comportamiento del kernel, cuando es muy pequeño, el modelo final es equivalente al obtenido con un kernel lineal, a medida que aumenta su valor, también lo hace la flexibilidad del modelo*" [137]. Cada tipo de kernel tiene unos parámetros que hay que ajustar en el entrenamiento, "*utilizando variación cruzada con algún conjunto de validación*" [142]- [143].

La clasificación de un nuevo punto de datos basada en el modelo SVM es la siguiente:

$$\mathrm{d}(\mathbf{x}) = \mathrm{sign}(\sum_i y_i \alpha_i k(\mathbf{x},\mathbf{z}) + b) \qquad (5.20)$$

Puede ser +b o –b según el criterio que se quiera usar.

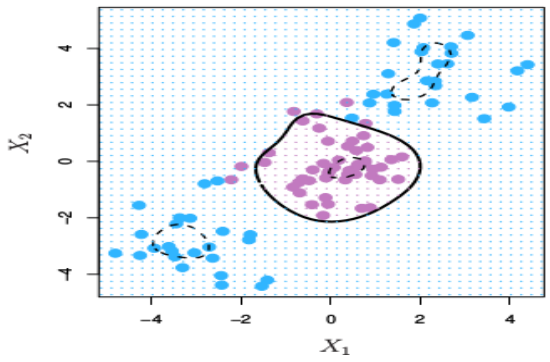


Figura 5-19. SVM con un kernel gaussiano radial. Fuente: [137].

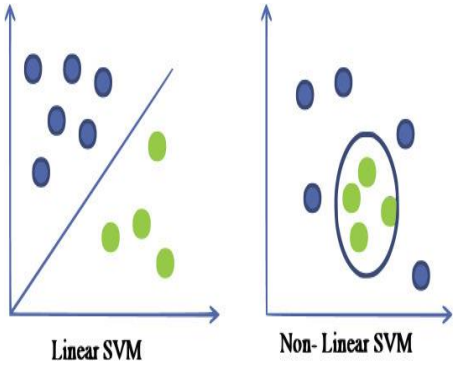


Figura 5-18. Existe lineal y no-lineal support vector machine. Fuente: [83].

Por otra parte, al igual que el clasificador Regularized Fisher LDA, un SVM utilizaría la regularización para evitar que el clasificador se adapte a conjuntos de datos posiblemente ruidosos, por ejemplo, usando variables de holgura [118].

El SVM ha sido ampliamente utilizada en BCI, porque es un clasificador simple que se desempeña bien y es robusto con respecto a los problemas de la dimensionalidad, lo que significa que no se requiere un gran conjunto de entrenamiento para conseguir buenos resultados, incluso con vectores de características de muy alta dimensión. Estas ventajas se producen a costa de la velocidad de ejecución. Sin embargo, el SVM es lo suficientemente rápido para las BCI en tiempo real [118].

También existen los Multiclass Support Vector Machine Models. Los cálculos subyacentes para la clasificación con SVM son para dos clases, pero se puede realizar una clasificación SVM multiclase creando un clasificador de

[44] También se le puede llamar función de base radial (RBF):

Códigos de Salida con Corrección de Errores (ECOC) [78], como se aprecia en Figura 5-20. Este método consigue convertir un problema de clasificación de varias clases en uno de clasificación binaria múltiple, *"lo que permite utilizar directamente modelos de clasificación binaria nativos"* [144].

"*SVM ha resultado ser uno de los mejores clasificadores para un amplio abanico de situaciones, por lo que se considera uno de los referentes dentro del ámbito de aprendizaje estadístico y machine learning*" [137].

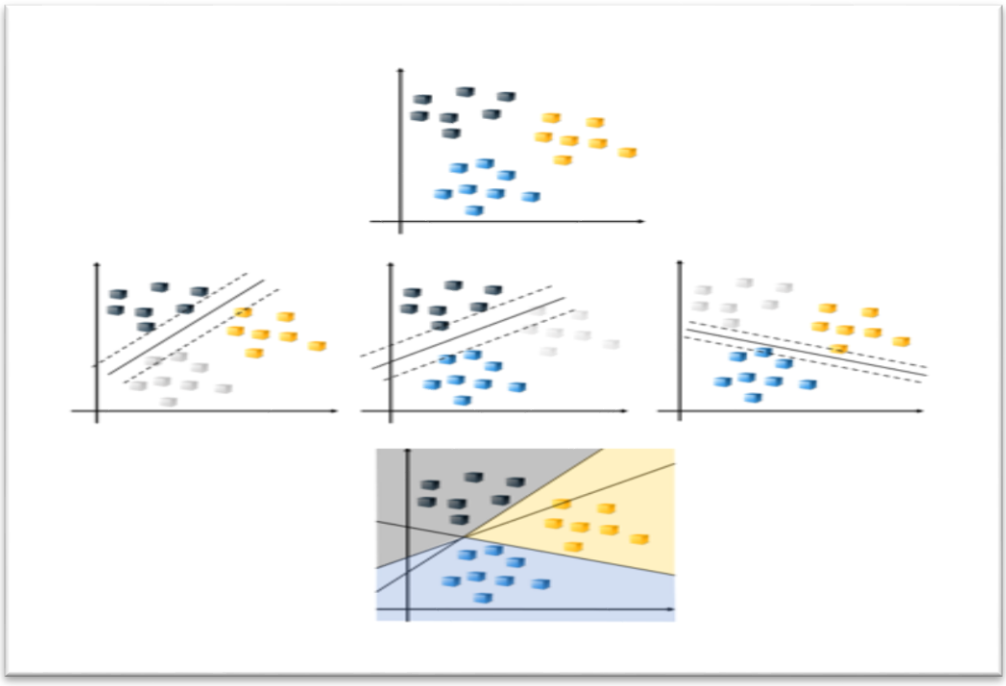


Figura 5-20. Ejemplo Multiclass Support Vector Machine Models. Fuente: [78].

Si se quiere profundizar más en los conceptos y cálculos de los SVM, es de información relevante el libro Support Vector Machines Succinctly de Alexandre Kowalczyk. También se puede consultar el apartado 17.5 del libro [89] y ver [145]- [146]- [147].

5.2.1 Resumen del clasificador SVM

El SVM tiene como objetivo hallar un hiperplano que permita separar las dos clases de forma que maximice el margen, la distancia entre hiperplano y muestras entrenamiento más cercanas de ambas clases. En casos perfectamente separables habrá varios hiperplanos posibles, habrá que escoger el óptimo, el que da mayor margen. Lo de perfectamente separable no es realista, así que se permitirá cometer unos pocos errores, algunas observaciones no estarán en el lado correcto del margen o el hiperplano. Además, el algoritmo se puede adaptar para múltiples clases.

En estos clasificadores se utiliza el método del Kernel, consiste en proyectar un conjunto de datos no separables linealmente en un espacio de dimensión mayor mediante una transformación no lineal, en donde probablemente sí se puedan separar linealmente.

La nueva muestra se clasificará según el lado en que caiga del hiperplano, usando el signo de la función de decisión.

5.3 Naïve Bayes

Un clasificador bayesiano consiste en clasificar según la probabilidad de que el patrón pertenezca a una clase u otra. "*Se basa en la aplicación del teorema de Bayes para predecir la probabilidad condicional de cada clase k, como el producto de la probabilidad a priori de la clase por la probabilidad condicional de las características (* $\mathbf{f}_\tau$ *) dada la clase* ***k****, dividido por la probabilidad de las características*" [140]:

$$P(C_k \mid \mathbf{f}_\tau) = \frac{P(C_k)P(\mathbf{f}_\tau \mid C_k)}{P(\mathbf{f}_\tau)} \qquad (5.21)$$

En otras palabras, "*el teorema de Bayes dice que la probabilidad condicional de un resultado se puede calcular usando la probabilidad condicional de la causa del resultado*" [148]. Se puede expresar como $\text{posterior} = \frac{\text{anterior } x \text{ probabilidad}}{\text{evidencia}}$, [149].

Para comprender este clasificador con claridad es necesario repasar algunos conceptos de la teoría de probabilidad. Por ello se recomienda leer el artículo: [150].

La probabilidad de un evento se denota como $p(\mathbf{f}_\tau)$, que es un número real en el intervalo [0,1]. La probabilidad condicional de $\mathbf{f}_\tau$ dado C se denota como $p(\mathbf{f}_\tau \mid C_k)$. La probabilidad de que ocurran dos eventos a la vez se denota como $p(\mathbf{f}_\tau \cap C_k)$, cumpliéndose que $p(\mathbf{f}_\tau \mid C_k) = \frac{p(C_k \cap \mathbf{f}_\tau)}{p(C_k)}$, de aquí se puede deducir el teorema de Bayes. Se puede entender que la probabilidad de que ocurra $\mathbf{f}_\tau$ y C_k, es la probabilidad de que ocurra $\mathbf{f}_\tau$ supuesto que ocurre C_k junto con la probabilidad de que ocurra C_k, $p(\mathbf{f}_\tau \cap C_k) = p(\mathbf{f}_\tau \mid C_k)\,p(C_k)$. De manera complementaria yo puedo decir que $p(C_k \cap \mathbf{f}_\tau) = p(C_k \mid \mathbf{f}_\tau)\,p(\mathbf{f}_\tau)$. Teniendo en cuenta que $p(\mathbf{f}_\tau \cap C_k) = p(C_k \cap \mathbf{f}_\tau)$, quedando verificado el teorema (5.21), solo habría que igualar y despejar lo que se quiera.

Suponiendo que se tienen dos clases, el teorema de Bayes se puede expresar de la siguiente forma:

$$P(C_k \mid \mathbf{f}_\tau) = \frac{P(C_k)P(\mathbf{f}_\tau \mid C_k)}{P(\mathbf{f}_\tau)} = \frac{P(C_k)P(\mathbf{f}_\tau \mid C_k)}{P(\mathbf{f}_\tau \mid C_1)P(C_1) + P(\mathbf{f}_\tau \mid C_2)P(C_2)} = \frac{P(\mathbf{f}_\tau \mid C_k)}{P(\mathbf{f}_\tau \mid C_1) + P(\mathbf{f}_\tau \mid C_2)} \qquad (5.22)$$

El modelo de probabilidad para un clasificador es $P(C_k \mid \mathbf{f}_\tau)$, en que la clase está condicionada por las características. "*El problema es que si el número de variables independientes (* $\mathbf{f}_\tau$ *) es grande o que éstas puedan tomar muchos valores, entonces basar este modelo en tablas de probabilidad se vuelve imposible*" [149]. Por lo que se reformula el teorema de Bayes asumiendo que las características son independientes entre sí, haciendo que el modelo se vuelva más manejable. El término Naive proviene de este hecho, de que este algoritmo asume una fuerte independencia entre características [149].

Otro hecho a recalcar, es que se presupone sobre la distribución estadística que siguen las clases. Habría que conseguir información o hacer hipótesis de cuáles son las funciones de densidad de cada clase. En resumidas

cuentas, se presupone que exhiben una distribución estadística gaussiana las características de cada clase (Figura 5-25). También se puede suponer que la distribución es una mezcla ponderada de distribuciones gaussianas [118].

"*Una ventaja del clasificador de Naive Bayes es que solo se requiere una pequeña cantidad de datos de entrenamiento para estimar los parámetros*[45] *necesarios para la clasificación. Como las variables independientes se asumen, solo es necesario determinar las varianzas de las variables de cada clase y no toda la matriz de covarianza"* [149].

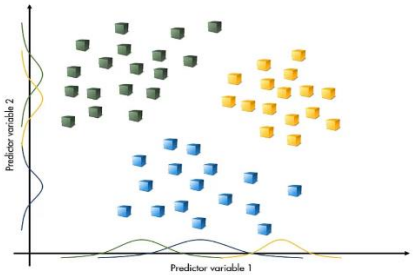


Figura 5-21. Los modelos KNN y los árboles de decisión no hacen suposiciones sobre la distribución de los datos subyacentes. Un clasificador Naive Bayes asume la independencia de los predictores dentro de cada clase. Este clasificador es una buena opción para problemas relativamente sencillos. Fuente: [78].

Se realizará las siguientes definiciones:

- $p(C_k)$: Probabilidad a priori de que se dé la clase. Por ejemplo, si se tiene un saco con objetos, la probabilidad que tiene que salga una clase sin ninguna condición previa.

- $p(\mathbf{f}_\tau)$: Probabilidad total de $\mathbf{f}_\tau$. Probabilidad de que me salga con unas características determinadas, sin importar la clase que sea.

- $p(\mathbf{f}_\tau \mid C_k)$: Probabilidad condicionada de $\mathbf{f}_\tau$. Probabilidad de que, supuesto de que tomamos un elemento de C_k, tenga el patrón $\mathbf{f}_\tau$.Por ejemplo, se tiene dos sacos, siendo el primero de la clase 1 y el segundo de la clase 2. Si saco un objeto del saco de clase 1, qué probabilidad habrá que sea un cierto patrón de características.

- $p(C_k \mid \mathbf{f}_\tau)$: Probabilidad a posteriori de la clase C_k. Probabilidad de que, dado un patrón $\mathbf{f}_\tau$, éste pertenezca a C_k. Por ejemplo, la probabilidad de que se haya sacado del saco un objeto de clase 1 si se ha obtenido un cierto patrón. Esta es la probabilidad que serviría para clasificar esencialmente, como se verá más adelante.

Los patrones de estudio son elementos de una población y se les asigna la clase que más probabilidad tenga de haberla producido, usando la función de máxima verosimilitud:

$$k_{asignada} = \arg\max_i \; p(C_k \mid \mathbf{f}_\tau) \qquad (5.23)$$

Donde un objeto (patrón) $\mathbf{f}_\tau$ se le asigna la clase $k_{asignada}$, la que ha dado probabilidad más alta.

Para asignar $\mathbf{f}_\tau$ a la clase C_k, debe cumplirse:

$$p(C_k \mid \mathbf{f}_\tau) > p(C_i \mid \mathbf{f}_\tau) \qquad \forall i = 1...N_c \quad (5.24)$$

[45] Las medias y las varianzas de las variables

Para calcular $p(C_k \mid \mathbf{f}_\tau)$ se usa el teorema de Bayes, (5.21). Este término indica la probabilidad de que la siguiente clase sea C_k condicionada por el hecho de que se haya recibido una determinada observación.

Las probabilidades de cada clase, $p(C_k)$, se consideran conocidas. A priori se pueden hallar asumiendo clases equiprobables[46], o si no lo son, se usa una estimación de la probabilidad de clase del conjunto de entrenamiento[47] [149]. En los BCI, la mayoría de las veces se consideran iguales, ya que se supone que el usuario no tiene predilección por ningún movimiento.

La probabilidad total de $\mathbf{f}_\tau$, $p(\mathbf{f}_\tau)$, se obtiene sabiendo que se debe cumplir:

$$p(\mathbf{f}_\tau) = \sum_{j=1}^{N_c} p(\mathbf{f}_\tau \mid C_j)\, p(C_j) \qquad (5.25)$$

Las probabilidades de aparición de $\mathbf{f}_\tau$ en cada clase, $p(\mathbf{f}_\tau \mid C_k)$, se pueden estimar a partir de observaciones de la población o suponiendo la distribución que sigue, como se comentó anteriormente.

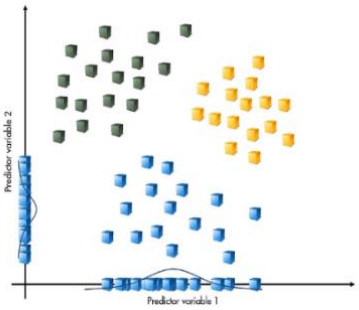


Figura 5-23. Se supone que sigue una distribución normal cada variable. Fuente: [146].

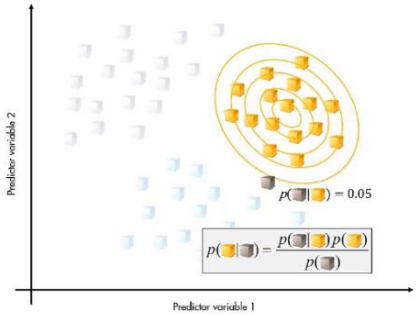


Figura 5-22. Ejemplo probabilidad de patrón condicionada a la clase. Fuente: [146].

El método estándar es presuponer que sigue una distribución Gaussiana [151], de la cual se tendrán que estimar sus parámetros[48]. Se va a explicar esto de forma más detallada:

"Para la estimación de los parámetros de la distribución de una característica, se debe asumir una distribución o generar modelos de estadística no paramétrica de las características del conjunto de entrenamiento. Las hipótesis sobre las distribuciones de características son llamadas el modelo de eventos del Clasificador Naive Bayes. La distribución multinomial y la distribución de Bernoulli son populares para características discretas. Cuando se trata con los datos continuos, una hipótesis típica es que los valores continuos asociados con cada clase se distribuyen según una ***Distribución normal***" [149]. En este caso se está presuponiendo que siguen distribuciones normales los patrones de cada clase.

- Si fuese un caso **unidimensional** (Figura 5-24), la función de densidad de distribución normal sería:

$$P(\mathbf{f}_\tau \mid C_k) = N(\mu_k, \sigma_k^2) = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma_k} e^{\frac{-(\mathbf{f}_\tau - \mu_k)^2}{2\sigma_k^2}} \qquad (5.26)$$

[46] 1/ (número de clases)
[47] *"(para una clase dada) = (número de muestras en la clase) / (número total de muestras)"* [149].
[48] la media y la varianza

Siendo $\mu_k = E[\mathbf{f}_\tau]$; $\sigma_k^2 = E[(\mathbf{f}_\tau - \mu_k)^2]$; $\forall \mathbf{f}_\tau \in C_k$. Donde μ_k es la media de $\mathbf{f}_\tau$ asociado a la clase C_k, y σ_k^2 es la varianza de $\mathbf{f}_\tau$ asociado a la clase C_k.

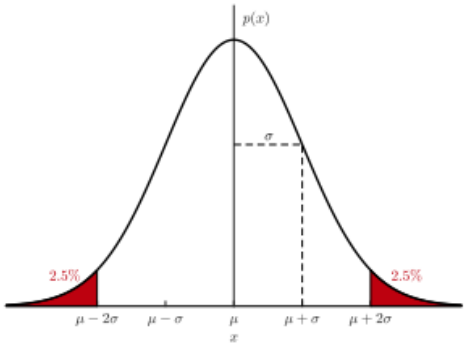


Figura 5-24. La densidad normal univariante está definida por dos parámetros: media μ y varianza σ^2. Tiene aproximadamente el 95 % de su área en el rango $|x - \mu| \leq 2\sigma$. El pico de la distribución tiene un valor $p(x)=1/\sqrt{2\pi}\sigma$. Fuente: [152].

"El teorema del límite central establece que el efecto agregado de la suma de un gran número de pequeños disturbios aleatorios independientes derivará en una distribución Gaussiana" [152].

Un ejemplo de clasificación con dos clases y una característica sería la Figura 5-25. La clase C_A viene definida con una función de densidad Gaussiana de probabilidad con una media μ_1 con una cierta dispersión. Lo mismo ocurre con clase C_B con media μ_2. Estas Gaussianas son $P(x|Clase)$. Caracterizan cada una de las clases de manera independiente. El eje horizontal de la gráfica es la característica x y la probabilidad concreta de un cierto patrón sería $p(x) = p(x|C_A)p(C_A) + p(x|C_B)p(C_B)$. Los valores $p(x|C_A)$ y $p(x|C_B)$ se podrían averiguar viendo donde corta en las funciones de densidad al trazar una línea vertical donde correspondería a la x que se quisiese. Si $p(C_A|x) > p(C_B|x) \rightarrow x \in C_A$.

De todas formas, la probabilidad $p(x)$ desaparecerá de la ecuación como se verá más adelante, no habrá que preocuparse por ella. Solo interesa $p(x|C)$ y $p(C)$.

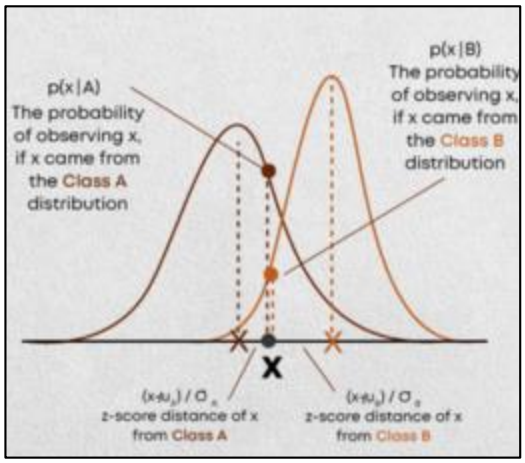


Figura 5-25. Ejemplo clasificador Naive Bayes con dos clases, A y B, y una característica, x. Fuente: [153].

- Si fuese un caso **multidimensional** (Figura 5-26), la distribución normal sería:

$$P(\mathbf{f}_\tau \mid C_k) = N(\bar{\mu}_k, \mathbf{V}_k) = \frac{1}{(2\pi)^{\frac{1}{2}} |\mathbf{V}_k|^{\frac{1}{2}}} e^{\frac{-1}{2}\left[(\bar{\mathbf{f}}_\tau - \bar{\mu}_k)^T \mathbf{V}_k^{-1} (\bar{\mathbf{f}}_\tau - \bar{\mu}_k)\right]} \quad (5.27)$$

Siendo $\bar{\mu}_k = E[\bar{\mathbf{f}}_\tau]$; $\mathbf{V}_k = E\left[(\bar{\mathbf{f}}_\tau - \bar{\mu}_k)^T (\bar{\mathbf{f}}_\tau - \bar{\mu}_k)\right]$; $\forall \mathbf{f}_\tau \in C_k$. La $\mathbf{V}_k$ es la matriz de covarianza de las características de la clase k, es decir, es la covarianza $\hat{\Sigma}_{f|c_x}$ que se explicó en LDA.

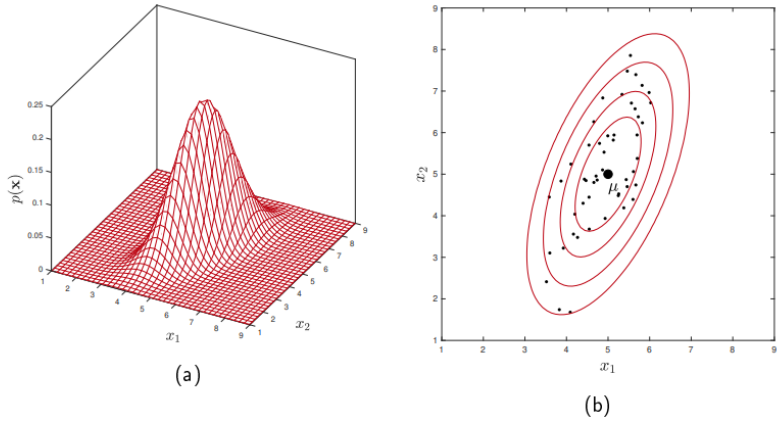


Figura 5-26. (a) Ejemplo Función de densidad de probabilidad Gaussiana centrada en μ, cuya forma está determinada por **V**. (b) Muestras tomadas aleatoriamente de la distribución Gaussiana en (a), caen en una nube centrada en μ y dispersas de acuerdo con **V**. Fuente: [152].

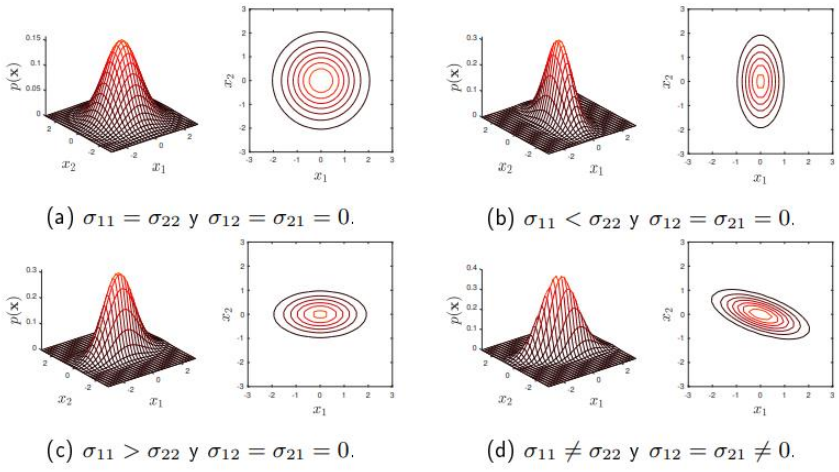


Figura 5-27. Ejemplo de efecto de la matriz de covarianza sobre una distribución de probabilidad en R^2. Fuente: [152].

En este trabajo se usa el caso multidimensional, ya que se usa un vector de más de una característica. Las próximas ecuaciones tendrán contemplado este hecho.

Continuando con la explicación principal, de (5.21) y (5.24) se deduce sustituyendo la expresión (5.28), anulándose $p(\bar{\mathbf{f}}_\tau)$ en ambos lados de la inecuación.

$$p(\bar{\mathbf{f}}_\tau \mid C_k)\, p(C_k) > p(\bar{\mathbf{f}}_\tau \mid C_i)\, p(C_i) \quad (5.28)$$

Se podría definir como función de decisión:

$$\hat{d}_k(\bar{\mathbf{f}}_\tau) = p(\bar{\mathbf{f}}_\tau \mid C_k)\, p(C_k) \qquad (5.29)$$

La norma de decisión (5.28) separa el espacio de características en regiones de decisión, separadas por fronteras de decisión que pueden ser no lineales. Para simplificar el análisis matemático, es más conveniente maximizar el logaritmo de la función de verosimilitud [152]. En Figura 5-28 se ve un ejemplo de esto, se observa que la frontera de decisión no varía.

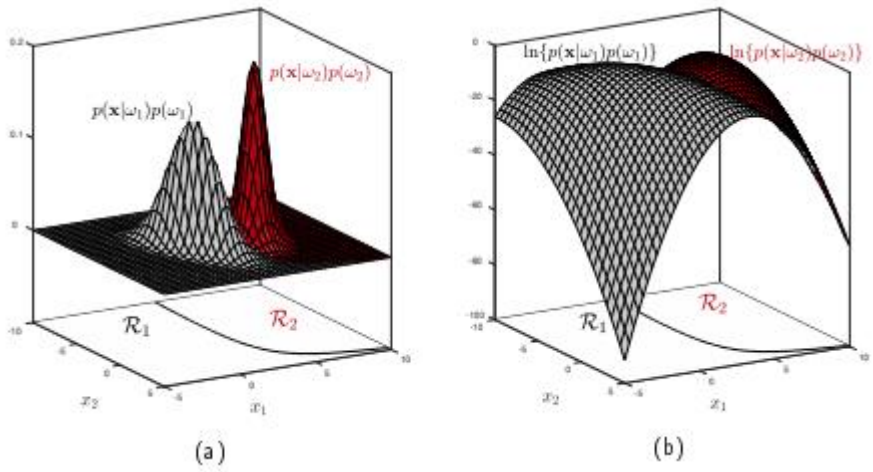


Figura 5-28. (a) Ejemplo Regiones de decisión sin aplicar logaritmo. (b) Las regiones de decisión no varían al aplicar logaritmo. Fuente: [152].

Se aplica logaritmo neperiano a ambos lados de la inecuación (5.28) , quedando la función discriminante bayesiana de siguiente modo:

$$\log p(\bar{\mathbf{f}}_\tau \mid C_k) + \log p(C_k) > \log p(\bar{\mathbf{f}}_\tau \mid C_i) + \log p(C_i) \qquad (5.30)$$

Con esta expresión de partida, que es la que necesitamos para comparar, se sustituye $p(\bar{\mathbf{f}}_\tau \mid C_k)$ por la función de densidad, asumiendo que sigue una normal. Darse cuenta de que al haber aplicado logaritmo se simplifica las expresiones, se puede quitar elementos exponenciales usando las propiedades de los logaritmos.

La función de decisión se puede conseguir con el siguiente procedimiento:

- En caso **unidimensional**:

$$P(\bar{\mathbf{f}}_\tau \mid C_k) = N(\mu_k, \sigma_k^2) = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma_k} e^{\frac{-(\bar{\mathbf{f}}_\tau - \mu_k)^2}{2\sigma_k^2}} = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma_k} e^{\frac{-1}{2}r_k^2} \qquad (5.31)$$

Se denomina r_k^2 a la distancia de Mahalanobis. En caso unidimensional $r_k^2 = r_k^2(\mathbf{f}_\tau) = \frac{(\mathbf{f}_\tau - \mu_k)^2}{\sigma_k^2}$. Se puede ver un ejemplo de esta distancia en caso unidimensional en la Figura 5-25, la distancia de cada media a la característica teniendo en cuenta dispersión de cada clase.

El desarrollo de $\hat{d}_k$ sería:

$$\log p(\bar{\mathbf{f}}_\tau \mid C_k) + \log p(C_k) > \log p(\bar{\mathbf{f}}_\tau \mid C_i) + \log p(C_i)$$

$$\left(\cancel{-\log\sqrt{2\pi}} - \log\sqrt{\sigma_k^2} - \frac{1}{2} r_k^2 \right) + \log p(C_k) > \left(\cancel{-\log\sqrt{2\pi}} - \log\sqrt{\sigma_i^2} - \frac{1}{2} r_i^2 \right) + \log p(C_i)$$

$$-\frac{1}{2} r_k^2 + \underbrace{\log p(C_k) - \frac{1}{2}\log\sigma_k^2}_{f_k} > -\frac{1}{2} r_i^2 + \underbrace{\log p(C_i) - \frac{1}{2}\log\sigma_i^2}_{f_i}$$

$$-\frac{1}{2} r_k^2 + f_k > -\frac{1}{2} r_i^2 + f_i$$

La función de decisión:

$$\hat{d}_k(\bar{\mathbf{f}}_\tau) = -\frac{1}{2} r_k^2 + f_k \qquad (5.32)$$

En la función se tienen dos términos, r_k que depende del patrón, y f_k que no depende del patrón sino de la clase [154].

- En caso **multidimensional**:

$$P(\bar{\mathbf{f}}_\tau \mid C_k) = N(\bar{\mu}_k, \mathbf{V}_k) = \frac{1}{(2\pi)^{\frac{1}{2}} |\mathbf{V}_k|^{\frac{1}{2}}} e^{\frac{-1}{2}\left[(\bar{\mathbf{f}}_\tau - \bar{\mu}_k)^T \mathbf{V}_k^{-1} (\bar{\mathbf{f}}_\tau - \bar{\mu}_k)\right]} = \frac{1}{(2\pi)^{\frac{1}{2}} |\mathbf{V}_k|^{\frac{1}{2}}} e^{\frac{-1}{2} r_k^2} \qquad (5.33)$$

Siendo $r_k^2 = (\mathbf{f}_\tau - \bar{\mu}_k)^T \mathbf{V}_k^{-1} (\mathbf{f}_\tau - \bar{\mu}_k)$, la distancia de Mahalanobis, la distancia desde un patrón a la media de la clase que se esté estudiando, la cual escala la distancia de acuerdo con la probabilidad. En lugar de usar distancia euclídea, se usa una distancia que es ponderada con la dispersión de la clase, como ya se comentó. "*Tiene en cuenta la correlación entre las variables aleatorias*" [155]. Un ejemplo se ve en Figura 5-29.

El procedimiento es análogo al unidimensional. Resultando como función de decisión (no lineal, cuadrática):

$$\hat{d}_k(\bar{\mathbf{f}}_\tau) = -\frac{1}{2} r_k^2 + f_k \qquad (5.34)$$

Siendo $f_k = \log p(C_k) - \frac{1}{2}\log|\mathbf{V}_k|$, dependiendo solo de la clase. Al ser funciones de decisión cuadráticas, tendrán gran capacidad de discriminación [154]. Si se diese el caso de que las matrices de covarianza fuesen iguales, $\mathbf{V}_k = \mathbf{V}_i = \mathbf{V}$, puede obtenerse una función de decisión lineal:

$$\hat{d}_k(\bar{\mathbf{f}}_\tau) = \bar{\mathbf{f}}_\tau \mathbf{V}^{-1} \bar{\mu}_k + \left[\log p(C_k) - \frac{1}{2} \bar{\mu}_k{}^T \mathbf{V}^{-1} \bar{\mu}_k \right] \rightarrow \hat{d}_k(\bar{\mathbf{f}}_\tau) = \tilde{w}^T \tilde{\bar{\mathbf{f}}}_\tau \qquad \textit{siendo} \quad \tilde{w} = \begin{bmatrix} \mathbf{V}^{-1}\bar{\mu}_k \\ \log p(C_k) - \frac{1}{2}\bar{\mu}_k{}^T \mathbf{V}^{-1} \bar{\mu}_k \end{bmatrix}$$

(5.35)

Siendo $\tilde{\bar{\mathbf{f}}}_\tau$ el vector de características ampliado.

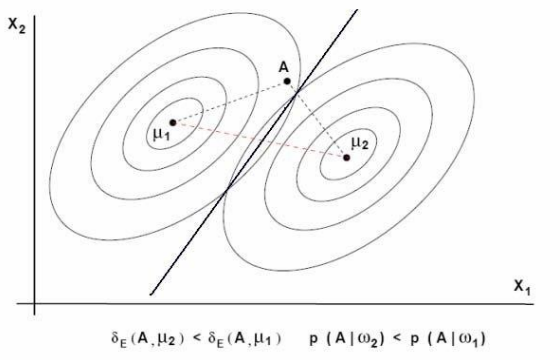


Figura 5-29. Ejemplo de distancia de Mahalanobis en Clasificación multidimensional. La clasificación depende de la distancia a las medias. Fuente: [156].

En resumen, la etapa de entrenamiento del clasificador consiste en hallar los parámetros de las funciones de densidad de probabilidad (funciones discriminantes Gaussianas), es decir, hallar la media $\overline{\mu}_k$, la matriz de covarianzas $\mathbf{V}_k$ y la probabilidad $p(C_i)$.

"*Naive Bayes funciona bien en muchos dominios, pero en ocasiones el rendimiento decrece debido a que los atributos no son condicionalmente independientes como inicialmente se asume*" [140]. Los clasificadores estadísticos bayesianos no se suelen usar en BCI. Sin embargo, hay un ejemplo para clasificar imaginación motora en el artículo: [157]. Para saber más sobre este clasificador se puede consultar: [158]- [159]- [89].

5.3.1 Resumen del clasificador Naïve Bayes

El clasificador Naive Bayes se inspira en el Teorema de Bayes. El término Naive proviene de que este algoritmo asume una fuerte independencia entre características. Además, se presupone que exhiben una distribución estadística gaussiana o una mezcla ponderada de distribuciones gaussianas las características de cada clase. A un patrón se le asigna la clase a la que tenga más probabilidad de corresponder, usando la función de máxima verosimilitud, $k_{asignada} = \arg\max_i \; p(C_k \mid \mathbf{f}_\tau)$.

La $p(\mathbf{f}_\tau \mid C_k)$ se estima suponiendo que sigue una distribución normal, teniendo en cuenta que si es un caso multidimensional se usarán matrices de covarianza. La $p(C_k)$ se considera conocida. La $p(\mathbf{f}_\tau)$ se puede calcular a partir de las dos probabilidades ya mencionadas.

El espacio de características se dividirá en fronteras de decisión no lineales. Para simplificar el cálculo se usa el logaritmo de la función de verosimilitud. Sustituyendo y sintetizando términos, se obtiene la función de decisión finalmente; la cual depende de la distancia de Mahalanobis desde el patrón estudiado hasta la media de la clase correspondiente.

El patrón nuevo se asigna a la clase que dé un valor mayor en la función de decisión, $\hat{d}_k(\overline{\mathbf{f}}_\tau) = -\frac{1}{2} r_k^2 + f_k$.

5.4 K-nearest neighbor (KNN)

Es una clasificación supervisada que asigna una clase a un patrón basándose en los patrones conocidos que lo rodean. "*Una de las formas más sencillas de clasificar una nueva muestra es encontrar muestras conocidas que sean similares a la nueva muestra, y asignar la nueva muestra a la misma clase. Esta es la idea básica que subyace a la clasificación por KNN (k- vecinos más cercanos)"* [78]. "*Puede usarse para clasificar nuevas muestras (valores discretos) o para predecir (regresión, valores continuos)*" [160]. "*La idea es memorizar todo el conjunto de datos y clasificar un punto en función de la clase de sus* k_{means} *vecinos más cercanos*" [148]. La k_{means} es la cantidad de patrones cercanos que se tienen en cuenta para clasificar.

Se basa en que las características diferentes corresponderán a diferentes clases, formándose "*grupos separados en el espacio de características*" [80]. Los vecinos cercanos son los patrones[49] que pertenecen a la misma clase.

Como se mencionó anteriormente, las características observadas de todos los elementos de una misma clase deben estar cerca en el espacio de características y a la vez estar lo más lejos posible de los de las otras clases. Es importante tener esto en cuenta al usar el clasificador.

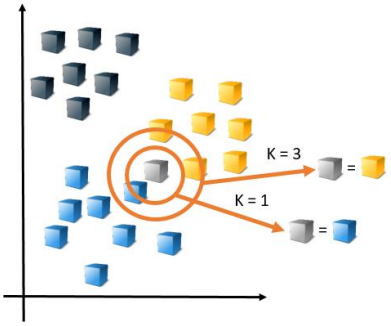


Figura 5-30.Una forma fácil de clasificar una observación es usar la misma clase que los ejemplos conocidos más cercanos. La clase que se asigne depende de cuántos patrones etiquetados se tengan en cuenta. Fuente: [78].

Comienza fijando un número k_{means} de patrones, pues se tomarán los k_{means} patrones más cercanos al que se quiere clasificar. Se calcula la distancia entre el patrón a clasificar ($\mathbf{f}_j$) y los patrones de entrenamiento ($\mathbf{f}_s$), es decir, se calcula la distancia euclidiana d_E en el espacio de características:

$$d_E(\mathbf{f}_s,\mathbf{f}_j) = \left\|\mathbf{f}_j - \mathbf{f}_s\right\| \qquad (5.36)$$

Se selecciona los k_{means} patrones con menor distancia, estos son los **'vecinos cercanos'**. Luego se le asigna al patrón $\mathbf{f}_j$ la clase más repetida entre los vecinos cercanos[50], siendo una votación de mayoría.

La distancia euclidiana no es la única posible a usar, también están las distancias Manhattan, Minkowski, Chebychev o Hamming.

Para decidir la clase de un elemento influye mucho el valor de k_{means}, sobre todo en las fronteras entre clases [160]. Tiene que ser un número entero impar, pues no hay un comportamiento definido en caso de haber un empate en la votación. Por otra parte, tomar un k_{means} muy grande no implica que mejore la precisión, si es muy grande considerará muestras más alejadas, que pueden no ser significativas. Lo que sí se puede afirmar es que cuanto más grande sea, más lento será la realización del algoritmo [160]- [38]. Por otro lado, si es muy pequeña puede afectar al rendimiento, ya que en algunas zonas puede ser una votación casi aleatoria.

[49] Conjunto de características
[50] Los patrones que son vecinos cercanos están etiquetados con su clase.

La mejor elección de k_{means} depende de cada caso [161]. El método para hallarla sería probando valores, hasta que se encuentre una que dé unos resultados de clasificación aceptables. Cuando se usa k_{means}=1, se le llama **Nearest Neighbor Algorithm**. *"La investigación ha demostrado que no existe un número óptimo de vecinos que se adapte a todo tipo de conjuntos de datos. Cada conjunto de datos tiene sus propios requisitos. Se ha demostrado que una pequeña cantidad de vecinos son los más flexibles, que tendrán un bajo sesgo[51], pero una alta varianza, y un gran número de vecinos tendrán un límite de decisión más suave, lo que significa una varianza más baja pero un sesgo más alto"* [162]. Se recomienda leer el capítulo 14 del libro [89].

Un ejemplo de su utilización se puede apreciar en el artículo [163], donde se usa un KNN, con valores de k_{means} desde 1 hasta 15, para un sistema BCI.

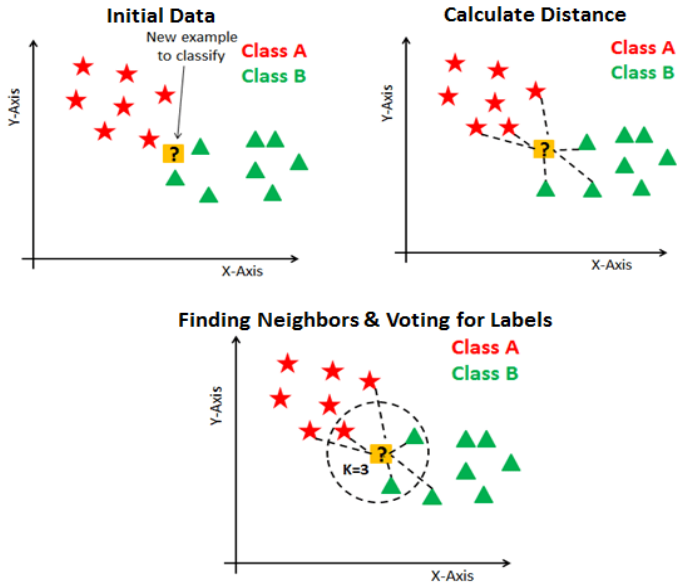


Figura 5-31. Ejemplo de proceso de clasificación KNN. Fuente: [148].

Una ventaja es su sencillez de implementación, pero "*requiere de uso de mucha memoria y recursos de procesamiento*" [160], además de ser más lento que otros clasificadores. El KNN funciona mejor con conjunto de datos pequeños [160]. La exactitud puede verse empeorada por "*la presencia de ruido o características irrelevantes, o si las escalas de características no son consistentes con lo que uno considera importante*" [161] . Eso no quiere decir que no se pueda usar para datos con ruido. Por ejemplo, el KNN es adecuado para clasificar los datos de EEG, ya que es una técnica robusta para grandes datos ruidosos [163].

En este clasificador no se actualiza ningún parámetro durante el entrenamiento, éste solo consistiría en ir almacenando las muestras. Memoriza los datos de entrenamiento, no aprende una función discriminatoria.

Existen variantes al algoritmo que tratan de ponderar la relevancia de cada vecino según la distancia en que se encuentre éste con respecto a la muestra a clasificar. Por ejemplo, "*ponderar el voto de cada vecino de acuerdo con el cuadrado inverso de la distancia*" [161] .

[51] *"En estadística se llama sesgo de un estimador a la diferencia entre su esperanza matemática y el valor numérico del parámetro que estima"* [242].

5.4.1 Resumen del clasificador KNN

En el espacio de características hay patrones con sus clases etiquetadas. Para asignarle una clase a un nuevo patrón, se busca el número (k_{means}) de patrones más cercanos que se le indique. Para ello se calcula la distancia a cada uno de ellos (Euclidiana, Manhattan, Minkowski, Chebychev o Hamming). Se le predice la clase basándose en la clase que más se repite de los datos que lo rodean, se hace una votación de mayoría de los vecinos. También existen otras variantes del algoritmo, como ponderar la contribución según la distancia de cada vecino.

Se memoriza los datos de entrenamiento, no aprende una función discriminatoria.

5.5 K-Means

Como curiosidad, hay que decir que existe el clasificador **K-Means** basado en el mismo concepto que el KNN, pero es un algoritmo no supervisado, trabaja con conjuntos no etiquetados [164]. "*Se llama k-Means porque comienza fijando un número k de agrupaciones y trata de clasificar el conjunto de entrenamiento de forma que se minimice la distancia cuadrada de cada punto a su centro de agrupación. Para obtener este mínimo, se utiliza el* ***algoritmo de Lloyd****. Este algoritmo consiste en inicializar los centroides con algunos puntos arbitrarios (normalmente dentro del conjunto), e itera sobre los dos pasos siguientes hasta la convergencia. A cada punto se le asigna el centroide más cercano"* [33]. Si se quiere comprender mejor se puede consultar [115]. Este clasificador no se llegó a implementar en MATLAB porque los no supervisados tienden a ser menos precisos, se puede consultar los resultados que daría en un BCI en el documento [38].

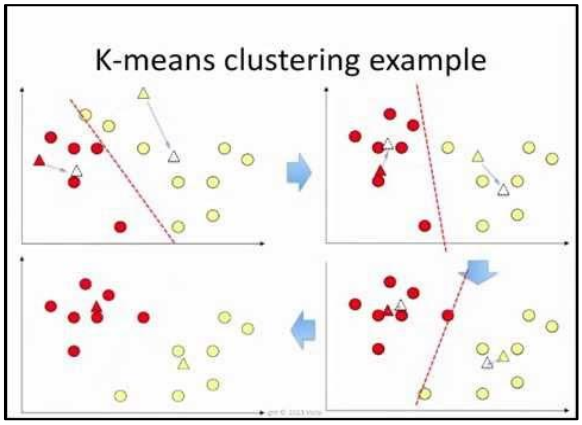


Figura 5-32. Ejemplo K-means (es no supervisado). Fuente: [115].

5.6 Decision Trees

"Los *árboles de decisión*[52] *son representaciones gráficas de posibles soluciones a una decisión basadas en ciertas condiciones, es uno de los algoritmos de aprendizaje supervisado más utilizados en machine learning*" [165]. Pueden manejar tanto datos numéricos como categóricos. Al no ser paramétrico, no necesita que se dé una distribución específica de los datos. "*Si para alguna observación, el valor de una característica no está disponible, a pesar de no poder llegar a ningún nodo terminal, se puede conseguir una predicción empleando todas las*

[52] En inglés Decision Trees

observaciones que pertenecen al último nodo alcanzado. La precisión de la predicción se verá reducida, pero al menos podrá obtenerse" [166].

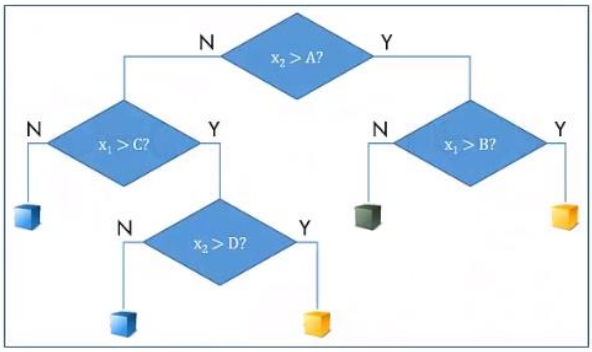


Figura 5-34.Ejemplo de árbol de decisión. Fuente: [146].

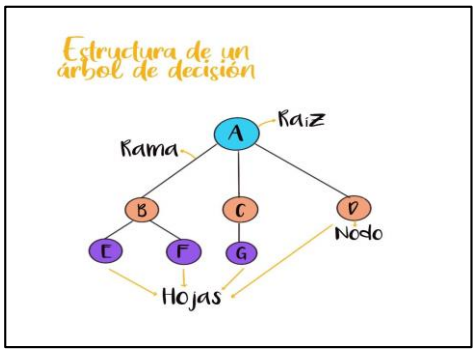


Figura 5-33. Estructura de un árbol de decisión. Fuente: [236].

"*En estas estructuras de árbol, las hojas representan etiquetas de clase y las ramas representan las conjunciones de características que conducen a esas etiquetas de clase. Los árboles de decisión, donde la variable de destino puede tomar valores continuos (por lo general números reales) se llaman árboles de regresión*" [167]. Cuando la variable respuesta es categórica se les llaman árboles de clasificación. En el caso del BCI sería un árbol de clasificación, siendo la clase 1 o 2.

Se resume en dos pasos el proceso de entrenamiento de un árbol de decisión: El primer paso sería la división continua del espacio de características generando regiones que no solapen (nodos terminales), $R_1, R_2, ..., R_j$. El método que se suele usar es el **recursive binary splitting** (división binaria recursiva). El segundo paso sería la "*predicción de la variable respuesta en cada región*" [166].

Los árboles de decisión binarios clasifican las observaciones creando una secuencia de cuestiones sí/no, como se ve en Figura 5-34, se divide en dos cada nodo y luego se vuelven a subdividir hasta que se llega a las hojas, los nodos finales [165]. Se explicará este proceso con más detalle.

Dado los datos de entrenamiento, se construye considerando todas las posibles divisiones en cada variable. Utilizando un criterio determinado sobre la calidad de una posible división, se elige la mejor división posible. A continuación, el proceso se repite en el siguiente nivel del árbol. Esto continúa hasta que todas las ramas terminan, esto ocurre cuando no hay más divisiones en esa rama que puedan mejorar el valor del criterio. El resultado final es un clasificador que divide el espacio del predictor en una colección de regiones rectangulares. Se puede ver un ejemplo de este procedimiento en la Figura 5-35. Realizado el proceso de entrenamiento, los patrones de entrenamiento quedan agrupados en los nodos terminales [166]- [146].

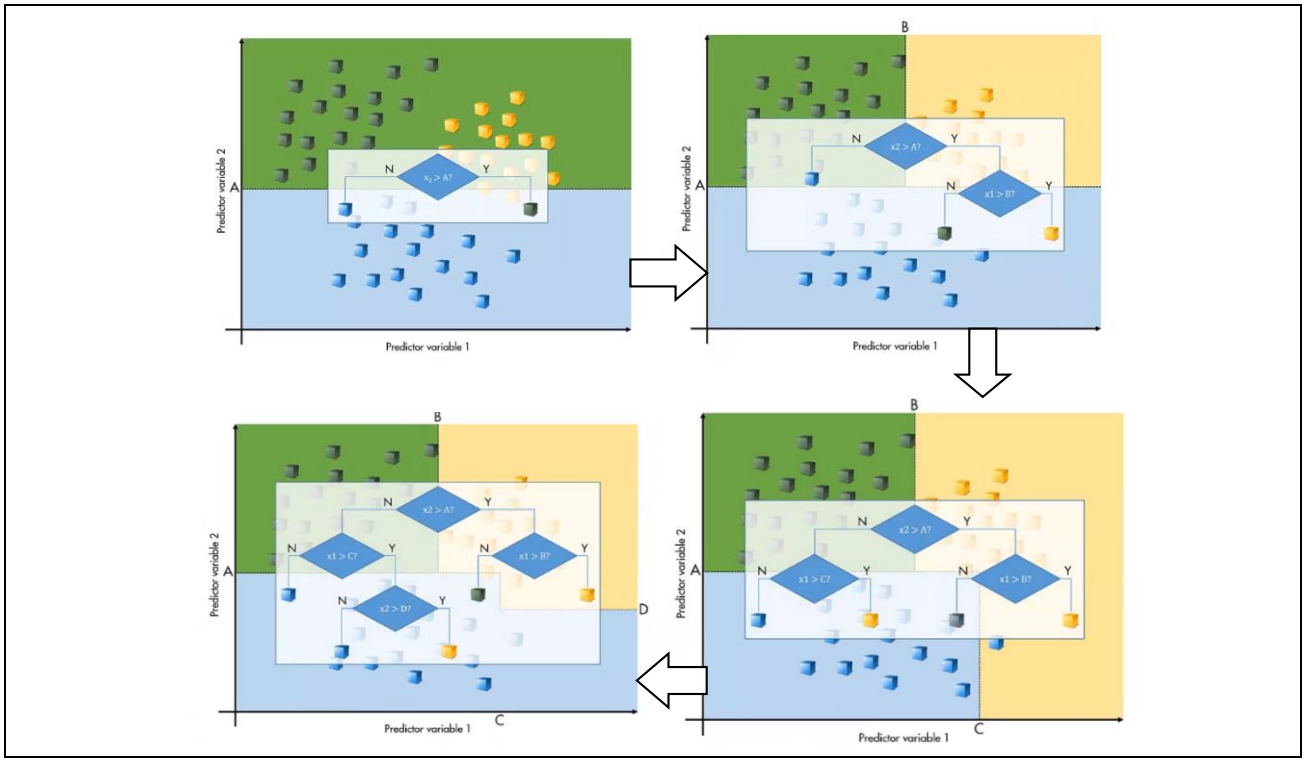


Figura 5-35. Proceso entrenamiento de un árbol de decisión. Fuente: [146].

Una vez que se ha entrenado un clasificador en forma de árbol, la realización de predicciones es rápida porque no requiere más que un conjunto de decisiones binarias. Para predecir un nuevo patrón, se recorre el árbol según los valores que tienen sus características hasta llegar a uno de los posibles nodos terminales [166]. En el caso de clasificación, suele usarse el criterio de "*moda de la variable respuesta como valor de predicción, es decir, la clase más frecuente del nodo*" [165].

Los datos realistas tendrán ruido. En teoría, un árbol suficientemente complejo podría ajustarse a un límite arbitrariamente complejo entre puntos. Sin embargo, esto suele significar que el modelo se ajusta en exceso a los datos, como se ve en Figura 5-36. Se produce un **overfitting**[53], reduciéndose la capacidad de predicción al usarlo con datos nuevos. "*Existen dos formas de prevenir el problema de overfitting de los árboles: limitar el tamaño del árbol (parada temprana o **early stopping**) y el proceso de podado (**pruning**)*" [165]. Se puede podar un árbol, es decir, reducir el número de divisiones para crear un modelo más simple, que puede tener una mayor pérdida de re-sustitución, pero una mejor generalización a los nuevos datos [146]. Un ejemplo de resultado sería la Figura 5-37.

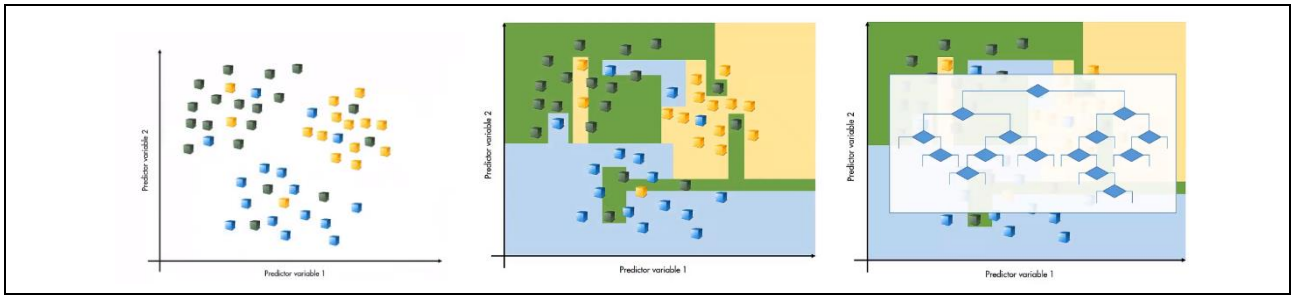


Figura 5-36. Overfitting de los datos, debido a ruido. Fuente: [146].

[53] Sobreajuste: "*Es el efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado*" [245].

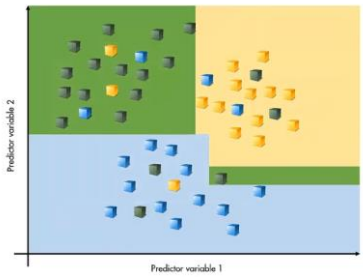


Figura 5-37. Resultado de podar un árbol, es decir, reducir el número de divisiones. Fuente: [146].

Con respecto al cálculo de las regiones, cuando se usan **árboles de regresión** se suele usar el criterio **Residual Sum of Squares** (RSS) para identificar las divisiones. "*La medida de selección de atributos es una heurística para seleccionar el criterio de división que divide los datos de la mejor manera posible*" [168] . El objetivo es hallar las J regiones que minimicen el RSS total, es decir, "*el sumatorio de las desviaciones al cuadrado entre las observaciones y la media de la región a la que pertenecen sea lo menor posible*" [169]:

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \qquad (5.37)$$

Donde $\hat{y}_{R_j}$ es la media de la variable respuesta en la región R_j y la y_i es el valor real de la variable a predecir.

El inconveniente es que es imposible analizar todas las posibles divisiones del espacio de características. Por esta razón, se utiliza el método recursive binary splitting: Consiste en "*encontrar en cada iteración el predictor* $\mathbf{f}_j$ *y el punto de corte (umbral)* s *tal que, si se distribuyen las observaciones en las regiones* $\{\mathbf{f}_\tau \mid \mathbf{f}_j < S\}$ *y* $\{\mathbf{f}_\tau \mid \mathbf{f}_j \geq S\}$*, se obtenga el menor posible RSS*" [169]. El procedimiento es:

- Primero se considera que todas las observaciones son de la misma clase [169].
- Se identifica los umbrales para cada una de las características, los predictores. "*Con predictores cualitativos, los posibles puntos de corte son cada uno de sus niveles. Para predictores continuos, se ordenan de menor a mayor sus valores, el punto intermedio entre cada par de valores se emplea como punto de corte*" [169].
- Se obtiene el RSS total, obteniéndose a partir de "*cada posible división identificada en el segundo paso*" [169].

$$RSS = \sum_{i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \quad (5.38)$$

Esta ecuación es RSS en caso de dos regiones, "*siendo cada una de las regiones el resultado de separar las observaciones acorde al predictor* j *y valor* s *"* [169].

- Se coge el punto de corte S y la característica $\mathbf{f}_j$ que dan mínima RSS total. Si hay más de una posible solución, la asignación será al azar [169].
- Se repiten una y otra vez todos los pasos anteriores para cada una de las regiones creadas en la repetición anterior hasta que se cumpla la condición de parada que se le establezca. Por ejemplo, "*que ninguna región contenga un mínimo de n observaciones, que el árbol tenga un máximo de nodos terminales o que la incorporación del nodo reduzca el error en al menos un porcentaje mínimo*" [169].

Darse cuenta de que, cada división estimada como solución, no toma en consideración si es la que permite obtener mejores árboles para divisiones futuras [169]. Además, "*en cada división se evalúa un único predictor haciendo preguntas binarias, hacer una pregunta sobre múltiples variables a la vez es equivalente a hacer múltiples preguntas sobre variables individuales*" [169].

Para los **árboles de clasificación** se usa también el procedimiento **recursive binary splitting**, pero no se usa RSS como juicio de escogimiento de las segmentaciones [169]. Las opciones que se suelen usar en su lugar son:

- **Classification Error Rate**:

"Se define como la proporción de observaciones que no pertenecen a la clase mayoritaria del nodo" [169].

$$E_j = 1 - \max_k(p_{jk}) \quad (5.39)$$

Donde p_{jk} es la proporción del nodo j que pertenece a la clase k.

- **Índice Gini**:

"*El índice de Gini es una métrica para medir la frecuencia con la que un elemento elegido al azar sería identificado incorrectamente"* [170] . En otras palabras, mide el grado de pureza del nodo. *"A mayor índice de Gini menor pureza"* [171]. *"Esto significa que se debe preferir un atributo con un índice de Gini más bajo"* [170].

$$Gini = 1 - \sum_{i=1}^{k}(p(c_i))^2 = \sum_{i=1}^{k} p_{ji}(1 - p_{ji}) \quad (5.40)$$

- **Entropía y ganancia de información**:

La entropía es la medida de impureza del conjunto de datos de entrada, cuanto más impuro mayor entropía.

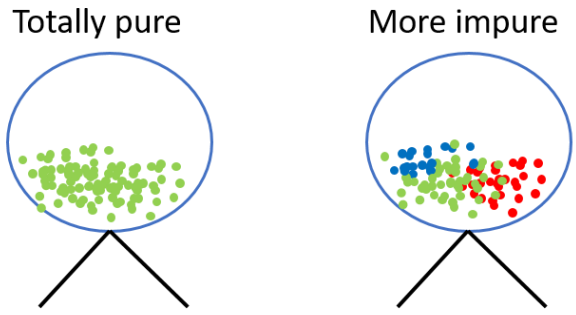


Figura 5-38. Ejemplo de entropía. Fuente: [168]

La entropía es 1 cuando una muestra posee misma cantidad de una clase y otra, sino la entropía estará entre 0 y 1.

$$Entropia = -p_i \log_2(p_i) - q \log_2(q) \quad (5.41)$$

Aquí la p_i y la q se definen como probabilidad de éxito y de fracaso respectivamente. Se escoge la división con "*la entropía más baja en comparación con el nodo principal y otras divisiones*" [168].

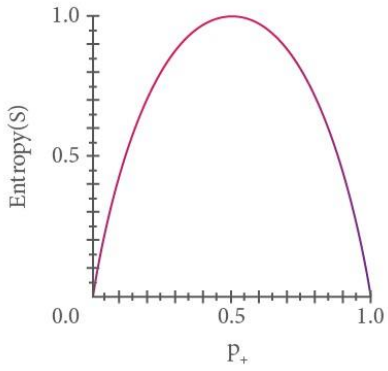


Figura 5-39. "*Función entropía en relación con una clasificación booleana*". Fuente: [168].

Otra forma de expresar la ecuación sería [169]:

$$Entropia = -\sum_{i=1}^{k} p_{ji} \log(p_{ji}) \quad (5.42)$$

La Ganancia de información mide lo bien que una característica separa los elementos de entrenamiento con respecto a la clasificación objetivo. "*La ganancia de información calcula la diferencia entre la entropía antes de la división y la entropía promedio después de la división del conjunto de datos en función de los valores de atributo dados*" [168].

$$\text{Ganancia de información} = Entropia(nodo\ padre) - [promedio\ Entropia(nodo\ hijo)] \quad (5.43)$$

- **Chi-Square**:

Radica en hallar si existe una disimilitud relevante entre nodo padre y nodos hijos, comprobar si la fragmentación del espacio logra mejoría. "*Para ello, se aplica un testeo estadístico* ***chi-square goodness of fit***[54] *empleando como distribución esperada (*H_0*) la frecuencia de cada clase en el nodo parental. Cuanto mayor el estadístico* χ^2 *, mayor la evidencia estadística de que existe una diferencia"* [169]:

$$\chi^2 = \sum_{k} \frac{(obsevado_k - esperado_k)^2}{esperado_k} \quad (5.44)$$

[54] La prueba de bondad de ajuste de chi-cuadrado es una prueba no paramétrica que se utiliza para averiguar cómo el valor observado de un fenómeno dado es significativamente diferente del valor esperado. el término bondad de ajuste se utiliza para comparar la distribución de la muestra observada con la distribución de probabilidad esperada [246].

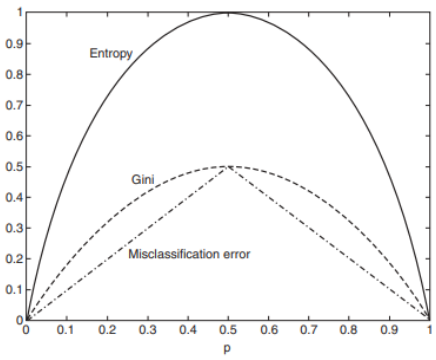


Figura 5-40. Ejemplo de comparación entre las medidas de impureza para problemas de clasificación binaria. La "p" se refiere a la fracción de registros que pertenecen a una de las dos clases. Observar que las tres medidas obtienen el máximo valor cuando la distribución de la clase es uniforme. Fuente: [172].

Una vez elegida la medida como criterio de selección de las divisiones, el proceso a realizar es:

- En "*cada posible división se calcula valor de la medida en cada uno de los dos nodos*" [169].
- Se "*suman los dos valores*" ponderándolos "*por la fracción de observaciones que contiene cada nodo*" [169]:

$$\frac{nº\ observaciones\ nodo\ A}{nº\ observaciones\ total} x\ pureza\ A + \frac{nº\ observaciones\ nodo\ B}{nº\ observaciones\ nodo\ total} x\ pureza\ B \quad (5.45)$$

- Se selecciona como solución la división con mayor o menor valor, depende de tipo de medida utilizada.

"Como los modelos de clasificación basados en árboles de decisión son fáciles de interpretar, no son robustos. Un problema importante con los árboles de decisión es su alta varianza o bajo sesgo. Un pequeño cambio en el conjunto de datos de entrenamiento puede generar un modelo de árbol de decisión completamente diferente" [148]. Son capaces de captar relaciones no lineales, no requiere mucho preprocesado[55], es un algoritmo no paramétrico, tienden al sobreajuste, es sensible a variaciones en los datos y es sensible al desbalanceo[56] [173].

Para saber más sobre este clasificador se puede consultar: [174]- [175]- [176]. Se puede visualizar un ejemplo en [177]. Existe una mejora del algoritmo llamado XGBoost[57] en el cual utiliza varios árboles de decisión a la vez, pero a diferencia de un Random Forest[58], tiene en cuenta los otros árboles para la clasificación, no actúan de manera individual [178]- [179].

En el documento [180] se encuentra un posible ejemplo de uso en BCI. En él se explica que las personas discapacitadas pueden escribir en un ordenador sin usar las manos o la voz. En la Figura 5-41 se ilustra este ejemplo, mostrando una secuencia para la elección de la letra N.

[55] Por ejemplo, no es necesario normalizar los datos
[56] Es preferible que haya un 50% de posibilidad de cada clase.
[57] Significa eXtreme Gradient Boosting. *"Una implementación de árboles de decisión con Gradient boosting. Basado en la idea de crear una regla de predicción altamente precisa combinando muchas reglas relativamente débiles e imprecisas"* [253].
[58] *"Los bosques aleatorios o los bosques de decisiones aleatorios son un método de aprendizaje conjunto para la clasificación, regresión y otras tareas que opera mediante la construcción de una multitud de árboles de decisión en el momento del entrenamiento. Para las tareas de clasificación, la salida del bosque aleatorio es la clase seleccionada por la mayoría de los árboles"* [252].

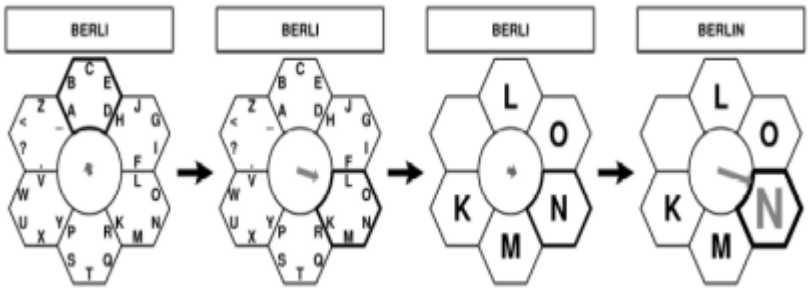


Figura 5-41. Paradigma empleado en HEX-o-Spell. Basado en un paradigma de dos estados, el conjunto de letras se selecciona basándose en un árbol de decisión binario. Fuente: [180].

5.6.1 Resumen del clasificador Decision Trees

Un árbol de decisión segmenta el espacio de características en zonas que no solapen, usando la división binaria recursiva. Clasifican las observaciones creando una secuencia de cuestiones sí/no.

Primero se consideran todas las observaciones de la misma clase, luego se segmenta el espacio identificando los umbrales de determinadas características (nodo de decisión), para poder separar clases. En cada nodo de decisión, hay que preguntarse si una característica es mayor o menor de un umbral, se considerará ese patrón más semejante a un tipo u otro. Cuando se termina de evaluar una característica, se pasa a la siguiente división del árbol, evaluando otra característica, así hasta llegar a una de las hojas del árbol, en la que se determina a cuál de las clases correspondería el patrón.

La medida de elección de características es una heurística que permite dividir los datos de la mejor manera posible. "*Esta medida establece un rango a cada característica*". La característica con mejor valoración se escogerá como característica de división [170] . Como medida para segmentar el espacio de características se puede usar **Classification Error Rate, Índice Gini, Entropía y ganancia de información** o **Chi-Square.** En cada posible división se suman los dos valores de medida de los dos nodos del árbol ponderándolos por la fracción de observaciones que posee cada nodo. La división con mayor o menor valor será la solución, esto es según el tipo de medida usada [169]. Una vez determinados los umbrales, para determinar la clase de un nuevo patrón solo hay que recorrer el árbol respondiendo a las cuestiones de sí/no, según las características correspondientes del patrón estudiado.

Hay que impedir que el modelo se amolde en exceso a los datos de entrenamiento (**overfitting**). Dos posibles soluciones son parada temprana o podado del árbol (reducir el número de divisiones).

5.7 Logistic Regression (LR)

La Regresión Logística binaria (Logistic Regression) es un algoritmo supervisado, se trata de "*una red neuronal con exactamente una neurona*" [181]. Este algoritmo es eficaz para solucionar problemas de clasificación cuya variable objetivo tiene dos resultados posibles. Se utiliza cuando la variable de salida es discreta, esa es la principal diferencia con una regresión lineal. Un hecho a recalcar es que se generan límites de decisión lineales en este algoritmo [126].

También existe la regresión logística multinomial y la regresión logística ordinal [182], para varias clases, pero este documento se centrará en la binaria.

"La Regresión Logística lleva el nombre de la función utilizada en el núcleo del método, la función logística es también llamada función Sigmoide" [182]. Esta función es una curva en forma de S la cual puede asignar un valor entre 0 y 1 a cualquier número real, pero jamás en esos límites exactos [183].

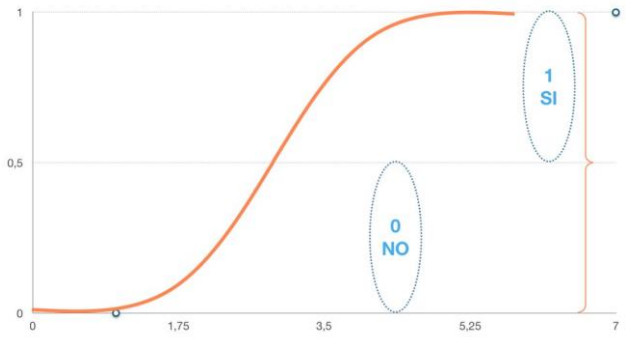


Figura 5-42. Función Sigmoide. El eje horizontal sería los valores de la combinación lineal de características y el eje vertical sería la probabilidad. Fuente: [182].

La ecuación de la función Sigmoide es:

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad (5.46)$$

Donde e es la base de los logaritmos naturales[59] y z es el valor numérico real que quieres transformar. El intervalo de números que se use, por ejemplo, rango entre -5 a 5, se verá transformado en el rango entre 0 y 1 usando la función logística [183].

Visualizando la Figura 5-42, la predicción se volverá 1 si la curva tiende a infinito positivo o si la salida de la función Sigmoide es mayor que 0.5. Se volverá 0 si la curva tiende a infinito negativo o si la salida es menor que 0.5 [182]. Las salidas de la función se pueden considerar como probabilidades, por ejemplo, probabilidad de 0.75 de que sea clase 2. Una representación de lo que realiza este algoritmo sería la Figura 5-43.

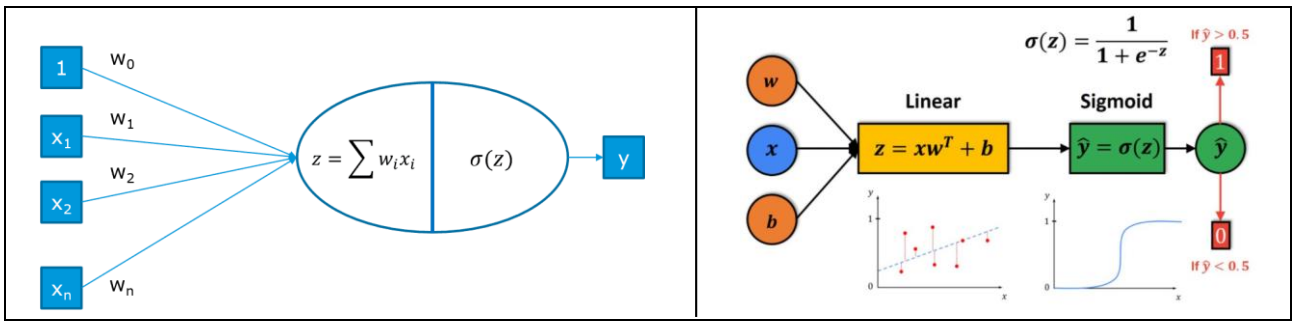


Figura 5-43. Regresión logística es una red neuronal de una neurona. Las dos imágenes muestran el esquema del procedimiento de este clasificador. Fuente: [181]- [184].

[59] El número de Euler

Los valores de x en la Figura 5-43 son las características $\mathbf{f}_\tau$, y n el número de características[60]. Siendo y la variable prevista, la expresión matemática quedaría:

$$y = \sigma(z) = \sigma(WX) = \sigma(\sum(w_i x_i)) = \sigma(\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)) \qquad (5.47)$$

Se aprecian dos pasos: Primero se realiza una combinación lineal que tiene alta similitud a una regresión lineal, los valores de entrada (x) se combinan linealmente utilizando valores de coeficientes[61] (w). Luego se aplica la función Sigmoide al resultado [181]. La ecuación de regresión logística quedaría sustituyendo valores:

$$p_i = y(x) = \frac{1}{1+e^{-\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}} = \frac{1}{1+e^{-(b+\mathbf{xd})}} \qquad (5.48)$$

La y sería equivalente a p_i, siendo p_i la probabilidad del suceso[62], siendo aproximada mediante una función logística, es un valor entre 0 y 1. La regresión logística modela la probabilidad de la clase predeterminada [183]. Teniendo en cuenta que analiza los datos con distribución binomial $Y_i \sim B(p_i, n_i) \quad para \quad i = 1, ..., m$, siendo n_i número de ensayos de Bernoulli, se calcula la probabilidad de ser de una clase para una de las dos categorías del conjunto de datos

Para entender esto mejor se puede consultar [185]. Haciendo cálculos se ve lo siguiente:

$$\begin{cases} p_i = \dfrac{e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}}{1+e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}} \\ 1 - p_i = \dfrac{1}{1} - \dfrac{e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}}{1+e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}} = \dfrac{1+ \cancel{e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}} - \cancel{e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}}}{1+e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}} = \dfrac{1}{1+e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)}} \end{cases}$$

$$\frac{p_i}{1-p_i} = e^{\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)} \rightarrow \ln(\frac{p_i}{1-p_i}) = \sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n)$$

El $\frac{p_i}{1-p_i}$ es la posibilidad en favor de acierto. Puede simplificarse al cálculo de una regresión lineal o una regresión exponencial.

Se denomina función **logit**[63] a la inversa del sigmoide, el logaritmo neperiano de la probabilidad de que ocurra el evento entre la probabilidad de que no ocurra ese evento. Toma valores de entrada en el rango 0 y 1 y los transforma en valores que están en todo el rango de números reales, que se pueden utilizar para plasmar una relación lineal entre las probabilidades logarítmicas y los valores de las características [186]. La operación **logit** de p_i se define como:

$$\text{logit}(p_i) = \ln(\frac{p_i}{1-p_i}) = \ln(p_i) - \ln(1-p_i) \qquad (5.49)$$

[60] Sería la p en el código realizado para LDA en MATLAB.
[61] Los w son los parámetros y las x son las variables independientes.
[62] Probabilidad del evento de interés, la probabilidad de ser de clase 1 o 2.
[63] Sería el logaritmo neperiano del Odds, medida de probabilidad relativa que tiene un evento de ocurrir frente a que no ocurra [255]- [256]- [191].

A partir de lo demostrado anteriormente, se puede deducir que es igual a una combinación lineal de los factores independientes. Sería la siguiente expresión [186]:

$$\text{logit}(P(p=1|\,x) = w_0x_0 + w_1x_1 + \ldots + w_nx_n = \sum_{i=0}^{n} w_ix_i \qquad (5.50)$$

Hay que recordar que la predicción de probabilidad se transformará en valores binarios [183].

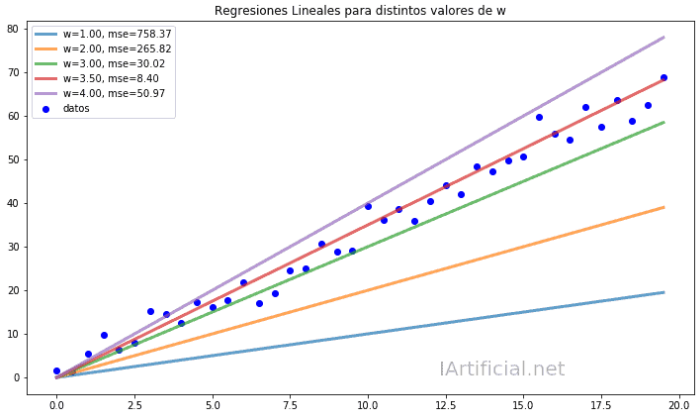


Figura 5-44. Ejemplo de regresión lineal hallando w, w = 3.5 es la solución para este ejemplo. Fuente: [187].

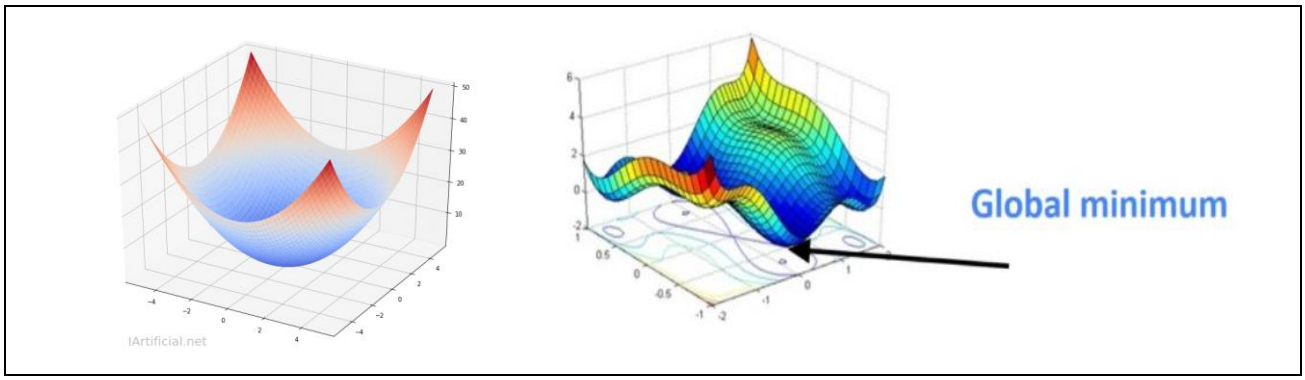


Figura 5-45. Gradiente descendente. Fuente: [187]- [184].

Hay que hallar los valores de los coeficientes w, para ello es necesario un aprendizaje automático. Se tomará de partida un número de muestras etiquetadas, siendo estas etiquetas 0 o 1. Obteniéndose un modelo que minimice el coste de errar al establecer las clases de las muestras que se tienen. Se entrena minimizando una función de error; una elección apropiada de dicha función para problemas de clasificación binaria podría ser el error de entropía cruzada[64] [188].

El coste de equivocarse al clasificar **x** se podría definir al usar la pérdida cuadrática[65] como:

$$\min_{\mathbf{w}} J(\mathrm{w}) = \min_{\mathbf{w}} \frac{1}{N_\tau} \sum_{\tau=1}^{N_\tau} (k_\tau - p_i(\mathrm{x}, \mathrm{w}))^2 \qquad (5.51)$$

[64] La función de "entropía cruzada categórica" es una función convexa con un solo mínimo. Los valores del parámetro pueden calcularse utilizando cualquier algoritmo de optimización como, por ejemplo, el descenso de gradiente [33].
[65] La pérdida al cuadrado (Squared loss, L_2) es una función de pérdida que se puede utilizar en el entorno de aprendizaje en el que estamos, predecir una variable de valor real ***y*** dada una variable de entrada ***x*** [257]- [258].

Siendo k_τ la clase real de cada ensayo de entrenamiento y N_τ el número de muestras. Se quiere hallar qué parámetros w minimizan la función de coste, es decir, hay que optimizar la función de coste.

No se puede despejar **w** de manera sencilla. No existe una fórmula que devuelva los coeficientes w óptimos, se tienen que estimar (ejemplo en Figura 5-44). El aprendizaje se realizará con optimización numérica, a partir de un valor inicial de **w** y unas ecuaciones de actualización, se conseguirá una posible solución. Hay diferentes métodos para ello, por ejemplo, con gradiente[66] descendiente. El gradiente descendiente es un procedimiento de optimización numérica que permitirá establecer los mejores coeficientes.

Uno de los aspectos a considerar es que la función de coste no es convexa, por ello se utilizará otro tipo de coste. Se establece que si $k_\tau = 1$ el coste será $-\ln(\hat{p}_i)$ y si $k_\tau = 0$ el coste será $-\ln(1-\hat{p}_i)$ [189]. Se establecerá una versión más compacta, definiéndola para cada una de las muestras de entrenamiento del siguiente modo:

$$\min_{\mathbf{w}} J(\mathrm{w}) = \min_{\mathbf{w}} \frac{-1}{N_\tau} \sum_{\tau=1}^{N_\tau} k_\tau \ln(\hat{p}_i(\mathbf{w})) + (1-k_\tau)\ln(1-\hat{p}_i(\mathbf{w})) \quad (5.52)$$

Ésta sí es convexa, la función de coste es la entropía cruzada. Una vez determinada la función de coste, se le aplicará una técnica de optimización numérica para minimizarla. El método de descenso de gradiente consistirá en ir viendo como son las pendientes de la función de coste convexa de manera iterativa, usando derivadas parciales, buscando cuándo se produce el mínimo, cuándo converge [190]. Se puede controlar los saltos que se realizan en este algoritmo con una constante llamada velocidad de aprendizaje. Pueden existir sucesos en que la función de coste sea sumamente plana, para evitar muchas iteraciones se pondría una constante de velocidad mayor que 1. Si pasa lo contrario, los gradientes muy grandes, se pondría una constante menor de 1 para reducir los saltos. Para determinar la mejor velocidad se utiliza el cross-validation.

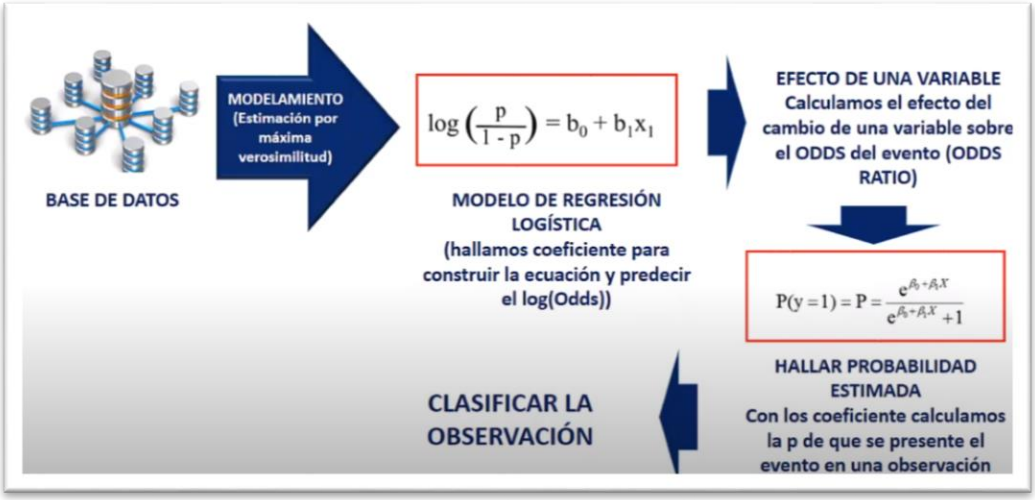


Figura 5-46. Esquema de proceso LR de clasificación. Las 'b' son lo que se denomina 'w' en este trabajo. Fuente: [191].

Se recomienda leer el artículo [192] y ver [191]. Además, se puede consultar: [193]- [194]- [195]- [196].

[66] "*El gradiente es una generalización de la derivada*" [264], "*el conjunto de todas las derivadas parciales de una función*" [187]. Se consideran varios planos paralelos entre sí, la intersección de la función con los planos crea curvas de nivel. Uno se puede preguntar por el gradiente de la función en cada punto. Vendrá determinado por la dirección y la magnitud del gradiente [187].

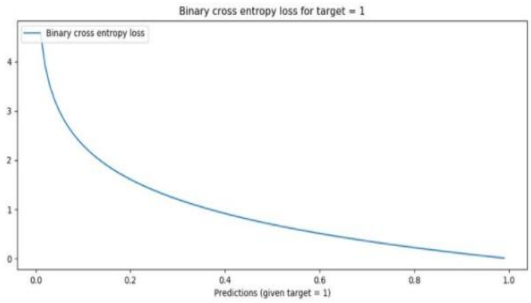


Figura 5-47. "*La pérdida de entropía cruzada aumenta a medida que la probabilidad predicha difiere de la etiqueta real. A medida que la probabilidad estimada se acerca al valor 1, la pérdida logarítmica disminuye lentamente. Un modelo perfecto tendría una pérdida logarítmica de 0*". Fuente: [184].

Es significativo escalar los datos antes de emplear este sistema de optimización numérica. Para entender mejor este concepto se puede consultar [187].

También se puede añadir términos de regularización en el procedimiento de aprendizaje, sería incluir una penalización a la función de coste. Esto equivaldría a tener una nueva función de coste que es igual a la anterior más otro término [197]. Esto permite tener modelos más sencillos que permiten generalizar mejor, es decir, ayuda a reducir el sobreajuste de los modelos. Las regularizaciones más empleadas en Machine Learning son **Lasso** (L1), **Ridge** (L2) y **ElasticNet** que combina Lasso y Ridge. Para más información consulte el documento [198].

Una vez realizado el aprendizaje, para cada nueva muestra se hallará la probabilidad estimada de que se exhiba la clase ($P(p=1|x)$) con los coeficientes calculados.

"*Los métodos de regresión logística y LDA están estrechamente relacionados y difieren principalmente en sus procedimientos de ajuste. En consecuencia, ambos suelen producir resultados similares. Sin embargo, el LDA asume que las observaciones se extraen de una distribución gaussiana con una matriz de covarianza común en cada clase, por lo que puede proporcionar algunas mejoras sobre la regresión logística cuando este supuesto se cumple aproximadamente. Por el contrario, la regresión logística puede superar al LDA si no se cumplen estos supuestos gaussianos. Tanto LDA como la regresión logística producen límites de decisión lineales, por lo que cuando los verdaderos límites de decisión son lineales, los enfoques LDA y de regresión logística tenderán a funcionar bien. El QDA, por otro lado, proporciona un límite de decisión cuadrático no lineal. Así, cuando el límite de decisión es moderadamente no lineal, el QDA puede dar mejores resultados"* [123].

5.7.1 Resumen del clasificador LR

El LR es apropiado cuando se tienen dos clases. La regresión logística proporciona una salida discreta. Lo que va a hacer el algoritmo es que a cada una de las variables les va a asignar una probabilidad que va a caer en el rango de 0 a 1. El modelo se utiliza para estimar la probabilidad de cierta clase dado un conjunto de variables independientes, a cada muestra se le asigna una probabilidad. Si está por debajo de 0.5, se le asigna una clase, y si está por encima, se le asigna la otra clase.

Para su realización se define la función $\text{logit}(p_i) = \ln(\frac{p_i}{1-p_i})$, la cual permite hallar una variable que se encuentre entre los valores de 0 y 1.

La probabilidad se obtiene usando la función sigmoide, la cual es la inversa de la función **logit**, permitiendo que sea el eje vertical el que esté entre valores de 0 y 1, y no el eje horizontal. El eje horizontal será ahora la combinación lineal de las características.

La probabilidad de que un suceso ocurra es $P(\mathrm{y}=1)=\frac{1}{1+e^{-\sum(w_0x_0+w_1x_1+\ldots+w_nx_n)}}$, ésta es la ecuación para calcular la probabilidad que se usará para clasificar, siendo **x** el vector de características. Se puede evaluar los valores de **x** en la función sigmoide para saber qué probabilidad dan. Para simplificar este cálculo, el modelo será una función lineal, se aplica una transformación de tal forma que sea igual a la combinación lineal de las variables. Entonces el modelo de LR quedaría como $\ln(\frac{p_i}{1-p_i})=w_0x_0+w_1x_1+\ldots+w_nx_n=\sum_{i=0}^{n}w_ix_i$. Modela la probabilidad de que sea de una clase dado un conjunto de características.

El objetivo para obtener el clasificador es obtener los valores de los coeficientes, **w**. Se busca minimizar la función de coste, es la función que indica el error que se comete al clasificar las muestras, en este caso la entropía cruzada. Para ello se utiliza optimización numérica, se puede usar el algoritmo tipo descenso de gradiente. Pudiendo utilizar la regulación para mejorar los resultados. Una vez realizado el aprendizaje, para cada nueva muestra se hallará la probabilidad estimada de que se presente la clase ($P(p=1|x)$) con los coeficientes calculados.

Es favorable usar LR cuando los valores para cada clase se distribuyen de forma que se separen en función de la variable independiente. La mayoría de las muestras de cada clase debe concentrarse respectivamente en lado izquierdo y derecho de la curva sigmoide.

5.8 Conjunto de clasificadores (Ensemble Classifiers)

Los **Ensemble Classifiers** combinan los resultados de muchos modelos de aprendizaje deficientes en un modelo de conjunto de alta calidad. Las calidades dependen de la elección del algoritmo [199]. Estos clasificadores se pueden dividir en dos categorías por su procedimiento: Secuencial o Paralelo. Los clasificadores Ensemble más destacados son los que se basan en los métodos bagging, boosting y stacking [200], pero existen más metodologías. Algunos ejemplos de clasificadores son Boosted Trees, Bagged Trees, Subespacio Discriminante, Subespacio KNN y RUSBoost Trees, entre otros.

Estos clasificadores no se van a explicar con detalle en esta memoria, son algoritmos más avanzados y complejos. Además, como se verá en capítulo 6, tardan demasiado en realizar el entrenamiento en comparación con los otros explicados, lo cual no es ventajoso para el propósito de este proyecto. Se puede consultar información sobre estos algoritmos en las siguientes referencias: [201]- [202]- [203]- [200]- [204]- [205]- [206]- [207]. Para saber los que están disponibles en MATLAB, se puede consultar la referencia [208].

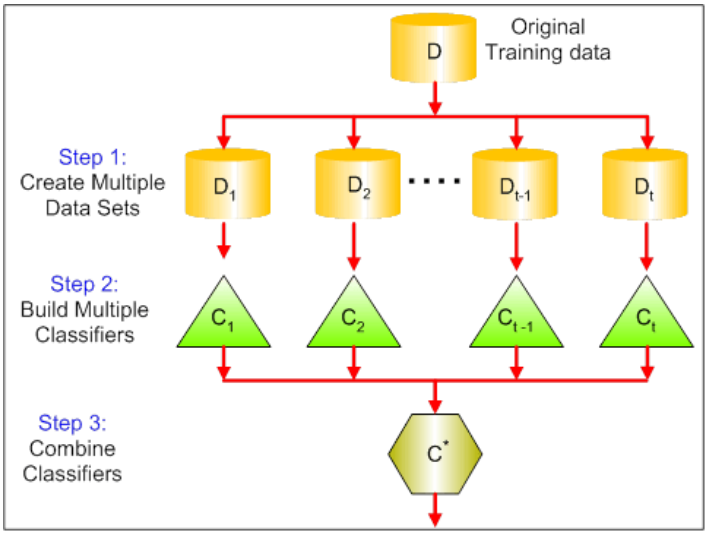


Figura 5-48. Estructura general de un Ensemble Classifiers. Fuente: [203].

5.9 Conclusiones

El aprendizaje automático consiste en ajustar unos parámetros durante una fase de entrenamiento, pudiendo utilizar estos para predecir la clase de patrones futuros a partir de sus características. Hay dos tipos de clasificación: supervisada y no supervisada.

Se han explicado diferentes métodos de clasificación que se pueden usar en el sistema MI-BCI : LDA, QDA, RDA, SVM, Naive Bayes, KNN, k-Means, Decision Trees, LR, Ensemble Classifiers. Siendo el más utilizado en este caso el LDA. Para la clasificación en tiempo real se asumirá que inicialmente hay una probabilidad acierto de cada clase del 50%. En este proyecto el K-Means y el RDA no se llegaron a implementar, pero se han querido explicar de forma breve.

En el LDA, se busca un hiperplano que permita separar las dos clases. El proceso se simplifica buscando una recta sobre la que proyectar los patrones, en el espacio de características, de forma que queden los de distinta clase lo más separados posible y los de las misma clase lo más cercanos posible. Hay que calcular la media, la covarianza y la probabilidad a priori de cada clase. Con ello se puede hallar la covarianza media total de los patrones, y así hallar el filtro óptimo para proyectar. Una vez realizado esto, se establece un umbral, un punto en la recta, que establezca la separación de las clases. Se determina el tipo de muestra de acuerdo con la posición del punto proyectado. Los valores que sean mayor de este umbral se considerarán de una clase y los menores de la otra clase. Finalmente, se consigue una función de decisión, que asigna la clase a partir del filtro óptimo de proyección, el umbral y el patrón que se esté estudiando.

En el QDA se pretende separar las clases utilizando curvas en lugar de rectas. Se mide la distancia de un patrón al centro de cada clase, asignándole la clase a la que menor distancia esté. La medida de distancia que se suele usar es la distancia de Mahalanobis.

El RDA es un compromiso entre el LDA y el QDA, cuya principal característica es que utiliza regularización. La regularización es el proceso de encontrar un pequeño conjunto de características que produzcan un modelo predictivo eficaz. El RDA se usa cuando número de observaciones es menor que la dimensión de cada observación.

El SVM busca un hiperplano que separe las dos clases de forma que se maximice la distancia entre el hiperplano y las muestras de entrenamiento más cercanas de ambas clases (margen). Se permitirá cometer unos pocos errores, algunas observaciones no estarán en el lado correcto del margen o el hiperplano. En SVM es común utilizar el

método del Kernel, consiste en proyectar un conjunto de datos no separables linealmente en un espacio de dimensión mayor mediante una transformación no lineal, en donde probablemente sí se puedan separar linealmente.

El Naive Bayes se basa en el Teorema de Bayes y presupone que las características de cada clase exhiben una distribución estadística gaussiana o una mezcla ponderada de distribuciones gaussianas. La clasificación consistiría en que a un patrón se le asigna la clase a la que tenga más probabilidad de pertenecer.

En KNN se asigna una clase a un nuevo patrón según los patrones que tiene alrededor. Se busca el número de patrones más cercanos que se le indique, calculando la distancia a cada uno de ellos. Se puede predecir la clase basándose en la que más se repite de los datos que lo rodean, pero también se pueden usar otros criterios similares.

El K-Means se basa en el mismo concepto que el KNN, pero para casos de aprendizaje no supervisado. Consiste en fijar un número de grupos y realizar el entrenamiento con el objetivo de minimizar la distancia al cuadrado de cada punto al centro de su grupo. Para ello usa el algoritmo de Lloyd.

El Decision Trees segmenta el espacio de características en zonas que no solapen, usando la división binaria recursiva (cuestiones de sí/no). En el entrenamiento, primero se consideran todas las observaciones de la misma clase, luego se van identificando los umbrales de determinadas características (nodo de decisión) para poder separar clases. Para elegir cuáles características ayudaran a determinar la segmentación del espacio de características (umbrales), se puede usar uno de los siguientes tipos de medida: **Classification Error Rate, Índice Gini, Entropía y ganancia de información** o **Chi-Square.** En cada posible división se suman los dos valores de medida de los dos nodos del árbol, ponderándolos por la fracción de observaciones que posee cada nodo. La división con mayor o menor valor será la solución, según el tipo de medida usada. Una vez determinados los umbrales, para determinar la clase de un nuevo patrón solo hay que recorrer el árbol respondiendo a las cuestiones de sí/no, según las características correspondientes del patrón estudiado. Por otra parte, para evitar el sobreajuste en el entrenamiento se utiliza la parada temprana o el podado del árbol.

El LR es una red neuronal con una neurona. Se asigna a cada entrada una probabilidad que va a caer en el rango de 0 a 1. Dependiendo de si está por debajo o por encima de 0.5, se le asigna una clase u otra. Este algoritmo estima la probabilidad de que se dé cierta clase, dado un conjunto de variables independientes. La probabilidad se obtiene usando la función sigmoide, la cual es la inversa de la función ***logit.*** Para simplificar este cálculo, el modelo será una función lineal, se aplicará una transformación de tal forma que el modelo sea igual a la combinación lineal de las características. El entrenamiento de este clasificador consiste en obtener los valores de los coeficientes de la combinación lineal, de forma que se minimice la función de coste (error que se comete al clasificar). Esa función será la entropía cruzada. Es un problema de optimización numérica, el cual se puede resolver usando el algoritmo de descenso de gradiente. Una vez realizado el entrenamiento, a cada nueva muestra se le halla la probabilidad estimada de que sea de una clase determinada, usando los coeficientes calculados.

En Ensemble Classifiers se combinan varios modelos de aprendizaje para determinar la clase de un patrón. Se pueden aplicar en serie o en paralelo. Existen varios algoritmos de este tipo, siendo todos bastantes complejos en comparación con los otros que se han explicado en este trabajo.

6 RESULTADOS EXPERIMENTALES

"El cerebro humano tiene 100 mil millones de neuronas, cada neurona conectada a otras 10 mil neuronas. Sentado sobre sus hombros está el objeto más complicado del universo."

Michio Kaku

Se hacen pruebas con distintos clasificadores para ver cuáles dan mejores resultados frente a la misma tarea de detección de imaginación motora, de la mano izquierda y derecha. Se quiere una probabilidad de acierto alta y que no tarde demasiado en realizar la clasificación. El tiempo es relevante porque, al ir destinado a un sistema BCI, no se puede dejar que el usuario espere demasiado tiempo la respuesta de su pensamiento. Por ejemplo, si se utilizase un BCI para mover una prótesis de un brazo robótico, no puede tardar una hora en clasificar la señal del cerebro para realizar el movimiento.

Con respecto a la base de datos utilizada, como ya se explicó en el apartado 2.2.2, se utilizará el MI-BCI dataset 2a de la BCI Competition IV [67], con las señales habiendo sido ya preprocesadas por la Universidad de Sevilla. Por comodidad para el lector, se hará un recordatorio sobre cómo se estructuraron los datos: Habrá datos de 9 usuarios y dos sesiones por usuario. Cada sesión está compuesta por 144 ensayos (trials) de 500 muestras cada uno, habiendo el mismo número de ensayos de cada clase en la sesión, utilizando dos clases: mano izquierda y derecha (1 y 2).

Los ensayos disponibles se dividirán en dos subconjuntos: el de entrenamiento y el de testeo. Es necesario hacer esto para ver cómo actúa realmente el clasificador ante ensayos nuevos. Si se ajusta demasiado bien al modelo de entrenamiento y no es capaz de clasificar valores nuevos, como se explicó en el clasificador Decision Trees, se produce sobreajuste. Hay que mencionar que, al ser el entrenamiento online, seguramente la probabilidad de acierto en el entrenamiento será menor que la de testeo, ya que el testeo se hace con el último clasificador y la última matriz de filtros obtenidos en el entrenamiento. En un caso offline, la probabilidad de acierto en el entrenamiento será mayor a la de testeo. Para corroborar el correcto funcionamiento del algoritmo, en algunos clasificadores se ha probado el clasificador final obtenido con los ensayos del entrenamiento, es decir, se comprueba qué probabilidad de acierto daría con los datos de entrenamiento. Esto sería similar a obtener la probabilidad de acierto del entrenamiento offline.

Aunque la base de datos estuviese preparada con la intención de usarse una sesión para entrenamiento y otra para testeo para cada usuario; en general, se ha considerado que de cada sesión unos 100 ensayos corresponderían al entrenamiento y 44 al testeo, tanto para los resultados obtenidos con un solo usuario como con varios usuarios.

Además, el número ensayos de entrenamiento y testeo se ha indicado en la introducción de cada simulación realizada, pues se cambió con respecto al caso general en algunas pruebas concretas.

Otro aspecto importante es que, dependiendo del clasificador a usar, se va a necesitar esperar a que haya dos o más de dos ensayos almacenados antes del entrenamiento, según se requiera. Hay que recordar que se quiere un entrenamiento en tiempo real, no hay que esperar a que estén casi todos almacenados. Por otra parte, es importante mencionar que, la probabilidad de acierto en entrenamiento online se calcula sin tener en cuenta los ensayos del inicio, pues de esos no se ha estimado su clase; los que ha habido que esperar a que se almacenen.

En cualquier caso, hay que establecer que haya un número suficientemente grande de muestras de entrenamiento para una correcta clasificación. A mayor número de muestras, mejor se estimarán los parámetros del clasificador. También se debe coger un número suficientemente grande de muestras de testeo para dar una estimación útil del clasificador, es decir, obtener una probabilidad de acierto realista.

En todos los clasificadores se usa CSP, si no se especifica lo contrario, usando el mismo número de filtros y, por tanto, de características. Se utilizan 8 filtros, las 4 primeras y las 4 últimas columnas de la matriz de filtros ordenada. Hay unas determinadas características relevantes, y si se añaden más darán lugar a información adicional. Por añadir información adicional, no tiene porqué mejorar el rendimiento del clasificador. Normalmente aplicar CSP produce mejoras si la base de datos proviene de muchos sensores. Cuando el número de sensores no es grande, la mejoría depende en gran medida del tipo de clasificador que se use. Se hicieron también pruebas para ver qué resultados se obtenían al no aplicar CSP, es decir, poniendo la matriz de filtros como una matriz identidad del tamaño del número de canales, como ya se mencionó, en este trabajo son 22 canales. También se realizaron pruebas sobre qué sucedía si no se establecía el comando de forzar a que los filtros estuviesen en orden ascendente.

Se ha probado en algunos clasificadores tres posibles maneras de programarlos (paso a paso, funciones especiales de MATLAB o con la aplicación[67] de MATLAB Classification Learner) y en otros casos dos o una de esas maneras, tanto para un usuario como para varios usuarios.

Cuando se quiso utilizar la App Classification Learner, lo que se hizo primero fue coger un conjunto de ensayos de una sesión, aplicarles el filtrado espacial (CSP) y extraer sus características, para luego meter esas características en la App. Una vez introducidos los datos, se entrenaron los clasificadores disponibles en la aplicación, realizando cross-validation folds[68] de 5 folds, gracias a una opción que viene disponible en la aplicación. Así se comprobaba cuáles daban mejores resultados de probabilidad de acierto. Tener presente que la App puede realizar reducción de dimensionalidad PCA [209], pero no se puede especificar que haga CSP, no permite pasarle la rutina de MATLAB. En ninguno de los experimentos realizados se aplicó la reducción de dimensionalidad PCA después del CSP, se deshabilitó esa opción. En lo que respecta al número de ensayos usados en la App, viene especificado junto a cada gráfico obtenido, los cuales se mostrarán más adelante.

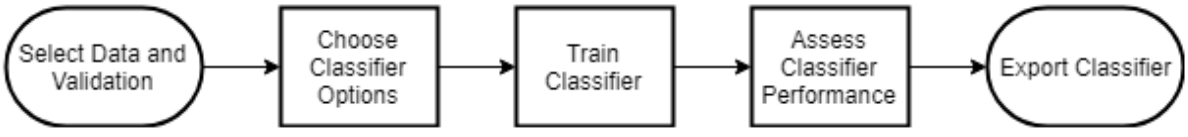


Figura 6-1. "*Este diagrama de flujo muestra un flujo de trabajo común para los modelos de clasificación de entrenamiento o clasificadores en la aplicación Classification Learner*". Fuente: [210].

[67] "*Programa o conjunto de programas informáticos que realizan un trabajo específico*" [250].
[68] Si se utiliza "*la validación cruzada k-fold, la aplicación calcula las puntuaciones de precisión utilizando las observaciones de los k-folds de validación e informa del error medio de validación cruzada. También realiza predicciones sobre las observaciones de estos folds (pliegues) de validación y calcula la matriz de confusión y la curva ROC basándose en estas predicciones*" [265] .

Los clasificadores disponibles en la App también se exportaron como archivos **.m**, para poder probarlos en las mismas condiciones que el resto, realizando el entrenamiento de forma que aumente el número de ensayos de manera sucesiva. Además, la función del clasificador devuelve, aparte del modelo entrenado, la probabilidad de acierto del modelo con cross-validation de 5 folds. Para más información sobre la aplicación se puede consultar: [211].

Los experimentos realizados se dividen en dos categorías:

- Para una sesión de un usuario determinado se realizaron varias pruebas en diferentes circunstancias y con diversos algoritmos de clasificación. Si no se especifica lo contrario, la actualización del CSP y del clasificador se hará de manera que imite a cómo se haría en tiempo real. Principalmente se tiene en cuenta las probabilidades de acierto y el tiempo en procesar un ensayo, en entrenamiento online y en testeo. Además, cada valor medio que se calculó se representó con su correspondiente desviación. Con respecto a los tiempos calculados en el entrenamiento online y en el testeo, se graficaron histogramas que permitían visualizar en torno a qué valores de tiempo se demora el procesamiento de un único ensayo en la mayoría de las iteraciones. También se indicó cuál fue el mayor tiempo que se tardó en procesar un ensayo, teniendo en cuenta todos los ensayos de entrenamiento. También se indicó cuál fue el mayor para el caso de los de testeo.

- Para diferentes sesiones y usuarios se probaron diversos clasificadores y se realizaron varias pruebas en diferentes circunstancias, de manera semejante al caso de un usuario, pero repitiéndose el proceso con varios usuarios. Se utilizaron las 18 sesiones de los 9 usuarios, pudiéndose obtener una media de las probabilidades de acierto obtenidas en las sesiones. También se representaron histogramas de tiempo. Las gráficas y los resultados de entrenamiento son, si no se especifica lo contrario, los obtenidos con el entrenamiento online.

Para los siguientes experimentos, se quitaron las pausas y las gráficas de comprobación que se pusieron en algunos clasificadores. Las gráficas eran para que el usuario visualizara si se estaba clasificando bien la señal en el entrenamiento. Las pausas eran para que el usuario tuviera tiempo para pensar, los tiempos calculados no tienen en cuenta estas pausas. En el capítulo 7 se profundizará sobre la programación realizada, volviéndose a mencionar este asunto.

6.1 Un usuario

Antes de obtener resultados de cada clasificador se realizaron una serie de comprobaciones con los datos:

Se evidenció que efectivamente había 72 ensayos (***trials***) de cada clase en una sesión, como se ve en Figura 6-2. El mismo número de ensayos (***trials***) para cada clase.

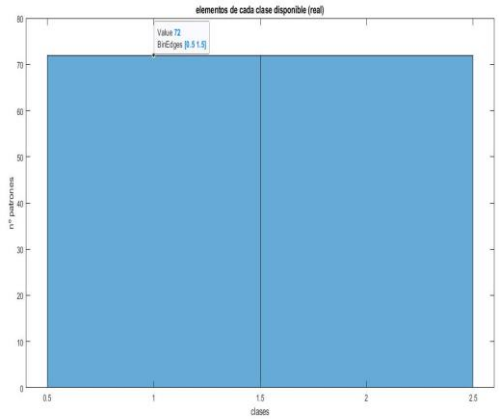


Figura 6-2. Número de ***trials*** que hay de cada clase en una sesión de un usuario.

Se hizo una comprobación de los resultados con diferentes clasificadores para una misma sesión de un usuario, el del archivo:

'c42a_7_class12_test.mat'.

Se quiso comprobar cómo se distribuían las características calculadas. En Figura 6-3 se han representado en orden las 8 características usadas, con diagrama de cajas, comparándose cada característica en ambas clases. Un diagrama de cajas es una forma sencilla de ver varias distribuciones. Las cajas representan la distribución de los valores de característica para cada una de las clases. Se usa para ver qué características diferencian mejor a las clases. Si los valores de las características son significativamente diferentes para una clase y otra, entonces esas características sirven para distinguir entre las clases [78].

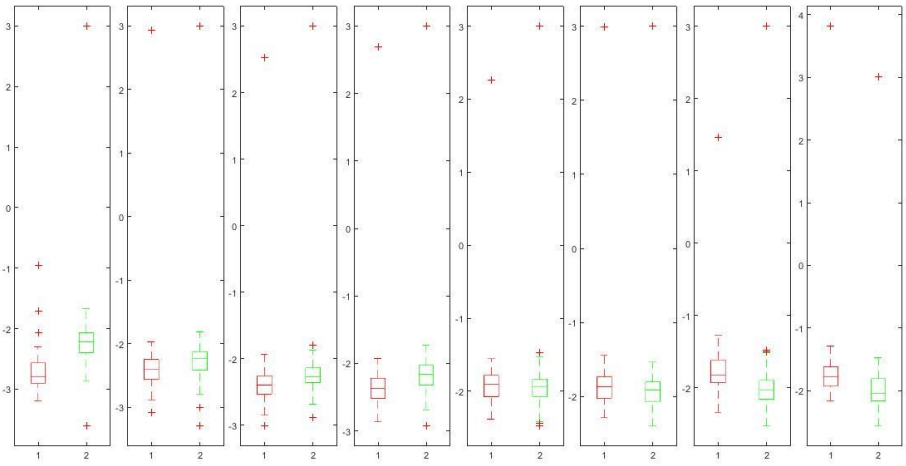


Figura 6-3. Características de cada clase, estas características son las obtenidas después de aplicar CSP.

Se puede apreciar que la primera característica es la que mejor diferencia entre las clases. Para esta gráfica se usaron las características de los 100 primeros ensayos (***trials***) del usuario ya mencionado.

Usando la App de MATLAB se comprobó el resultado de probabilidad acierto de todos los clasificadores disponibles con los 100 ***trials*** de entrenamiento y usando cross-validation folds de 5 folds en una sesión:

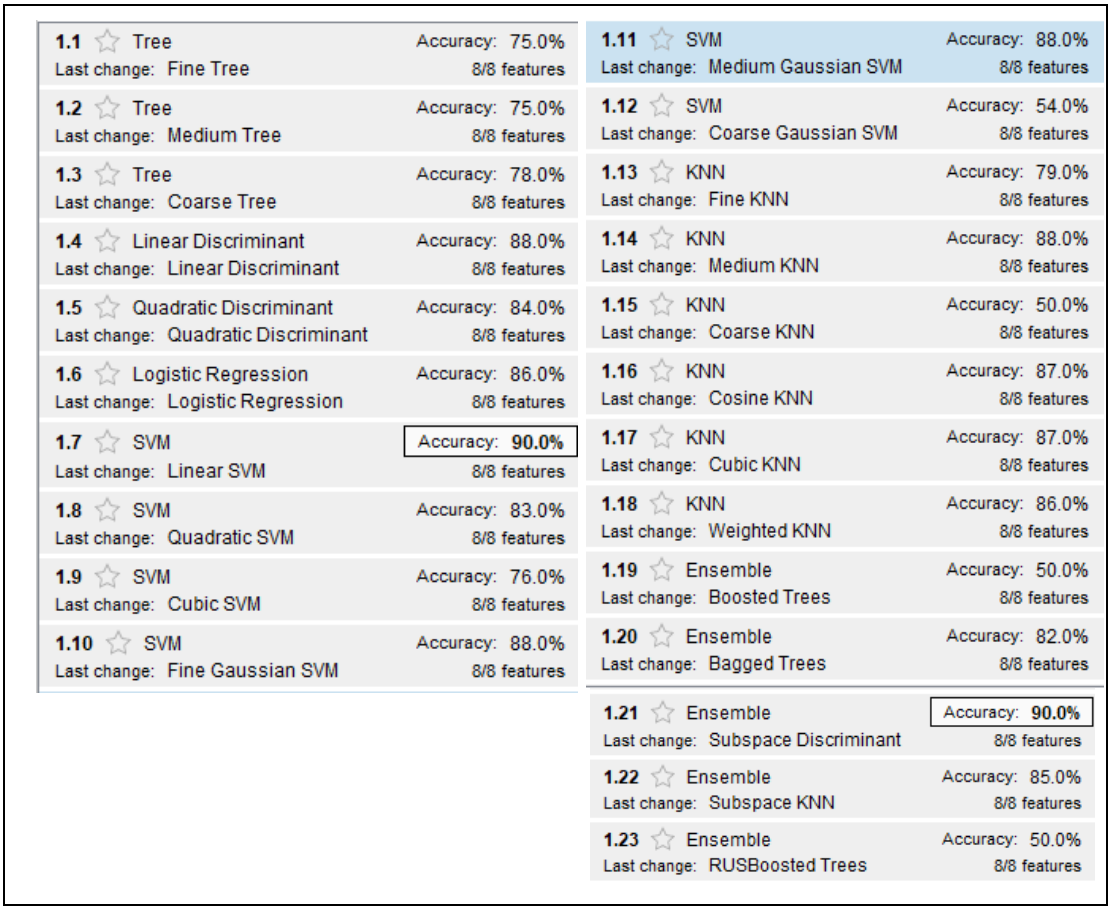


Tabla 6-1. Resultados usando App de MATLAB con los 100 ensayos de entrenamiento.

6.1.1 Análisis discriminante

6.1.1.1 LDA

6.1.1.1.1 Código realizado paso por paso

6.1.1.1.1.1 Pruebas sin ordenar matriz de filtros W

Se realizaron pruebas con o sin incluir el comando de ordenamiento de los filtros, el cual se mencionó en el apartado 4.4. Si no se fuerza la ordenación, sería como tener un menor efecto del CSP. En otras palabras, sin poner en el código:

```
[B,indice]=sort(diag(D));
 V=V(:,indice);
```

Un dato para recalcar es que, como se verá próximamente en el documento, se comprobó que algunos clasificadores no se veían afectados al quitar estas líneas en el código. Se comprobó que ya aparecían ordenados desde el principio los elementos de la matriz en esos casos. En otras palabras, los clasificadores que daban mismos resultados en las pruebas de con o sin comando ordenar, podría deberse a que ya aparecen ordenados de menor a mayor los elementos.

6.1.1.1.1.1.1 Prueba 100 de entrenamiento y 44 de testeo

Se tomó 100 ***trials*** para entrenamiento y 44 de testeo, procedentes de una sesión. Se tomaron estos valores porque, al hacer varias pruebas con diferentes números de ***trials***, se vio que daba resultados aceptables en LDA.

```
p_acierto: 0.7 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento:    0.871498 segundos
```

tiempo máximo un trial test: 0.015564 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 77.551020 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 22.448980%

Tasa clasificación errónea test: 15.909091 %

La tasa de predicción (y la de clasificación errónea) ofrece un único valor para el rendimiento global del modelo, pero puede resultar de interés ver de manera más detallada los aciertos en cada clase. Una matriz de confusión muestra el número de observaciones por cada combinación de clase predicha y real. Se visualiza normalmente sombreando los elementos según su valor. A menudo, los elementos diagonales (las clasificaciones correctas) están sombreados en un color y el resto de los elementos (las clasificaciones incorrectas) en otro color [78].

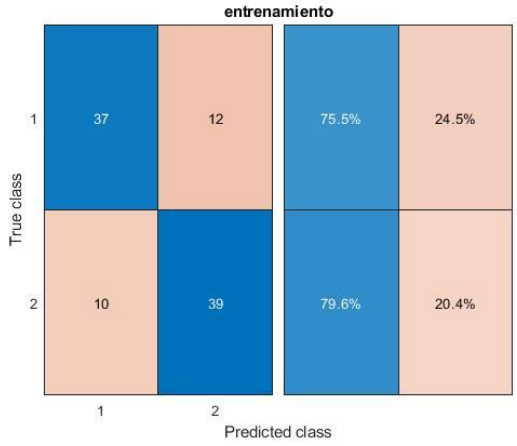


Figura 6-4. Matriz de confusión entrenamiento LDA paso a paso.

Se puede ver en la Figura 6-4 que 12 de la clase 1 fueron clasificadas como clase 2, y que 10 de la clase 2 fueron clasificadas como clase 1, durante entrenamiento. Mientras que un total de 76 patrones fueron correctamente clasificados.

Cuando se crea una gráfica de confusión, se puede agregar información sobre la tasa de falsos negativos y falsos positivos de cada clase agregando resúmenes de filas o columnas, respectivamente. El resumen de filas muestra la tasa de falsos negativos para cada clase.

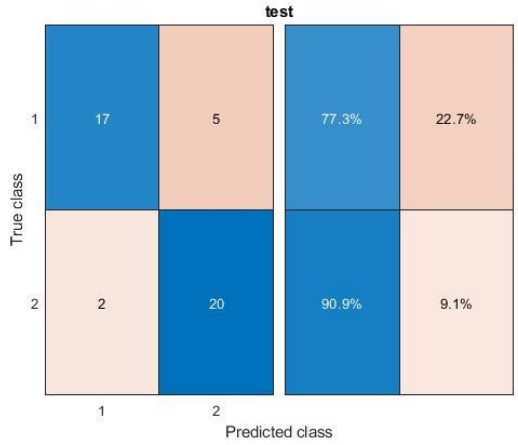


Figura 6-5. Matriz de confusión test LDA paso a paso.

Se puede ver en la Figura 6-5 que 5 de la clase 1 fueron clasificadas como clase 2, y que 2 de la clase 2 fueron clasificadas como clase 1, durante la validación del clasificador.

Se decidió representar los tiempos máximos de procesamiento de un trial en forma de histograma:

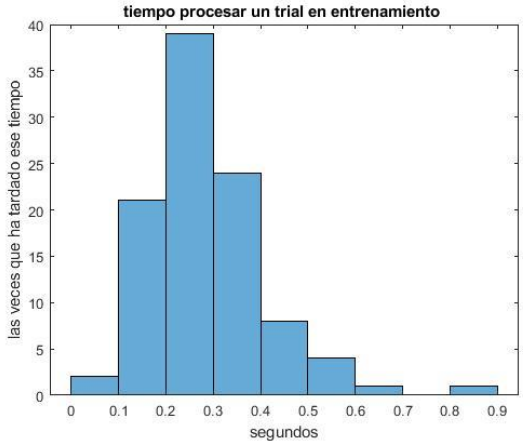


Figura 6-6. Histograma de tiempo en ensayos LDA paso a paso entrenamiento.

Como se ve en la Figura 6-6, unas 39 veces ha tardado entre 0.2 y 0.3 segundos en procesar un ensayo durante el entrenamiento online.

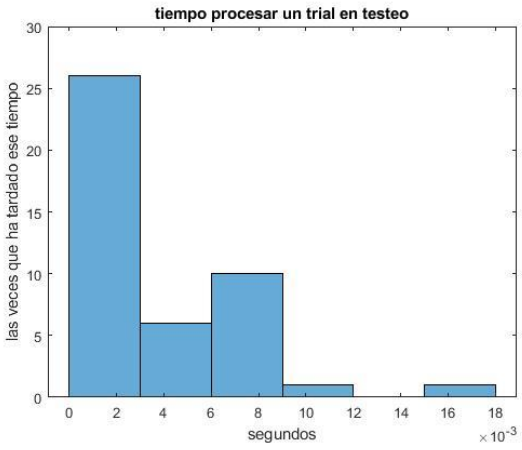


Figura 6-7. Histograma de tiempo en testeo de ensayos en LDA paso a paso.

6.1.1.1.1.2 Prueba con 111 de entrenamiento y 33 de testeo

tiempo máximo un trial entrenamiento: 0.622655 segundos

tiempo máximo un trial test: 0.035691 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 78.899083 %

Tasa predicción (probabilidad acierto conjunto test) : 90.909091 %

Tasa clasificación errónea entrenamiento: 21.100917%

Tasa clasificación errónea test: 9.090909 %

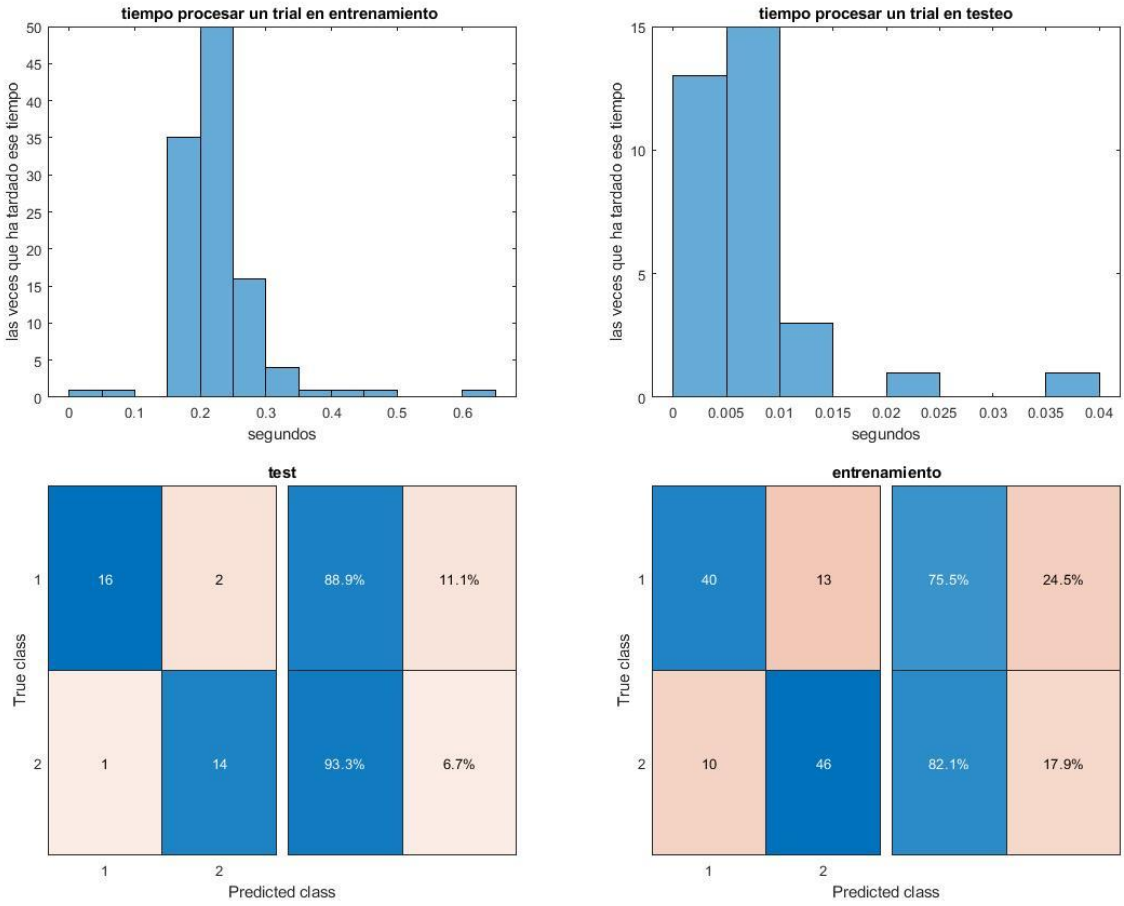


Figura 6-8. Resultados LDA con 111 entrenamiento y 33 de testeo.

Se puede apreciar cómo afecta a la probabilidad de acierto, el número de ensayos que se usa. Ha subido la probabilidad de acierto con respecto a la prueba anterior de 100 ***trials*** de entrenamiento. Sin embargo, en las siguientes pruebas se ha querido seguir usando 100 para entrenamiento y 44 para testeo, ya que fue la condición que se estableció al principio.

6.1.1.1.1.3 Prueba de aplicar testeo a los datos de entrenamiento

Se utiliza los datos de entrenamiento como testeo para comprobar qué resultado da. Como el resultado obtenido anteriormente era de un entrenamiento online, dará un resultado distinto ahora. Para esta comprobación se usa el archivo **mi_script_Isabel_BUFFER_3_testeo.m,** después de haber ejecutado el archivo en que se obtenía los parámetros del clasificador y la matriz de filtros espaciales. Tener presente que los dos primeros datos o más, los cuales había que esperar a tener antes de empezar a clasificar, se están teniendo en cuenta en este testeo.

```
tiempo máx. un trial test:     0.272469 segundos

probabilidad acierto conjunto test: 94.000000 %
```

Da un resultado coherente, la probabilidad sobre los datos con los que se hizo el entrenamiento es mayor a los obtenidos con otros datos distintos (94% > 84%).

6.1.1.1.1.1.4 Prueba cambiando cálculo de covarianza media total (sigma) en LDA

En el LDA realizado se calcula la covarianza media total igualándola a la covarianza de la matriz de características. Se comprobará que sucede, si en lugar de eso, se realiza la suma ponderada de las covarianzas de cada clase, ponderándolas con las probabilidades de cada una, lo cual se programaría en MATLAB del siguiente modo:

```
sum1=0;
sum2=0;
for j=1:i-1
    if TrLabels(j)==1
        a= (TrFeatures(:,j)-mu1);
        sum1=sum1+(a*a');
        %                   size(sum1)%8      8
    else
        a= (TrFeatures(:,j)-mu2);
        sum2=sum2+(a*a');
    end
end
sigma1=(1/n_c1)*sum1;
sigma2=(1/n_c2)*sum2;

pc1=n_c1/(n_c1+n_c2);
pc2=n_c2/(n_c1+n_c2);

sigma=pc1*sigma1+pc2*sigma2;
```

Si se asume clases equiprobables, la versión teórica de la covarianza sería ***sigma=(sigma1+sigma2)/2***, siendo ***sigma1*** la covarianza de las características de clase 1 y ***sigma2*** la de clase 2. La versión que coincide con la que se implementa internamente en MATLAB es ***sigma=cov(TrFeatures')***.

Se obtienen los siguientes resultados:

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 79.591837 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 20.408163%

Tasa clasificación errónea test: 15.909091 %

Sube la probabilidad de acierto en el entrenamiento online, la de testeo permanece igual.

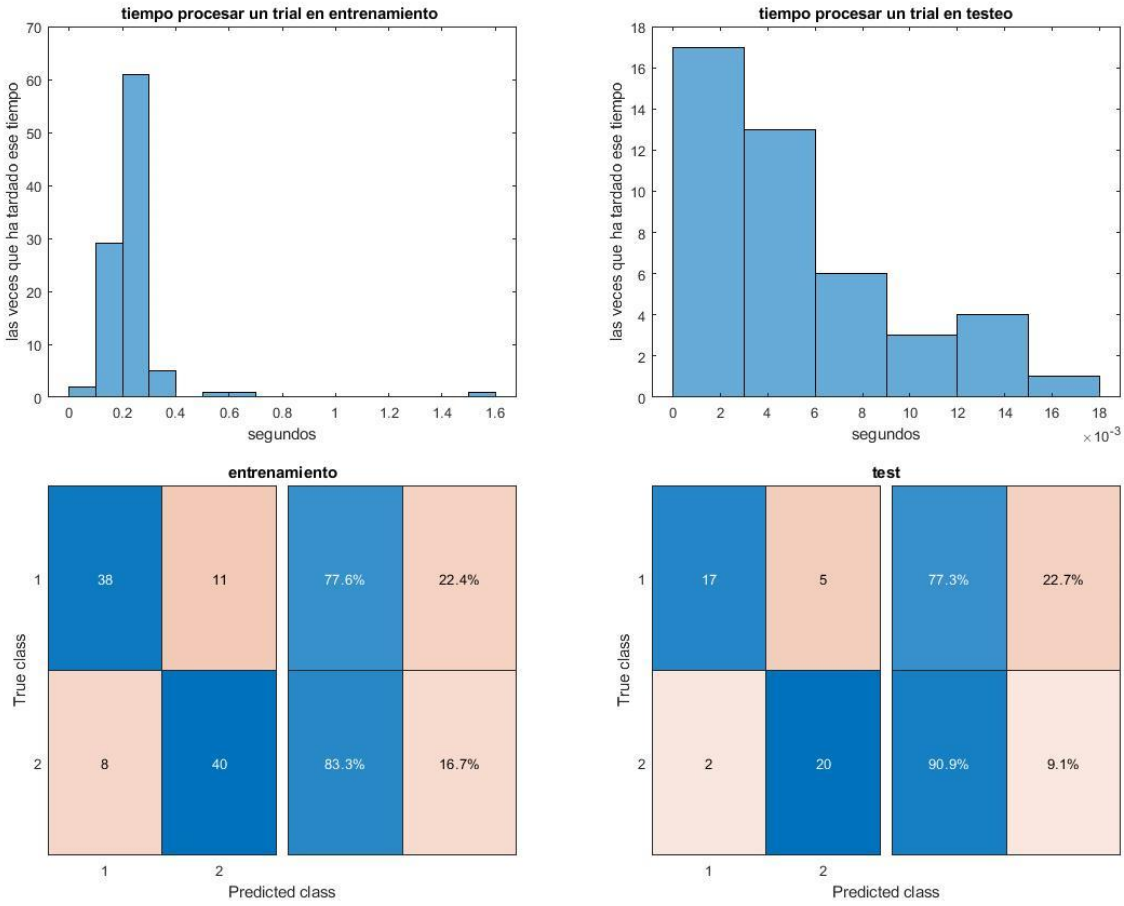


Figura 6-9. Resultados LDA cuando se calcula sigma de manera diferente.

6.1.1.1.2 Pruebas con matriz de filtros W ordenada

6.1.1.1.2.1 Prueba 100 de entrenamiento y 44 de testeo

tiempo máximo un trial entrenamiento: 0.154038 segundos

tiempo máximo un trial test: 0.114536 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 80.612245 %

Tasa predicción (probabilidad acierto conjunto test) : 86.363636 %

Tasa clasificación errónea entrenamiento: 19.387755%

Tasa clasificación errónea test: 13.636364 %

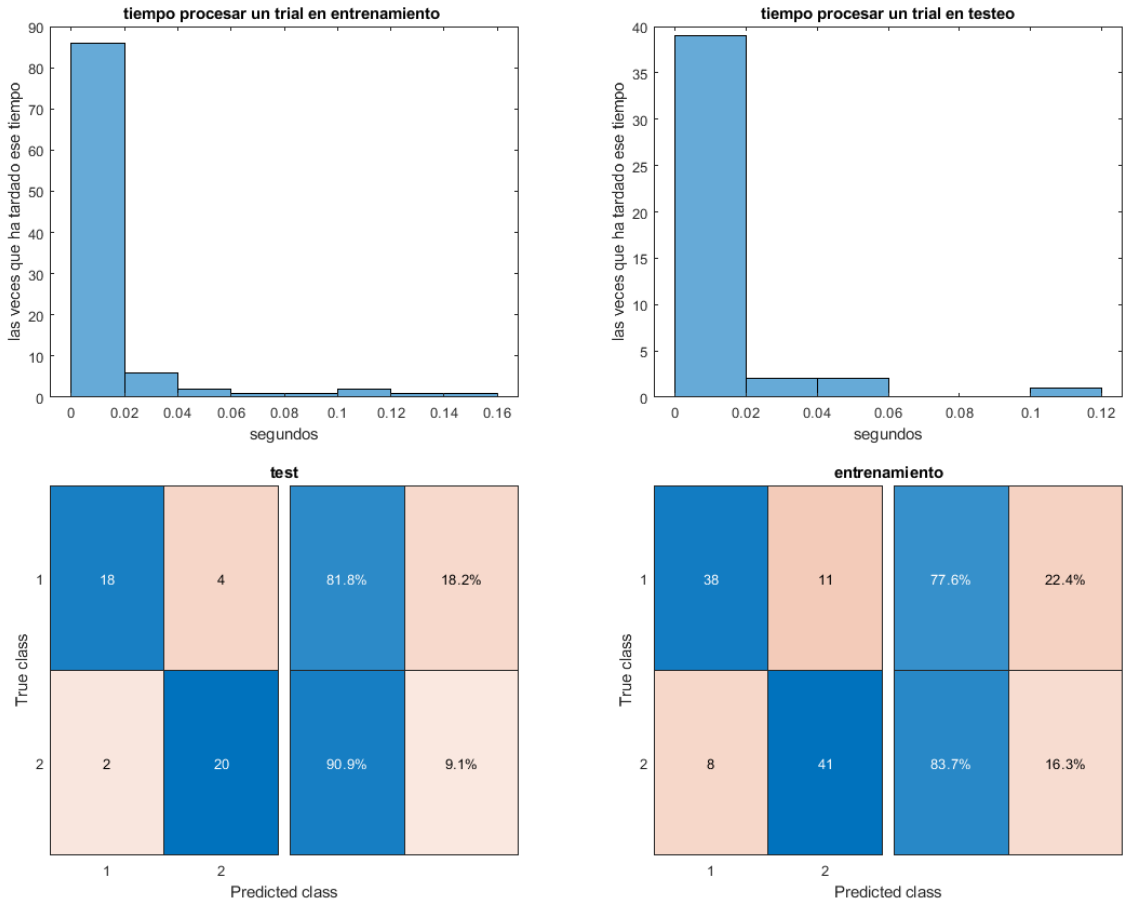


Figura 6-10. Resultados LDA paso a paso 100 entrenamiento y 44 de testeo.

Se comprobó también en el código que, si se quitaba la ordenación de la matriz de filtros **W** antes de aplicar LDA, y solo se establecía al principio, cuando se obtiene la matriz de filtros con los ensayos en los que hay que esperar que haya un mínimo de uno de cada clase, se obtenían los mismos resultados que cuando se fuerza la ordenación en cada iteración. Por otra parte, cuando se quita también la ordenación para hallar **W** en los primeros ensayos (siendo esos resultados los obtenidos en el apartado 6.1.1.1.1.1), empeoran los resultados del clasificador LDA.

6.1.1.1.1.2.2 Cross-validation

También se hizo una adaptación del código LDA anterior para probar a hacer cross-validation con kfolds=5, decidiendo desde el inicio cuáles serían los ensayos de entrenamiento y de testeo del conjunto ya almacenado. Es solo para comprobar la eficacia del clasificador. En este caso se realiza una sola vez la validación, lo ideal sería repetirlo varias veces cogiendo ensayos distintos, para después hacer una media de todos los valores de probabilidad de acierto obtenida, es decir, hacer análisis de Monte Carlo. El análisis de Monte Carlo consiste en hacer varias iteraciones del algoritmo cogiendo diferentes ensayos de entrenamiento y testeo, es decir, para cada sesión y clasificador la prueba se repite un número de veces. Hacer este análisis sería estadísticamente más significativo, pues el conjunto de datos es limitado y es aconsejable probar cada clasificador más de una sola vez, para ver si mejora su eficiencia. En conclusión, si no se hace la clasificación y el testeo en tiempo real, es recomendable usar k-fold cross-validation y análisis de Monte Carlo.

El archivo es **mi_script_Isabel_BUFFER_3_cross_vali.m,** dando de resultado para un usuario:

tiempo máximo un trial entrenamiento: 1.201484 segundos tiempo máximo un trial test: 0.170134 segundos

Tasa predicción online (probabilidad acierto conjunto entrenamiento) : 74.561404 %

Tasa predicción (probabilidad acierto conjunto test) : 92.857143 %

En la carpeta '**basico2**', adjunta con este trabajo, se encuentra la realización del LDA sin ser en tiempo real (offline). En el archivo **MI_BCI_cv9.m** se encuentra como se usa cross-validatión y el análisis de Monte Carlo en MATLAB. Dando resultado de probabilidad acierto final, con varios usuarios, de 77.1 %±3.4 en el caso offline. Hay que ejecutar **MI_BCI_basic9.m** para comprobarlo. Los archivos que contienen el clasificador, la reducción de dimensionalidad y obtención de características offline del LDA son: **mi_lda.m, MI_DimReduction.m, MI_Features.m.** Se usan varias sesiones y usuarios. El resultado final que se obtiene de probabilidad de acierto media de testeo y de cada sesión es:

<Acc>(Classif. 1):exactitud 77.1 Â±3.4 | Acc(user): 85.9 79.6 57.7 57.0 96.1 93.7 76.8 68.0 63.4 58.5 64.9 64.1 93.3 76.4 95.4 93.3 90.5 74.0

6.1.1.1.1.2.3 Probando diferentes números de ensayos de entrenamiento y testeo

Para esta prueba se utiliza el archivo **mi_script_Isabel_BUFFER_3_numeros.m**.

Se observará cómo varía la probabilidad de acierto en entrenamiento online y test según el número de ensayos que se cojan para test en una sesión, siendo el resto para entrenamiento, tener en consideración que son 144 ensayos en total.

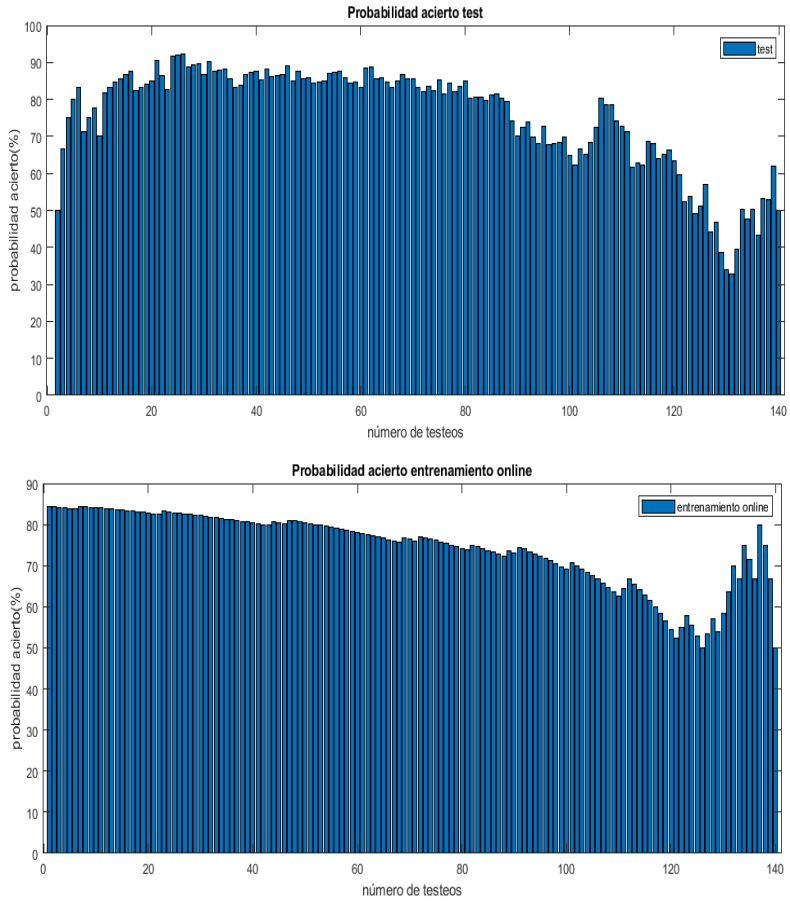


Figura 6-11. Probabilidad de acierto con respecto a número de ensayos cogidos para el testeo.

Se ve en la Figura 6-11 que cuantos más ensayos se cojan para el entrenamiento online, mayor será la probabilidad de acierto en entrenamiento y testeo. Cuando hay muy pocos ensayos para el entrenamiento, los resultados son aleatorios.

Por otra parte, se aprecia que la probabilidad en testeo sube a medida que aumentan los ensayos de entrenamiento (disminuyendo los de testeo) hasta un determinado momento, en el cuál empieza a aparecer probabilidades bajas, debido a que hay pocos ensayos para test, no es un resultado de probabilidad fiable. Tiene que haber una cantidad suficientemente grande tanto para entrenamiento como para test. El objetivo es conseguir una probabilidad de testeo alta y fiable, esta se consigue, por ejemplo, con 26 ensayos para testeo y 118 para entrenamiento, teniendo una probabilidad de 92.31%. Observando los resultados de la gráfica, se deduce que se podría considerar usar valores entre 26 y 46 para testeo aproximadamente.

6.1.1.1.1.2.4 Prueba diferentes valores de limite siendo 100 ensayos entrenamiento y 44 testeo

Se denominará límite al número de ensayos que hay que esperar que se almacene antes de empezar a clasificar. Como ya se comentó, se está considerando que espere a que haya uno de cada clase antes de empezar a clasificar. Si se cambia para que espere más ensayos antes de empezar a clasificar, se obtendrán los resultados de probabilidad que se mostrarán a continuación. Se utiliza el archivo **mi_script_Isabel_BUFFER_3_limit.m** para su obtención.

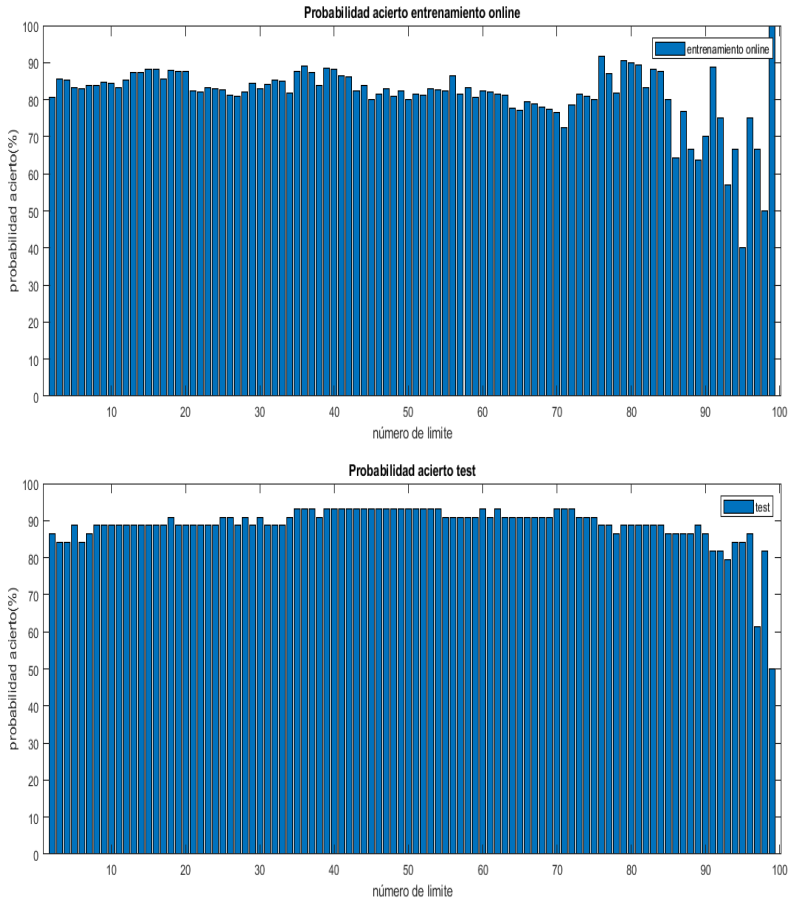


Figura 6-12. Resultados probando varios límites de ensayos en LDA.

Hay que recordar que, la probabilidad de acierto en entrenamiento se está calculando sin tener en cuenta los ensayos que ha habido que esperar que se almacenen al principio, antes de empezar el entrenamiento.

Cada barra que se muestra en la Figura 6-12, representa la probabilidad de acierto desde el caso de esperar a que haya 2 ensayos almacenados (uno de cada clase) hasta el caso de esperar a que haya 99 ensayos almacenados al inicio. Darse cuenta de que se había establecido los 100 primeros ensayos como entrenamiento, así que el límite

no puede ser mayor de 99. Esperando un límite de 39 ensayos se obtiene una probabilidad de acierto en testeo de 93.18%.

Los resultados de probabilidad de acierto en testeo son muy similares, pero cuando se espera a que haya 99, el algoritmo programado no funciona correctamente. Si se esperase a que haya 99 ensayos, los resultados serían:

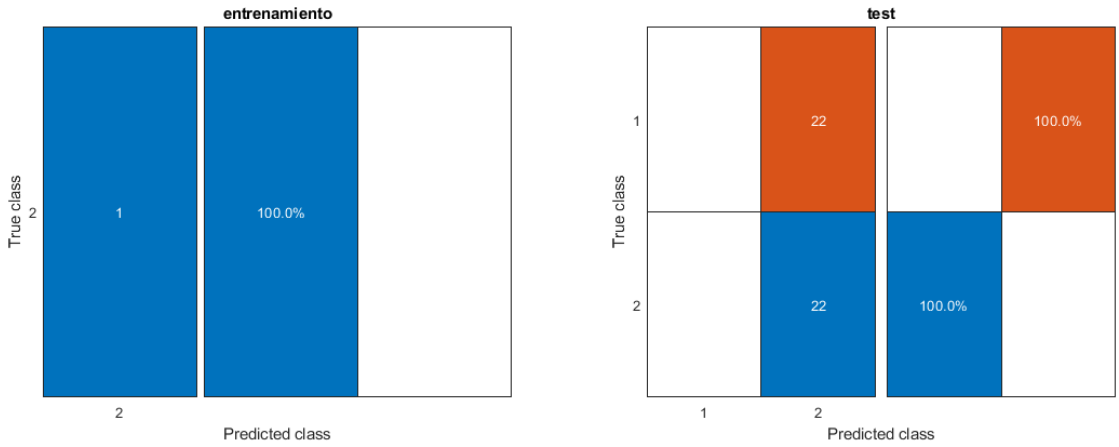
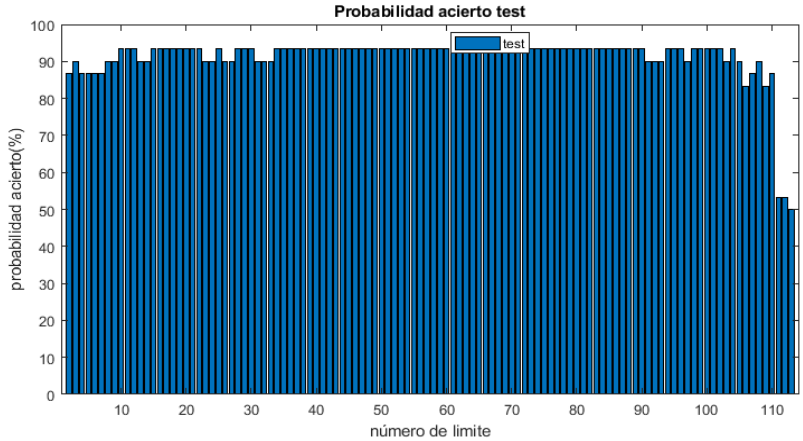


Figura 6-13. Resultados LDA con limit=99.

Cuando entra por primera vez en el cálculo de los parámetros del LDA, tiene almacenado 99 vectores de características. Lo que se muestra en matriz de confusión es el resultado de probabilidad de los que no se han ido almacenando al principio, en este caso solo está estimando clase de un vector de características en el entrenamiento, el último. En el testeo se aprecia que todo lo clasifica como clase 2, la misma clase correspondiente al único que se estimó su clase en el entrenamiento. Sin embargo, todos los vectores de características almacenados se están teniendo en cuenta a la hora de actualizar el clasificador y la matriz de filtros espaciales. El problema presentado puede radicar en la forma en que se está realizando la programación del algoritmo. En el caso de esperar a que haya uno de cada clase, no presenta problemas el algoritmo. Parece que solo presenta problemas cuando el valor del límite es cercano al número establecido como cantidad de ensayos de entrenamiento.

Si se establece 30 ensayos para testeo, 114 para entrenamiento y los límites desde 2 hasta 113, se obtiene:



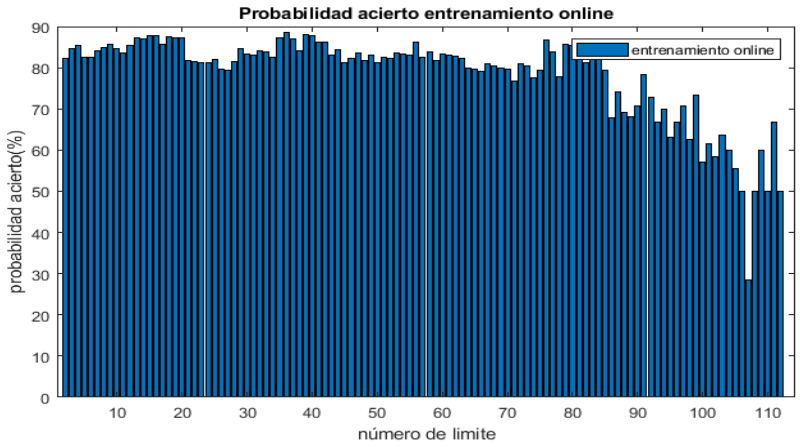


Figura 6-14. Resultados probando varios límites de ensayos en LDA con 114 entrenamiento y 30 de testeo.

6.1.1.1.1.2.5 Prueba diferentes números de filtros

Se prueban diferentes números de filtros y se obtiene la probabilidad de testeo de cada caso. Se usa el archivo **mi_script_Isabel_BUFFER_3_nFilters.m**. En la siguiente gráfica se muestran las probabilidades de acierto en testeo cuando se tiene de 2 a 22 filtros, por tanto, de 2 a 22 características.

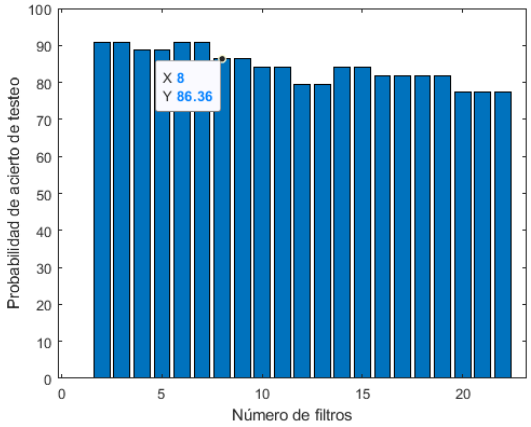


Figura 6-15. Probabilidad de acierto cambiando número de filtros en LDA.

Se aprecia que al aumentar el número de filtros la probabilidad de acierto en test baja. Con 22 filtros se obtenía 77.27%, mientras que con 2 filtros se obtenía 90.91%. En caso de 8 filtros, se ve que efectivamente da la probabilidad de acierto que se obtuvo con 100 de entrenamiento y 44 de testeo (apartado 6.1.1.1.1.2.1), ya que en este proyecto se decidió usar 8 filtros.

6.1.1.1.1.2.6 Prueba de aplicar testeo a los datos de entrenamiento

Se usa el archivo **mi_script_Isabel_BUFFER_3_testeo.m** .

```
tiempo máx. un trial test:     0.115323 segundos

probabilidad acierto conjunto test: 95.000000  %
```

Da un resultado coherente, la probabilidad sobre los datos con los que se hizo el entrenamiento es mayor a los obtenidos con otros datos distintos (95% > 86%).

6.1.1.1.1.2.7 Prueba cambiando cálculo de covarianza media total (sigma) en LDA

Se obtienen los siguientes resultados:

tiempo máximo un trial entrenamiento: 0.135649 segundos

tiempo máximo un trial test: 0.076044 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 77.551020 %

Tasa predicción (probabilidad acierto conjunto test) : 86.363636 %

Tasa clasificación errónea entrenamiento: 22.448980%

Tasa clasificación errónea test: 13.636364 %

La probabilidad de acierto en entrenamiento online ha bajado, la de testeo ha permanecido igual.

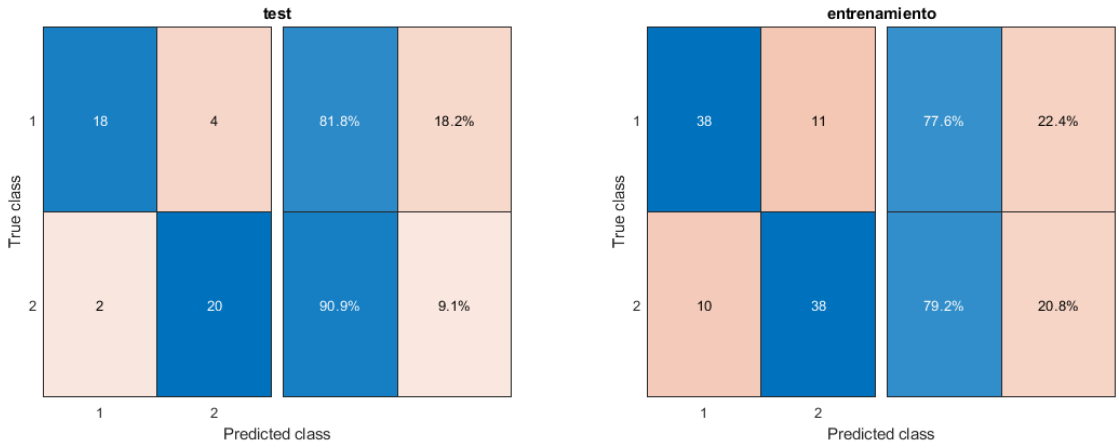


Figura 6-16. Resultados LDA cuando se calcula sigma de manera diferente.

6.1.1.1.1.2.8 Prueba recalculando características

Al ser un entrenamiento online, se estableció que reutilizase cálculos obtenidos anteriormente para reducir coste computacional. Por ejemplo, reutilizar las características calculadas anteriormente para hallar los parámetros del LDA.

El hecho de no recalcular los valores de las características implica que se están teniendo en cuenta los datos obtenidos en los primeros ensayos, los cuales no eran buenos. Se poseían pocos datos de entrenamiento, así que la **W** inicial no era aceptable. Se va a comprobar que pasa si se recalculan todas las características en cada iteración. Para su realización se usa el archivo **mi_script_Isabel_BUFFER_3_recalculando.m**.

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 82.653061 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 17.346939%

Tasa clasificación errónea test: 15.909091 %

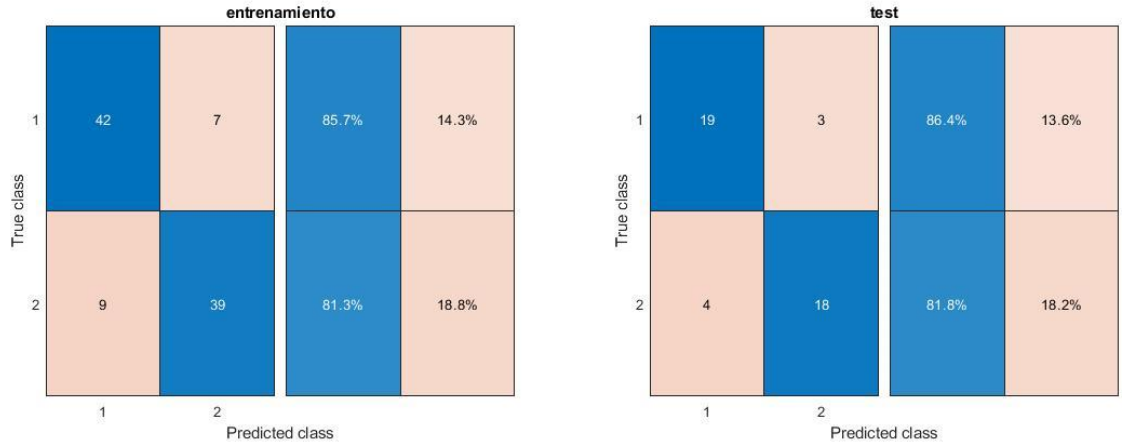


Figura 6-17. Resultados en LDA si se pone que recalcule todas características cada vez que se actualice **Wo**.

Se puede apreciar que la matriz de confusión ha cambiado un poco con respecto al LDA sin recalcular. La probabilidad de acierto media de entrenamiento online ha subido, pero la de testeo no. Se supone que debería subir la probabilidad de testeo, puede ser que haya un error en el código realizado para que recalcule las características en cada iteración.

6.1.1.1.1.3 Prueba realizada sin aplicar CSP

Se va a comprobar los resultados que daría sin aplicar la reducción de dimensión CSP. Para ello se establece, como ya se mencionó, que la matriz **W** es la matriz identidad de tamaño *22x22*, ya que son 22 canales. Para realizar esto en MATLAB simplemente es usar el comando **eye().** Los resultados obtenidos son los siguientes:

```
tiempo máximo un trial entrenamiento:     0.588221 segundos

tiempo máximo un trial test:     0.044372 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 69.387755  %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182  %



Tasa clasificación errónea entrenamiento: 30.612245%

Tasa clasificación errónea test: 18.181818 %
```

La probabilidad de acierto en testeo es de 81.81%, cuando se realizaba con CSP daba 84.09%. La probabilidad de acierto ha bajado tanto en testeo como entrenamiento.

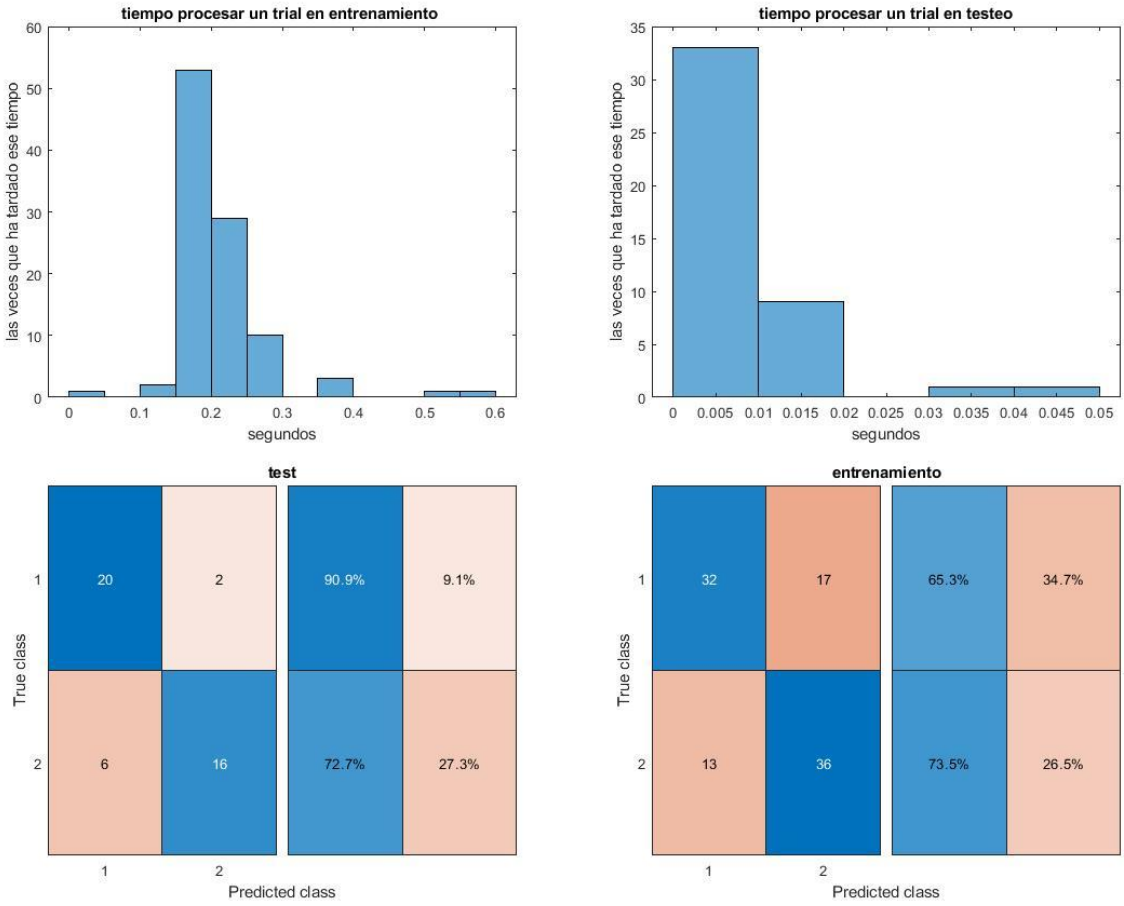


Figura 6-18. Resultados LDA paso a paso sin CSP.

6.1.1.1.2 Aprovechando propiedades de MATLAB

6.1.1.1.2.1 Pruebas sin ordenar matriz de filtros W

6.1.1.1.2.1.1 Lineal

Se considera que todas las clases tienen la misma matriz de covarianza.

p_acierto: 0.7 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 2.350708 segundos

tiempo máximo un trial test: 0.114342 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento): 88.659794 %

Tasa predicción (probabilidad acierto conjunto test): 81.818182 %

Tasa clasificación errónea entrenamiento: 11.340206%

Tasa clasificación errónea test: 18.181818 %

Training Error (perdida por sustitución en el modelo): 0.080808

Test Error (perdida basada en la distribución de datos): 0.18228

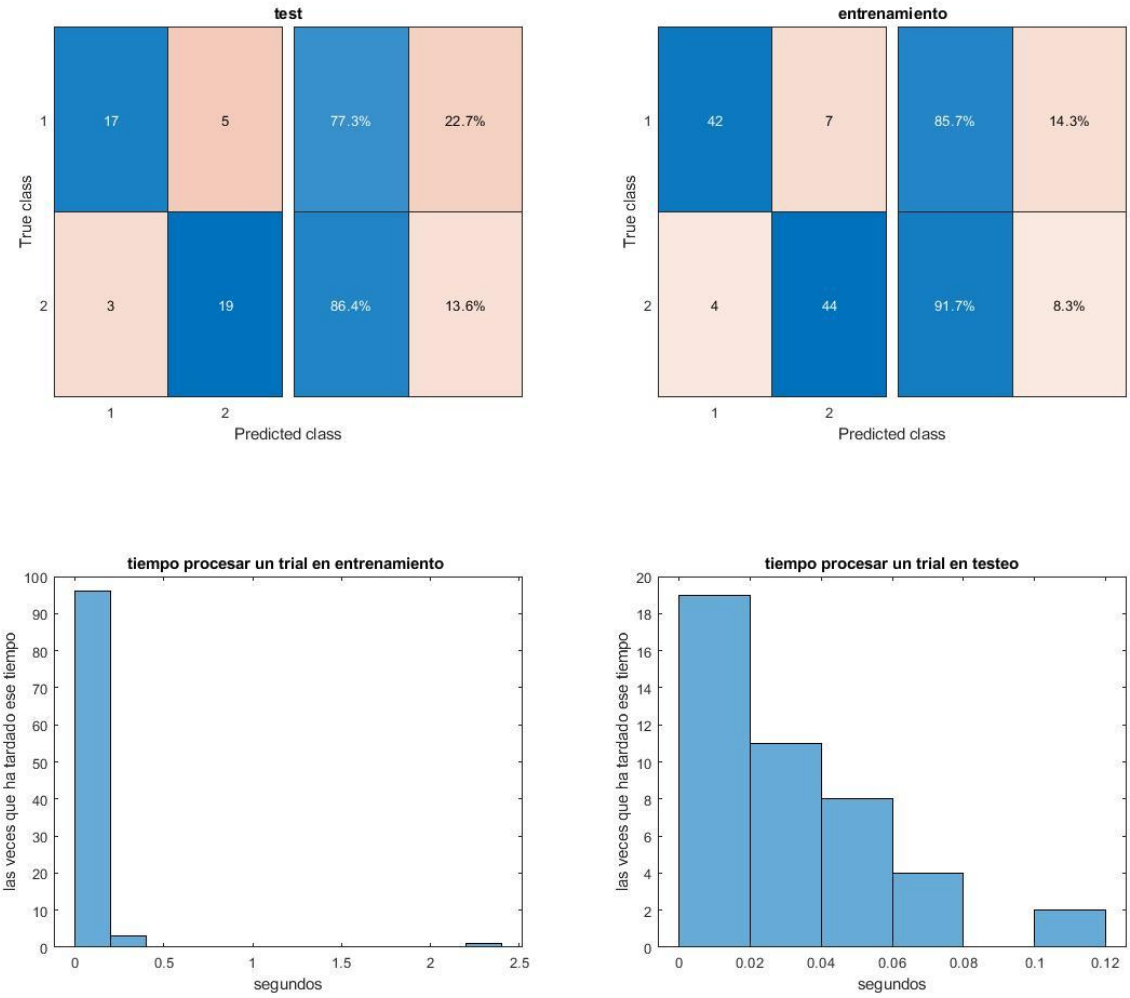


Figura 6-19. Gráficas resultado LDA lineal.

El modelo que realiza MATLAB da mejores resultados en probabilidad de acierto, en comparación con el del apartado 6.1.1.1.1. Sin embargo, puede llegar a ser más lento el procesamiento de un ensayo. De todas formas, el tiempo que tarda tampoco es demasiado grande, teniendo en cuenta lo que se quiere realizar.

6.1.1.1.2.1.1.1 Prueba de aplicar testeo a los datos de entrenamiento

Para testear con datos de entrenamiento, los clasificadores hechos con las funciones de MATLAB, se usa **mi_script_Isabel_BUFFER_machine_funciones_testeo.m**. El resultado que se obtuvo fue:

tiempo máximo un trial test: 0.494128 segundos

Tasa predicción (probabilidad acierto conjunto test) : 96.000000 %

Tasa clasificación errónea test: 4.000000 %

Test Error (perdida basada en la distribución de datos): 0.040202

Un 96 % es mayor al 81.81% de probabilidad de acierto de testeo con datos desconocidos. El resultado es coherente.

6.1.1.1.2.1.2 Diaglinear

Todas las clases se considera que tienen la misma matriz de covarianza diagonal.

tiempo máximo un trial entrenamiento: 7.319140 segundos

tiempo máximo un trial test: 0.071717 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 60.824742 %

Tasa predicción (probabilidad acierto conjunto test) : 88.636364 %

Tasa clasificación errónea entrenamiento: 39.175258%

Tasa clasificación errónea test: 11.363636 %

Training Error (perdida por sustitución en el modelo): 0.15152

Test Error (perdida basada en la distribución de datos): 0.11295

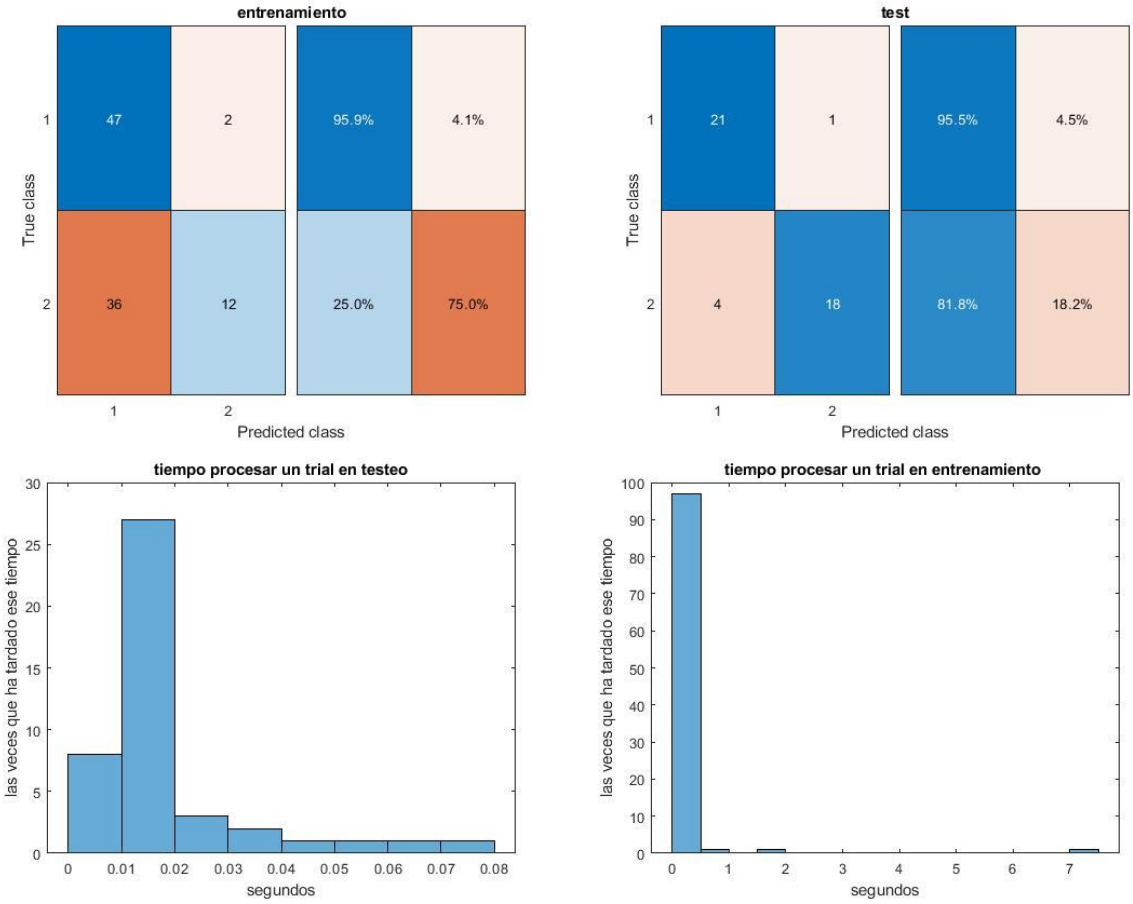


Figura 6-20. Gráficas resultado LDA diaglineal.

6.1.1.1.2.1.3 Pseudolinear

En este caso se considera que todas las clases tienen la misma matriz de covarianza. El software de MATLAB invertirá la matriz de covarianza utilizando la pseudoinversa.

tiempo máximo un trial entrenamiento: 1.824277 segundos

tiempo máximo un trial test: 0.102855 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 87.628866 %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182 %

Tasa clasificación errónea entrenamiento: 12.371134%

Tasa clasificación errónea test: 18.181818 %

Training Error (perdida por sustitución en el modelo): 0.080808

Test Error (perdida basada en la distribución de datos): 0.18228

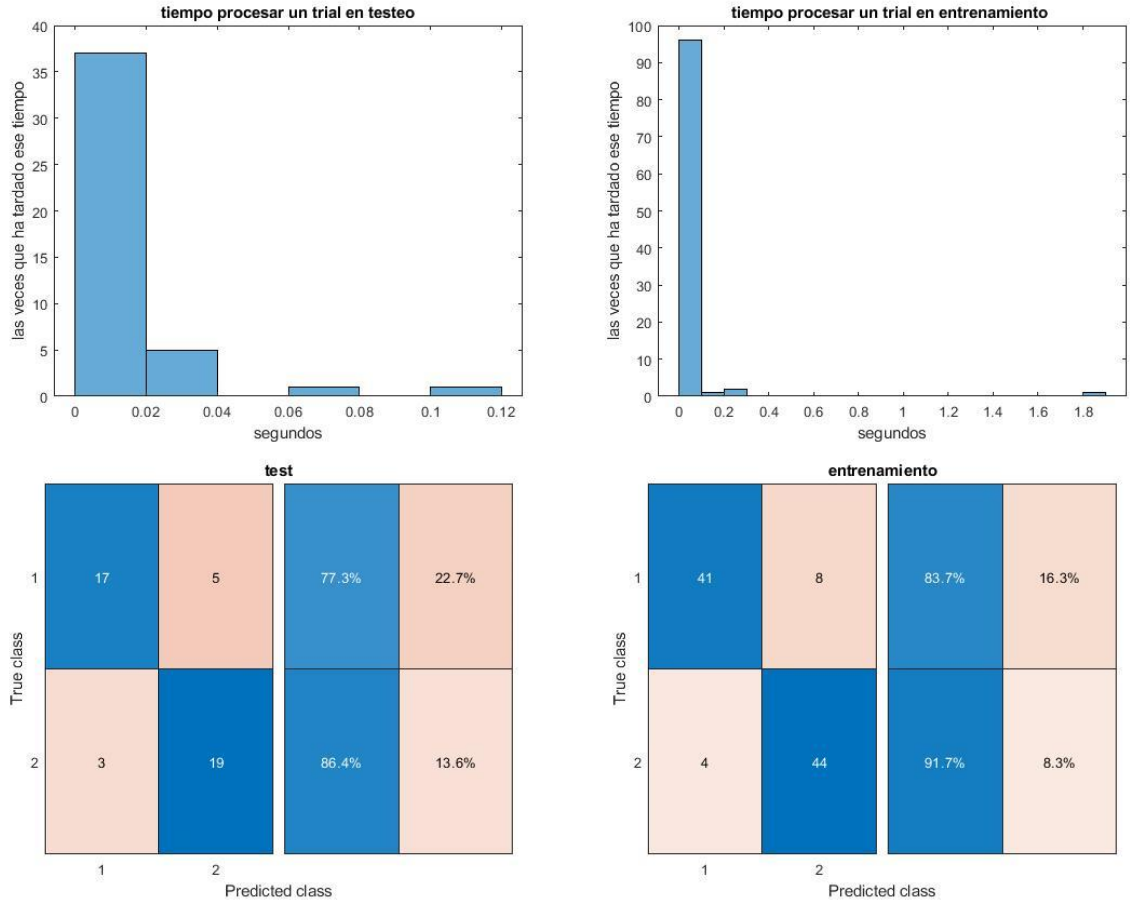


Figura 6-21. Gráficas resultado LDA pseudolineal.

6.1.1.1.2.2 Pruebas con matriz de filtros W ordenada

6.1.1.1.2.2.1 Lineal

Se considera que todas las clases tienen la misma matriz de covarianza.

tiempo máximo un trial entrenamiento: 4.417390 segundos

tiempo máximo un trial test: 0.016149 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 84.536082 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 15.463918%

Tasa clasificación errónea test: 15.909091 %

Training Error (perdida por sustitución en el modelo): 0.080808

Test Error (perdida basada en la distribución de datos): 0.15932

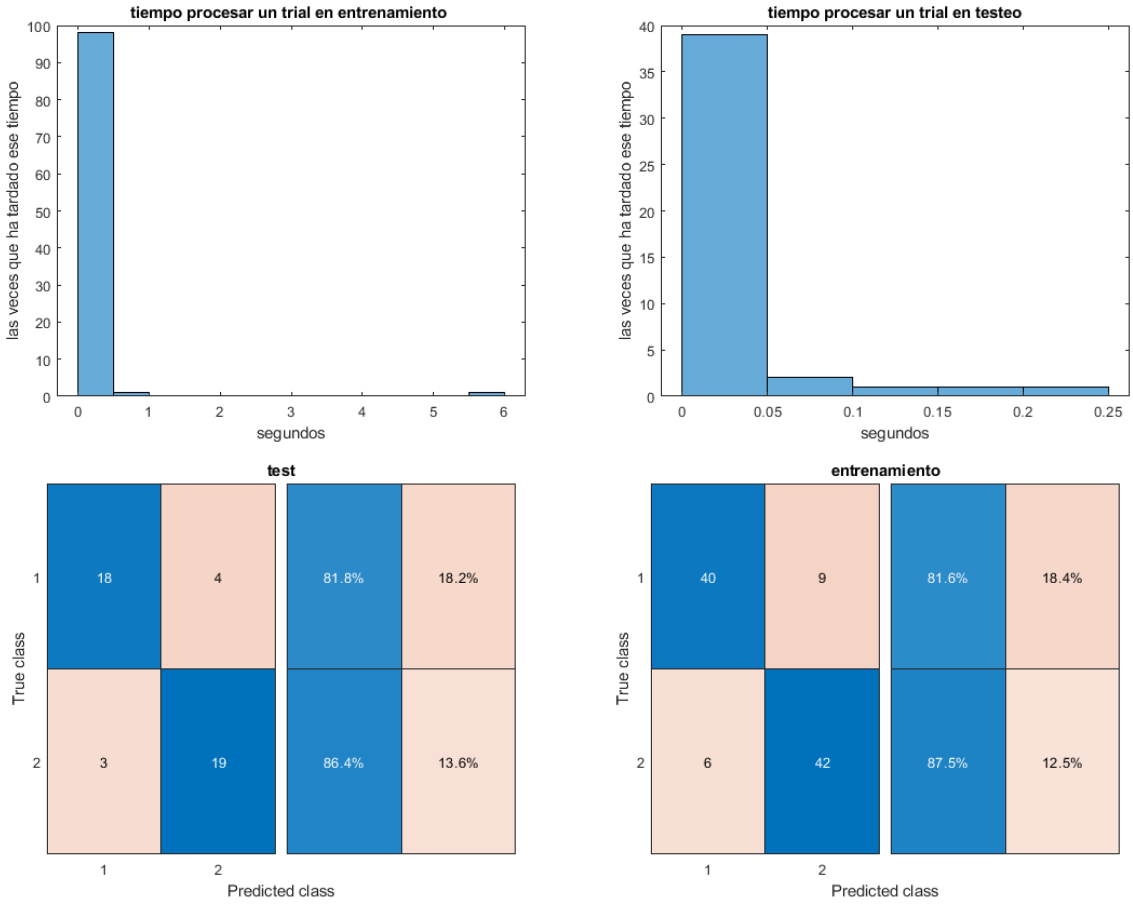


Figura 6-22. Gráficas resultado LDA lineal.

6.1.1.1.2.2.1.1 Prueba de aplicar testeo a los datos de entrenamiento

Se usa **mi_script_Isabel_BUFFER_machine_funciones_testeo.m**. El resultado que se obtiene es:

tiempo máximo un trial test: 0.207383 segundos

Tasa predicción (probabilidad acierto conjunto test) : 95.000000 %

Tasa clasificación errónea test: 5.000000 %

Test Error (perdida basada en la distribución de datos): 0.050303

Un 95 % es mayor al 84% del testeo con datos desconocidos.

6.1.1.1.2.2.2 Diaglinear

Todas las clases se considera que tienen la misma matriz de covarianza diagonal.

tiempo máximo un trial entrenamiento: 2.176920 segundos

tiempo máximo un trial test: 0.051411 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 61.855670 %

Tasa predicción (probabilidad acierto conjunto test) : 88.636364 %

Tasa clasificación errónea entrenamiento: 38.144330%

Tasa clasificación errónea test: 11.363636 %

Training Error (perdida por sustitución en el modelo): 0.15152

Test Error (perdida basada en la distribución de datos): 0.11295

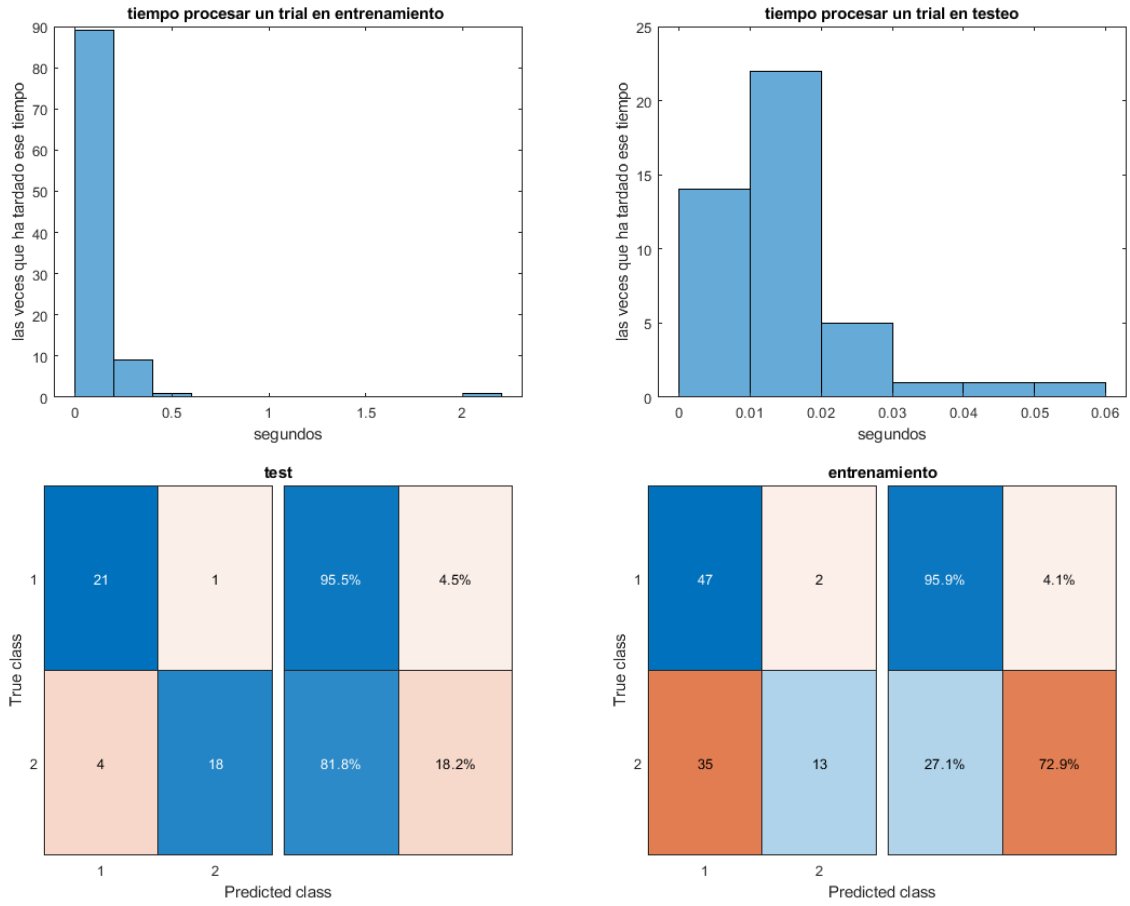


Figura 6-23. Gráficas resultado LDA diaglineal.

6.1.1.1.2.2.3 Pseudolinear

En este caso se considera que todas las clases tienen la misma matriz de covarianza. El software de MATLAB invertirá la matriz de covarianza utilizando la pseudoinversa.

tiempo máximo un trial entrenamiento: 2.087461 segundos

tiempo máximo un trial test: 0.097826 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 83.505155 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 16.494845%

Tasa clasificación errónea test: 15.909091 %

Training Error (perdida por sustitución en el modelo): 0.080808

Test Error (perdida basada en la distribución de datos): 0.15932

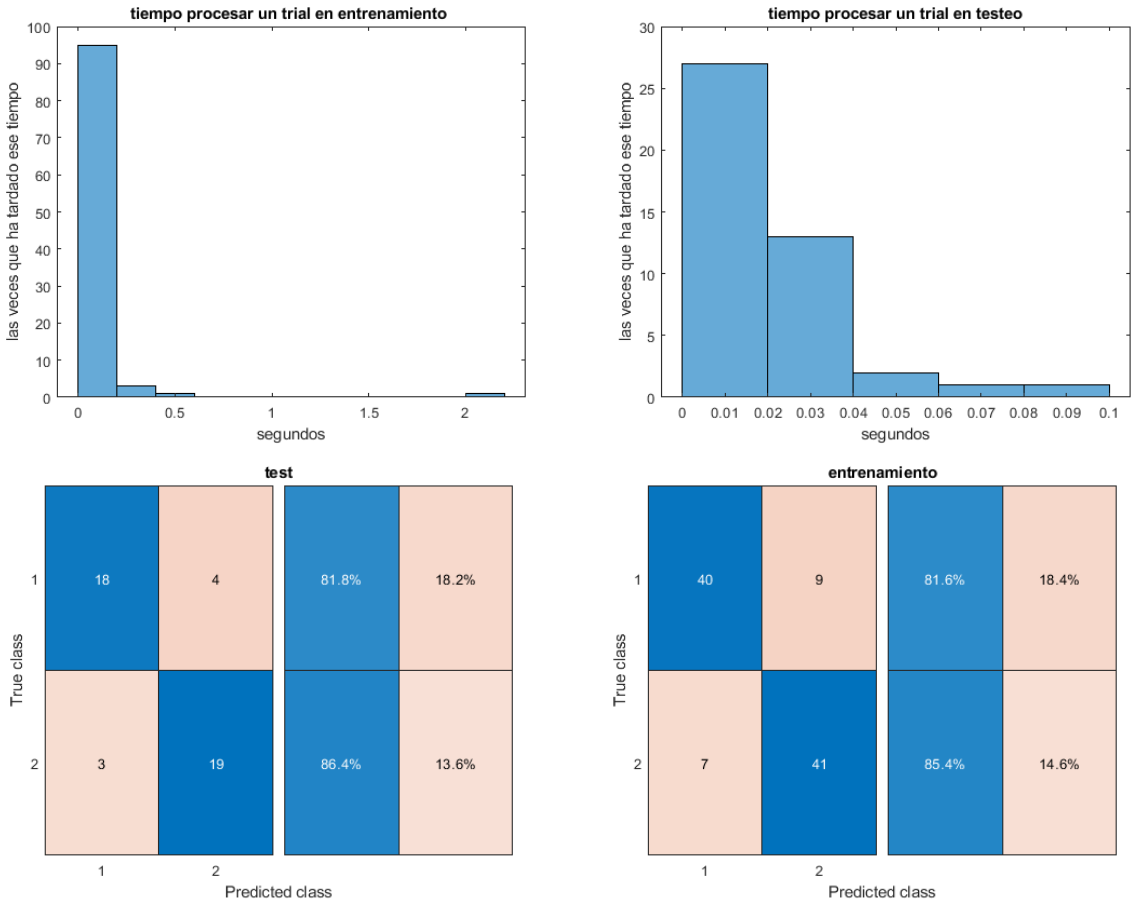


Figura 6-24. Gráficas resultado LDA pseudolineal.

6.1.1.1.3 Usando App de MATLAB

6.1.1.1.3.1 Pruebas sin ordenar matriz de filtros W

Como se mencionó anteriormente, se utilizó CSP antes de pasar los datos a la aplicación. Usando 100 ***trials*** de entrenamiento y cross-validation, en la aplicación se obtiene:

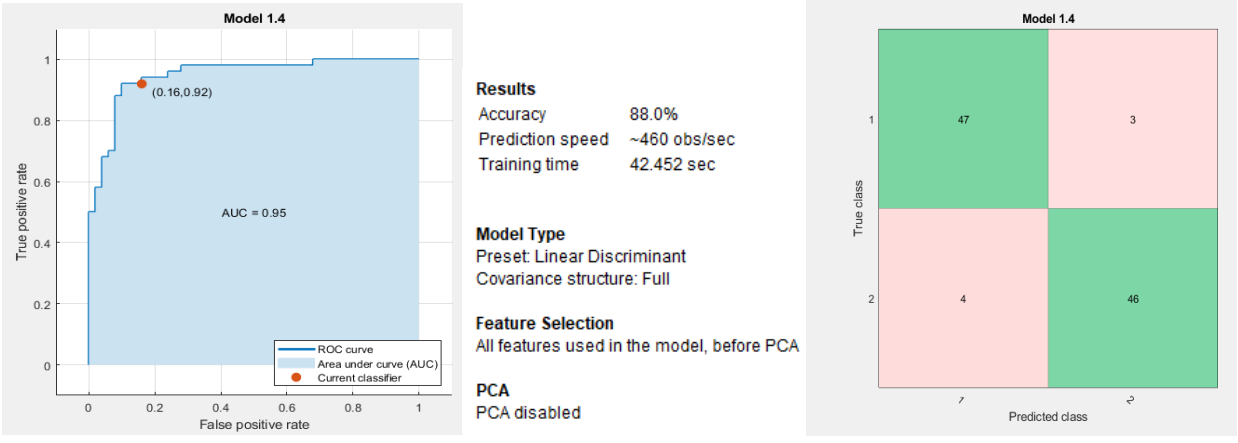


Figura 6-25. Matriz de confusión (a la derecha) usando App y ROC (a la izquierda) en LDA.

Si se ve cómo se clasifican solo las dos primeras características:

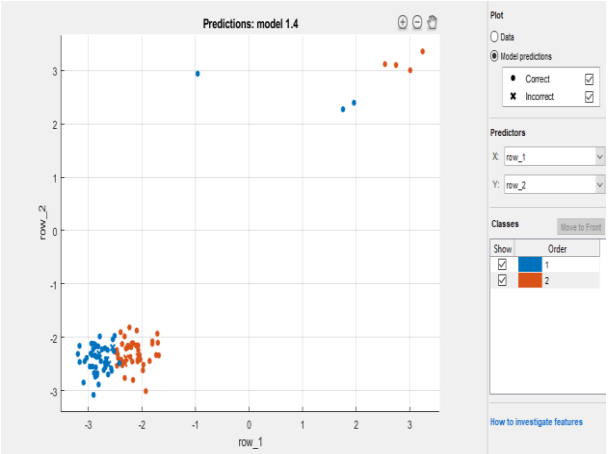


Figura 6-26. Representación de dos características usando la App de MATLAB.

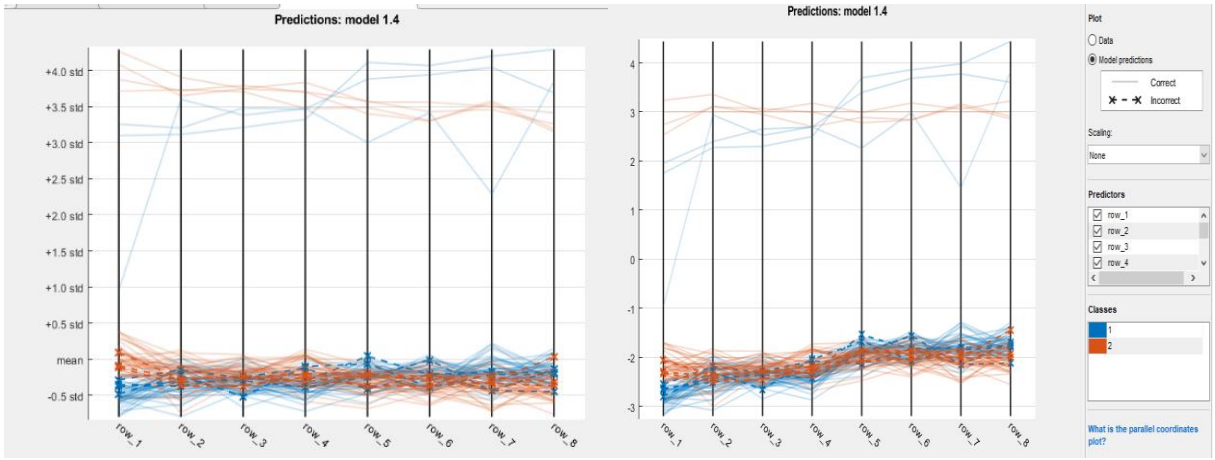


Figura 6-27. Valores de las dos primeras características y comprobando cuáles se han predicho correctamente.

Después se exportó el algoritmo del clasificador para poder entrenarlo de manera online. Los resultados después de ejecutar el código de entrenamiento online, usando este clasificador exportado, especificando 100 de entrenamiento y 44 de testeo:

tiempo máximo un trial entrenamiento: 12.037645 segundos

tiempo máximo un trial test: 0.169305 segundos

validationAccuracy final : 0.909091 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 85.416667 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 14.583333%

Tasa clasificación errónea test: 15.909091 %

Tener presente que ***validationAccuracy*** es la precisión del modelo de clasificación, el resultado de probabilidad de acierto usando cross-validation; la cual se calcula internamente en el código del clasificador exportado. El resultado de probabilidad de acierto mostrado es el último que se obtuvo, el de la última vez que se llamó a función de entrenamiento exportada. En otras palabras, es el resultado de aplicar cross-validation a los 100 ***trials*** que hay especificados para el entrenamiento, obtenidos de manera online. En cambio, la tasa de predicción de testeo se refiere a la obtenida probando el clasificador en los 44 ***trials*** especificados para testeo.

Se aprecia que con cross-validation se obtiene un mayor resultado de probabilidad de acierto, en comparación con el otro método de testeo que se ha utilizado.

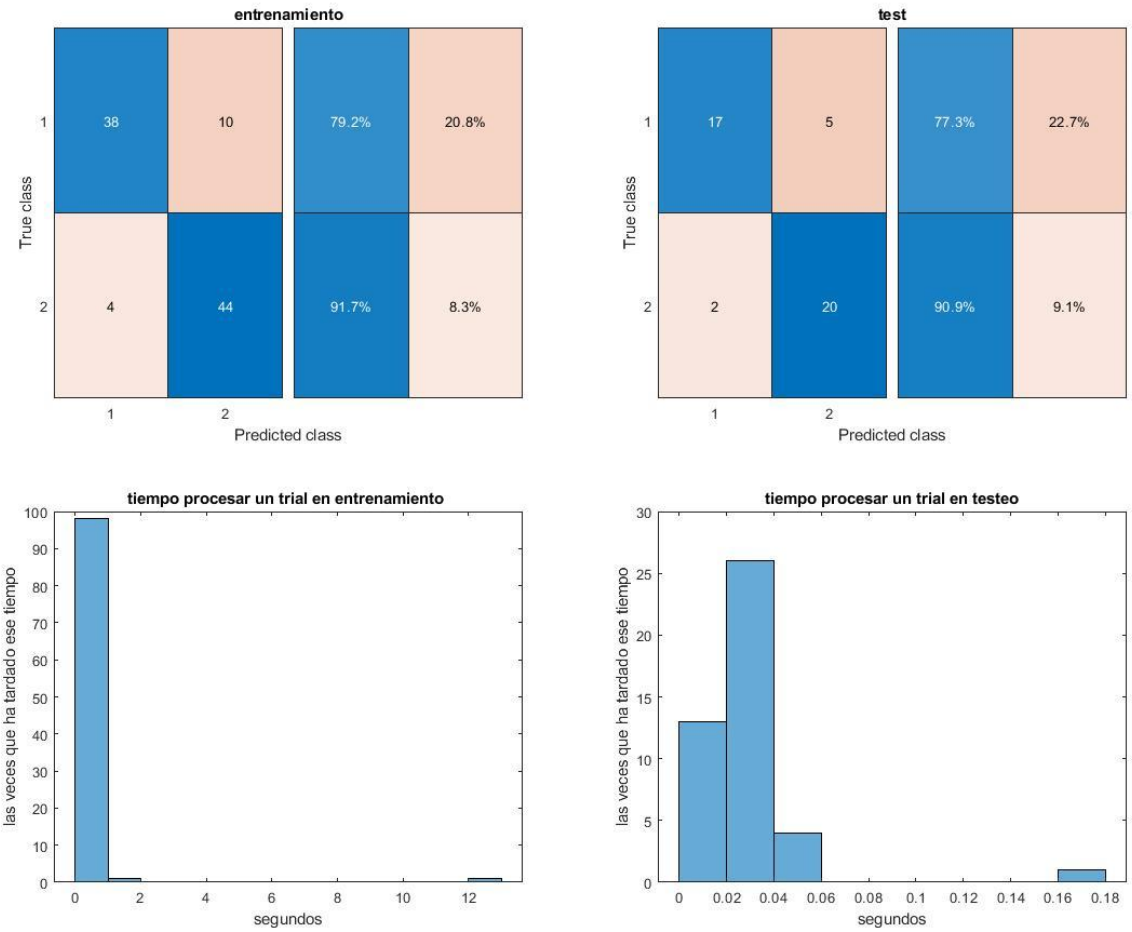


Figura 6-28. Gráficas resultado LDA App.

6.1.1.1.3.2 Pruebas con matriz de filtros W ordenada

tiempo máximo un trial entrenamiento: 8.461646 segundos

tiempo máximo un trial test: 0.141368 segundos

validationAccuracy final : 0.909091 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 85.416667 %

Tasa predicción (probabilidad acierto conjunto test) : 86.363636 %

Tasa clasificación errónea entrenamiento: 14.583333%

Tasa clasificación errónea test: 13.636364 %

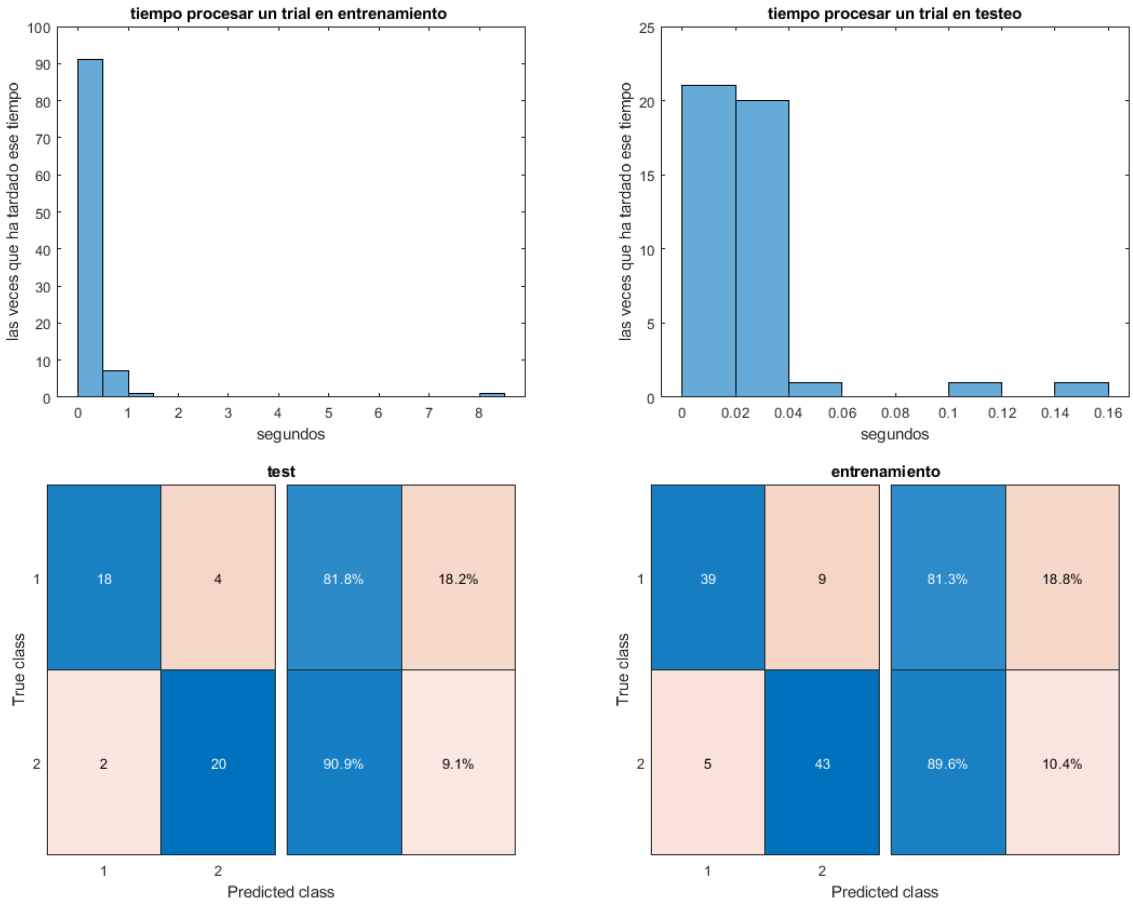


Figura 6-29. Gráficas resultado LDA App.

6.1.1.2 QDA

6.1.1.2.1 Aprovechando propiedades de MATLAB

6.1.1.2.1.1 Pruebas sin ordenar matriz de filtros W

En este clasificador se considera que las matrices de covarianza pueden variar entre clases.

tiempo máximo un trial entrenamiento: 2.181506 segundos

tiempo máximo un trial test: 0.125958 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 76.250000 %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182 %

Tasa clasificación errónea entrenamiento: 23.750000%

Tasa clasificación errónea test: 18.181818 %

Training Error (perdida por sustitución en el modelo): 0.060606

Test Error (perdida basada en la distribución de datos): 0.18228

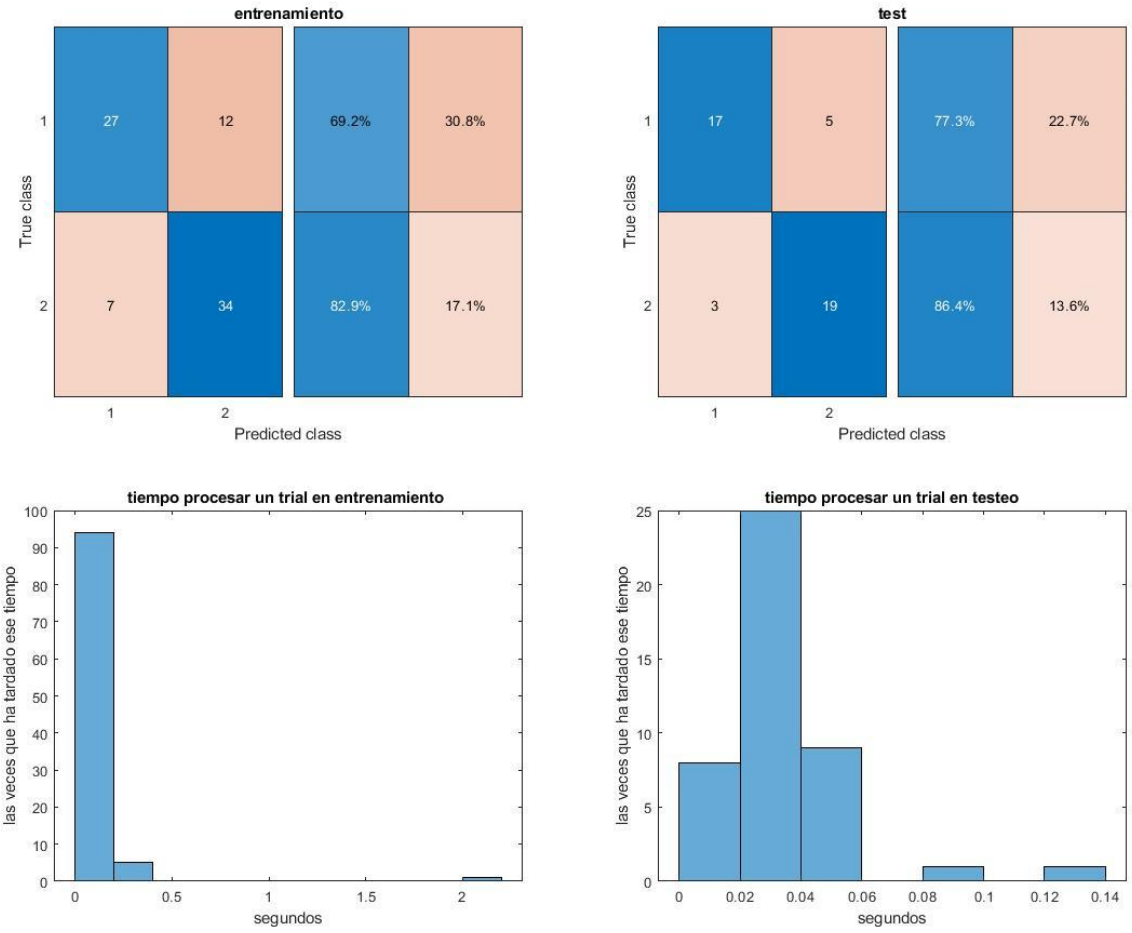


Figura 6-30. Gráficas resultado QDA.

6.1.1.2.1.2 Pruebas con matriz de filtros W ordenada

tiempo máximo un trial entrenamiento: 5.741397 segundos

tiempo máximo un trial test: 0.082648 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 76.250000 %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182 %

Tasa clasificación errónea entrenamiento: 23.750000%

Tasa clasificación errónea test: 18.181818 %

Training Error (perdida por sustitución en el modelo): 0.060606

Test Error (perdida basada en la distribución de datos): 0.18228

La probabilidad de acierto no se ha visto afectada por ordenar matriz de filtros.

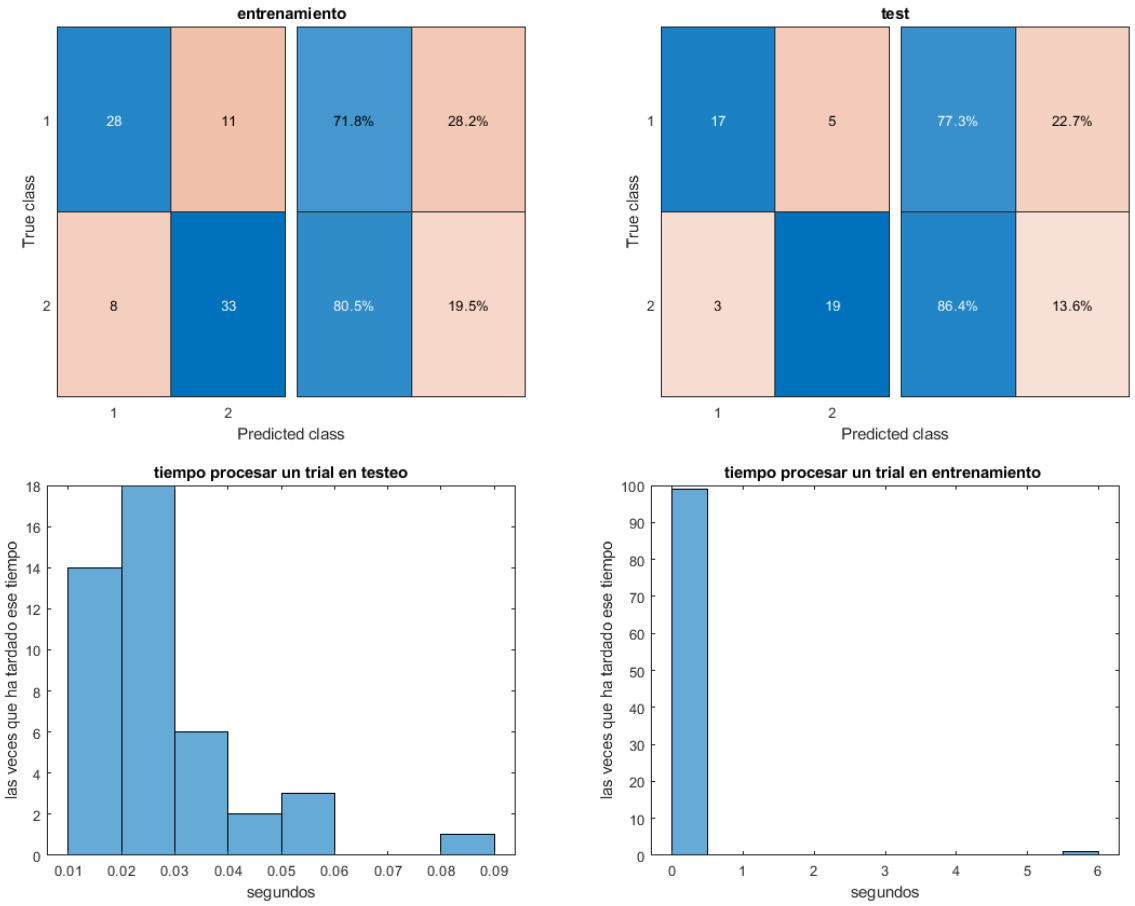


Figura 6-31. Gráficas resultado QDA.

6.1.1.2.2 Usando App de MATLAB

6.1.1.2.2.1 Pruebas sin ordenar matriz de filtros W

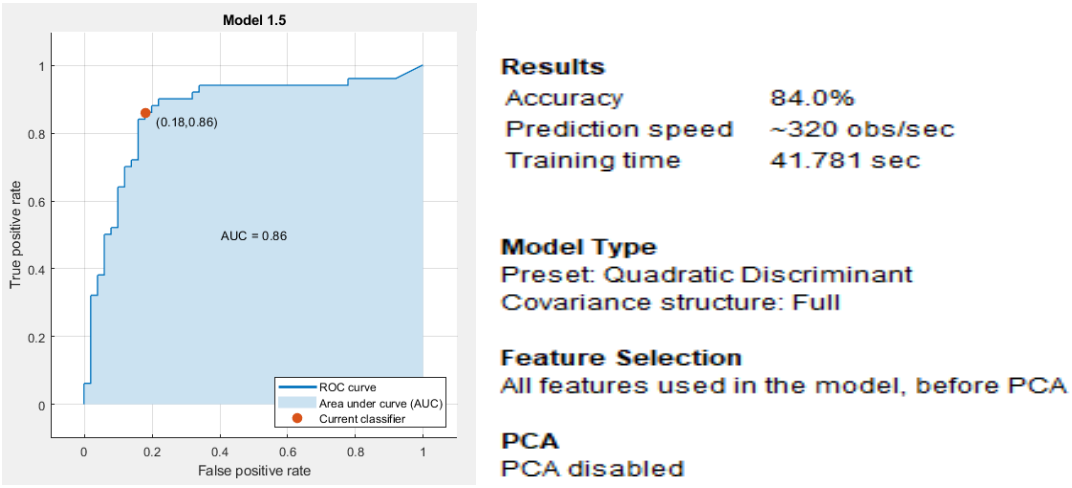


Figura 6-32. Curva ROC QDA.

p_acierto: 0.7 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 5.853773 segundos

tiempo máximo un trial test: 0.357311 segundos

validationAccuracy final: 0.878788 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 72.727273 %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182 %

Tasa clasificación errónea entrenamiento: 27.272727%

Tasa clasificación errónea test: 18.181818 %

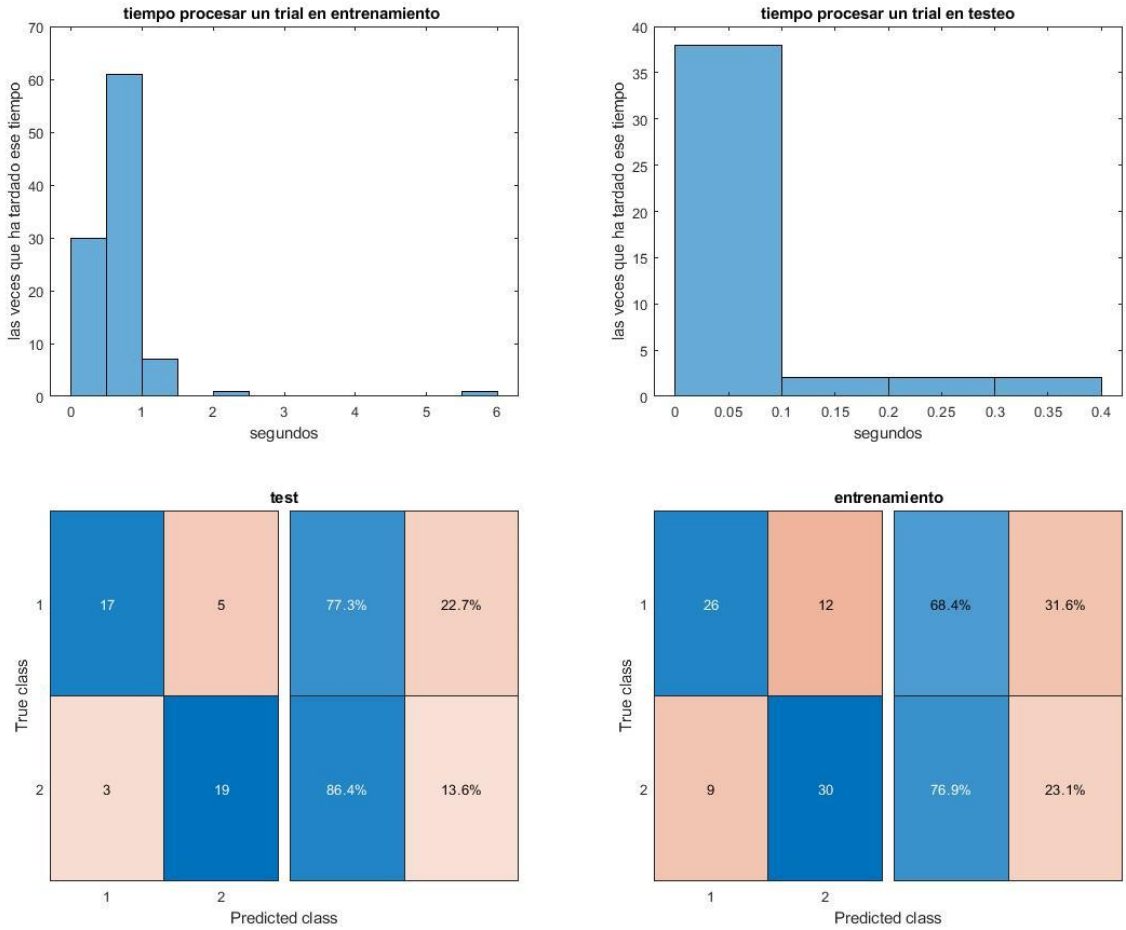


Figura 6-33.Gráficas resultado QDA App.

6.1.1.2.2.2 Pruebas con matriz de filtros W ordenada

tiempo máximo un trial entrenamiento: 13.428240 segundos

tiempo máximo un trial test: 0.309500 segundos

validationAccuracy final : 0.868687 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 74.025974 %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182 %

Tasa clasificación errónea entrenamiento: 25.974026%

Tasa clasificación errónea test: 18.181818 %

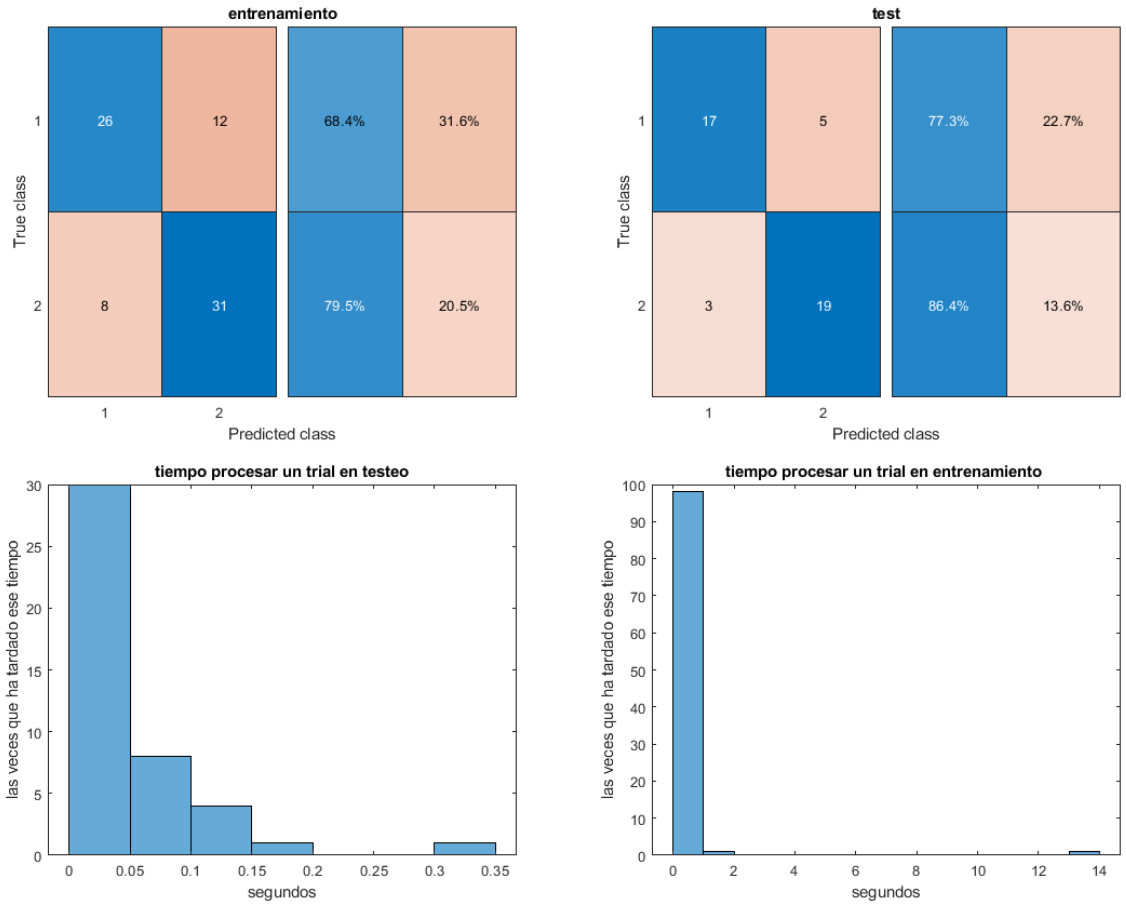


Figura 6-34. Gráficas resultado QDA App.

6.1.2 Bayes

6.1.2.1 Código realizado paso por paso

6.1.2.1.1 Pruebas sin ordenar matriz de filtros W

Se tomó 100 ***trials*** para entrenamiento y 44 de testeo:

p_acierto: 0.6 | acierto = 0 | acierto_previo = 0

tiempo máximo un trial entrenamiento: 0.564049 segundos

tiempo máximo un trial test: 0.090539 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 55.102041 %

Tasa predicción (probabilidad acierto conjunto test) : 65.909091 %

Tasa clasificación errónea entrenamiento: 44.897959%

Tasa clasificación errónea test: 34.090909 %

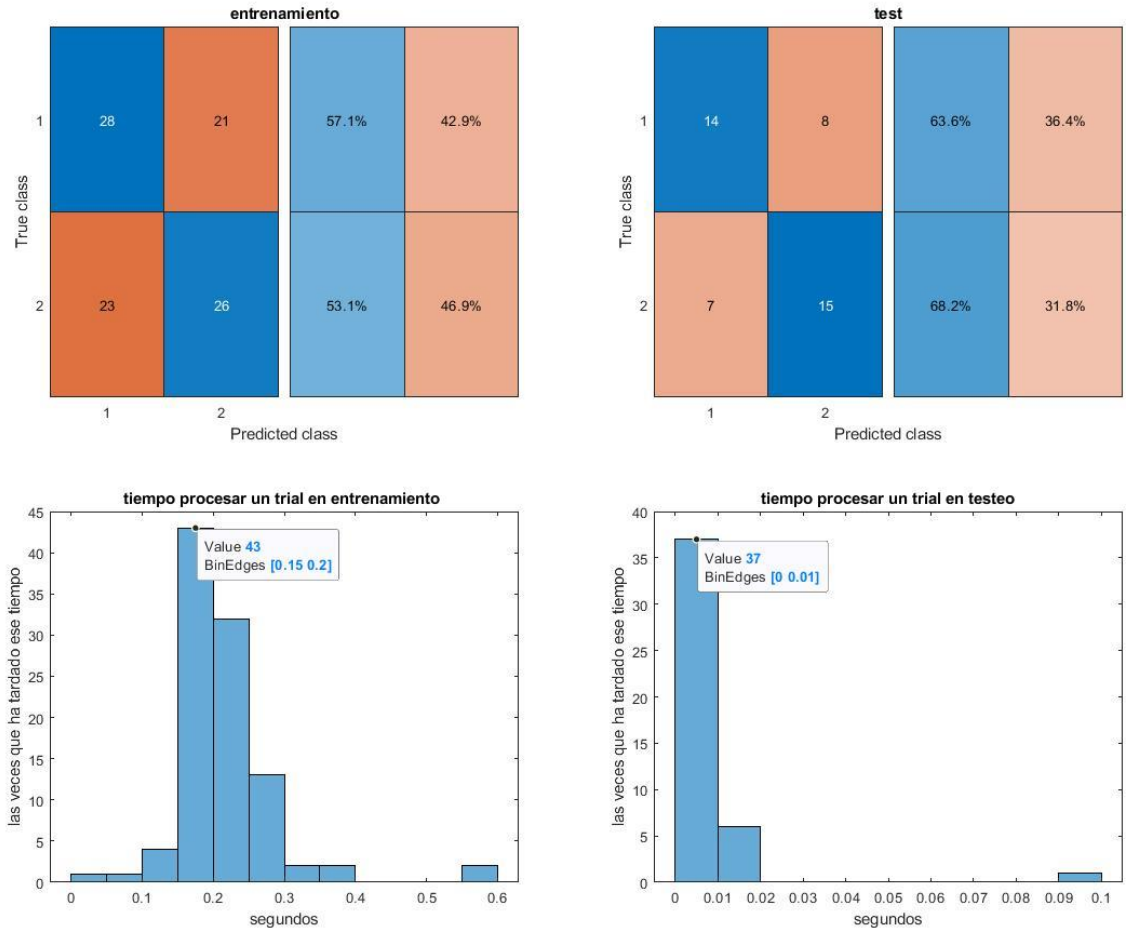


Figura 6-35.Gráficas resultado Bayes paso a paso.

6.1.2.1.2 Pruebas con matriz de filtros W ordenada

tiempo máximo un trial entrenamiento: 1.587962 segundos

tiempo máximo un trial test: 0.093181 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 54.081633 %

Tasa predicción (probabilidad acierto conjunto test) : 65.909091 %

Tasa clasificación errónea entrenamiento: 45.918367%

Tasa clasificación errónea test: 34.090909 %

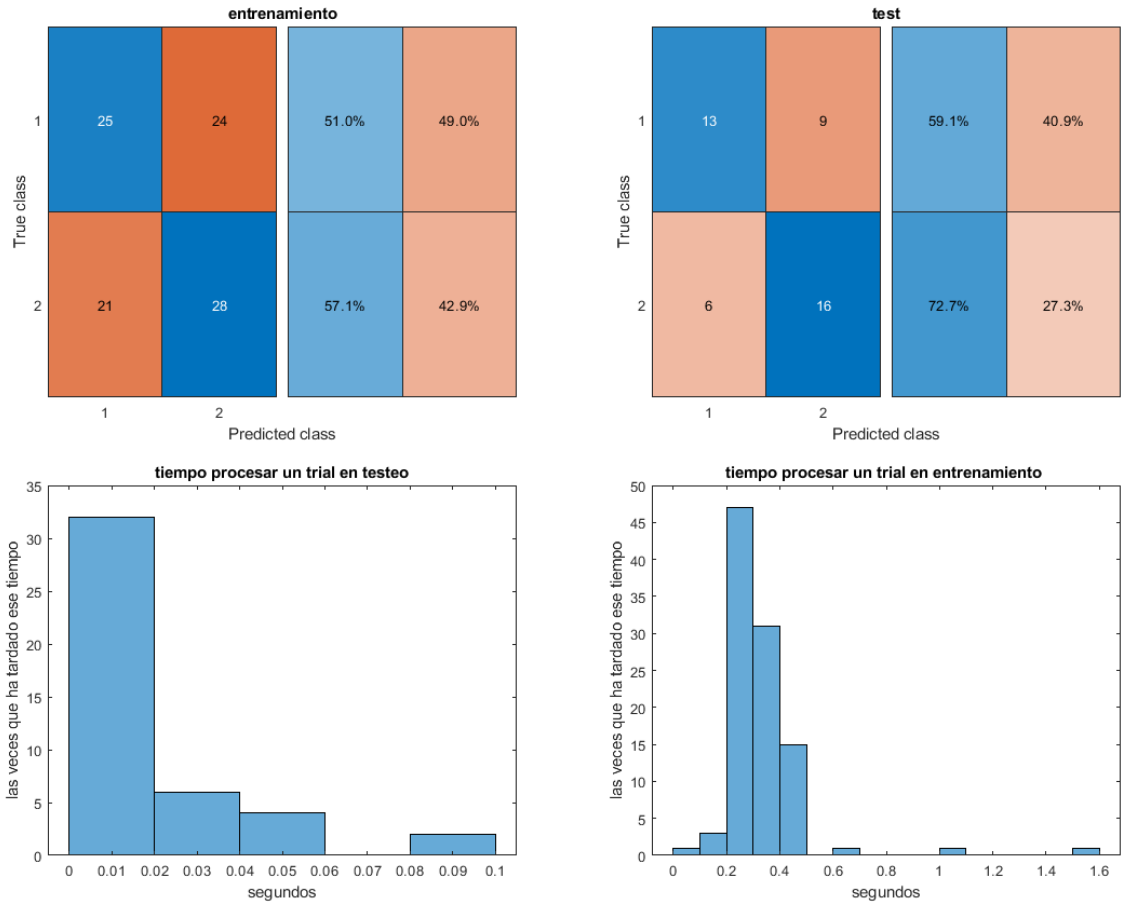


Figura 6-36.Gráficas resultado Bayes paso a paso.

6.1.2.2 Aprovechando propiedades de MATLAB

6.1.2.2.1 Pruebas sin ordenar matriz de filtros W

6.1.2.2.1.1 Gaussian Naive Bayes

No se puede cambiar ningún parámetro para controlar la flexibilidad del modelo.

p_acierto: 0.5 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 4.828474 segundos

tiempo máximo un trial test: 0.143744 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 54.166667 %

Tasa predicción (probabilidad acierto conjunto test) : 61.363636 %

Tasa clasificación errónea entrenamiento: 45.833333%

Tasa clasificación errónea test: 38.636364 %

Training Error (perdida por sustitución en el modelo): 0.35354

Test Error (perdida basada en la distribución de datos): 0.38292

El resultado es aproximadamente igual al obtenido con el código hecho con los cálculos paso por paso.

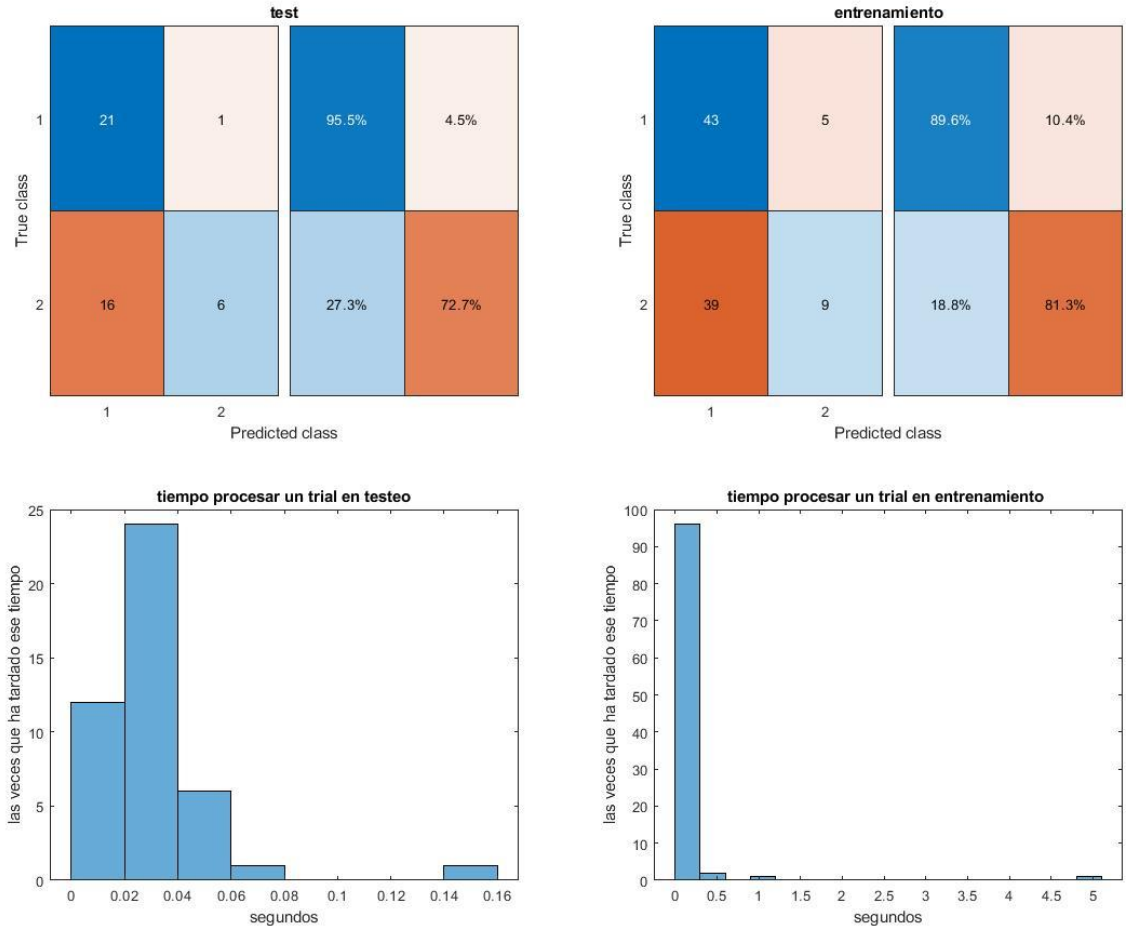


Figura 6-37.Gráficas resultado gaussian naive bayes.

6.1.2.2.1.2 Kernel Gaussiano Naive Bayes.

Se puede cambiar los ajustes de **Tipo de núcleo** y **Soporte** para controlar cómo el clasificador modela las distribuciones de los predictores.

El método de este apartado es uno no paramétrico basado en kernel[69] para realizar la regla de Bayes, basado en representaciones de probabilidades en la reproducción de espacios de Hilbert kernel [212]. En otras palabras, genera un modelo de clasificación Kernel Naive Bayes utilizando densidades de kernel estimadas [213].

Para ver un ejemplo de combinación de Kernel y Bayes se puede consultar [214].

p_acierto: 0.8 | acierto = 0 | acierto_previo = 1 tiempo máximo un trial entrenamiento: 3.681398 segundos tiempo máximo un trial test: 0.193084 segundos

[69] *"Un kernel es una función de ponderación utilizada en técnicas de estimación no paramétricas. Los kernels se utilizan en la estimación de la densidad del kernel para estimar las funciones de densidad de las variables aleatorias, o en la regresión del kernel para estimar la expectativa condicional de una variable aleatoria"* [213].

Tasa predicción (probabilidad acierto conjunto entrenamiento): 81.250000 %

Tasa predicción (probabilidad acierto conjunto test): 88.636364 %

Tasa clasificación errónea entrenamiento: 18.750000%

Tasa clasificación errónea test: 11.363636 %

Training Error (perdida por sustitución en el modelo): 0.090909

Test Error (perdida basada en la distribución de datos): 0.11478

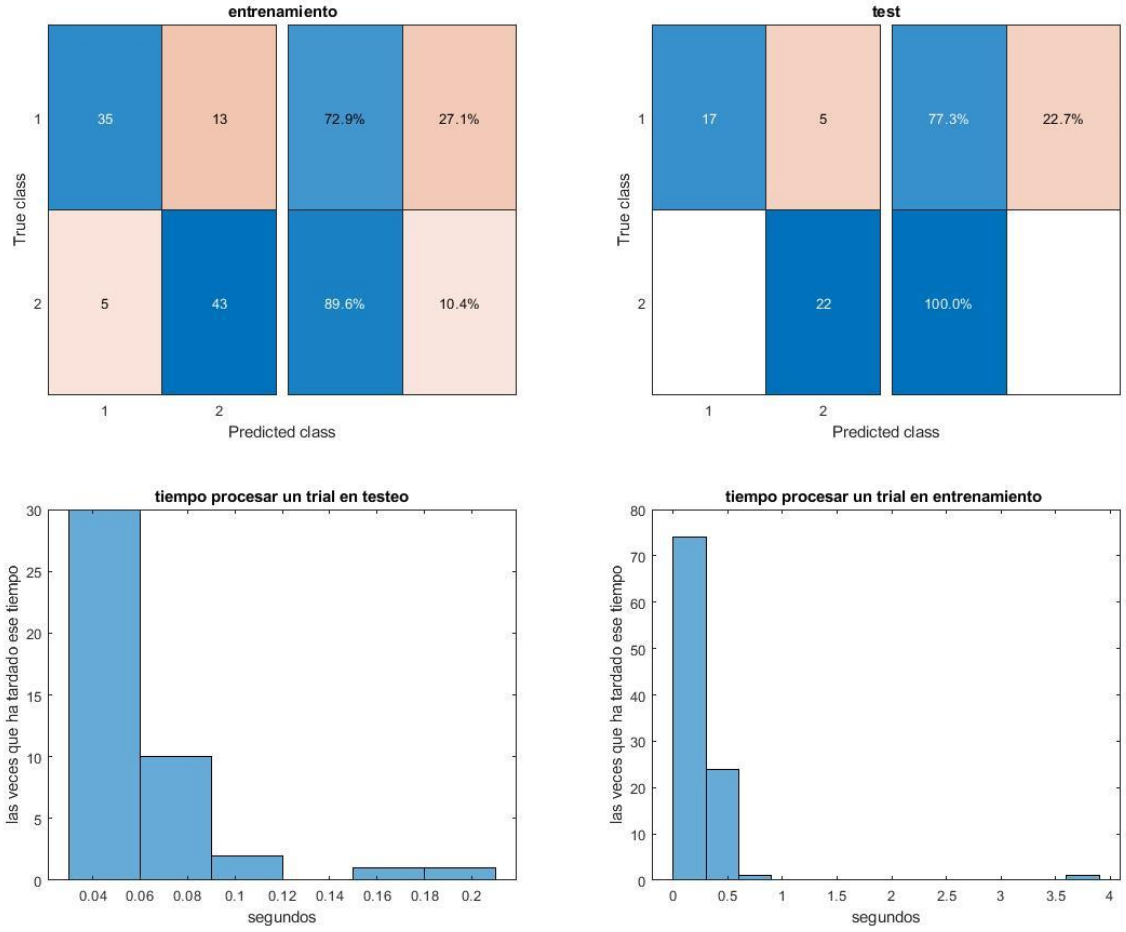


Figura 6-38.Gráficas resultado Kernel Gaussiano.

6.1.2.2.2 Pruebas con matriz de filtros W ordenada

6.1.2.2.2.1 Kernel Gaussiano Naive Bayes

tiempo máximo un trial entrenamiento: 6.610933 segundos

tiempo máximo un trial test: 0.160380 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 82.291667 %

Tasa predicción (probabilidad acierto conjunto test) : 88.636364 %

Tasa clasificación errónea entrenamiento: 17.708333%

Tasa clasificación errónea test: 11.363636 %

Training Error (perdida por sustitución en el modelo): 0.090909

Test Error (perdida basada en la distribución de datos): 0.11478

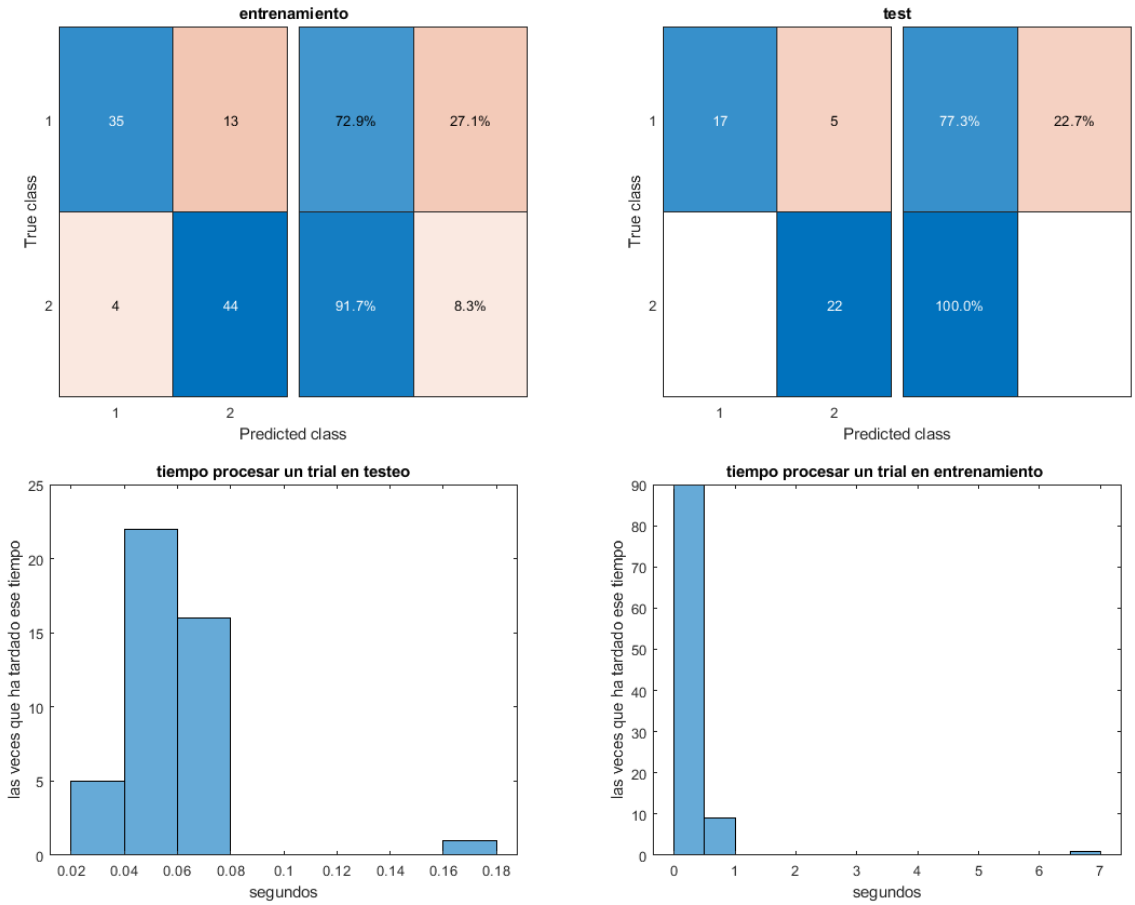


Figura 6-39.Gráficas resultado Kernel Gaussiano.

6.1.3 KNN

6.1.3.1 Aprovechando propiedades de MATLAB

6.1.3.1.1 Pruebas sin ordenar matriz de filtros W

6.1.3.1.1.1 Probando con diferentes k

Probando con diferentes valores de k (es el k_{means} explicado en apartado 5.4) se obtiene estos resultados de probabilidad de acierto:

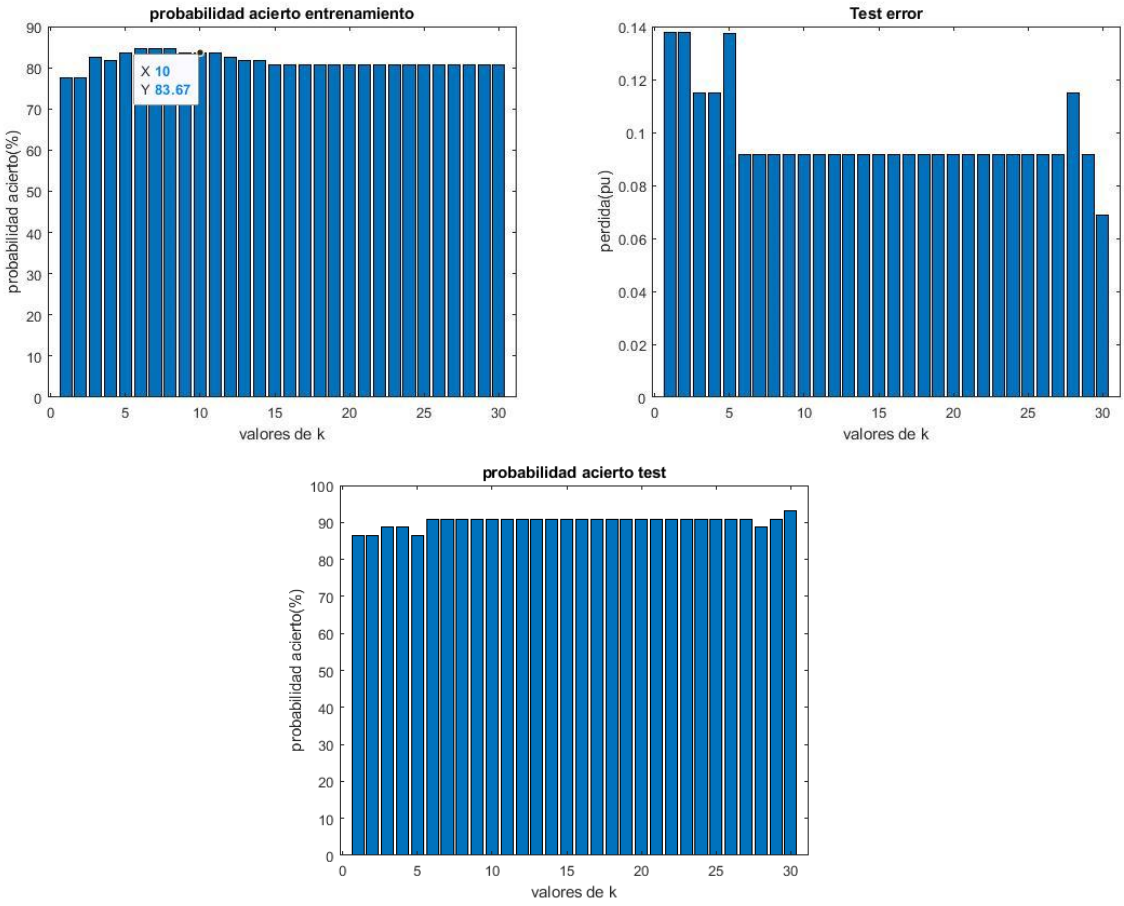


Figura 6-40. Probando diferentes k en KNN.

El valor de k=7 sería una buena opción, ya que la probabilidad de acierto es alta y hay pocas perdidas en el modelo. El **Test Error** es la perdida basada en la distribución de datos que se produce al usar este modelo, esto se volverá a mencionar en el capítulo 7.

6.1.3.1.1.2 Resultados con k=10

Usando '**nearest**' y con ponderación de la distancia de '**squaredinverse**' se obtiene:

p_acierto: 0.8 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 2.007965 segundos

tiempo máximo un trial test: 0.027945 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 83.673469 %

Tasa predicción (probabilidad acierto conjunto test) : 90.909091 %

Tasa clasificación errónea entrenamiento: 16.326531%

Tasa clasificación errónea test: 9.090909 %

Training Error (perdida por sustitución en el modelo): 0

Test Error (perdida basada en la distribución de datos): 0.091827

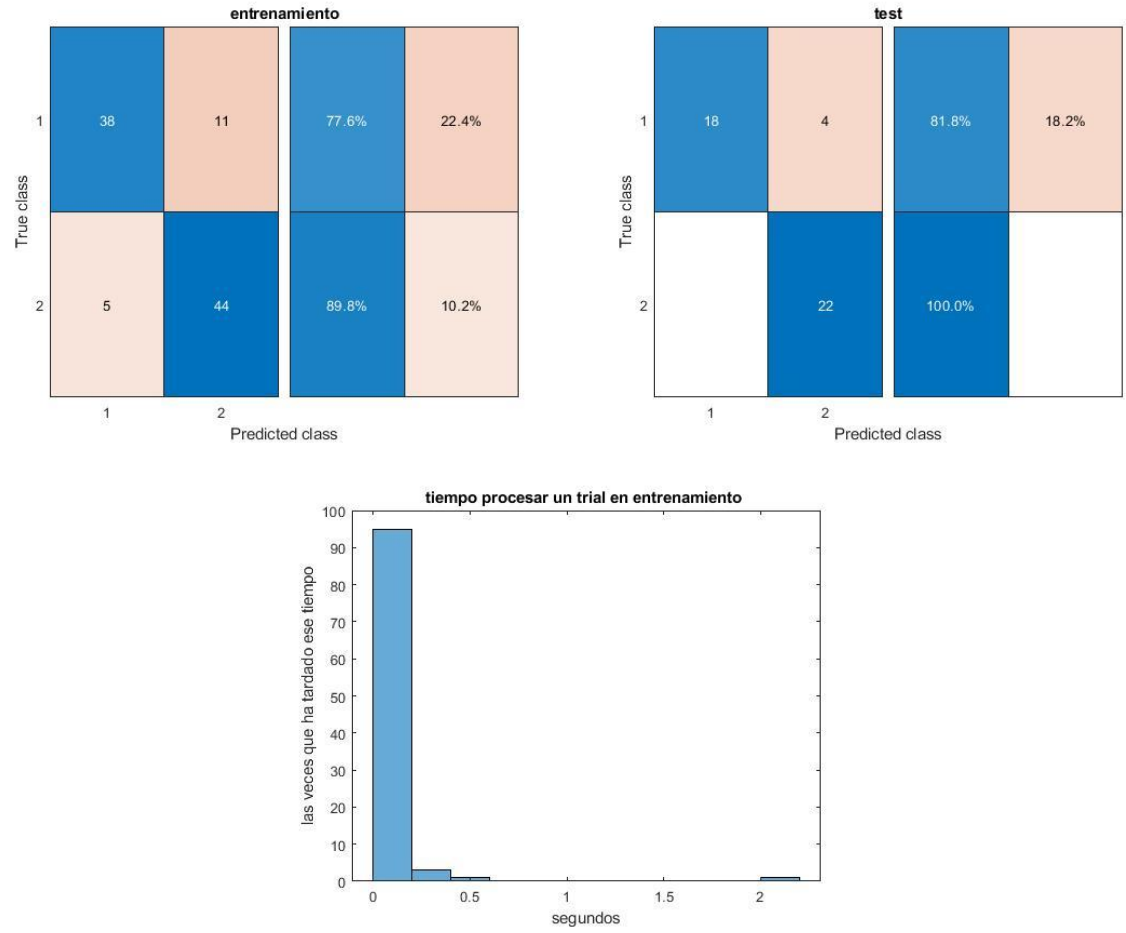


Figura 6-41.Gráficas resultado KNN k=10.

6.1.3.1.1.3 Resultados con k=7

tiempo máximo un trial entrenamiento: 2.238563 segundos

tiempo máximo un trial test: 0.027682 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 84.693878 %

Tasa predicción (probabilidad acierto conjunto test) : 90.909091 %

Tasa clasificación errónea entrenamiento: 15.306122%

Tasa clasificación errónea test: 9.090909 %

Training Error (perdida por sustitución en el modelo): 0

Test Error (perdida basada en la distribución de datos): 0.091827

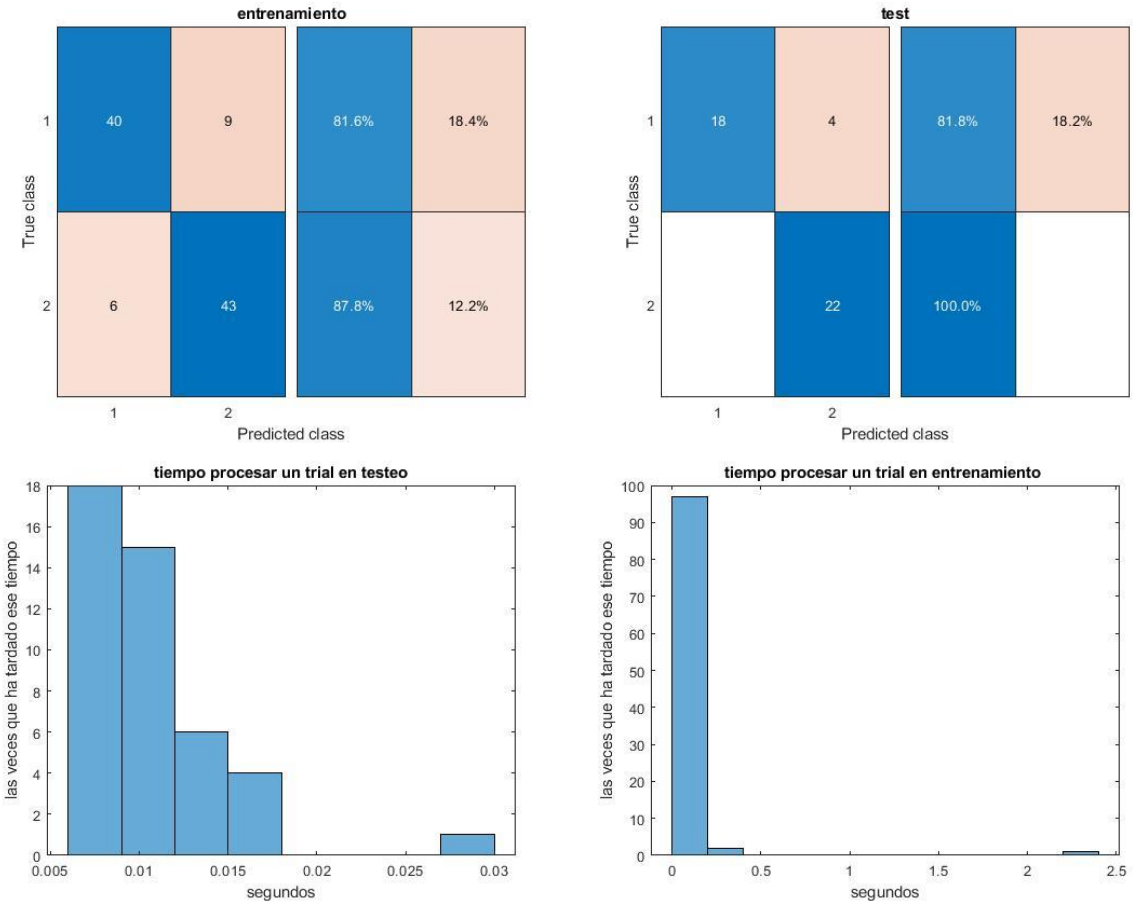


Figura 6-42.Gráficas resultado KNN k=7.

6.1.3.1.1.4 Resultados con k=1

tiempo máximo un trial entrenamiento: 1.621981 segundos

tiempo máximo un trial test: 0.060320 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 77.551020 %

Tasa predicción (probabilidad acierto conjunto test) : 86.363636 %

Tasa clasificación errónea entrenamiento: 22.448980%

Tasa clasificación errónea test: 13.636364 %

Training Error (perdida por sustitución en el modelo): 0

Test Error (perdida basada en la distribución de datos): 0.13774

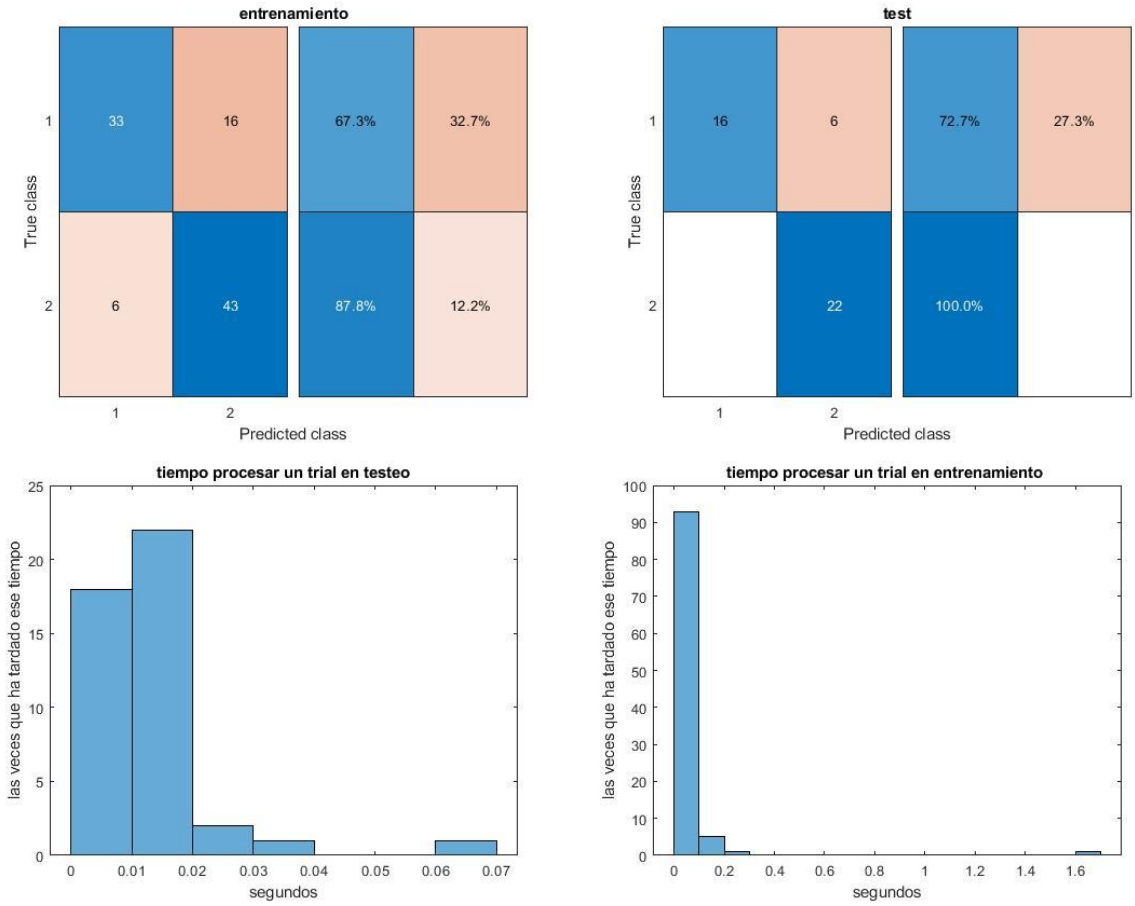


Figura 6-43.Gráficas resultado KNN k=1.

6.1.3.1.2 Pruebas con matriz de filtros W ordenada

6.1.3.1.2.1 Probando con diferentes k

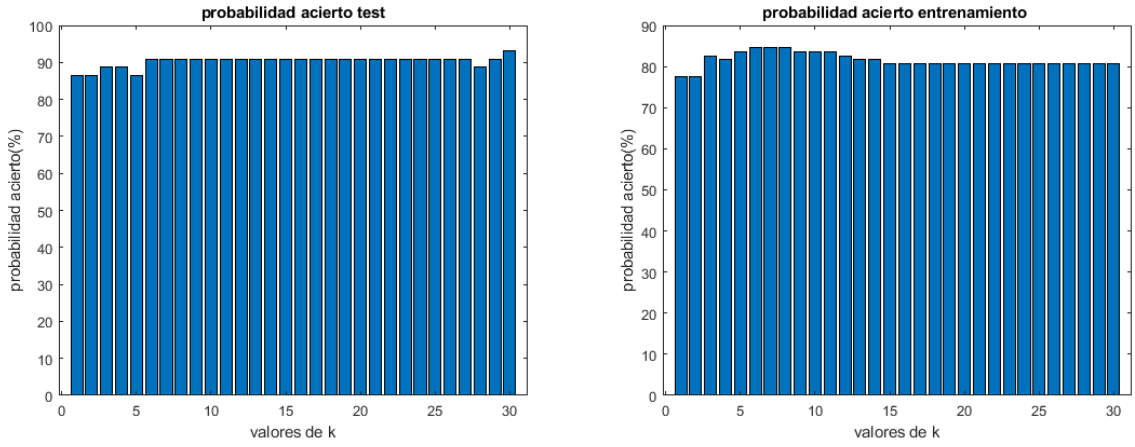


Figura 6-44. Probando diferentes k en KNN.

6.1.3.1.2.2 Resultados con k=7

tiempo máximo un trial entrenamiento: 7.116263 segundos
tiempo máximo un trial test: 0.087348 segundos
Tasa predicción (probabilidad acierto conjunto entrenamiento) : 84.693878 %
Tasa predicción (probabilidad acierto conjunto test) : 90.909091 %

Tasa clasificación errónea entrenamiento: 15.306122%

Tasa clasificación errónea test: 9.090909 %

Training Error (perdida por sustitución en el modelo): 0

Test Error (perdida basada en la distribución de datos): 0.091827

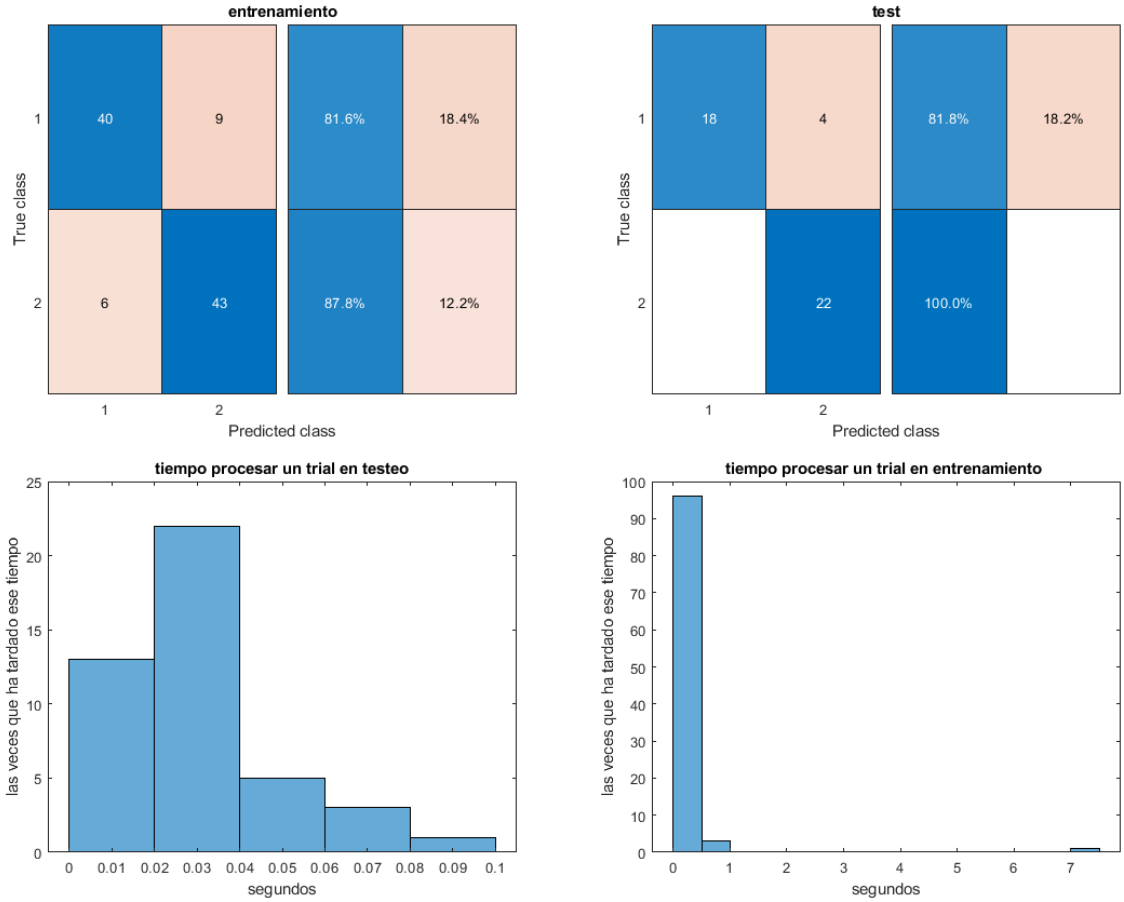


Figura 6-45.Gráficas resultado KNN k=7.

Se ha podido apreciar que no afecta mucho el hecho de ordenar la matriz **W** del CSP.

6.1.3.1.3 Prueba sin CSP

6.1.3.1.3.1 Resultados con k=7

tiempo máximo un trial entrenamiento: 2.829673 segundos

tiempo máximo un trial test: 0.041959 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 52.040816 %

Tasa predicción (probabilidad acierto conjunto test) : 70.454545 %

Tasa clasificación errónea entrenamiento: 47.959184%

Tasa clasificación errónea test: 29.545455 %

Training Error (perdida por sustitución en el modelo): 0

Test Error (perdida basada en la distribución de datos): 0.29522

Se ve como al no utilizar CSP empeora el resultado.

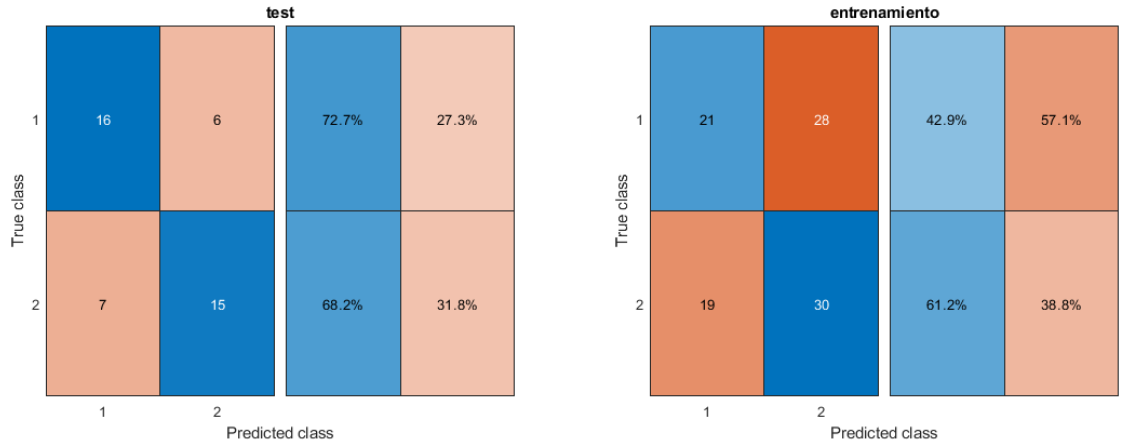


Figura 6-46. KNN sin CSP y k=7.

6.1.4 Decision Trees

6.1.4.1 Aprovechando propiedades de MATLAB

6.1.4.1.1 Pruebas sin ordenar matriz de filtros W

6.1.4.1.1.1 Si no se poda el árbol

tiempo máximo un trial entrenamiento: 2.054717 segundos

tiempo máximo un trial test: 0.069482 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 72.448980 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 27.551020%

Tasa clasificación errónea test: 15.909091 %

Training Error (perdida por sustitución en el modelo): 0.080808

Test Error (perdida basada en la distribución de datos): 0.1607

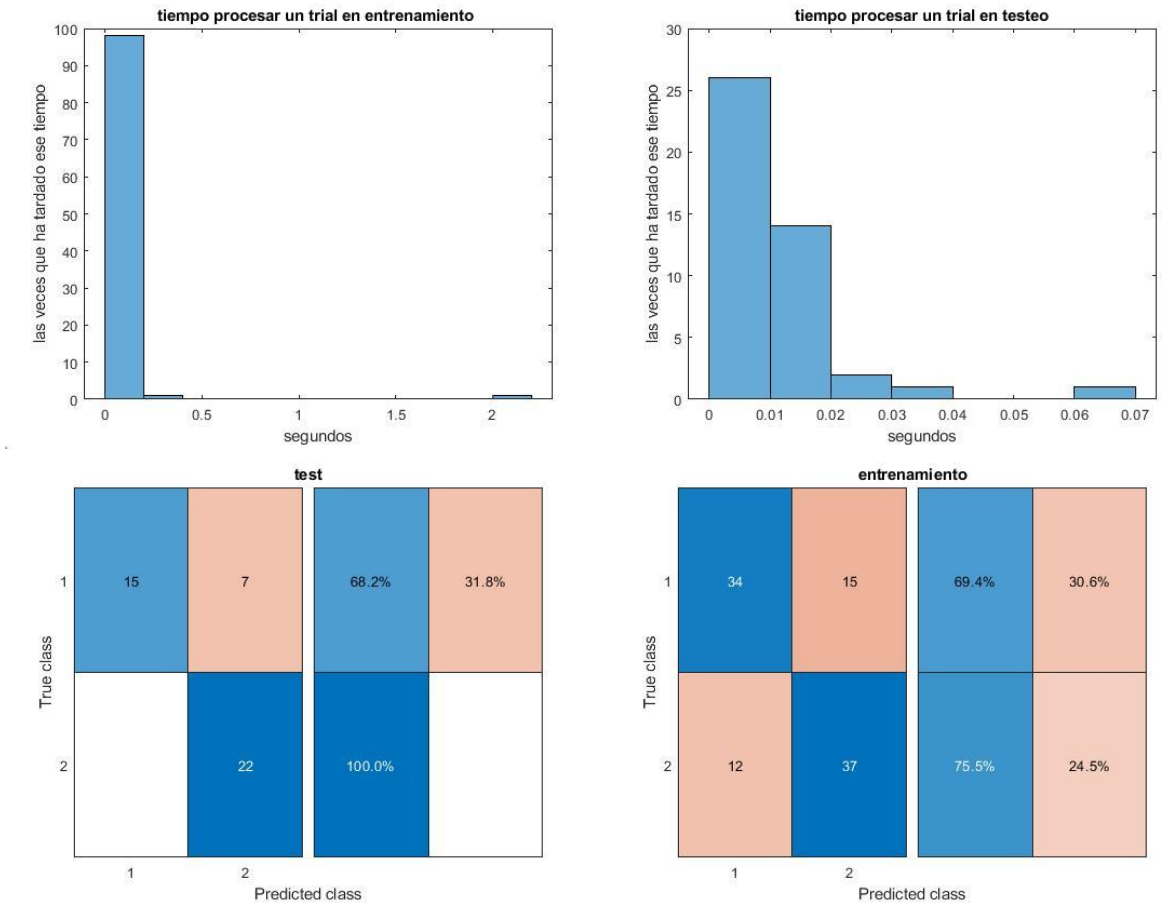


Figura 6-47.Gráficas resultado Tree sin podar.

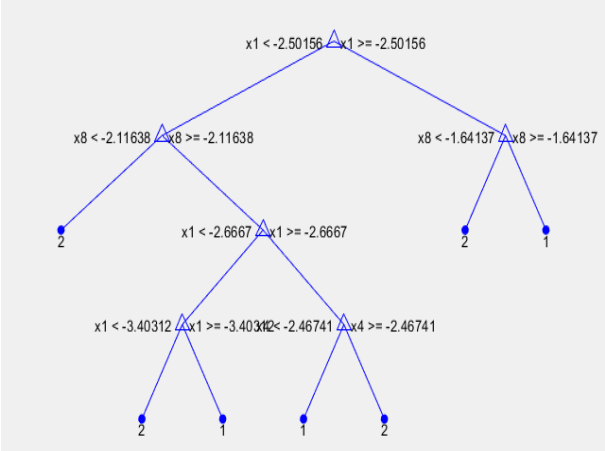


Figura 6-48. Árbol sin podar.

6.1.4.1.1.2 Se prueban diferentes niveles de podado

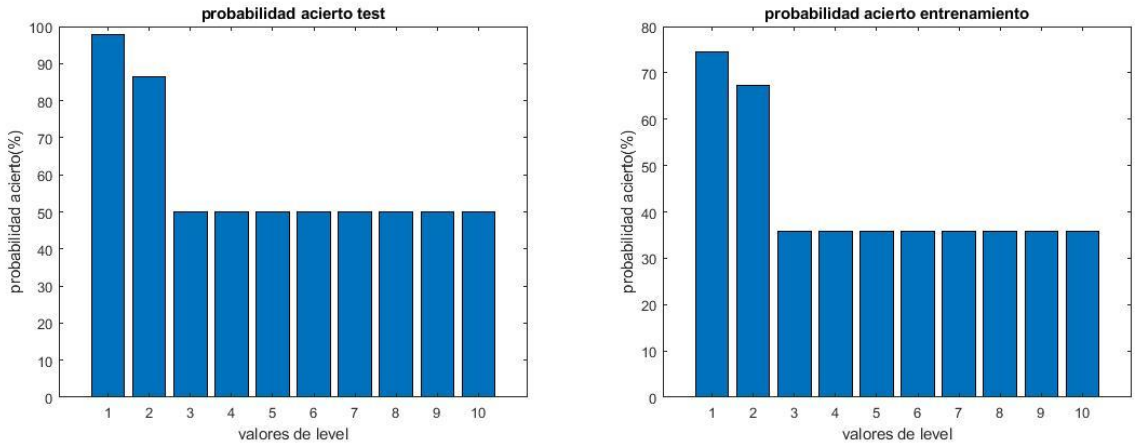


Figura 6-49. Diferentes niveles de podado de Tree.

Viendo la gráficas obtenidas, se deduce que es mejor el de nivel 1.

6.1.4.1.1.3 Usando nivel de 3

Se está podando el árbol completo, va a ser un clasificador aleatorio.

p_acierto: 0.6 | acierto = 1 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 3.284993 segundos

tiempo máximo un trial test: 0.032263 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 35.714286 %

Tasa predicción (probabilidad acierto conjunto test) : 50.000000 %

Tasa clasificación errónea entrenamiento: 64.285714%

Tasa clasificación errónea test: 50.000000 %

Training Error (perdida por sustitución en el modelo): 0.49495

Test Error (perdida basada en la distribución de datos): 0.49495

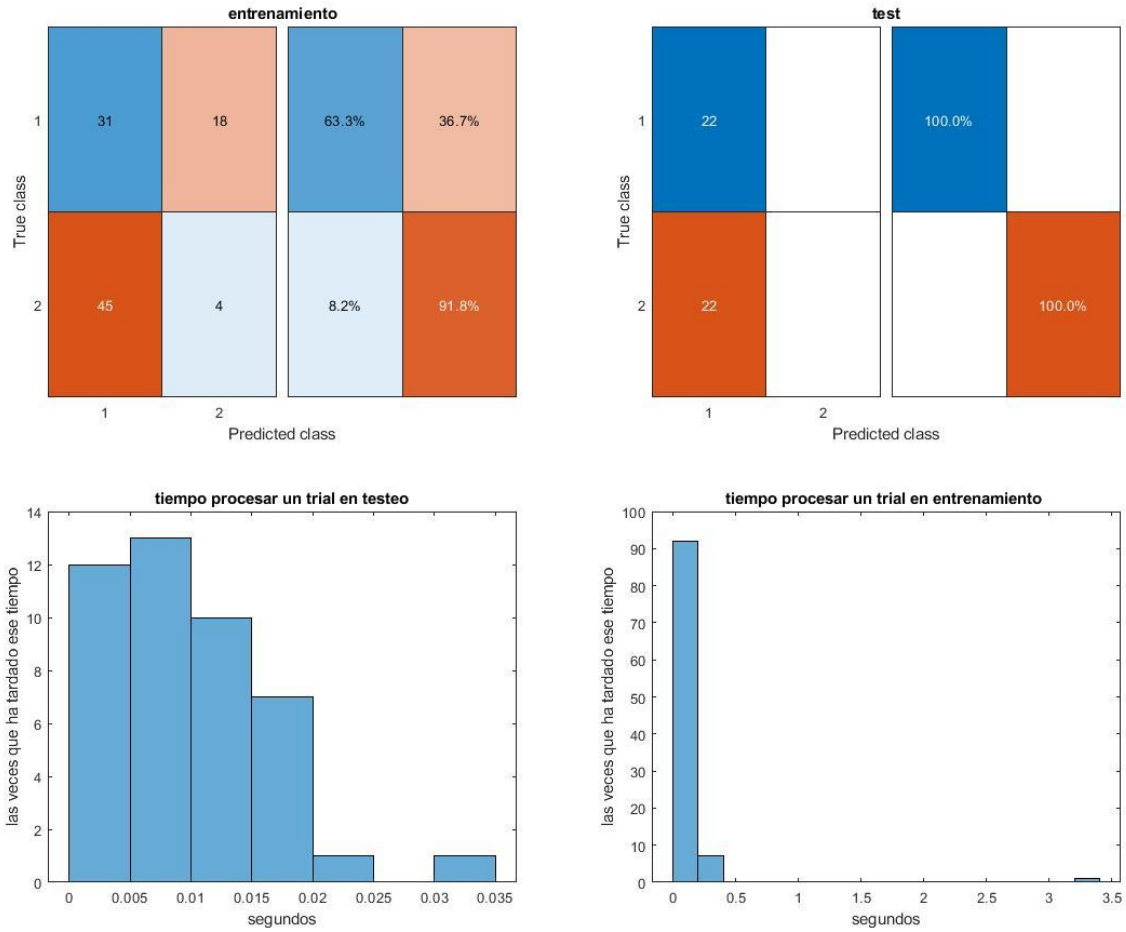


Figura 6-50.Gráficas resultado Tree nivel 3.

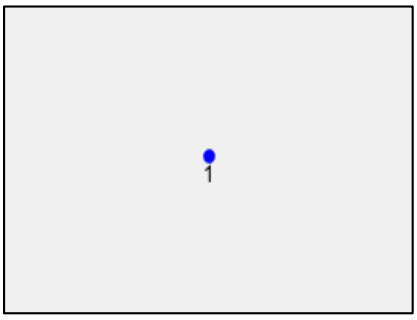


Figura 6-51. Árbol al ser podado a nivel 3. No hay árbol.

6.1.4.1.1.4 Usando nivel de 1

Si se cambia a nivel 1 da lo siguientes resultados:

p_acierto: 0.9 | acierto = 1 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 8.075236 segundos

tiempo máximo un trial test: 0.049958 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 74.489796 %

Tasa predicción (probabilidad acierto conjunto test) : 97.727273 %

Tasa clasificación errónea entrenamiento: 25.510204%

Tasa clasificación errónea test: 2.272727 %

Training Error (perdida por sustitución en el modelo): 0.10101

Test Error (perdida basada en la distribución de datos): 0.022957

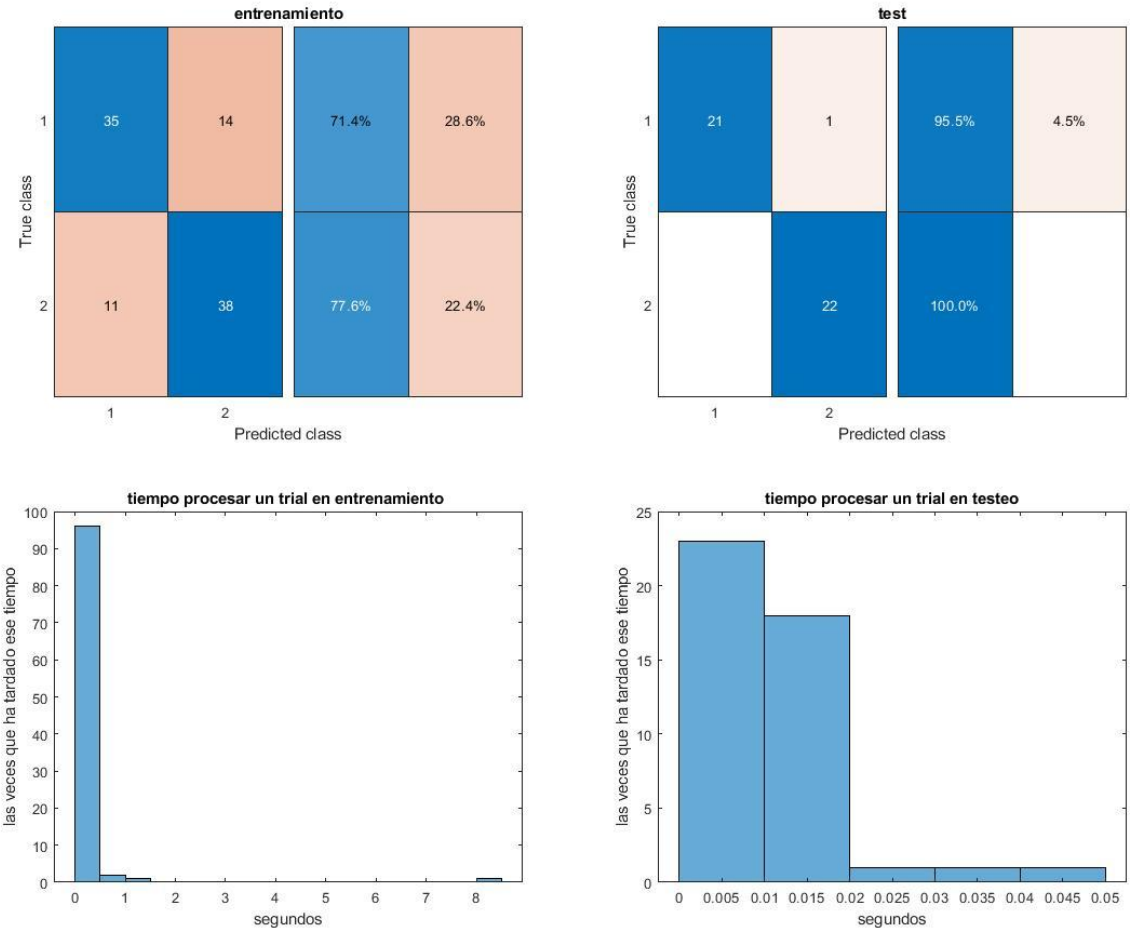


Figura 6-52.Gráficas resultado Decision tree de nivel 1.

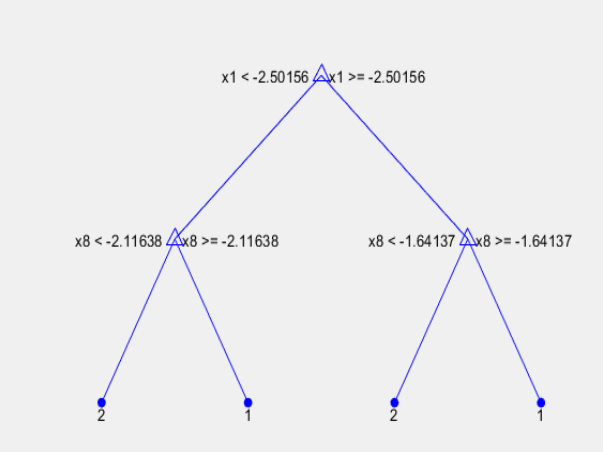


Figura 6-53. Árbol nivel 1.

Da una probabilidad de acierto más alta que sin podar. Además, cuantas menos ramas se tengan, es más sencillo de aplicar.

6.1.4.1.2 Pruebas con matriz de filtros W ordenada

6.1.4.1.2.1 Si no se poda el árbol

tiempo máximo un trial entrenamiento: 2.674718 segundos

tiempo máximo un trial test: 0.079813 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 72.448980 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 27.551020%

Tasa clasificación errónea test: 15.909091 %

Training Error (perdida por sustitución en el modelo): 0.080808

Test Error (perdida basada en la distribución de datos): 0.1607

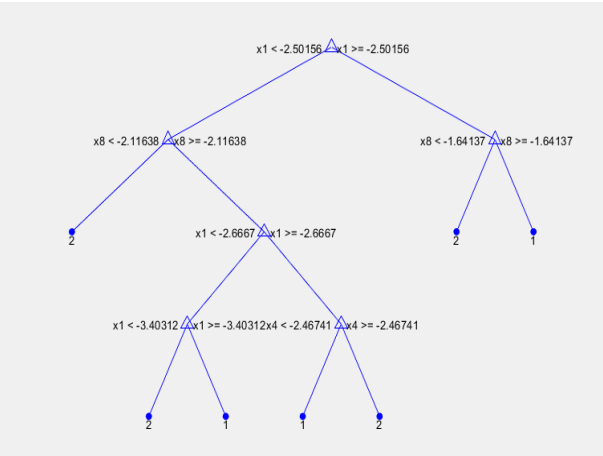


Figura 6-54. Árbol sin podar.

Da los mismos resultados que sin el comando de ordenar **W**.

6.1.4.1.2.2 Usando nivel de 1

tiempo máximo un trial entrenamiento: 8.700948 segundos

tiempo máximo un trial test: 0.026996 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 74.489796 %

Tasa predicción (probabilidad acierto conjunto test) : 97.727273 %

Tasa clasificación errónea entrenamiento: 25.510204%

Tasa clasificación errónea test: 2.272727 %

Training Error (perdida por sustitución en el modelo): 0.10101

Test Error (perdida basada en la distribución de datos): 0.022957

6.1.4.2 Usando App de MATLAB.

Se recomienda consultar la referencia [215].

6.1.4.2.1 Pruebas sin ordenar matriz de filtros W

6.1.4.2.1.1 Fine Tree

"Pocas hojas para hacer distinciones gruesas entre clases (el número máximo de divisiones es 4)" [216] .

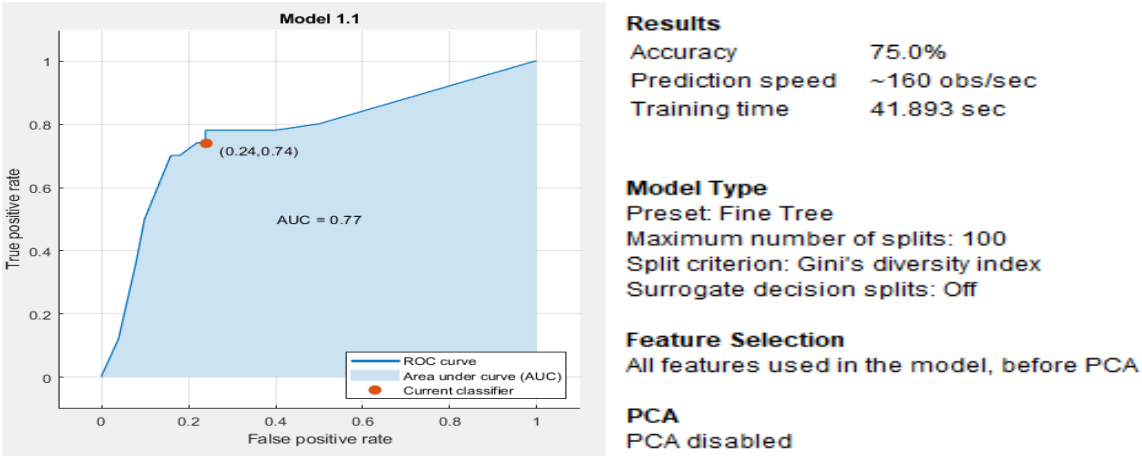


Figura 6-55. ROC curva con clase positiva la 1 y negativa la 2, Fine Tree.

p_acierto: 0.8 | acierto = 1 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 5.533414 segundos

tiempo máximo un trial test: 0.220056 segundos

validationAccuracy final: 0.727273 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 72.448980 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 27.551020%

Tasa clasificación errónea test: 15.909091 %

Se aprecia que la probabilidad de acierto obtenida con validación cruzada ha dado un valor menor que la obtenida por el otro método, en este caso.

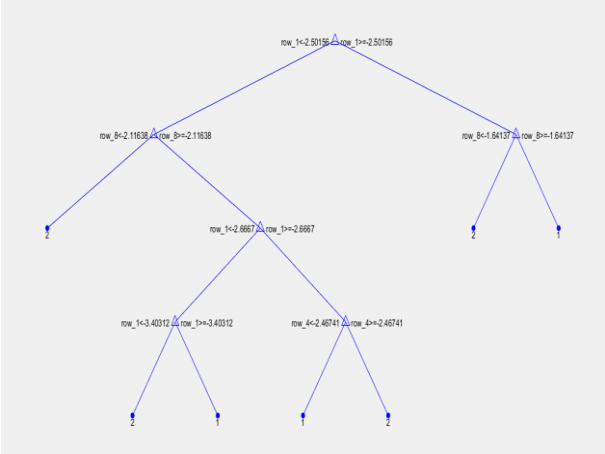
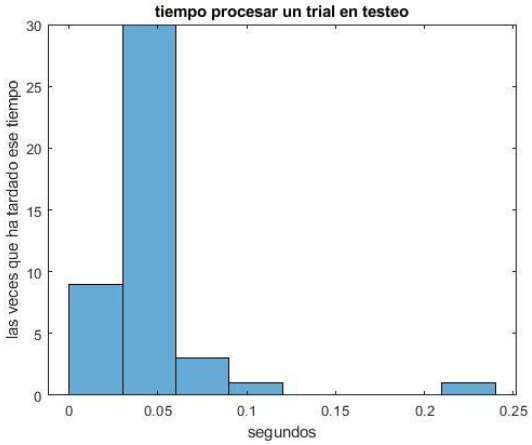
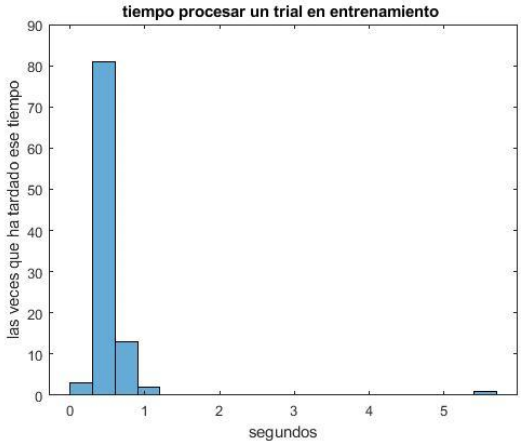


Figura 6-56. Árbol fine tree.





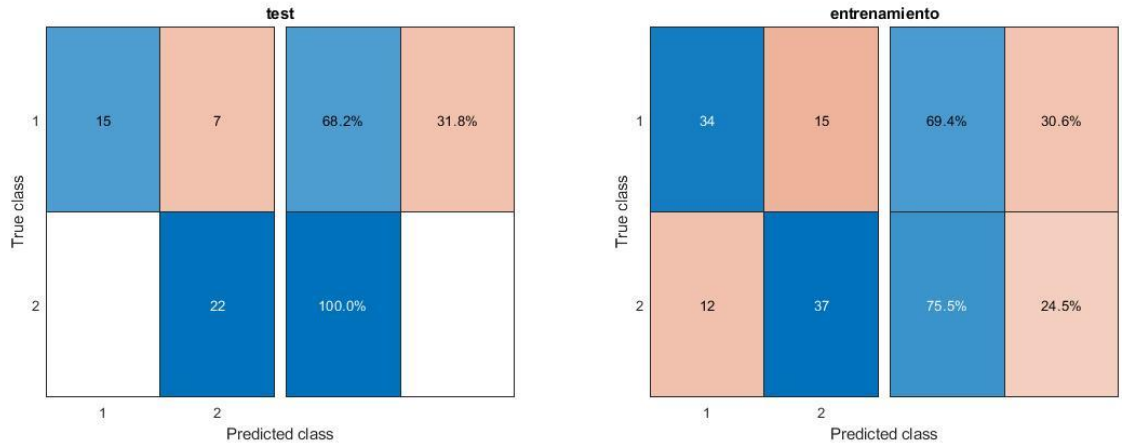


Figura 6-57.Gráficas resultado Fine Tree.

6.1.4.2.1.2 Medium Tree

"Número medio de hojas para distinciones más finas entre clases (el número máximo de divisiones es 20)" [216] .

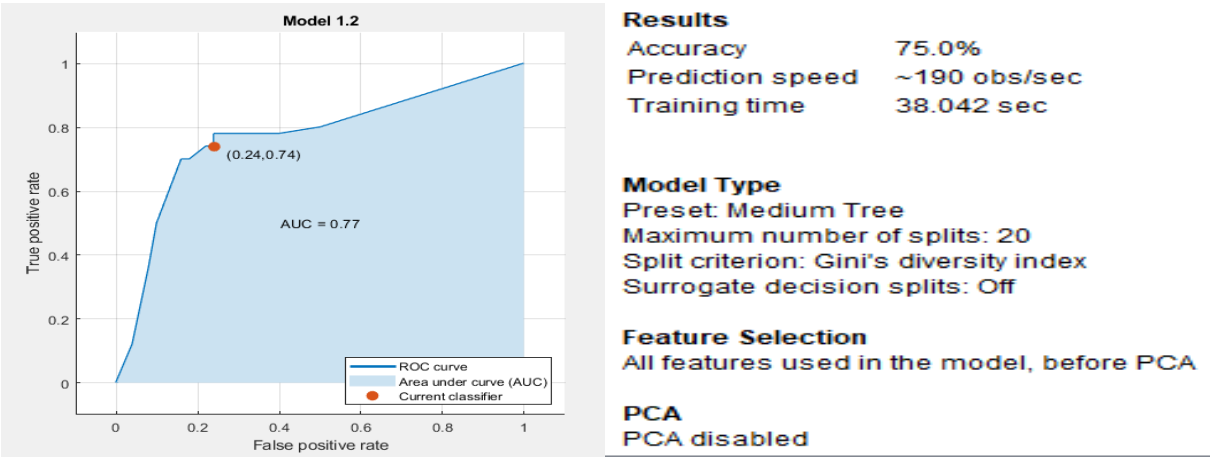


Figura 6-58. Curva ROC Medium tree.

p_acierto: 0.8 | acierto = 1 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 5.461757 segundos

tiempo máximo un trial test: 0.062764 segundos

validationAccuracy final : 0.727273 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 72.448980 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 27.551020%

Tasa clasificación errónea test: 15.909091 %

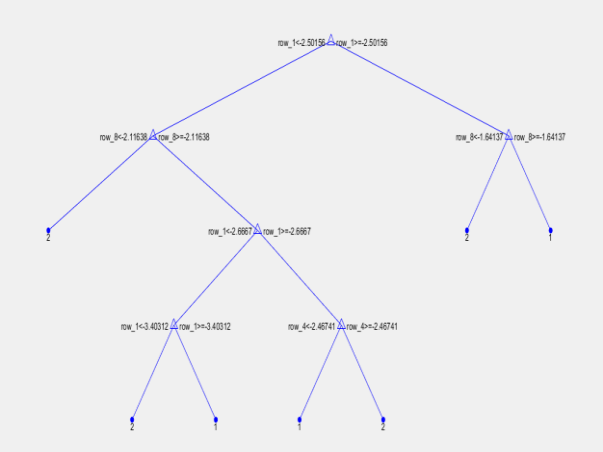


Figura 6-59. Árbol Medium Tree.

Se obtiene el mismo esquema de árbol de decisión que Fine Tree. El fine tree y el médium tree dan el mismo árbol de decisión.

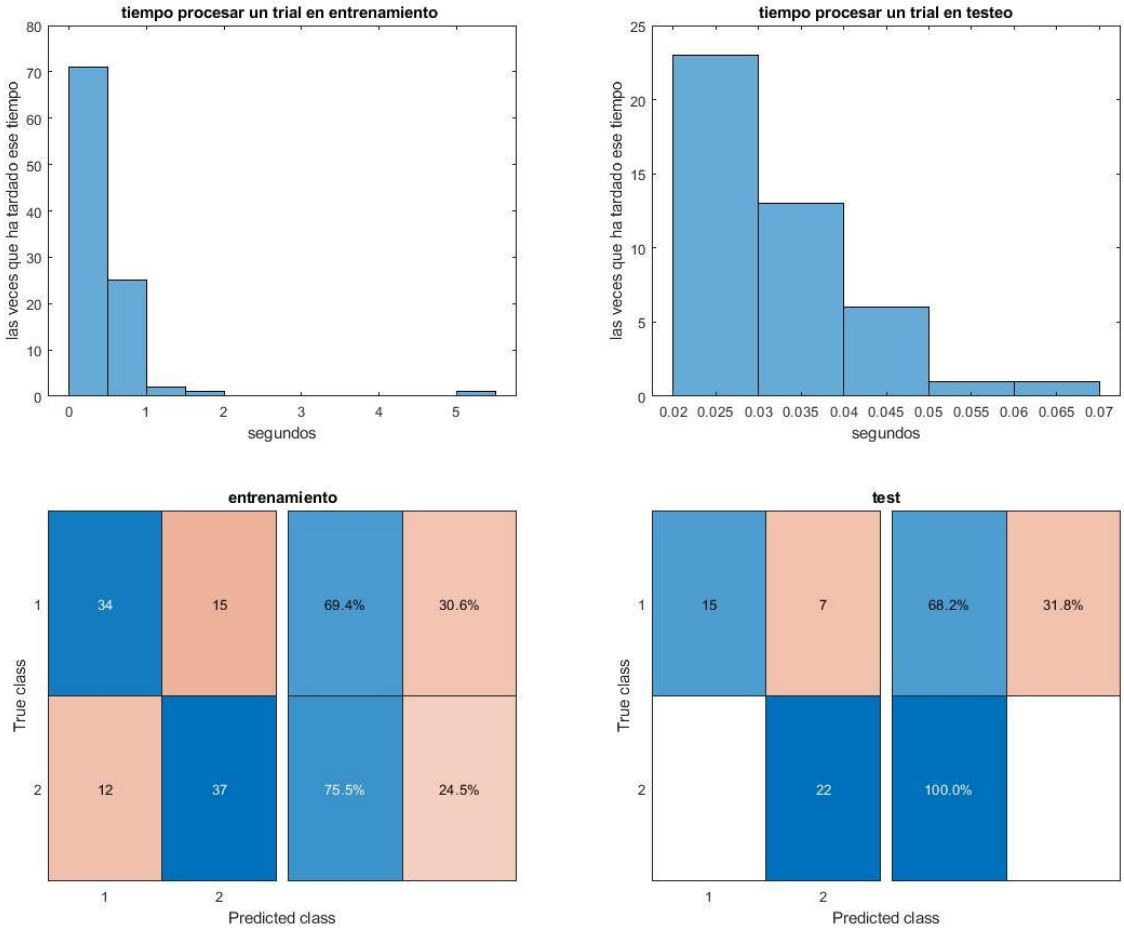


Figura 6-60.Gráficas resultado Medium Tree.

6.1.4.2.1.3 Coarse Tree

"*Pocas hojas para hacer distinciones gruesas entre clases (el número máximo de divisiones es 4)"* [216] .

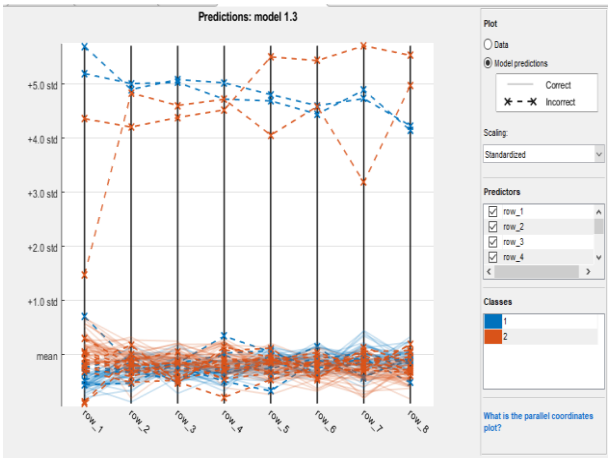
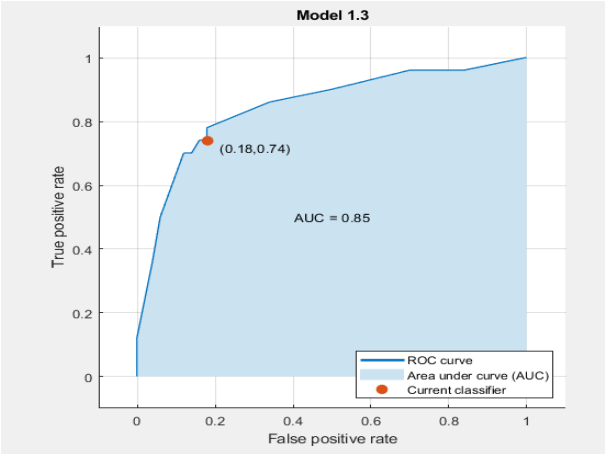


Figura 6-61. Representación real y predicha de dos características con Coarse Tree.



Results

Accuracy	78.0%
Prediction speed	~1500 obs/sec
Training time	43.737 sec

Model Type
Preset: Coarse Tree
Maximum number of splits: 4
Split criterion: Gini's diversity index
Surrogate decision splits: Off

Feature Selection
All features used in the model, before PCA

PCA
PCA disabled

Figura 6-62.Curva ROC Coarse Tree.

p_acierto: 0.9 | acierto = 1 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 6.953753 segundos

tiempo máximo un trial test: 0.056727 segundos

validationAccuracy final : 0.777778 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 72.448980 %

Tasa predicción (probabilidad acierto conjunto test) : 97.727273 %

Tasa clasificación errónea entrenamiento: 27.551020%

Tasa clasificación errónea test: 2.272727 %

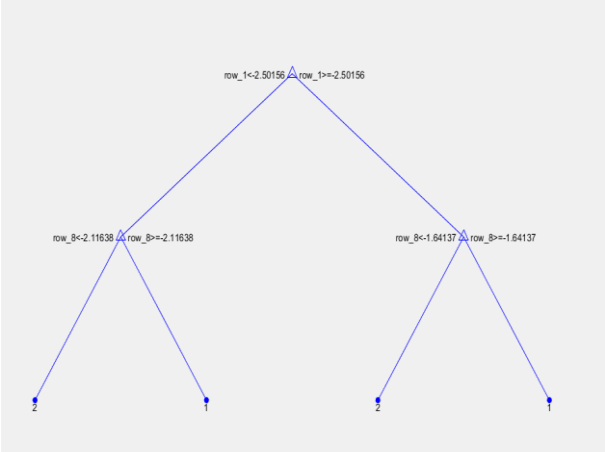


Figura 6-63. Árbol Coarse Tree.

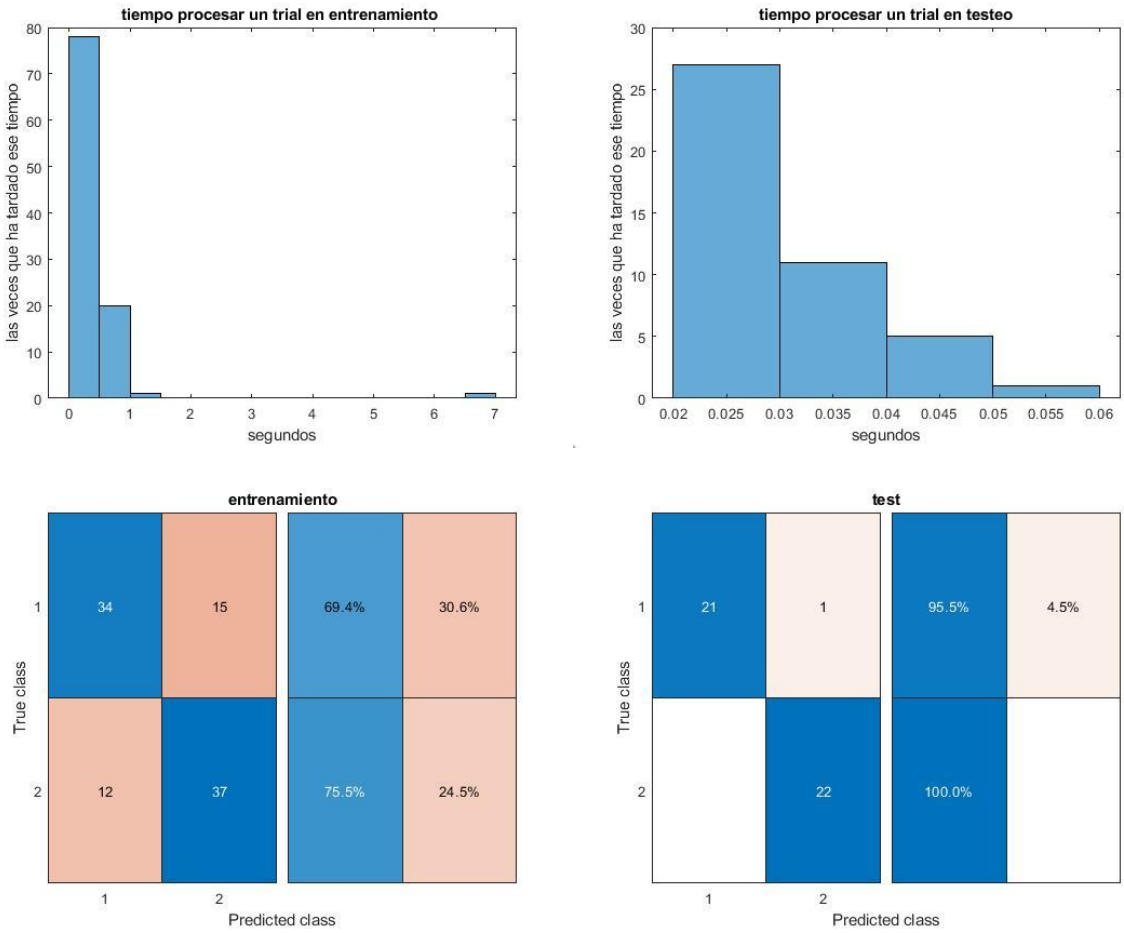


Figura 6-64.Gráficas resultado Coarse Tree.

De los tres Decision Tree disponibles en la App de MATLAB, el Coarse Tree es el que da mejores resultados.

6.1.4.2.2 Pruebas con matriz de filtros W ordenada

6.1.4.2.2.1 Coarse Tree

tiempo máximo un trial entrenamiento: 18.959916 segundos tiempo máximo un trial test: 0.048203 segundos

validationAccuracy final : 0.777778 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 72.448980 %

Tasa predicción (probabilidad acierto conjunto test) : 97.727273 %

Tasa clasificación errónea entrenamiento: 27.551020%

Tasa clasificación errónea test: 2.272727 %

6.1.4.2.3 Prueba sin CSP

No se puede realizar esta prueba a los clasificadores se exportadon de la App, ya que estos se hicieron esperando 8 características. Si no se quiere aplicar CSP a éstos, hay que volver a crearlos con la App, haciendo que tenga en cuenta los 22 canales.

6.1.5 SVM

6.1.5.1 Aprovechando propiedades de MATLAB

6.1.5.1.1 Pruebas sin ordenar matriz de filtros W

6.1.5.1.1.1 Lineal

p_acierto: 0.8 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 9.317267 segundos

tiempo máximo un trial test: 0.114600 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 82.653061 %

Tasa predicción (probabilidad acierto conjunto test) : 88.636364 %

Tasa clasificación errónea entrenamiento: 17.346939%

Tasa clasificación errónea test: 11.363636 %

Training Error (perdida por sustitución en el modelo): 0.10101

Test Error (perdida basada en la distribución de datos): 0.11478

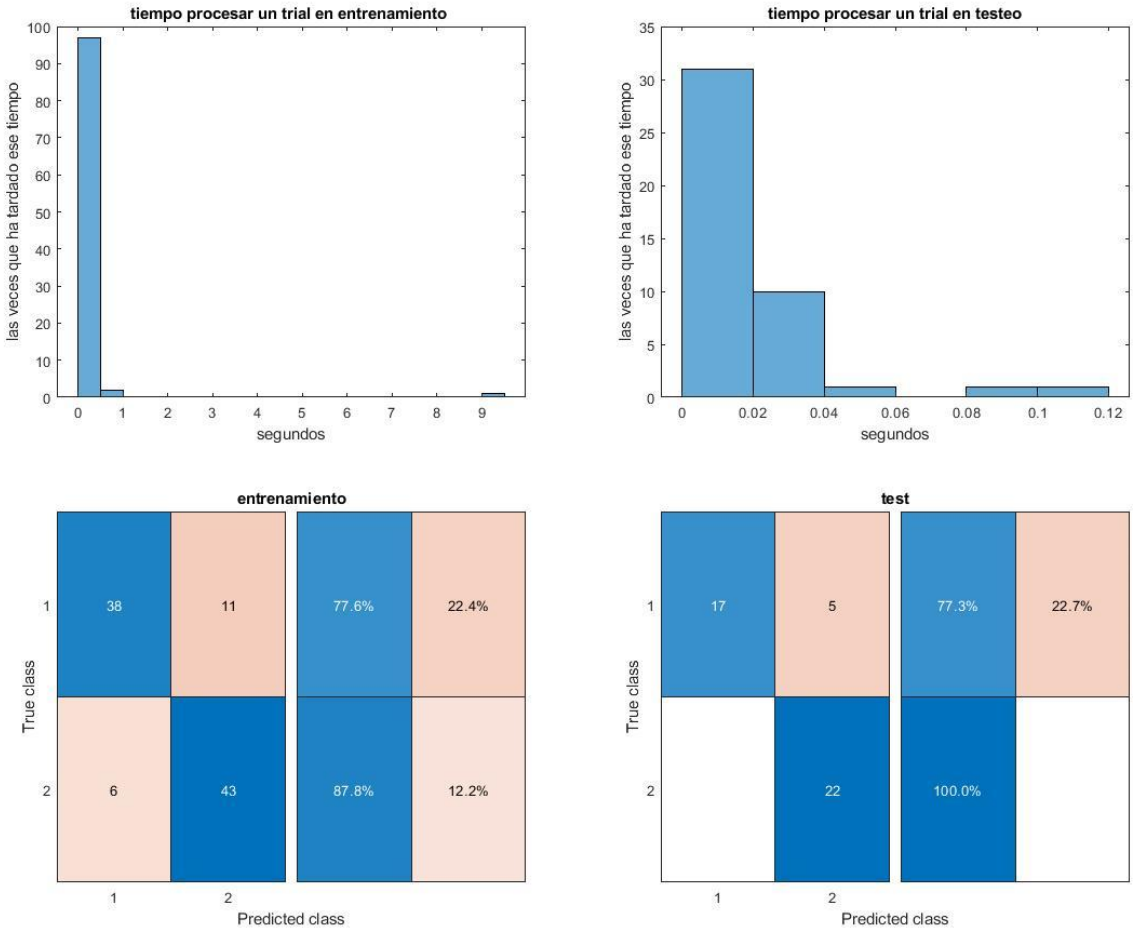


Figura 6-65.Gráficas resultado SVM lineal.

6.1.5.1.1.2 Usando función kernel gaussiana

p_acierto: 0.8 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 2.402447 segundos

tiempo máximo un trial test: 0.060629 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 83.673469 %

Tasa predicción (probabilidad acierto conjunto test) : 88.636364 %

Tasa clasificación errónea entrenamiento: 16.326531%

Tasa clasificación errónea test: 11.363636 %

Training Error (perdida por sustitución en el modelo): 0.080808

Test Error (perdida basada en la distribución de datos): 0.11478

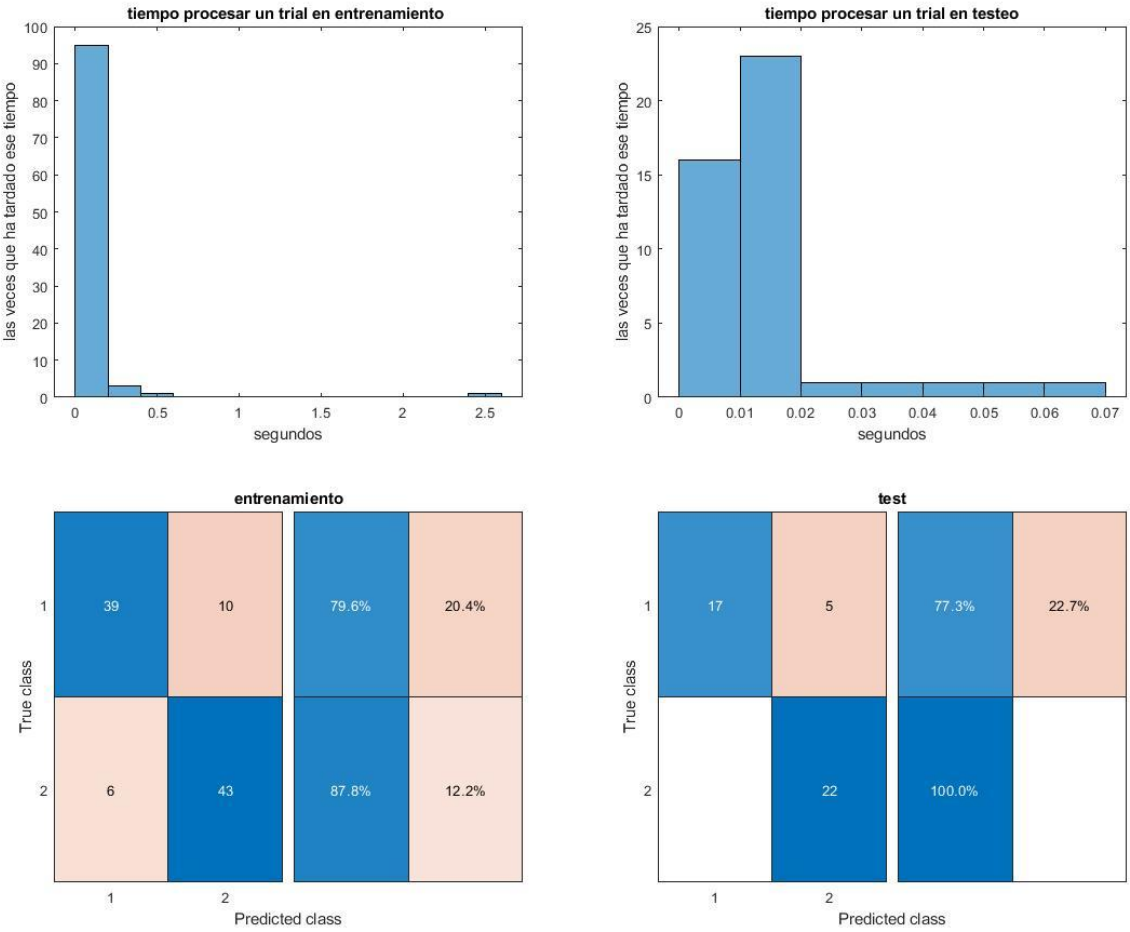


Figura 6-66.Gráficas resultado SVM kernel gaussiano.

6.1.5.1.1.3 Usando función kernel polinomial

p_acierto: 0.8 | acierto = 1 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 4.546310 segundos

tiempo máximo un trial test: 0.031637 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 78.571429 %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182 %

Tasa clasificación errónea entrenamiento: 21.428571%

Tasa clasificación errónea test: 18.181818 %

Training Error (perdida por sustitución en el modelo): 0.050505

Test Error (perdida basada en la distribución de datos): 0.1832

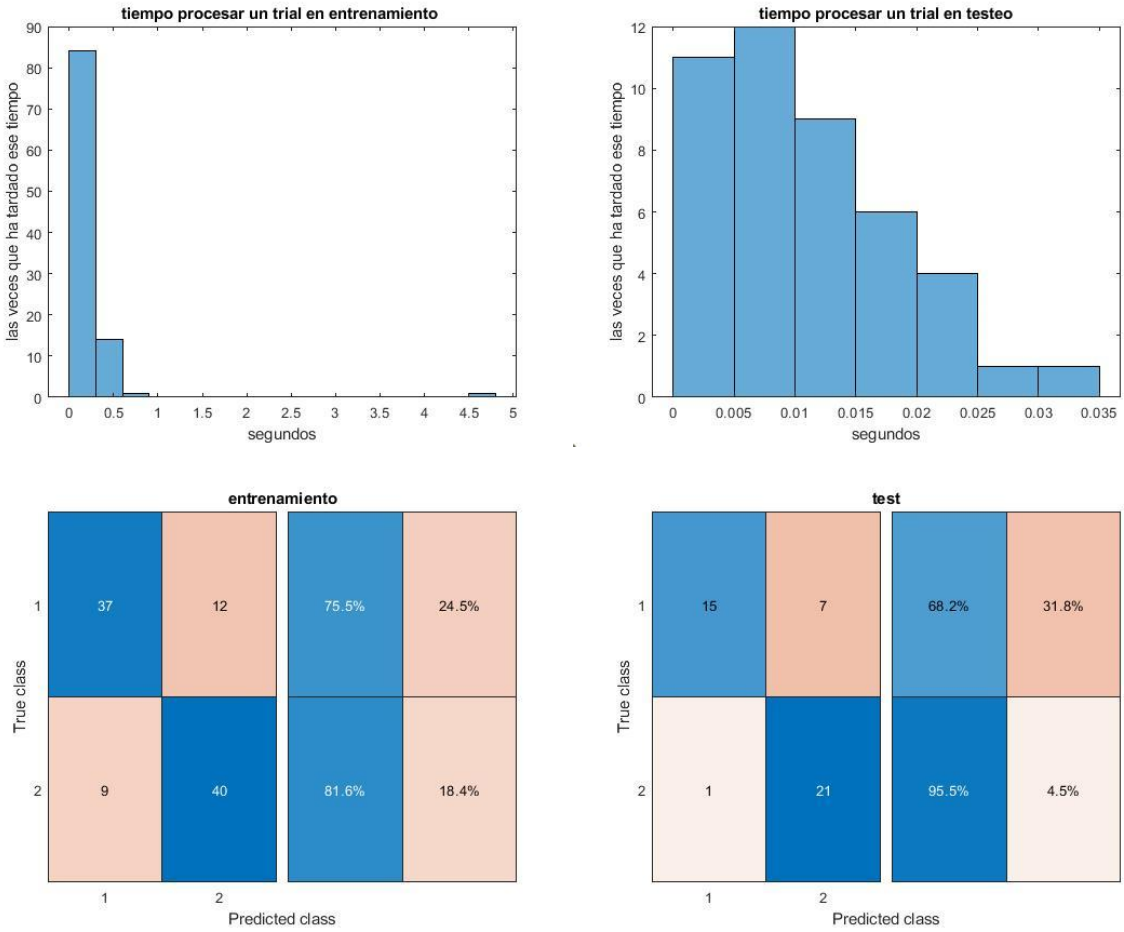


Figura 6-67.Gráficas resultado SVM kernel polinomial.

6.1.5.1.2 Pruebas con ordenación de matriz de filtros W

6.1.5.1.2.1 Usando función kernel gaussiana

tiempo máximo un trial entrenamiento: 2.422924 segundos

tiempo máximo un trial test: 0.078769 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 82.653061 %

Tasa predicción (probabilidad acierto conjunto test) : 88.636364 %

Tasa clasificación errónea entrenamiento: 17.346939%

Tasa clasificación errónea test: 11.363636 %

Training Error (perdida por sustitución en el modelo): 0.080808

Test Error (perdida basada en la distribución de datos): 0.11478

La probabilidad de entrenamiento online ha salido un poco más baja, en comparación con el caso de no ordenar **W**. La probabilidad de acierto de testeo ha salido igual. Se puede suponer que se acaban obteniendo aproximadamente los mismos valores de los parámetros para la clasificación.

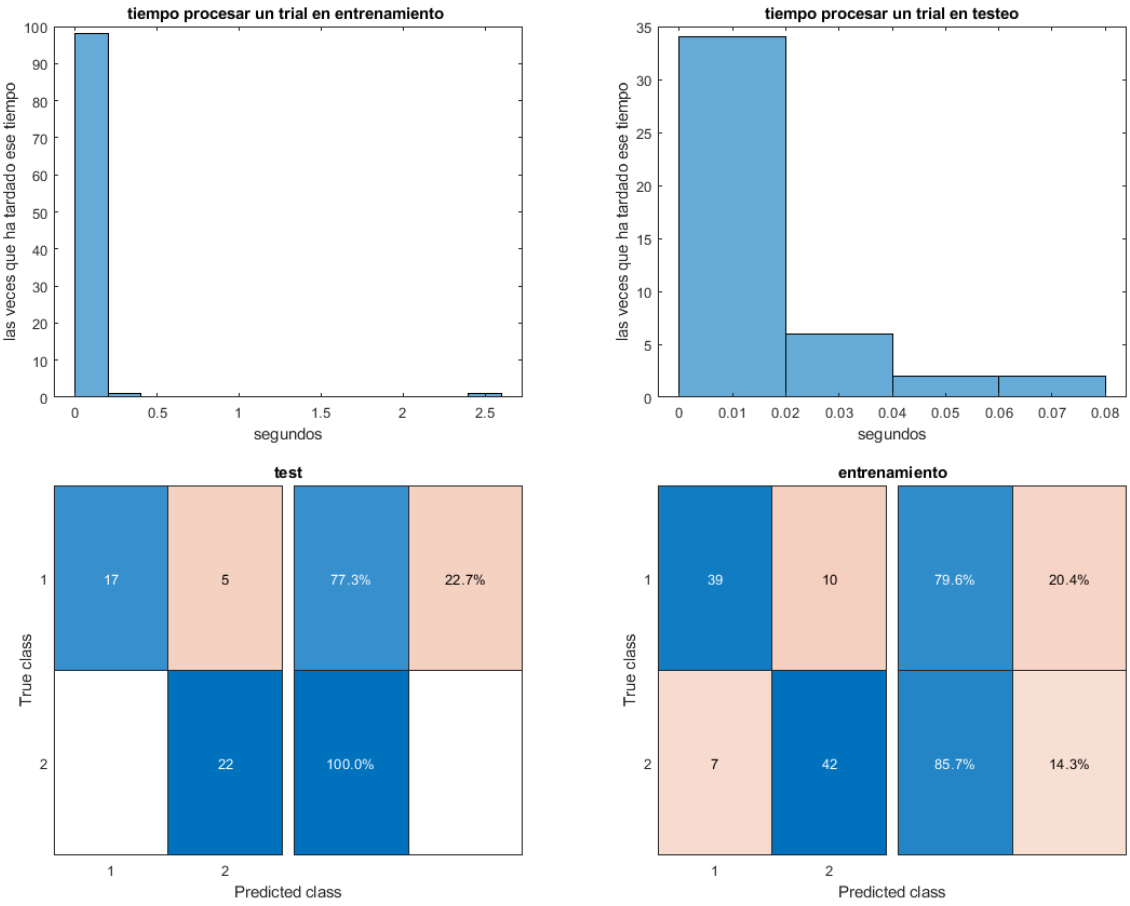


Figura 6-68.Gráficas resultado SVM kernel gaussiano.

6.1.5.1.2.1.1 Prueba de aplicar testeo a los datos de entrenamiento

tiempo máximo un trial test: 0.125384 segundos

Tasa predicción (probabilidad acierto conjunto test) : 96.000000 %

Tasa clasificación errónea test: 4.000000 %

Test Error (perdida basada en la distribución de datos): 0.040202

6.1.5.1.2.1.2 Prueba sin CSP

tiempo máximo un trial entrenamiento: 1.576637 segundos

tiempo máximo un trial test: 0.088247 segundos

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 48.979592 %

Tasa predicción (probabilidad acierto conjunto test) : 70.454545 %

Tasa clasificación errónea entrenamiento: 51.020408%

Tasa clasificación errónea test: 29.545455 %

6.1.5.2 Usando App de MATLAB

6.1.5.2.1 Pruebas sin ordenar matriz de filtros W

6.1.5.2.1.1 Linear SVM

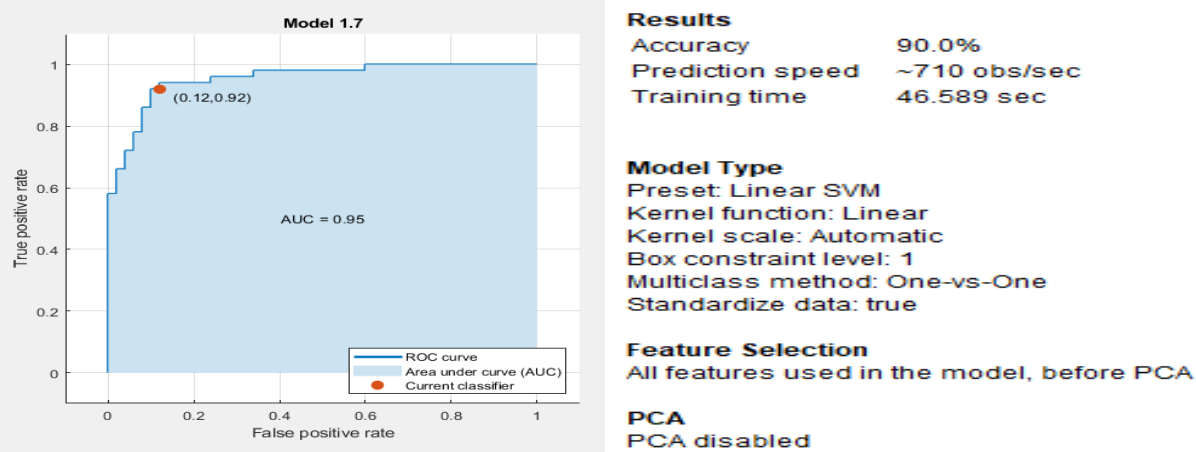


Figura 6-69. Curva ROC SVM lineal.

p_acierto: 0.8 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 7.810734 segundos

tiempo máximo un trial test: 0.107483 segundos

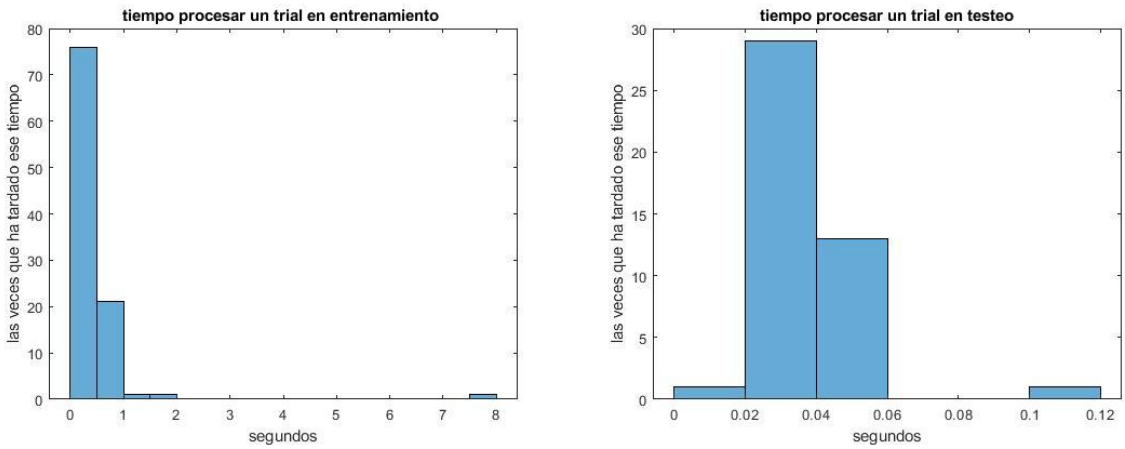
validationAccuracy final : 0.898990 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 79.591837 %

Tasa predicción (probabilidad acierto conjunto test) : 88.636364 %

Tasa clasificación errónea entrenamiento: 20.408163%

Tasa clasificación errónea test: 11.363636 %



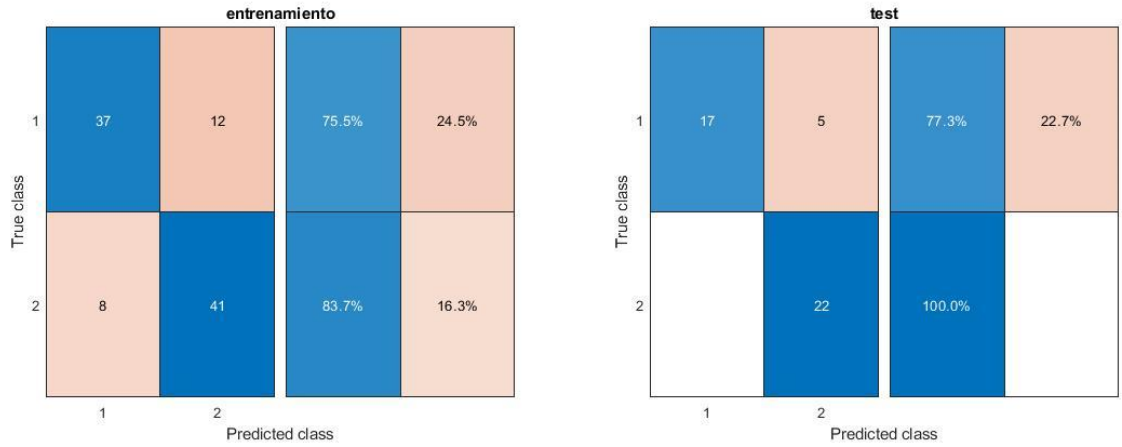


Figura 6-70. Gráficas resultado SVM lineal App.

6.1.5.2.2 Pruebas con ordenación de matriz de filtros W

6.1.5.2.2.1 Linear SVM

tiempo máximo un trial entrenamiento: 11.109479 segundos

tiempo máximo un trial test: 0.070007 segundos

validationAccuracy final : 0.898990 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 80.612245 %

Tasa predicción (probabilidad acierto conjunto test) : 88.636364 %

Tasa clasificación errónea entrenamiento: 19.387755%

Tasa clasificación errónea test: 11.363636 %

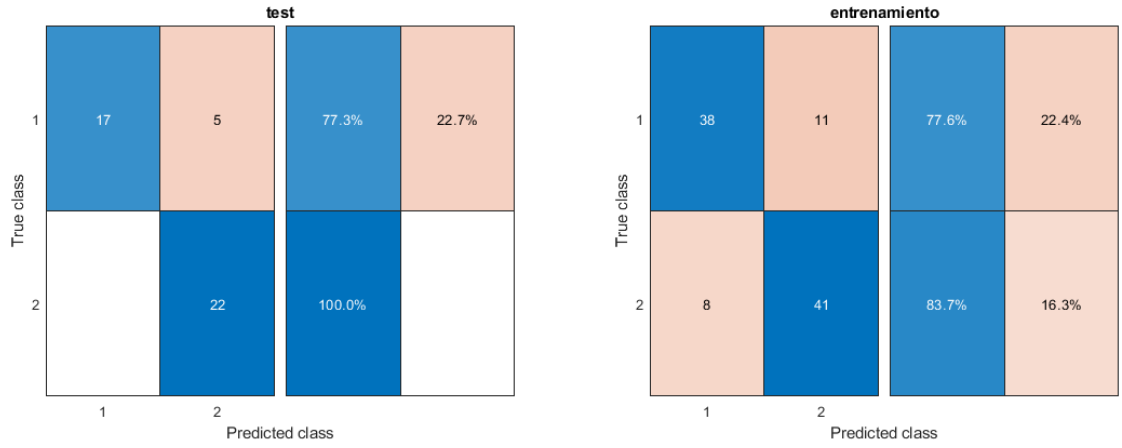


Figura 6-71. Gráficas resultado SVM lineal App.

6.1.6 Logistic Regression

6.1.6.1 Usando App de MATLAB

6.1.6.1.1 Pruebas sin ordenar matriz de filtros W

En la App de MATLAB se probaron todos los clasificadores que hay disponible entrenando con los 144 ***trials*** de un usuario. Concluyéndose que el Logistic Regression era de los mejores clasificadores en ese caso. Habiendo seleccionado Cross-validation de 5 folds (Figura 6-73), la probabilidad de acierto en Logistic Regression fue de un 91%.

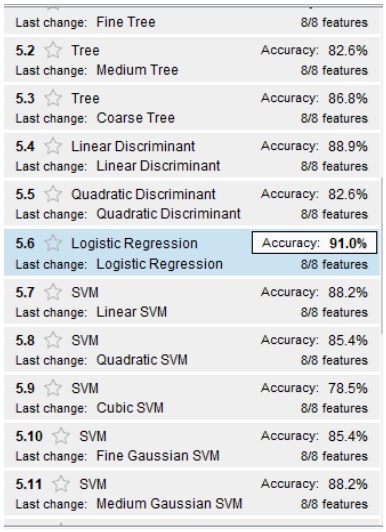


Tabla 6-2. Probabilidad acierto con app MATLAB, 144 ***trials.***

Figura 6-72. Logistic Regression resultados en App.

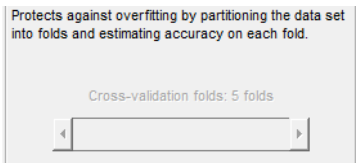


Figura 6-73. Panel donde se indica si se usa cross-validation folds y el números de folds, usando App de MATLAB.

Los resultados del Logistic Regression exportado, siendo en tiempo real la clasificación (se le indica en código 144 ***trials*** entrenamiento en tiempo real y 0 de testeo), son:

```
p_acierto: 0.8 | acierto = 0 | acierto_previo = 1
```

tiempo máximo un trial: 6.270301 segundos

probabilidad acierto entrenamiento conjunto: 81.751825

validationAccuracy final: 0.909091

Los resultados usando este clasificador con 100 de entrenamiento y 44 de testeo son:

p_acierto: 0.7 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 6.804348 segundos

tiempo máximo un trial test: 0.191624 segundos

validationAccuracy final : 0.888889 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 81.720430 %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182 %

Tasa clasificación errónea entrenamiento: 18.279570%

Tasa clasificación errónea test: 18.181818 %

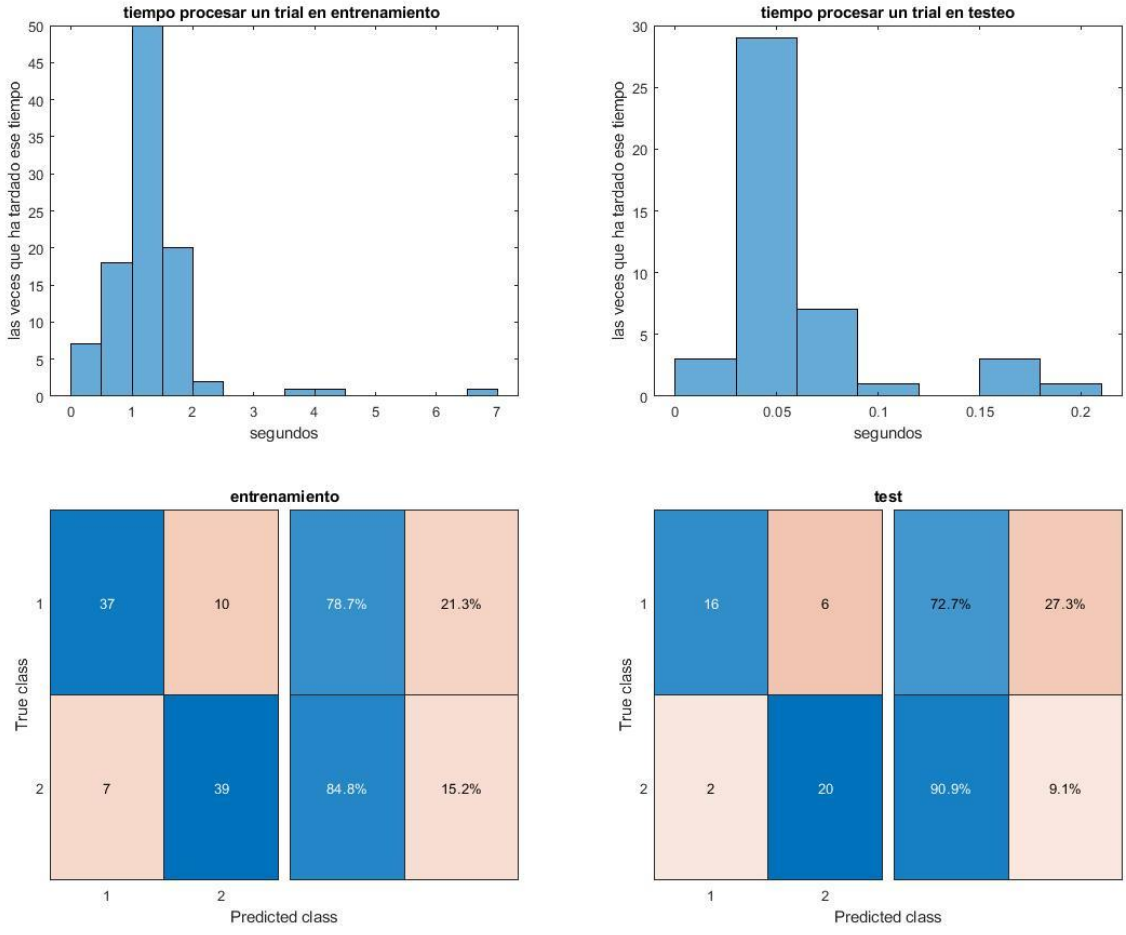


Figura 6-74.Gráficas resultado Logistic Regression.

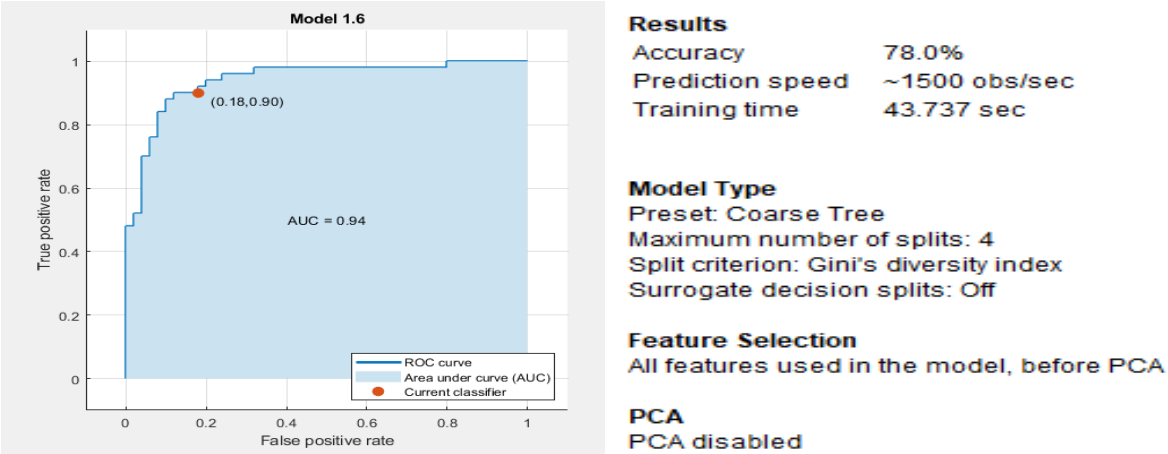


Figura 6-75. Curva ROC Logistic Regression.

6.1.6.1.2 Pruebas con ordenación de matriz de filtros W

tiempo máximo un trial entrenamiento: 12.056496 segundos

tiempo máximo un trial test: 0.061613 segundos

validationAccuracy final : 0.888889 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 80.645161 %

Tasa predicción (probabilidad acierto conjunto test) : 81.818182 %

Tasa clasificación errónea entrenamiento: 19.354839%

Tasa clasificación errónea test: 18.181818 %

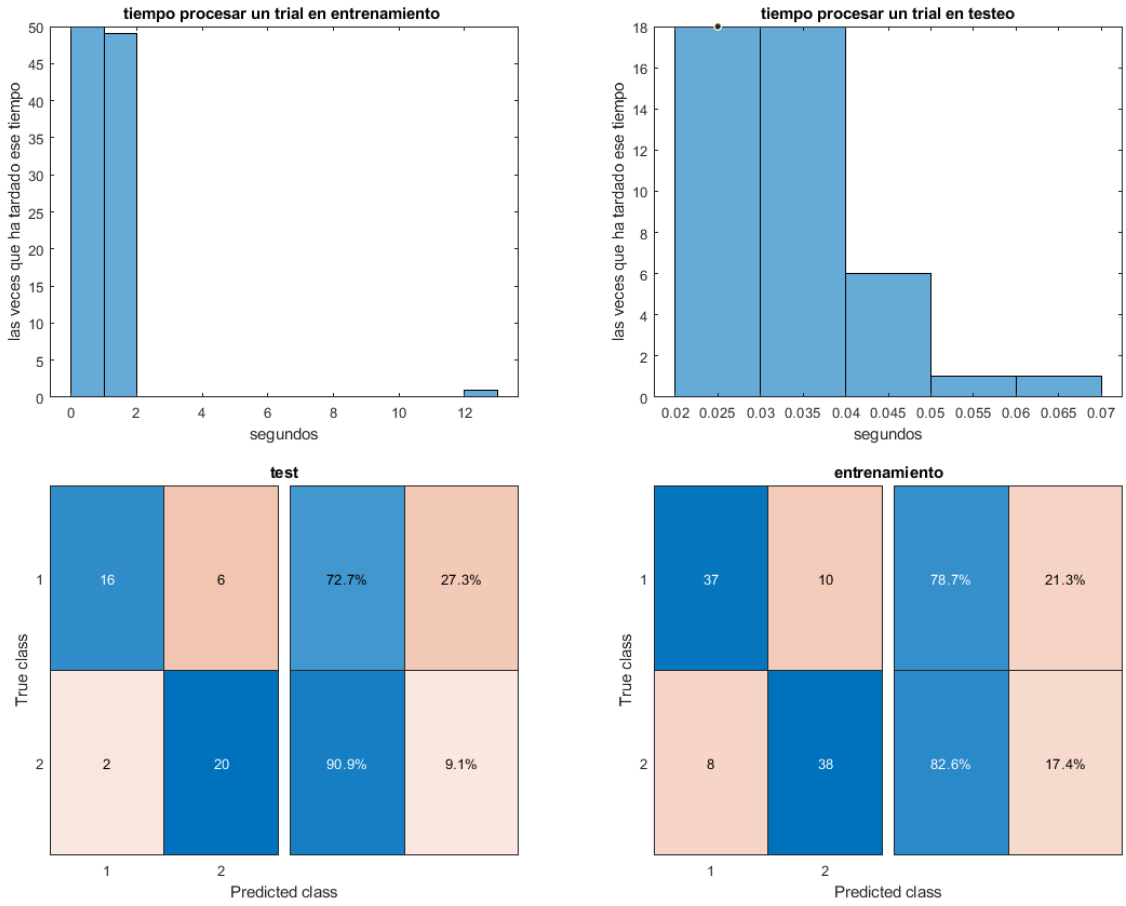


Figura 6-76.Gráficas resultado Logistic Regression.

6.1.7 Ensemble (Conjunto de clasificadores)

Estos clasificadores no serían muy recomendables para un BCI con entrenamiento online, por ser lentos en el entrenamiento; sin embargo, se ha querido comprobar el resultado de algunos de los que dieron buenas probabilidades de acierto cuando se comprobaron en la App.

6.1.7.1 Subspace Discriminant

6.1.7.1.1 Usando App de MATLAB

6.1.7.1.1.1 Pruebas sin ordenar matriz de filtros W

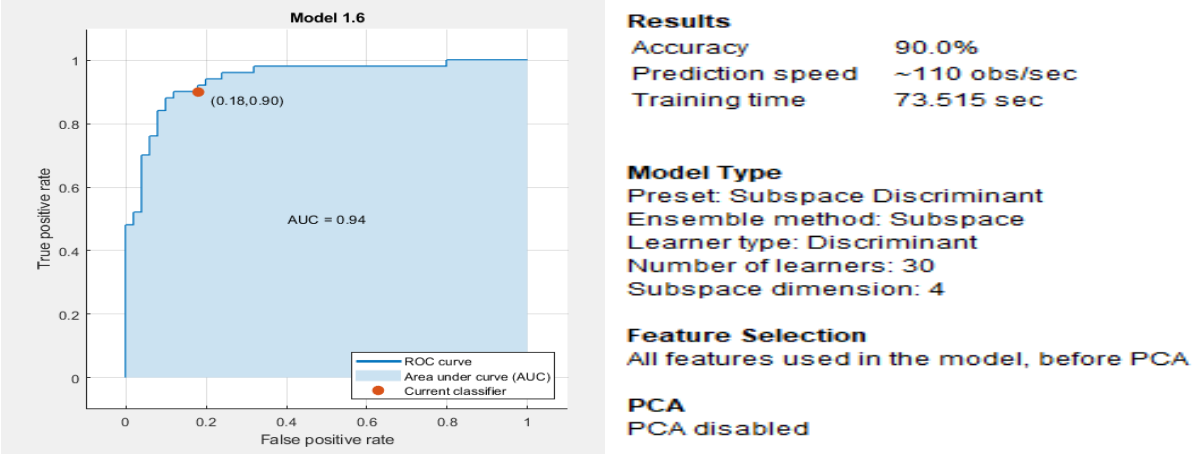


Figura 6-77. Curva ROC Subspace Discriminant.

p_acierto: 0.8 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 19.484968 segundos

tiempo máximo un trial test: 0.577971 segundos

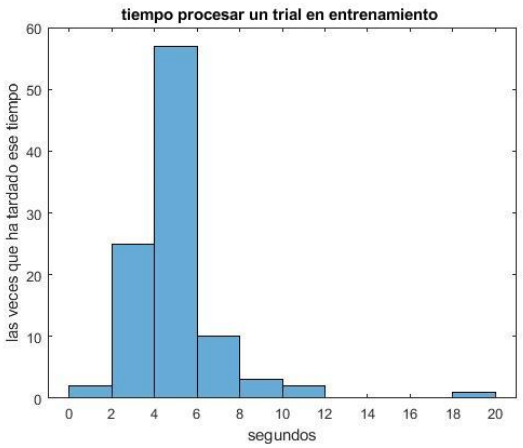
validationAccuracy final : 0.878788 (pu)

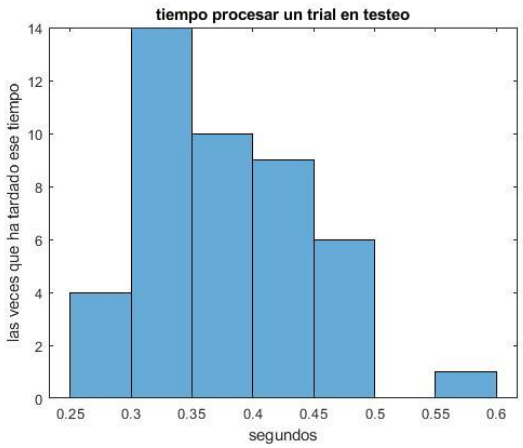
Tasa predicción (probabilidad acierto conjunto entrenamiento) : 79.591837 %

Tasa predicción (probabilidad acierto conjunto test) : 90.909091 %

Tasa clasificación errónea entrenamiento: 20.408163%

Tasa clasificación errónea test: 9.090909 %





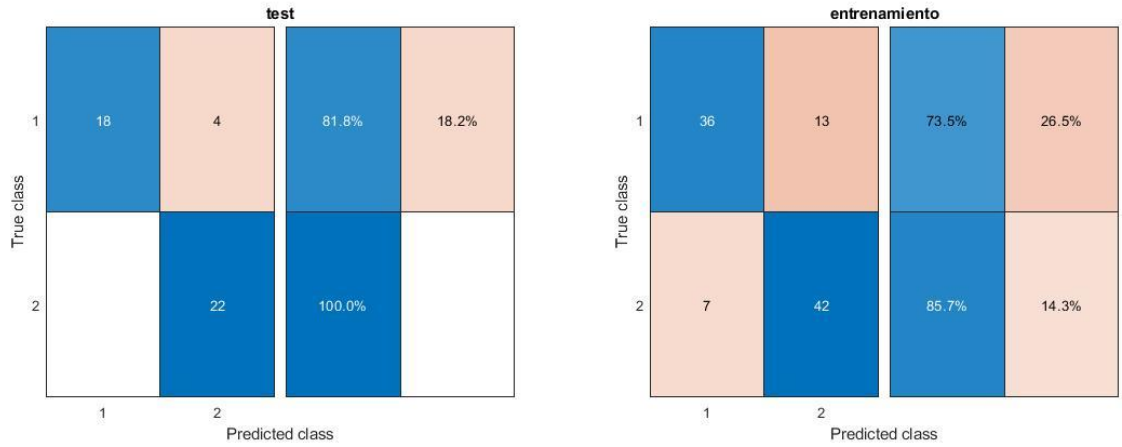


Figura 6-78.Gráficas resultado Subspace Discriminant.

6.1.7.1.1.2 Pruebas con ordenación de matriz de filtros W

tiempo máximo un trial entrenamiento: 29.561961 segundos

tiempo máximo un trial test: 0.438221 segundos

validationAccuracy final : 0.888889 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 80.612245 %

Tasa predicción (probabilidad acierto conjunto test) : 90.909091 %

Tasa clasificación errónea entrenamiento: 19.387755%

Tasa clasificación errónea test: 9.090909 %

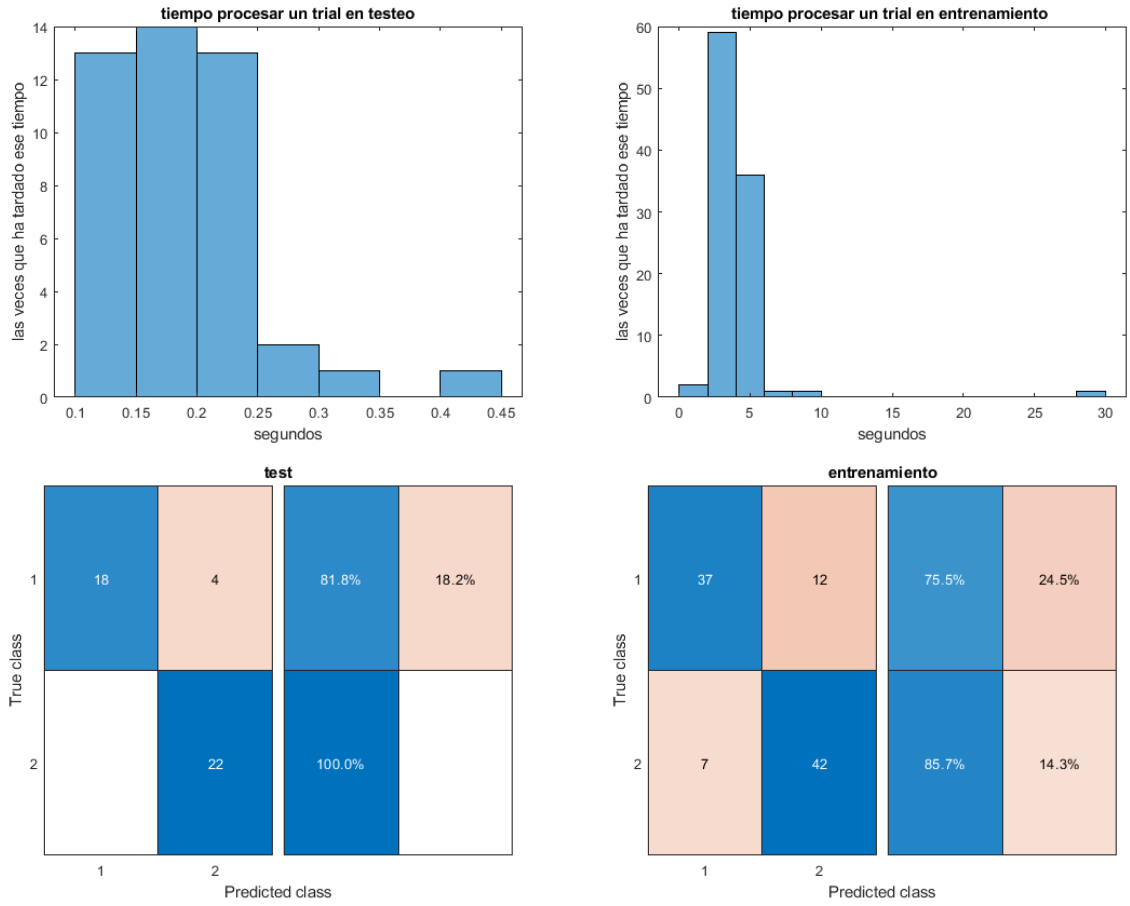


Figura 6-79.Gráficas resultado Subspace Discriminant.

6.1.7.2 Subspace KNN

6.1.7.2.1 Usando App de MATLAB

6.1.7.2.1.1 Pruebas sin ordenar matriz de filtros W

Se comprobó en la App qué clasificador daría mejores resultados con solo 7 ***trials*** de entrenamiento:

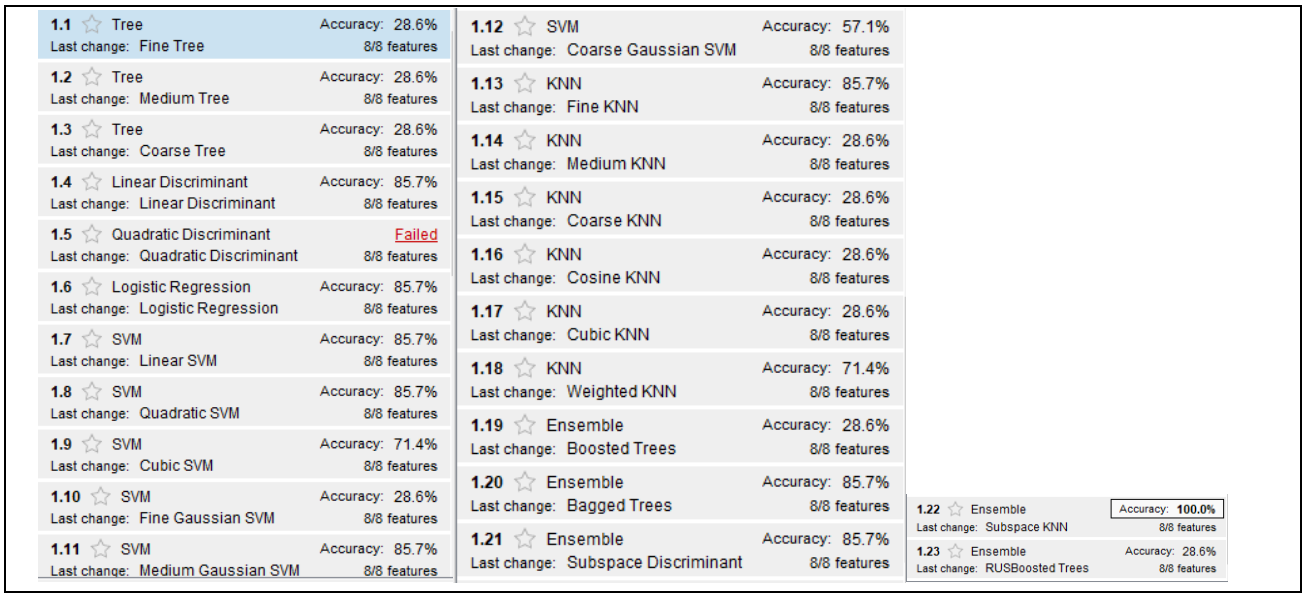


Tabla 6-3. Resultados en la App con 7 ***trials.***

El que dio mejor probabilidad fue Ensemble Subspace KNN.

p_acierto: 0.8 | acierto = 0 | acierto_previo = 1

tiempo máximo un trial entrenamiento: 16.086151 segundos

tiempo máximo un trial test: 0.340682 segundos

validationAccuracy final : 0.838384 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 85.416667 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 14.583333%

Tasa clasificación errónea test: 15.909091 %

Tiene una probabilidad de acierto alta, pero es muy lento el entrenamiento.

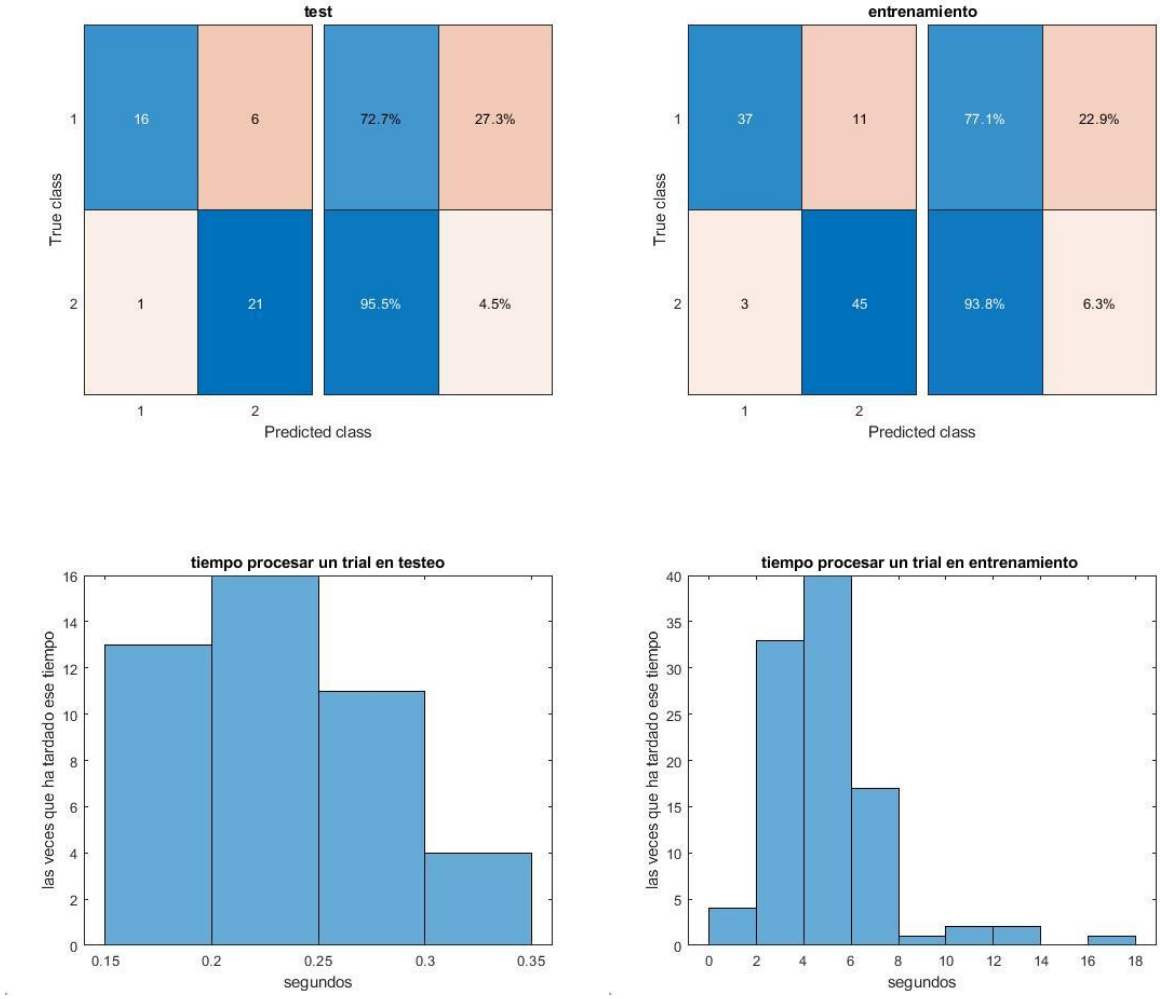


Figura 6-80. Gráficas resultado Subspace KNN.

6.1.7.2.1.2 Pruebas con ordenación de matriz de filtros W

tiempo máximo un trial entrenamiento: 11.526000 segundos

tiempo máximo un trial test: 0.393976 segundos

validationAccuracy final : 0.848485 (pu)

Tasa predicción (probabilidad acierto conjunto entrenamiento) : 85.416667 %

Tasa predicción (probabilidad acierto conjunto test) : 84.090909 %

Tasa clasificación errónea entrenamiento: 14.583333%

Tasa clasificación errónea test: 15.909091 %

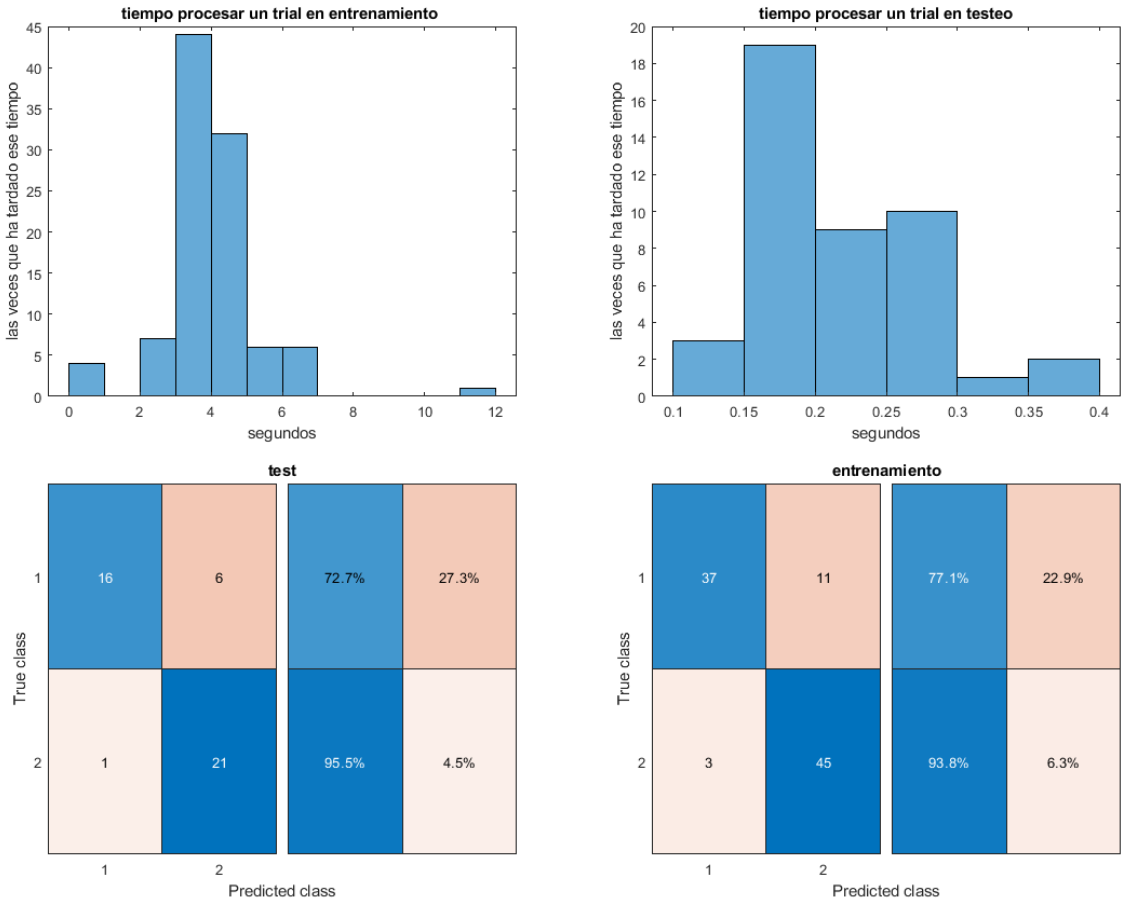


Figura 6-81. Gráficas resultado Subspace KNN.

6.1.8 Conclusiones: un usuario en una sesión

Se hicieron diferentes pruebas con algunos clasificadores, como cambiar el número de filtros; cambiar el número de elementos para entrenamiento y test; cambiar cuántos ensayos hay que almacenar antes de empezar a clasificar; hacerlo con o sin CSP; hacerlo con o sin forzar la ordenación de los filtros; probar diferentes métodos de validación; usar programación sin o con rutinas internas de MATLAB; probar diferentes niveles de podado en los Decision Tree; probar diferentes valores de *k* en el KNN; y qué pasaría si fuera offline en lugar de online.

Se verificó con algunos clasificadores que, cuando se prueba el clasificador final obtenido con el conjunto de datos del entrenamiento, se consigue una probabilidad de acierto mayor que la obtenida en el testeo y que la obtenida en el entrenamiento online. El entrenamiento online se adapta de forma dinámica al número de ensayos disponible en cada iteración. Al no disponer de todos los ensayos al principio, éste tiende a infravalorar los

resultados de acierto en entrenamiento sobre el conjunto completo de datos. Por esta razón, la probabilidad de acierto en test, en general, puede llegar a ser superior a la del entrenamiento online. Sin embargo, eso no supone una contradicción teórica en el funcionamiento de los algoritmos.

En el LDA la covarianza media total se probó a calcularla de dos formas. El primer método era igualándola a la covarianza de la matriz de características. El segundo método era igualándola a la suma ponderada de las covarianzas de cada clase, ponderando con las probabilidades de cada clase, como se explicó en apartado 4.4. Varió un poco la probabilidad de acierto en el entrenamiento online, pero la de testeo permaneció igual. Se podría suponer que no afecta mucho realizar el cálculo de una manera o de la otra.

Se comprobó también que, si se quitaba la ordenación de la matriz de filtros antes de aplicar el entrenamiento del LDA, y solo "se establecía al principio, cuando se obtiene la matriz de filtros con los ensayos en los que hay que esperar que haya un mínimo de uno de cada clase, se obtenían los mismos resultados que cuando se fuerza la ordenación en cada iteración". Se comprobó que se mantenían ordenados en las actualizaciones de la matriz de filtros. Por otra parte, se vio que, si se quitaba también la ordenación en los primeros ensayos, empeoraban los resultados del clasificador LDA. Se podría considerar que se reduce el efecto del CSP.

Algunos clasificadores daban mismos resultados de probabilidad de acierto en las pruebas de con o sin comando de ordenar la matriz de filtros, podría deberse a que se acaban obteniendo los mismos valores de los parámetros para la clasificación, no se ven afectados por esta modificación.

Además, se verificó que al aumentar el número de filtros en LDA, la probabilidad de acierto en el testeo bajaba. Por otra parte, con respecto al hecho de usar o no el CSP, se comprobó que en varios clasificadores afectaba negativamente a la probabilidad de acierto el no usarlo.

Se comprobó cómo varía la probabilidad de acierto en entrenamiento online y test según el número de ensayos que se cojan para cada grupo. Cuantos más ensayos se cojan para el entrenamiento, mayor será la probabilidad de acierto en entrenamiento y en testeo. Si hay pocos ensayos para el entrenamiento, los resultados serán aleatorios. También hay que considerar que el resultado de probabilidad no será fiable si hay pocos ensayos para testear el algoritmo. "Tiene que haber una cantidad suficientemente grande tanto para entrenamiento como para test. El objetivo es conseguir una probabilidad de testeo alta y fiable."

Con respecto al método de testeo, con cross-validation a veces daba mejor y otras veces peor probabilidad de acierto, en comparación con el otro método de testeo utilizado, el cual es simplemente probar el clasificador con un conjunto de datos y hallar la probabilidad de aciertos.

Se muestran a continuación unas tablas resumen de las probabilidades de acierto obtenidas con 100 ensayos para entrenamiento online y 44 para el testeo. Son los resultados obtenidos con el CSP forzando la ordenación del filtro y sin forzarla.

- Pruebas sin ordenar matriz de filtros **W**:

Clasificador	P.A. Entr. Online (%)	P.A. Test.(%)	T.max Ent. (s)	T.max Test (s)
LDA paso	77.55	84.09	0.87	0.01
LDA función linear	88.67	81.82	2.35	0.11
LDA función diaglinear	60.82	88.64	7.32	0.072
LDA función Pseudolinear	87.63	81.82	1.82	0.10
LDA App	85.41	84.09	12.03	0.16
QDA función	76.25	81.81	2.18	0.13
QDA App	72.73	81.81	5.85	0.36
Bayes paso	55.10	65.91	0.56	0.09

Gaussian Naive Bayes func.	54.17	61.36	4.83	0.14
Kernel Naive Bayes Gaussiano func.	81.25	88.64	3.68	0.19
KNN func. k=1	77.55	86.36	1.62	0.06
KNN func. k=7	84.69	90.91	2.24	0.028
KNN func. k=10	83.67	90.91	2.00	0.028
Tree func. Sin podar	72.45	84.09	2.05	0.07
Tree func. Nivel 3	35.71	50.00	3.28	0.03
Tree func. Nivel 1	74.49	97.73	8.07	0.05
Tree App fine	72.45	84.09	5.53	0.22
Tree App medium	72.45	84.09	5.46	0.063
Tree App Coarse	72.45	97.73	6.95	0.057
SVM función lineal	82.65	88.64	9.32	0.115
SVM func. Con kernel gaussiana	83.67	88.64	2.40	0.06
SVM func. Con kernel polinomial	78.57	81.82	4.54	0.03
SVM App linear	79.59	88.63	7.81	0.107
Logistic Regression App	81.72	81.82	6.8	0.19
Subspace Discriminant App	79.59	90.91	19.48	0.58
Subspace KNN App	85.42	84.09	16.09	0.34

Tabla 6-4. Resultados un usuario sin forzar ordenación de la matriz de filtros.

El mejor resultado en probabilidad de acierto en entrenamiento online ha sido el obtenido con LDA (lineal), usando la función de MATLAB. En lo que se refiere al testeo, el Decision Tree Coarse y el Decision Tree de nivel 1 dan la mayor probabilidad de acierto, es relevante tener más en consideración la de testeo, que la de entrenamiento. Es importante destacar que en esos dos Decisión Tree se acababa obteniendo el mismo esquema de árbol de decisión, pero el Coarse se realizó aplicando la App y el otro con funciones de MATLAB.

Tener presente que, como ya se comentó, la probabilidad de acierto del entrenamiento online es más baja que la de testeo debido a que al principio se tenía pocos datos, por tanto, los resultados del principio eran malos.

En general, en esta sesión se consiguieron probabilidades de acierto altas en la mayoría de los clasificadores, excepto Decision Tree de nivel 3 (es un árbol completamente podado, clasificador aleatorio) y Naive Bayes.

En lo que se refiere al tiempo máximo de entrenamiento, los que dan valores menores son los que se programaron paso por paso, LDA y Naive Bayes Gaussiano, sin usar las funciones de MATLAB para clasificación. El de LDA es, además, el que menos tarda en el testeo. No obstante, los tiempos calculados pueden variar según las condiciones del ordenador en que se ejecute. Lo que se obtiene es una idea aproximada de lo máximo que podría llegar a tardar su ejecución.

- Pruebas ordenando matriz de filtros **W**:

Clasificador	P.A. Entr. online(%)	P.A. Test.(%)	T.max Ent (s)	T.max Test (s)
LDA paso	80.61	86.36	0.15	0.11
LDA función linear	84.53	84.09	4.42	0.01
LDA función diaglinear	61.85	88.64	2.17	0.05
LDA función Pseudolinear	83.50	81.82	2.08	0.09
LDA App	85.41	86.36	8.46	0.14
QDA función	76.25	81.81	5.74	0.08
QDA App	74.02	81.81	13.43	0.31

Bayes paso	54.08	65.91	1.58	0.09
Kernel Naive Bayes Gaussiano func.	82.29	88.64	6.61	0.16
KNN func. k=7	84.69	90.91	7.12	0.09
Tree func. Sin podar	72.45	84.09	2.67	0.08
Tree func. Nivel 1	74.49	97.73	8.70	0.03
Tree App Coarse	72.45	97.73	18.9	0.05
SVM func. Con kernel gaussiana	82.65	88.64	2.42	0.08
SVM App linear	80.61	88.64	11.1	0.07
Logistic Regression App	80.64	81.82	12.05	0.06
Subspace Discriminant	80.61	90.91	29.56	0.44
Subspace KNN	85.42	84.09	11.53	0.34

Tabla 6-5. Resultados un usuario con ordenación de la matriz de filtros.

En conclusión, en la sesión de este usuario, los mejores clasificadores, en lo que respecta a probabilidad de acierto, son el Decision Tree de nivel 1 y el Coarse Tree.

No es muy significativo comprobar resultados con una sola sesión, por ello se realizarán pruebas con diferentes sesiones y usuarios a continuación.

6.2 Un conjunto de usuarios

Se mostrará el valor medio de probabilidad de acierto en entrenamiento online y en testeo de las sesiones con cada clasificador. Además, se mostrará la probabilidad acierto media de cada usuario, sabiendo que en la base de datos venían las dos sesiones de cada usuario guardada de manera seguida. También se mostrará el histograma de tiempo máximo de las sesiones en el entrenamiento y el testeo. Adicionalmente, se graficará un diagrama de caja de la probabilidad de acierto en entrenamiento.

6.2.1 LDA

Se usa **mi_script_Isabel_BUFFER_LDA_usuarios.m** y

mi_script_Isabel_BUFFER_funcion_lda_lineal_usuarios.m.

6.2.1.1 Usando una sesión de mismo usuario como entrenamiento y otra sesión como testeo.

En el siguiente experimento, se quiso comprobar los resultados que da el LDA cuando se considera una sesión del mismo usuario para entrenamiento online (144 ensayos) y otra de testeo (144 ensayos). Los resultados gráficos de probabilidad de acierto muestran la media de cada usuario. Además, en este caso, los histogramas muestran el tiempo máximo medio de cada usuario. Este tipo de prueba solo se realizó con el LDA hecho paso a paso, para comprobar que resultados saldrían.

6.2.1.1.1 Código realizado paso por paso

6.2.1.1.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 69.0 Â± 5.0

Tiempo máx. medio entrenamiento: 0.141225 Â± 0.043212

Porcentaje medio de acierto test: 68.0 Â± 4.5

Tiempo máx. medio test: 0.081111 Â± 0.014679

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual (media test y entrenamiento): 68.5 Â± 4.4

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual (media test y entrenamiento): 0.111168 Â± 0.023783

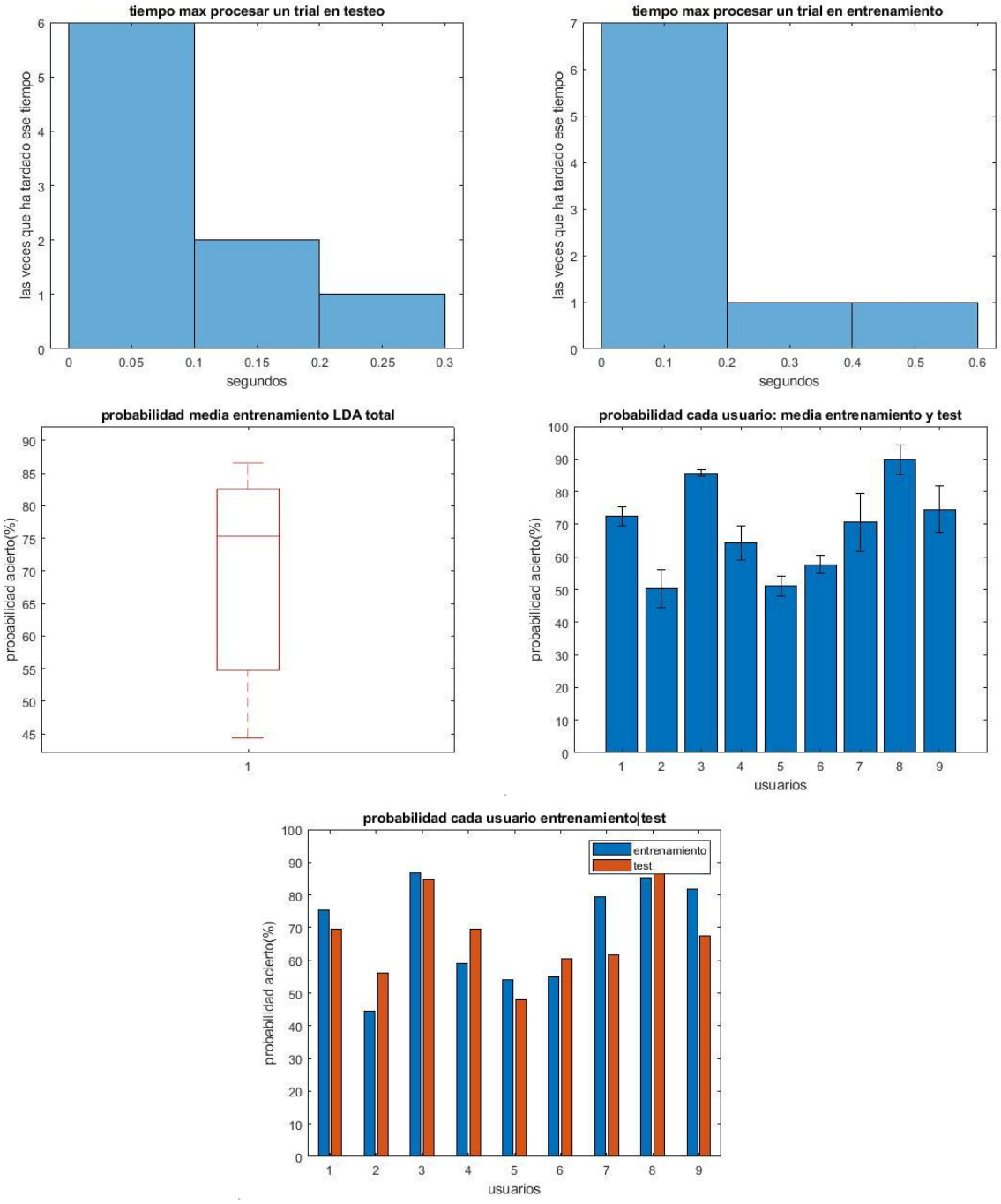


Figura 6-82. Gráficas resultado varios usuarios LDA con 144 para entrenamiento y 144 para testeo.

6.2.1.2 Usando 100 de la sesión como entrenamiento y 44 como testeo.

Los siguientes experimentos son semejantes a las pruebas realizadas con un usuario, pero siendo cada resultado una media de varias sesiones, con distintos usuarios, como ya se comentó. El mismo procedimiento de análisis se llevará a cabo con los demás clasificadores.

6.2.1.2.1 Código realizado paso por paso

6.2.1.2.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 65.8 Â± 3.7

Tiempo máx. medio entrenamiento: 0.060929 Â± 0.009300

Porcentaje medio de acierto test: 65.5 Â± 3.2

Tiempo máx. medio test: 0.015137 Â± 0.005073

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 65.8 Â± 4.5

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.060929 Â± 0.008346

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 65.8 Â± 4.5

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.060929 Â± 0.008346

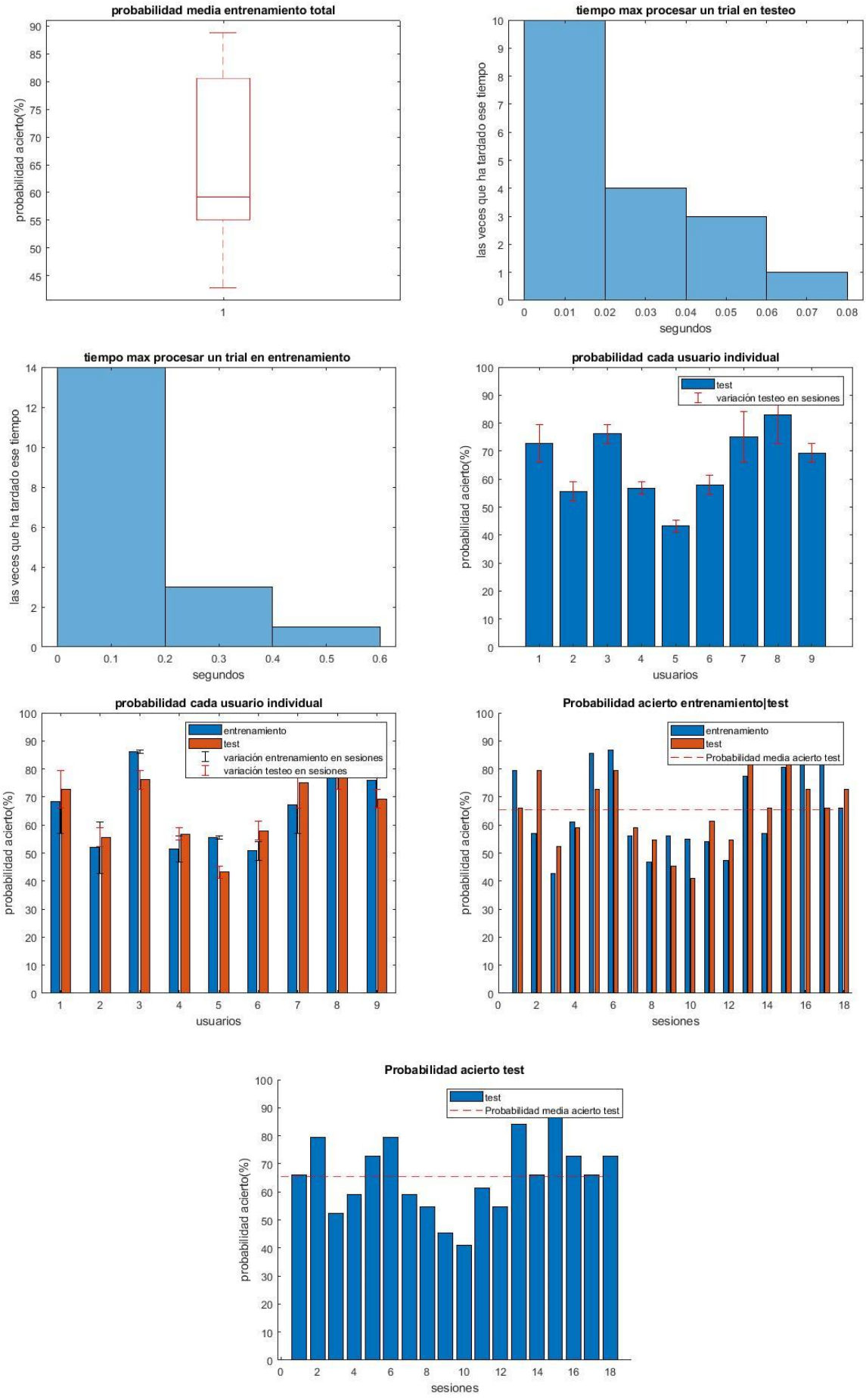


Figura 6-83. Gráficas resultado de varios usuarios con LDA paso a paso.

6.2.1.2.1.1.1 Prueba de aplicar testeo a los datos de entrenamiento

Se probará el clasificador final obtenido con los datos del entrenamiento. Para su realización se usa el archivo **my_repit_sesiones_testing.m.**, el cual se le indicará que llame a la función **mi_script_Isabel_BUFFER_3_testeof.m** para que sea el caso de LDA realizado paso a paso. Si se quiere realizar este tipo de prueba a modelos de entrenamiento realizados con funciones de MATLAB, hay que usar **mi_script_Isabel_BUFFER_funcion_testing_usuarios.m**. Como en el caso de un usuario, los ensayos que había que esperar a poseer antes de empezar a clasificar, se están teniendo en cuenta.

Porcentaje medio de acierto test: 77.2 Â± 3.7

Tiempo máx. medio test: 0.067048 Â± 0.011779

La probabilidad de testeo de los datos de entrenamiento es en torno al 77.2%, es mayor a la probabilidad de testeo de 65.5% con datos desconocidos.

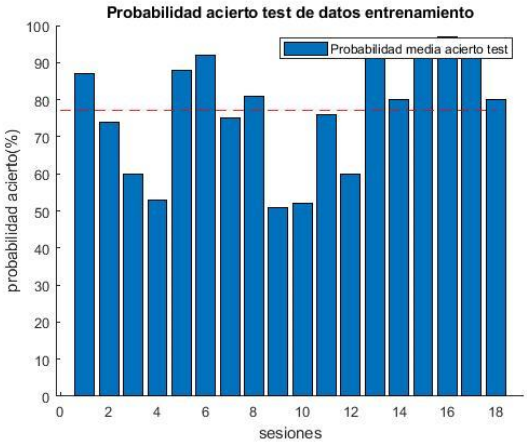


Figura 6-84. Probabilidad de acierto testeando datos de entrenamiento con LDA.

6.2.1.2.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 65.9 Â± 3.7

Tiempo máx. medio entrenamiento: 0.110266 Â± 0.030282

Porcentaje medio de acierto test: 64.6 Â± 3.4

Tiempo máx. medio test: 0.107208 Â± 0.045484

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 65.9 Â± 4.5

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.110266 Â± 0.030555

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 64.6 Â± 4.0

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.107208 Â± 0.060506

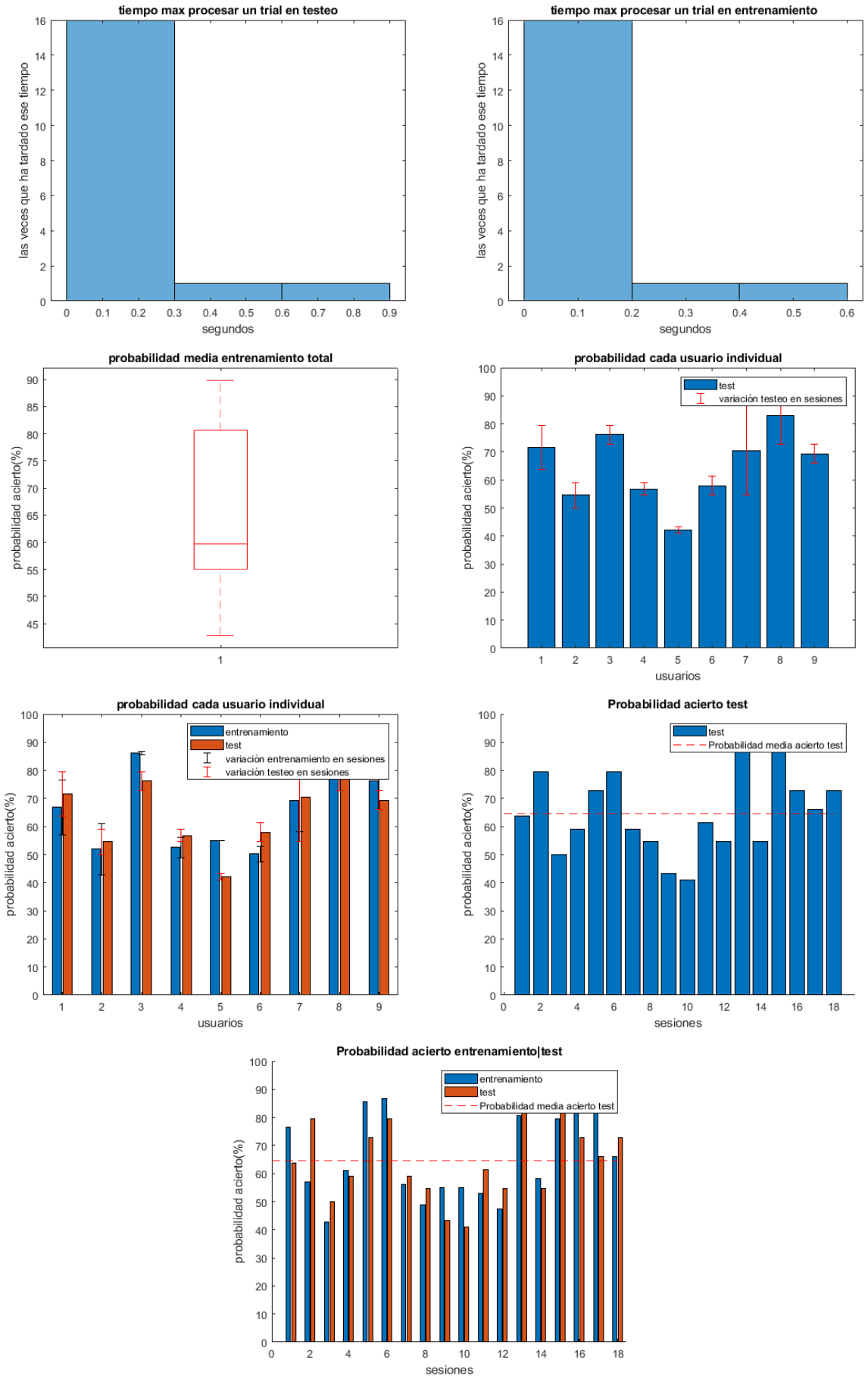


Figura 6-85. Gráficas resultado de varios usuarios con LDA paso a paso.

6.2.1.2.1.2.1 Prueba de aplicar testeo a los datos de entrenamiento

Porcentaje medio de acierto test: 77.0 Â± 3.7

Tiempo máx medio test: 0.084694 Â± 0.012579

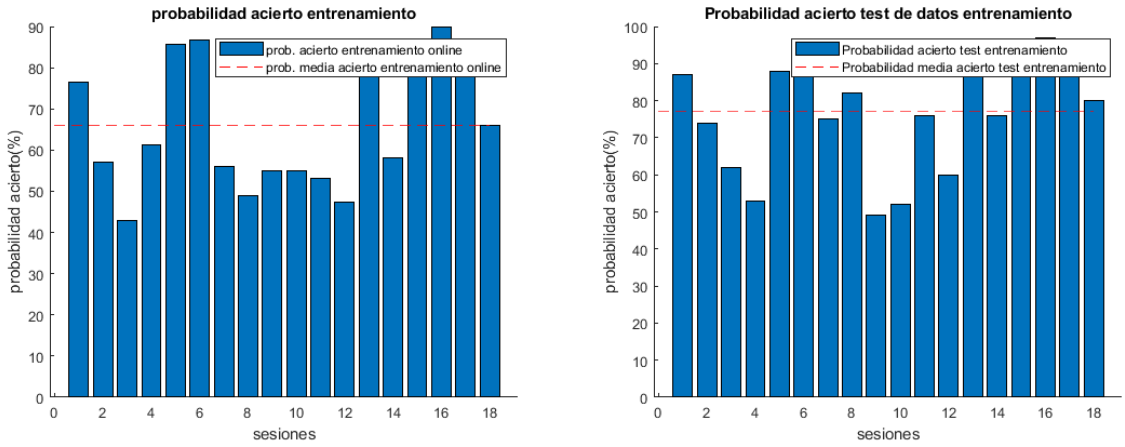


Figura 6-86. Probabilidad de acierto testeando datos de entrenamiento con LDA.

6.2.1.2.1.3 Prueba realizada sin aplicar CSP

Porcentaje medio de acierto entrenamiento: 63.4 Â± 2.2

Tiempo max medio entrenamiento: 0.058305 Â± 0.008721

Porcentaje medio de acierto test: 67.8 Â± 2.8

Tiempo máx. medio test: 0.023626 Â± 0.006873

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 63.4 Â± 2.7

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.058305 Â± 0.009614

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 67.8 Â± 3.3

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.023626 Â± 0.006107

En el testeo, ha salido una probabilidad media de acierto mayor (67.8%) cuando no se usa CSP, usando CSP salió 65.3%.

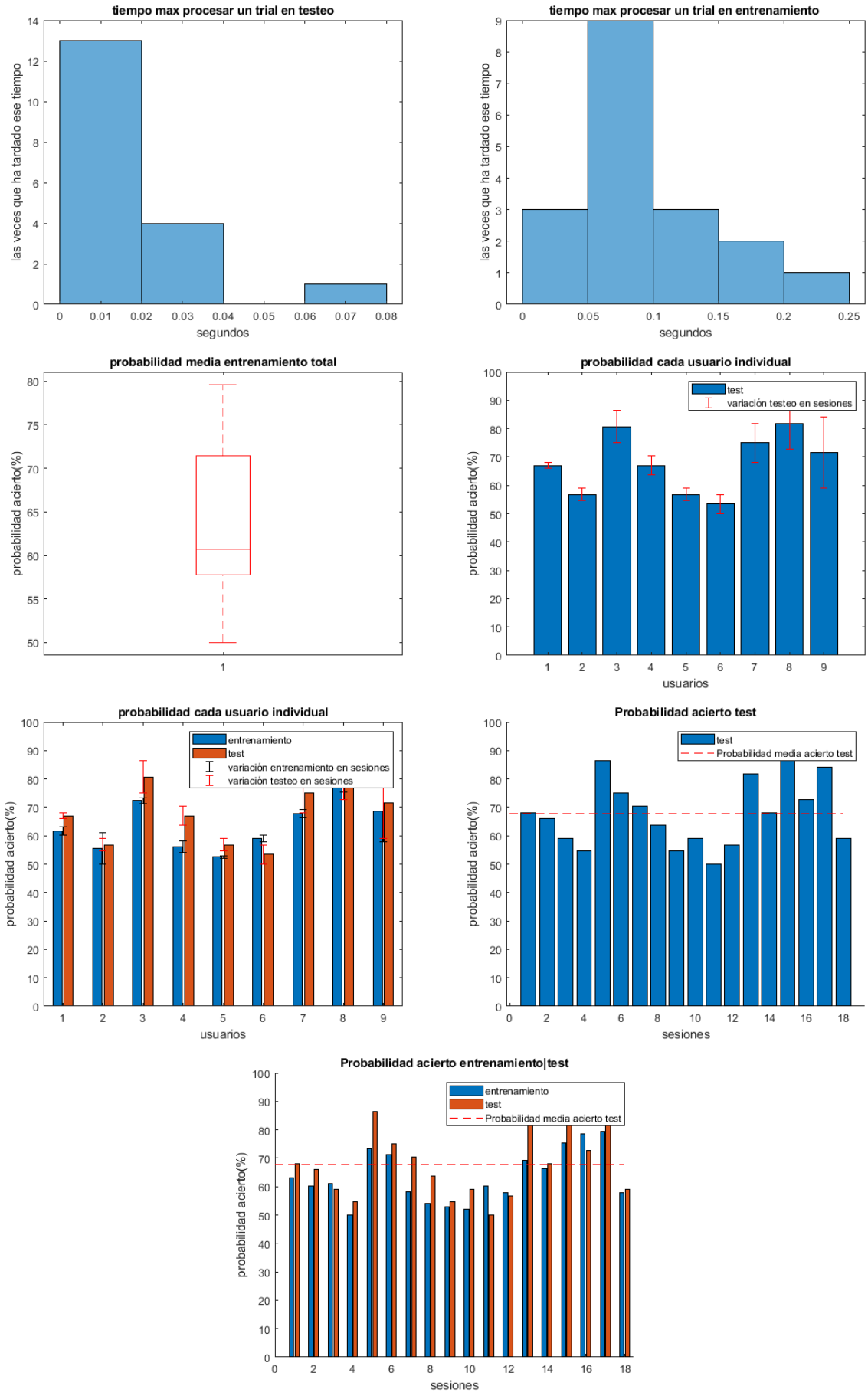


Figura 6-87. Resultados LDA sin CSP para varios usuarios y sesiones.

6.2.1.2.1.3.1 Prueba de aplicar testeo a los datos de entrenamiento

Porcentaje medio de acierto test: 80.8 Â± 2.4

Tiempo máx. medio test: 0.037283 Â± 0.012258

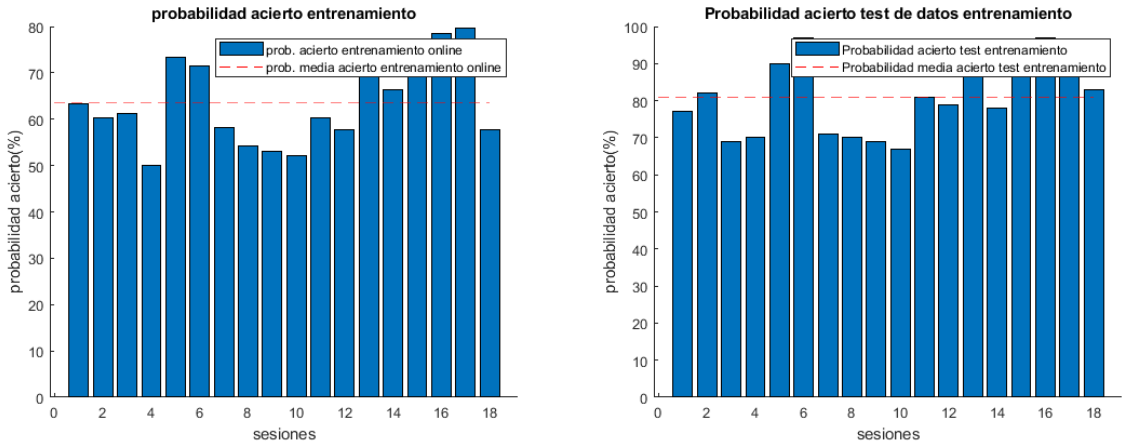


Figura 6-88. Probabilidad de acierto testeando datos de entrenamiento con LDA.

6.2.1.2.2 Aprovechando propiedades de MATLAB

6.2.1.2.2.1 Lineal

6.2.1.2.2.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 66.7 Â± 4.0

Tiempo máx. medio entrenamiento: 0.298162 Â± 0.121473

Porcentaje medio de acierto test: 64.8 Â± 3.4

Tiempo máx. medio test: 0.049313 Â± 0.012622

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 66.7 Â± 4.8

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.298162 Â± 0.126878

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 64.8 Â± 4.3

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.049313 Â± 0.011943

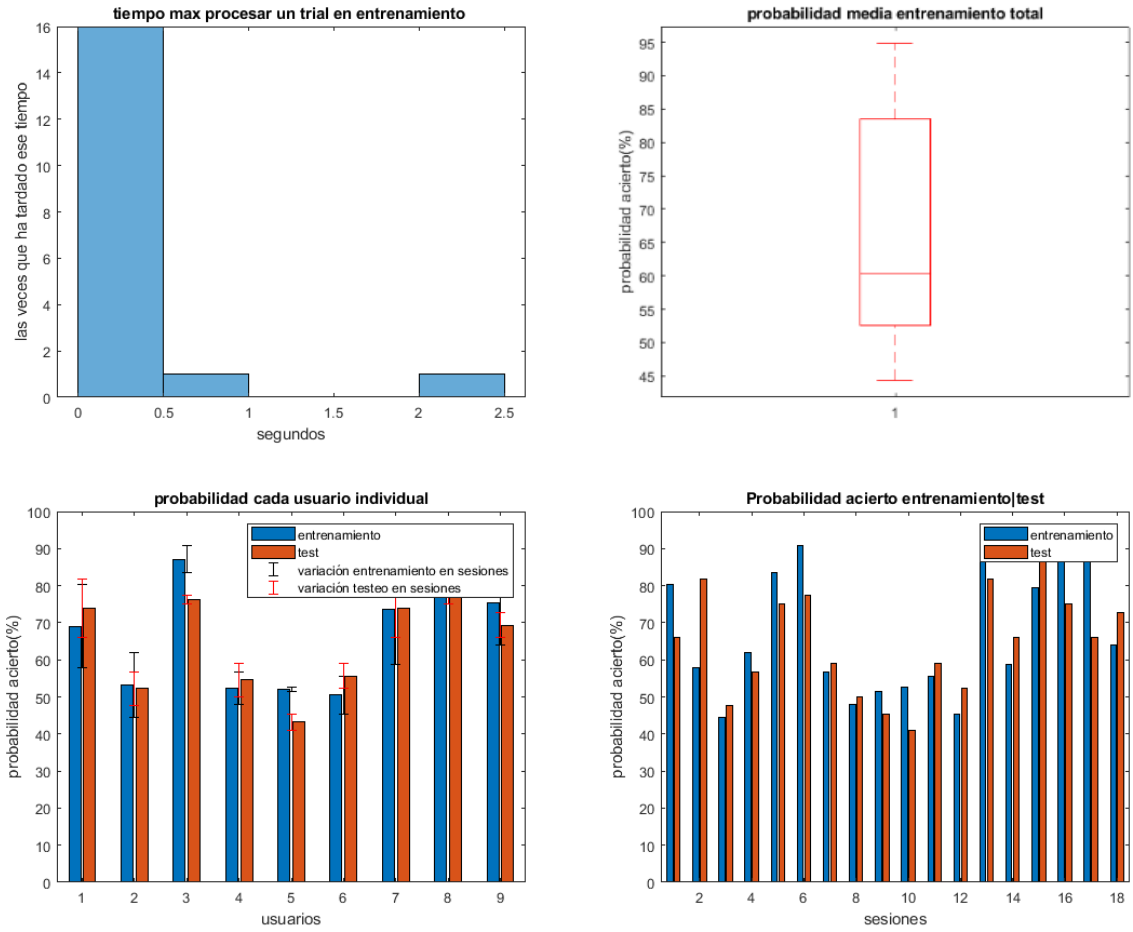


Figura 6-89. Gráficas resultado de varios usuarios con LDA lineal.

6.2.1.2.2.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 66.1 Â± 4.1

Tiempo max medio entrenamiento: 0.616155 Â± 0.419778

Porcentaje medio de acierto test: 64.8 Â± 3.4

Tiempo máx medio test: 0.076150 Â± 0.026435

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 66.1 Â± 4.8

Tiempo máx medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.616155 Â± 0.395308

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 64.8 Â± 4.2

Tiempo máx medio test, teniendo en cuenta media de cada usuario individual: 0.076150 Â± 0.026130

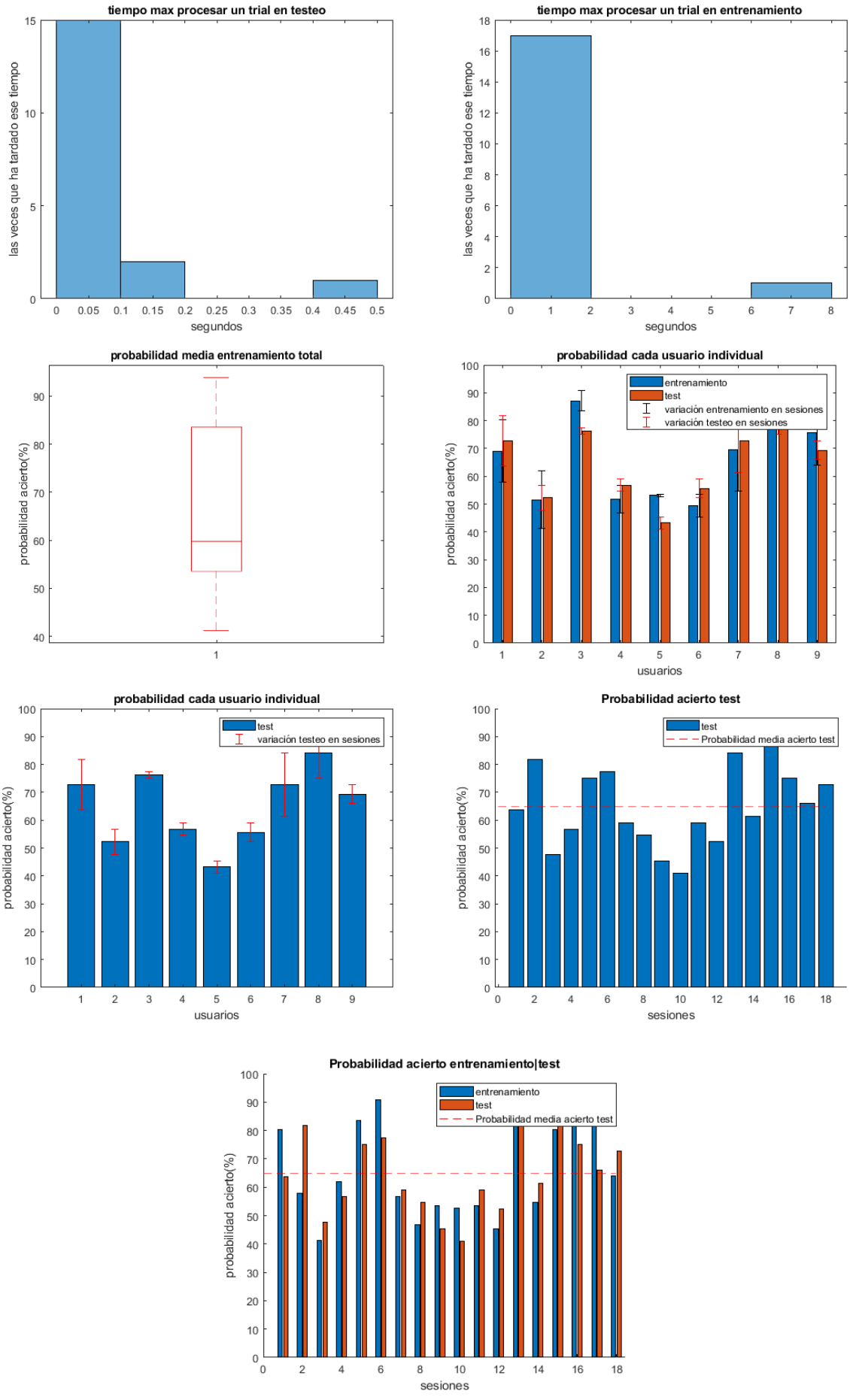


Figura 6-90. Gráficas resultado de varios usuarios con LDA lineal.

6.2.1.2.2.1.2.1 Prueba de aplicar testeo a los datos de entrenamiento

Porcentaje medio de acierto test: 77.8 Â± 3.6 Tiempo máx. medio test: 0.115455 Â± 0.026938

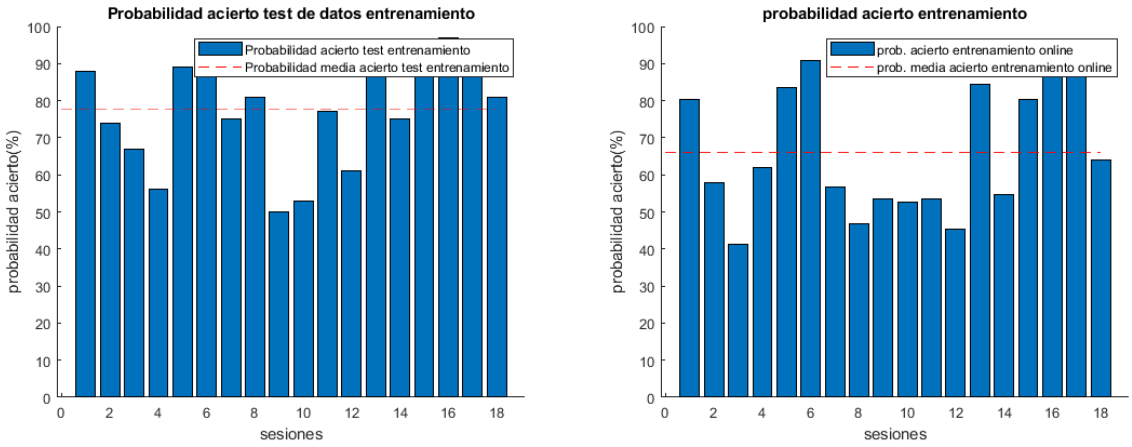


Figura 6-91. Probabilidad de acierto testeando datos de entrenamiento con LDA.

6.2.1.2.2.1.3 Prueba realizada sin aplicar CSP

Porcentaje medio de acierto entrenamiento: 63.2 Â± 2.8 Tiempo máx. medio entrenamiento: 0.684544 Â± 0.462783 Porcentaje medio de acierto test: 67.9 Â± 2.8 Tiempo máx. medio test: 0.068585 Â± 0.014511 Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 63.2 Â± 3.3 Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.684544 Â± 0.439415 Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 67.9 Â± 3.3 Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.068585 Â± 0.018196

La probabilidad de acierto medio en test es del 67.9%, con CSP era de 64.8%.

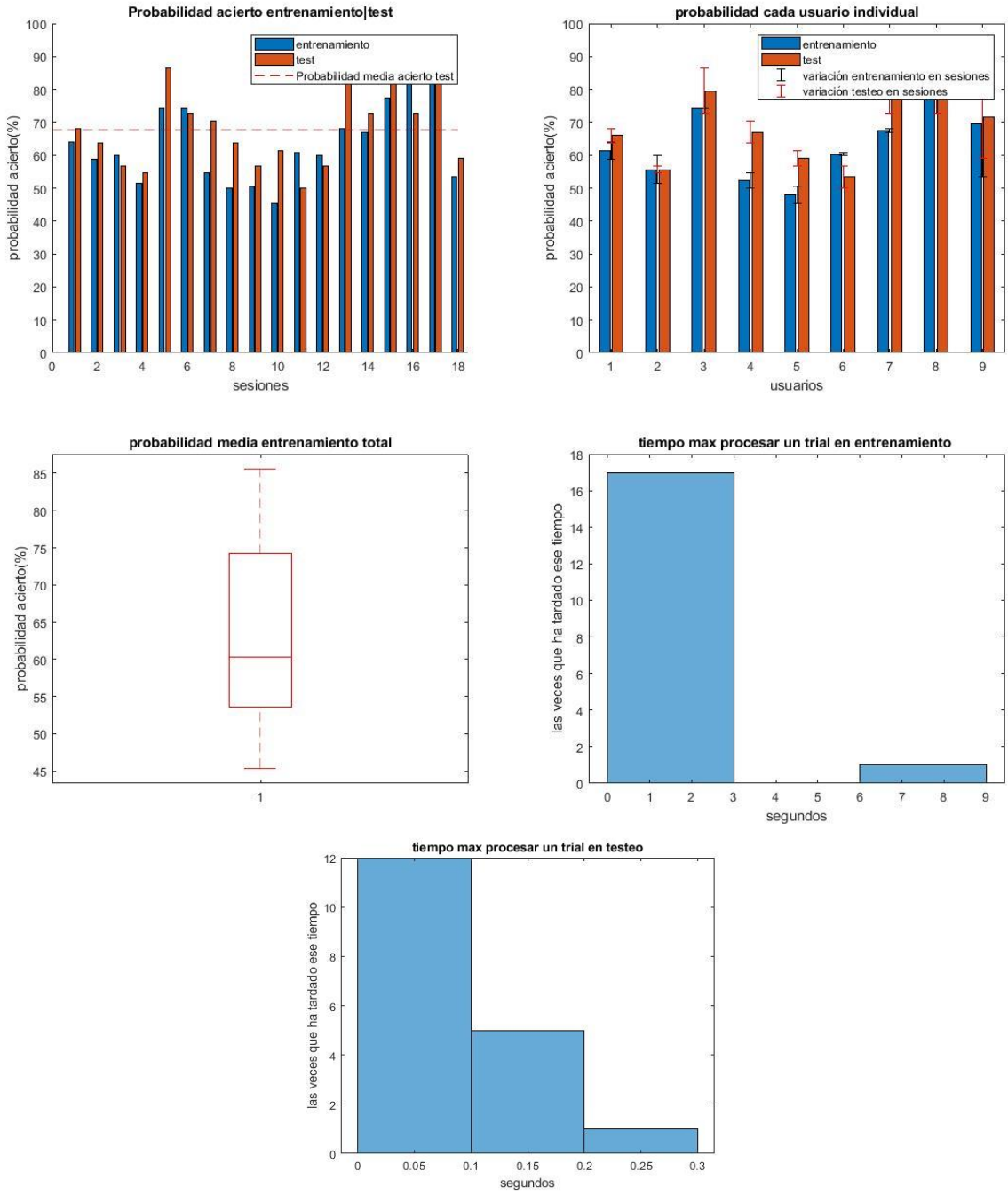


Figura 6-92. Resultado LDA con funciones de MATLAB y sin usar CSP.

6.2.1.2.2.2 Diaglinear

6.2.1.2.2.2.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 59.5 Â± 2.9

Tiempo máx. medio entrenamiento: 0.234936 Â± 0.093709

Porcentaje medio de acierto test: 66.4 Â± 3.9

Tiempo máx. medio test: 0.059964 Â± 0.015975

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 59.5 Â± 3.7

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.234936 Â± 0.088996

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 66.4 Â± 4.7

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.059964 Â± 0.016355

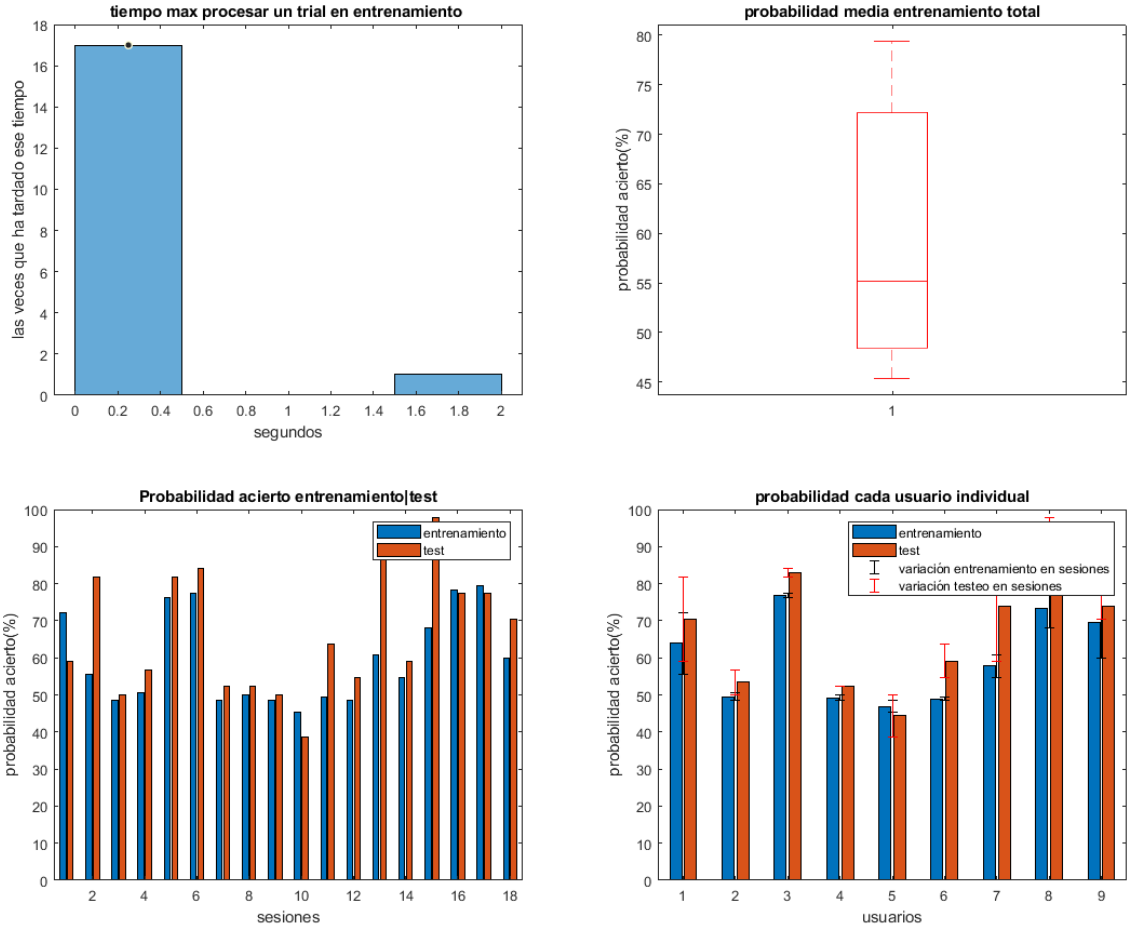


Figura 6-93.Gráficas resultado de varios usuarios con LDA diagllineal.

6.2.1.2.2.2.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 59.7 Â± 2.9

Tiempo máx. medio entrenamiento: 0.406371 Â± 0.117875

Porcentaje medio de acierto test: 66.7 Â± 3.8

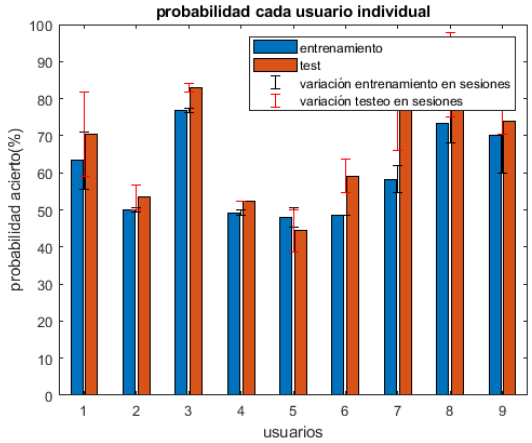
Tiempo máx. medio test: 0.081970 Â± 0.021593

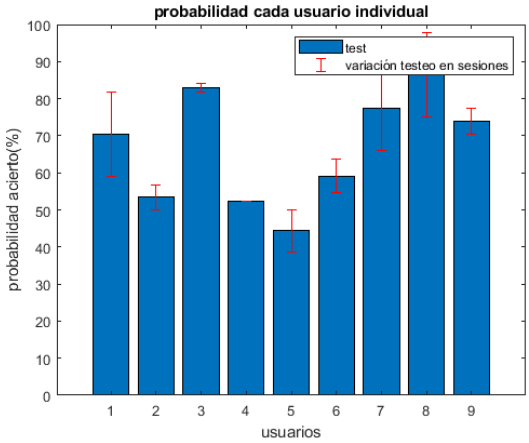
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 59.7 Â± 3.7

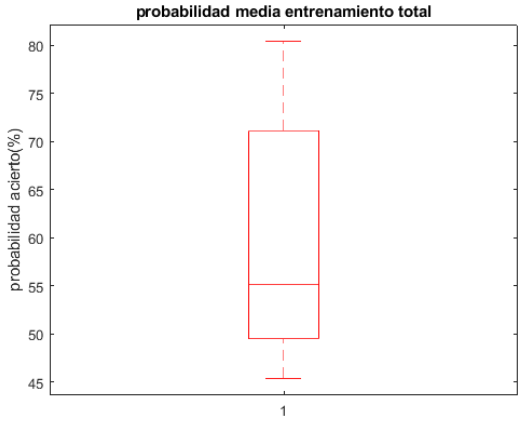
Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.406371 Â± 0.112215

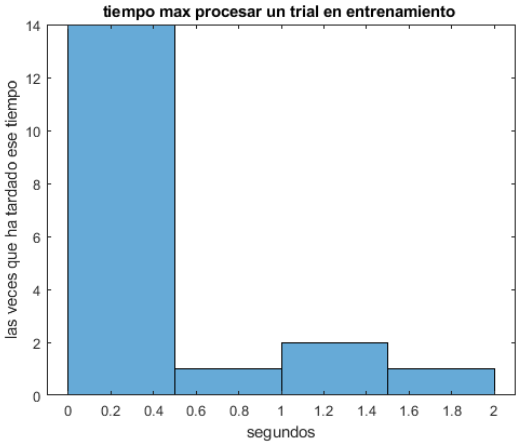
Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 66.7 Â± 4.7

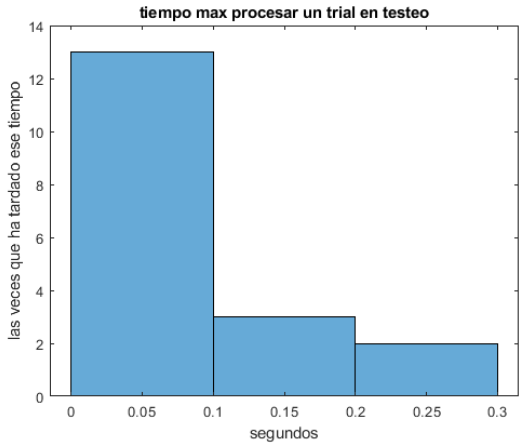
Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.081970 Â± 0.023640

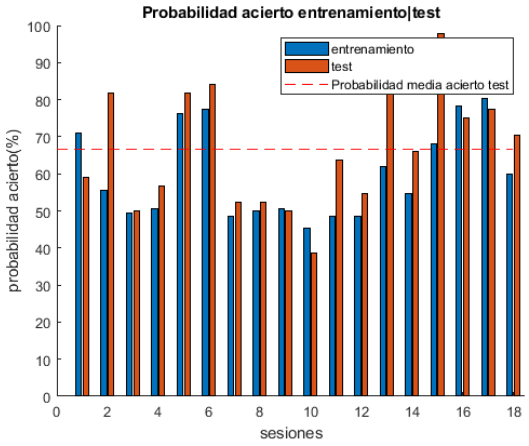












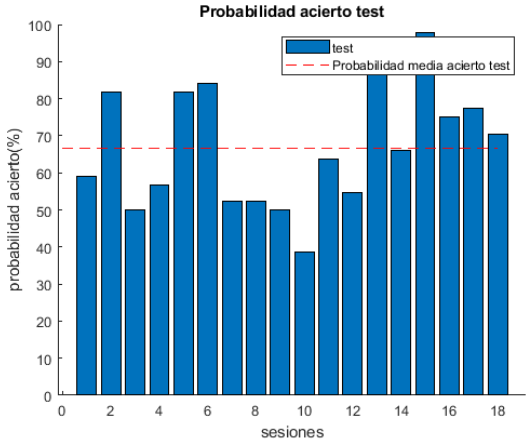


Figura 6-94.Gráficas resultado de varios usuarios con LDA diagllineal.

6.2.1.2.2.3 Pseudolinear

6.2.1.2.2.3.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 67.0 Â± 4.1

Tiempo máx. medio entrenamiento: 0.326468 Â± 0.174463

Porcentaje medio de acierto test: 64.8 Â± 3.4

Tiempo máx. medio test: 0.086346 Â± 0.020873

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 67.0 Â± 4.9

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.326468 Â± 0.174877

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 64.8 Â± 4.3

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.086346 Â± 0.024821

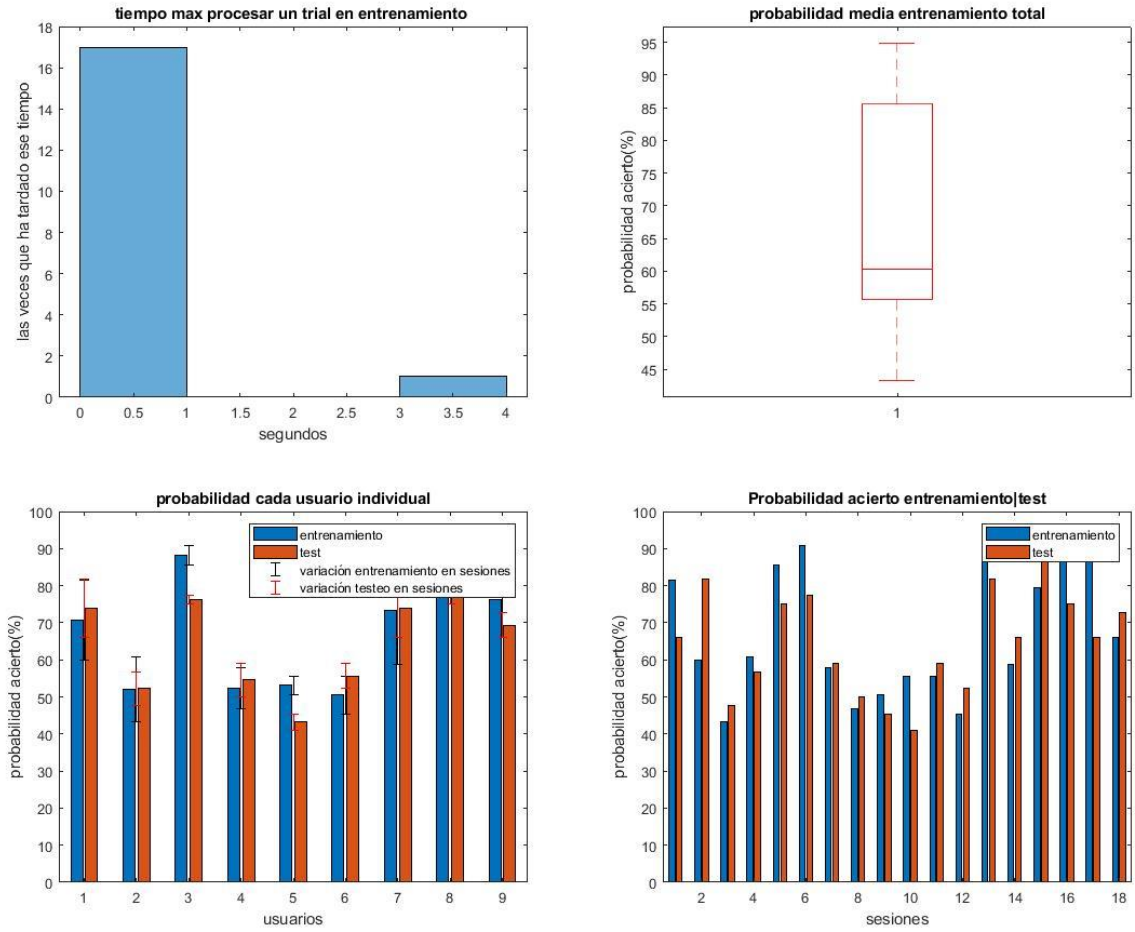


Figura 6-95. Gráficas resultado varios usuarios LDA pseudolineal.

6.2.1.2.2.3.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 66.8 Â± 4.0

Tiempo máx. medio entrenamiento: 0.464551 Â± 0.246802

Porcentaje medio de acierto test: 64.8 Â± 3.4

Tiempo máx. medio test: 0.063218 Â± 0.011760

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 66.8 Â± 4.9

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.464551 Â± 0.241677

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 64.8 Â± 4.2

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.063218 Â± 0.012144

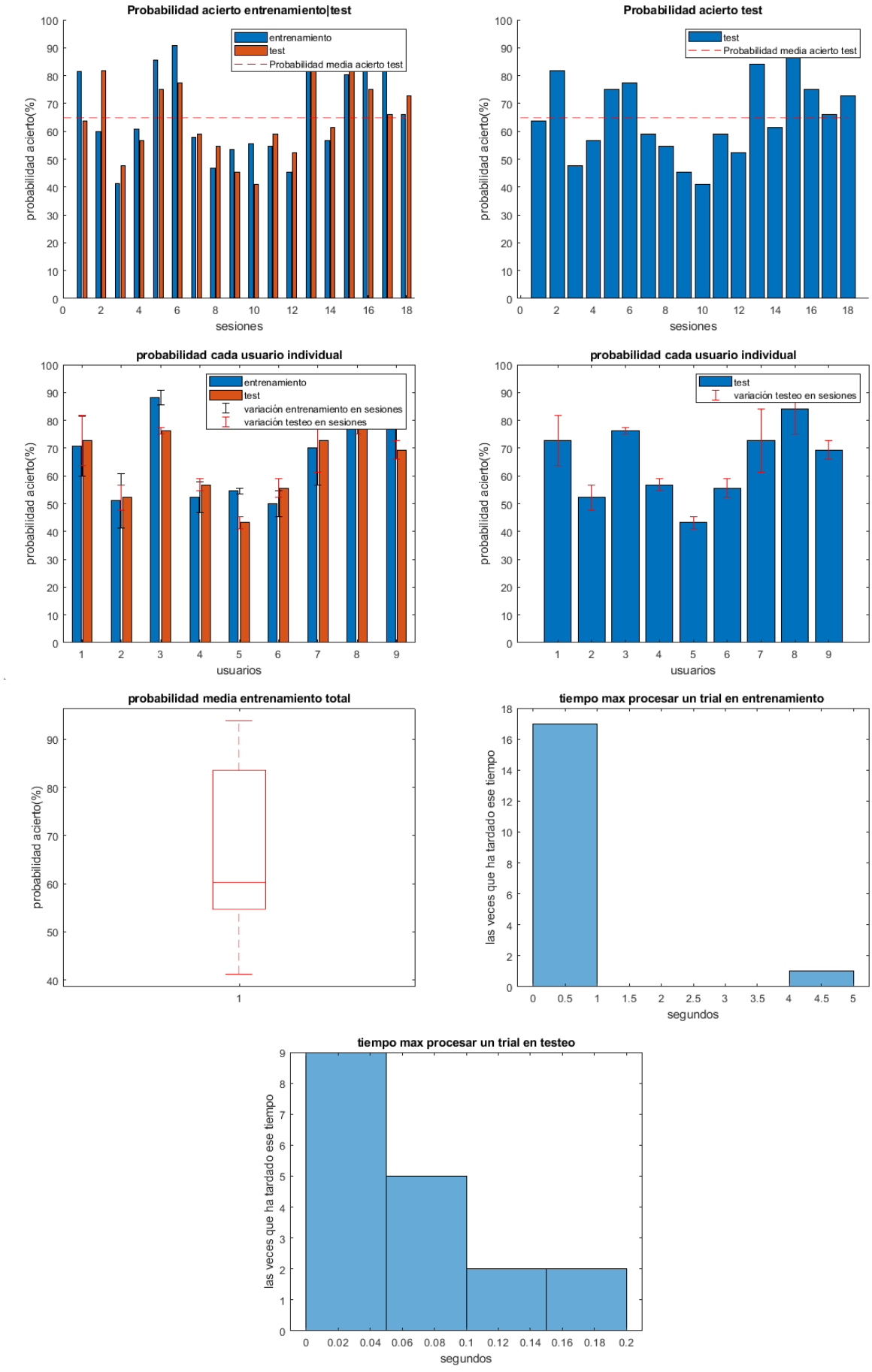


Figura 6-96. Gráficas resultado varios usuarios LDA pseudolineal.

6.2.2 QDA

Se usa **mi_script_Isabel_BUFFER_QDA_usuarios.m**.

6.2.2.1 Usando 100 de la sesión como entrenamiento y 44 como testeo.

6.2.2.1.1 Aprovechando propiedades de MATLAB

6.2.2.1.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 64.6 Â± 3.5

Tiempo máx. medio entrenamiento: 0.237994 Â± 0.094109

Porcentaje medio de acierto test: 65.3 Â± 3.7

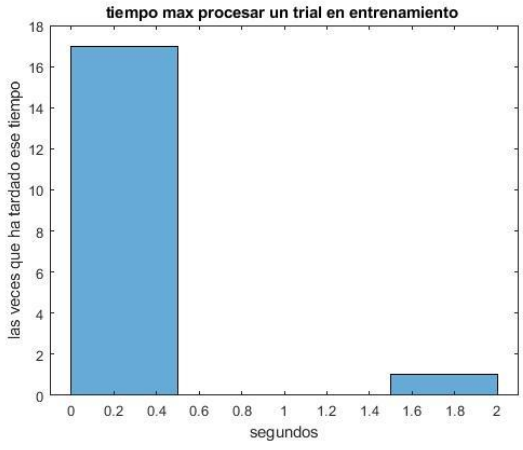
Tiempo máx. medio test: 0.056579 Â± 0.011515

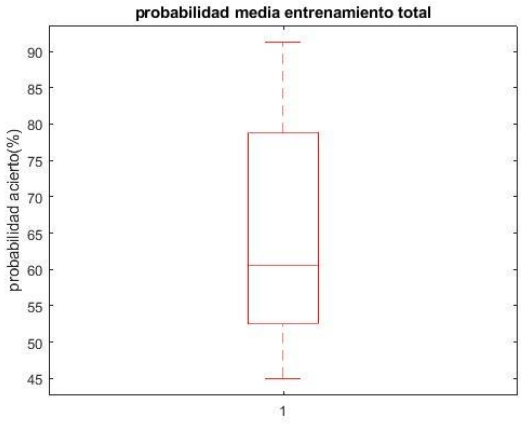
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 64.6 Â± 4.3

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.237994 Â± 0.089636

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 64.6 Â± 4.3

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.237994 Â± 0.089636





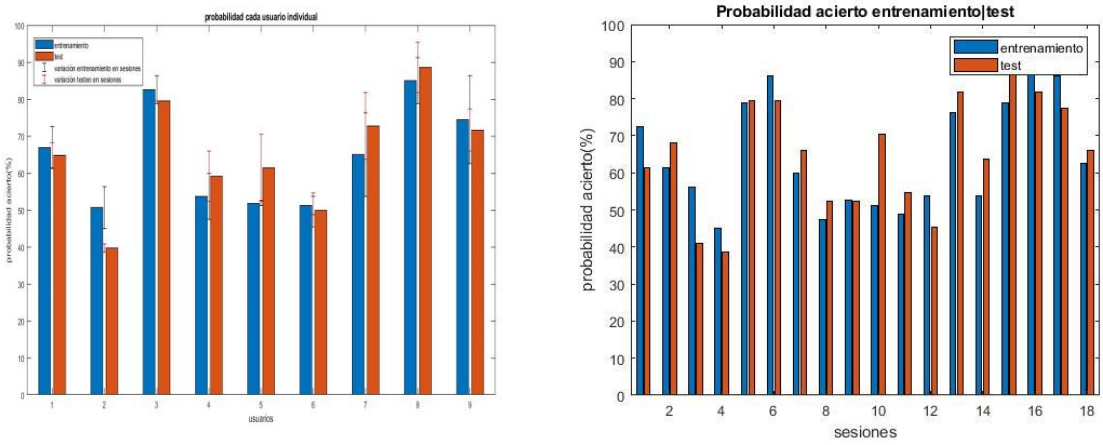


Figura 6-97. Gráficas resultado varios usuarios QDA.

6.2.2.1.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 65.1 Â± 3.4

Tiempo máx. medio entrenamiento: 0.459479 Â± 0.268649

Porcentaje medio de acierto test: 65.8 Â± 3.6

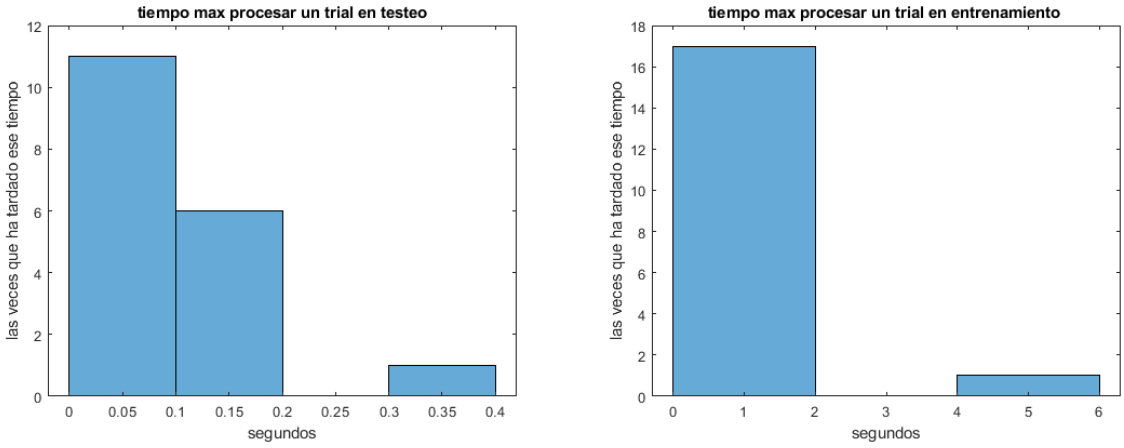
Tiempo máx. medio test: 0.105491 Â± 0.018586

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 65.1 Â± 4.2

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.459479 Â± 0.253919

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 65.8 Â± 4.6

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.105491 Â± 0.014220



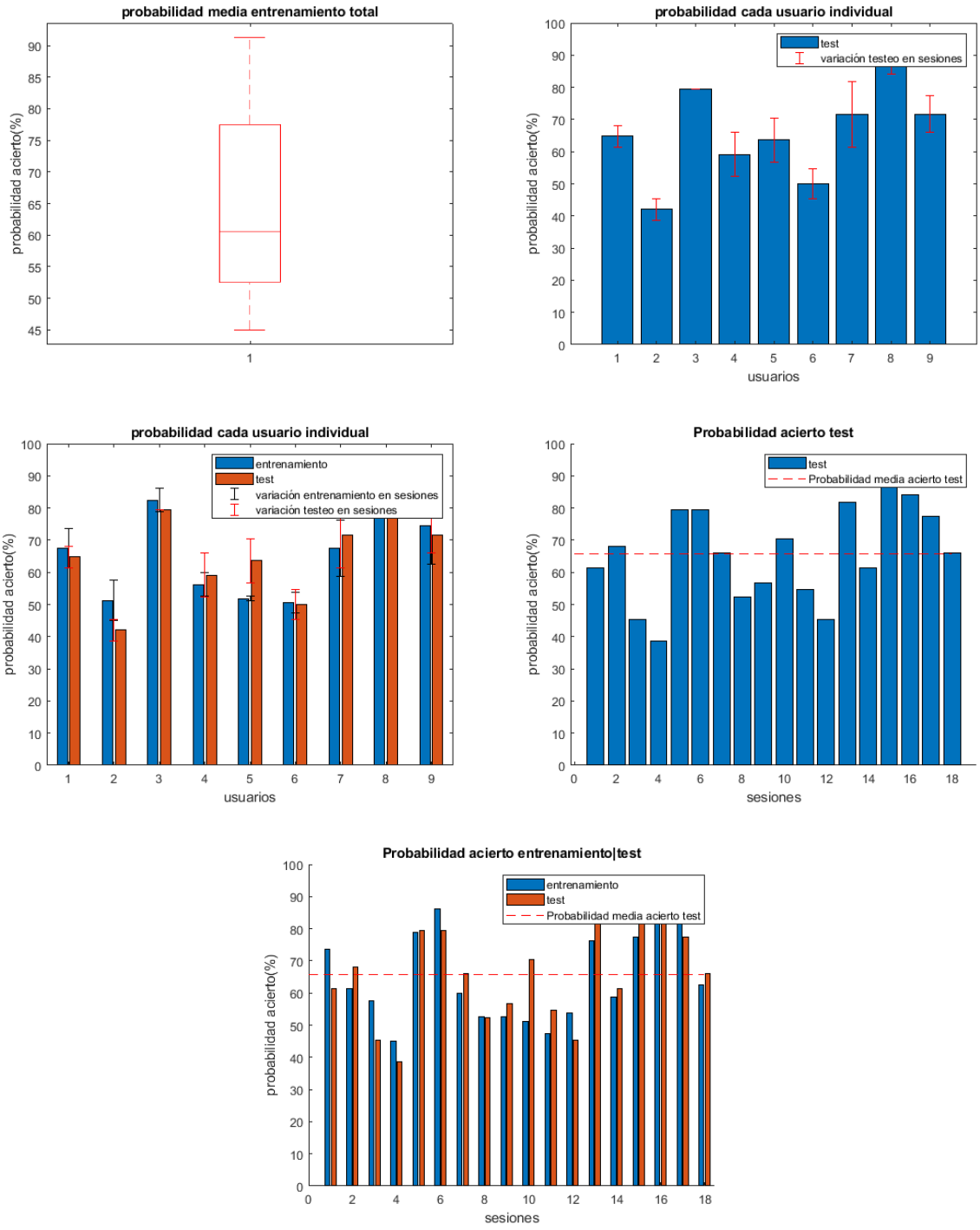


Figura 6-98. Gráficas resultado varios usuarios QDA.

6.2.3 Naive Bayes

6.2.3.1 Usando 100 de la sesión como entrenamiento y 44 como testeo.

6.2.3.1.1 Código realizado paso por paso

Se usa **mi_script_Isabel_BUFFER_bayes_usuarios.m**.

6.2.3.1.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 55.5 Â± 1.8

Tiempo máx. medio entrenamiento: 0.221541 Â± 0.137692

Porcentaje medio de acierto test: 59.7 Â± 3.4

Tiempo máx. medio test: 0.015113 Â± 0.002902

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 55.5 Â± 2.3

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.221541 Â± 0.130244

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 55.5 Â± 2.3

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.221541 Â± 0.130244

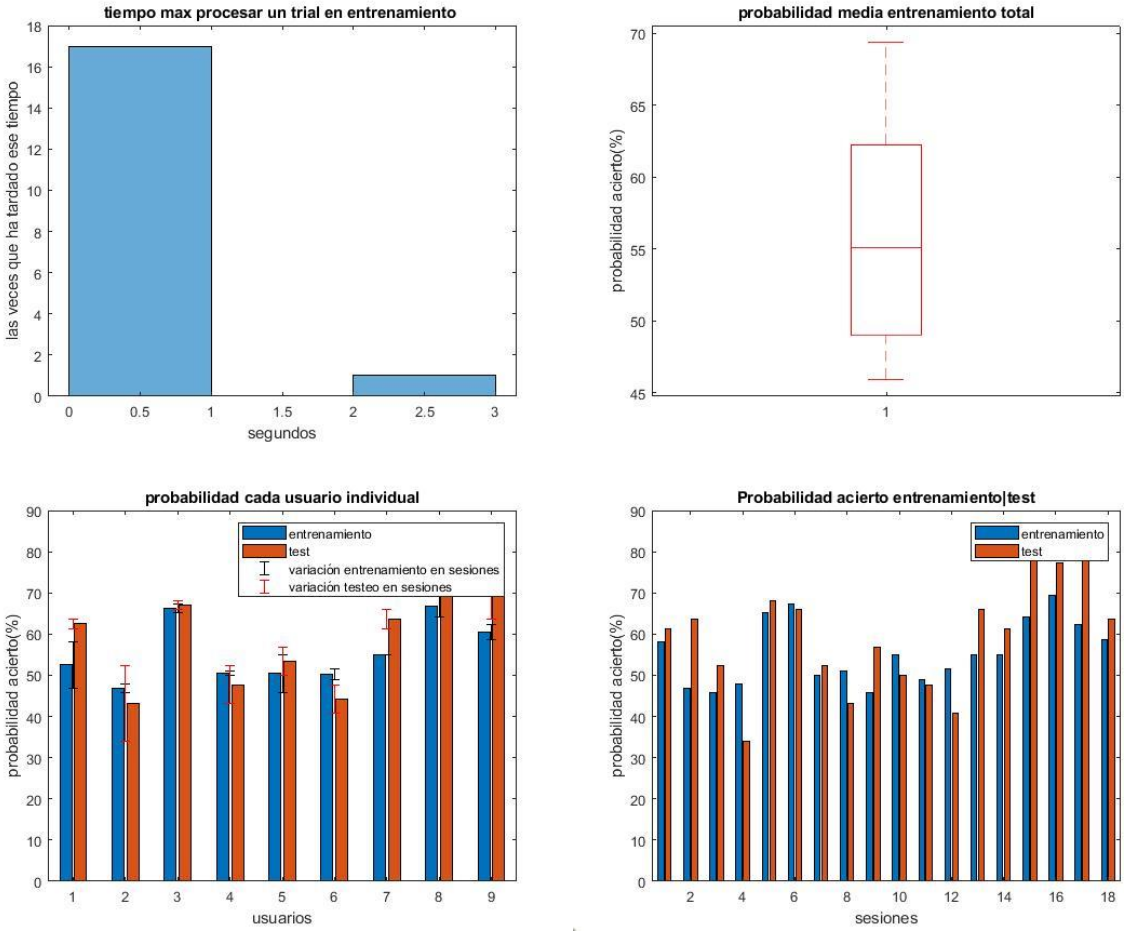


Figura 6-99. Gráficas resultado varios usuarios Naive Bayes paso a paso.

6.2.3.1.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 55.4 Â± 1.9

Tiempo máx. medio entrenamiento: 0.226509 Â± 0.044379

Porcentaje medio de acierto test: 59.8 Â± 3.5

Tiempo máx. medio test: 0.043353 Â± 0.012740

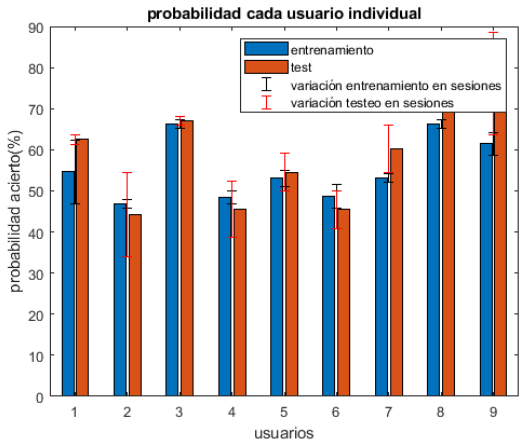
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 55.4 Â± 2.4

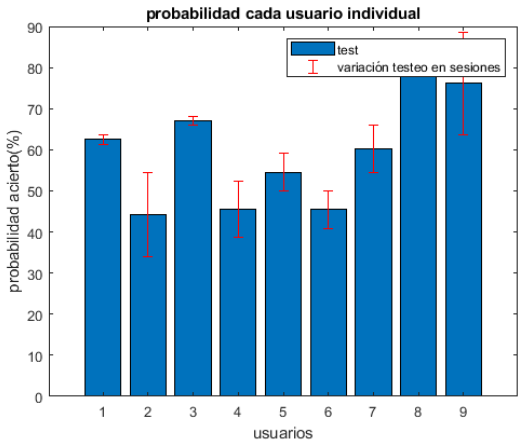
Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.226509 Â± 0.045880

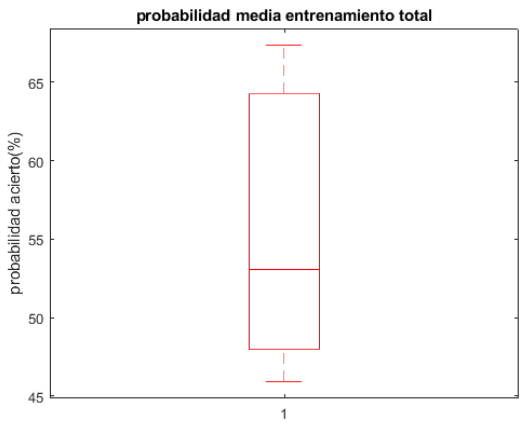
Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 59.8 Â± 4.4

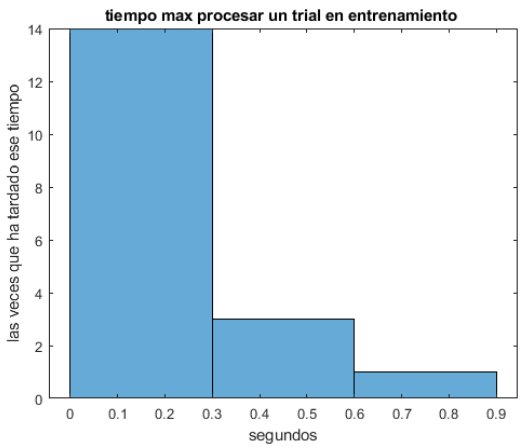
Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.043353 Â± 0.010500

No varía mucho el resultado con o sin el comando de ordenar los filtros.









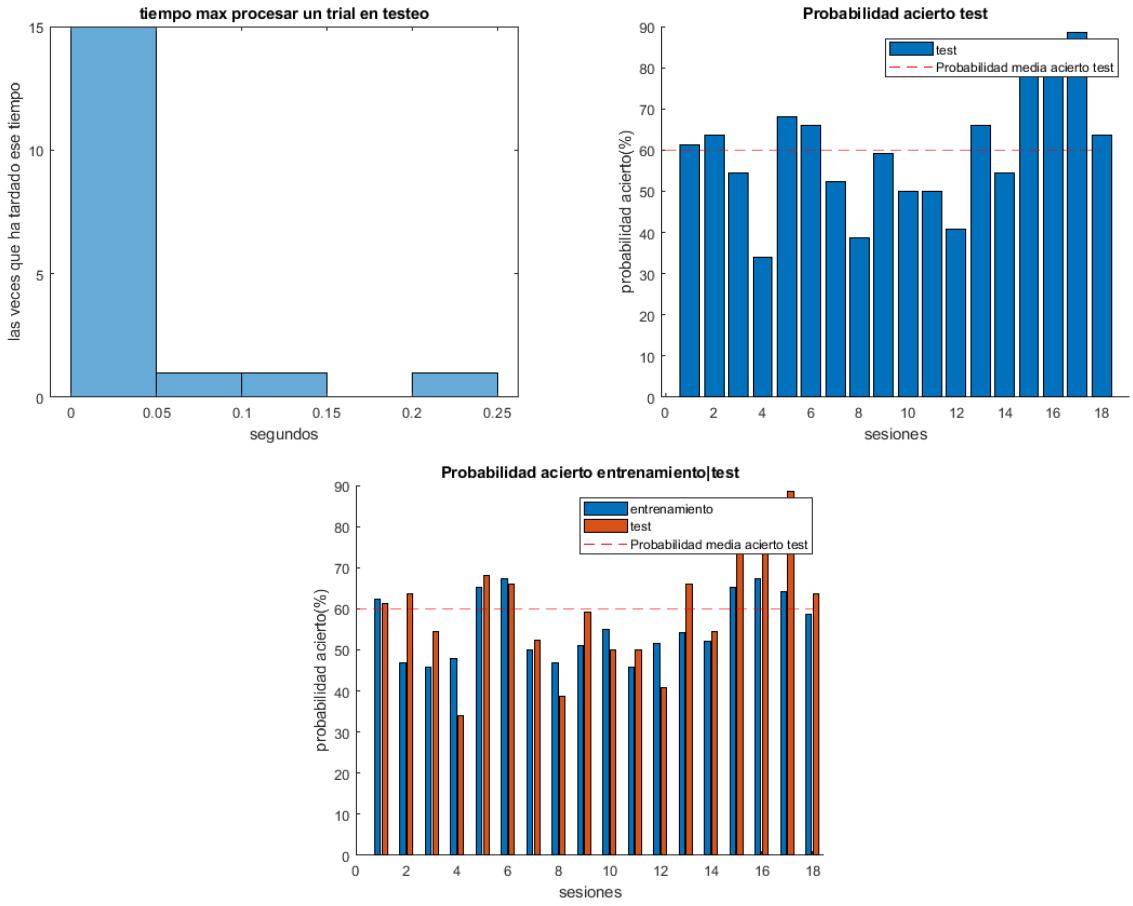


Figura 6-100. Gráficas resultado varios usuarios Naive Bayes paso a paso.

6.2.3.2 Aprovechando propiedades de MATLAB

6.2.3.2.1 Kernel Gaussiano Naive Bayes

Se usa **mi_script_Isabel_BUFFER_naiveBayesfuncion_usuarios.m**.

6.2.3.2.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 64.2 Â± 3.9

Tiempo max medio entrenamiento: 0.796755 Â± 0.246288

Porcentaje medio de acierto test: 66.8 Â± 3.3

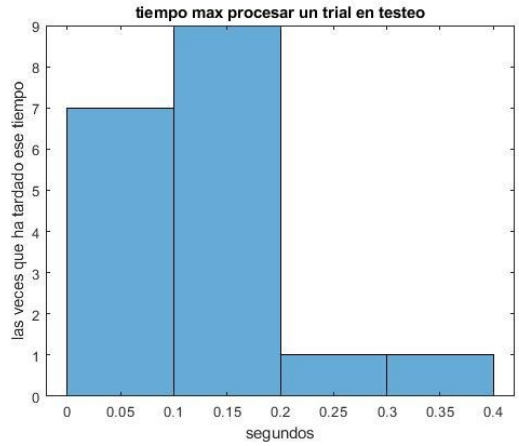
Tiempo máx medio test: 0.132073 Â± 0.017856

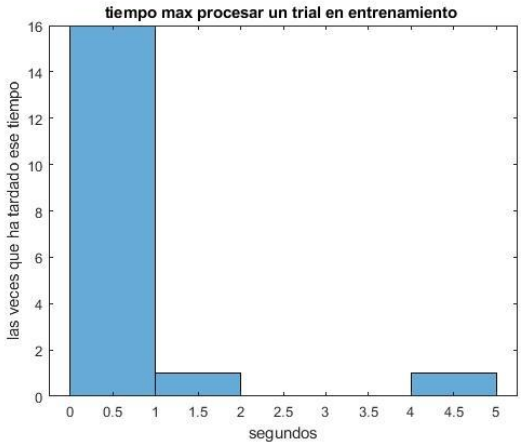
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 64.2 Â± 4.8

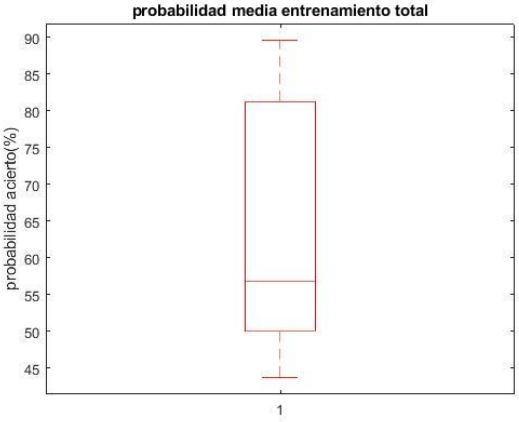
Tiempo máx medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.796755 Â± 0.250128

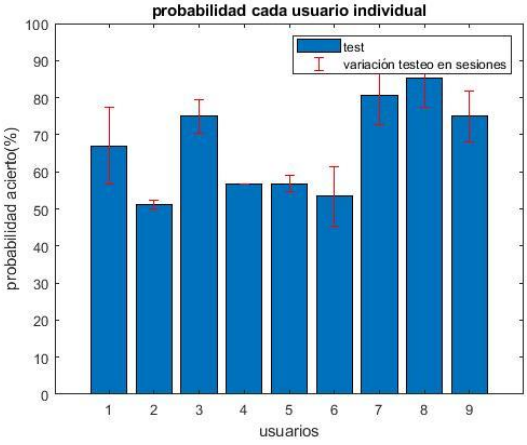
Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 66.8 Â± 4.0

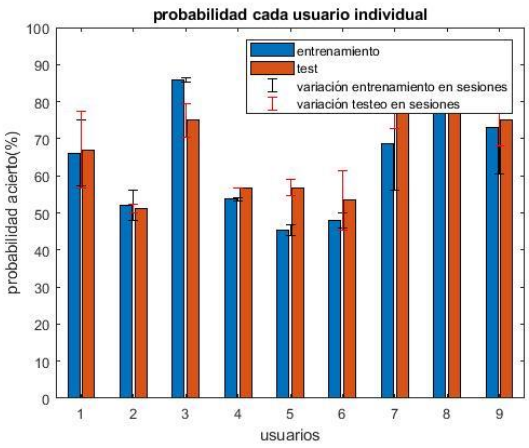
Tiempo máx medio test, teniendo en cuenta media de cada usuario individual: 0.132073 Â± 0.017246

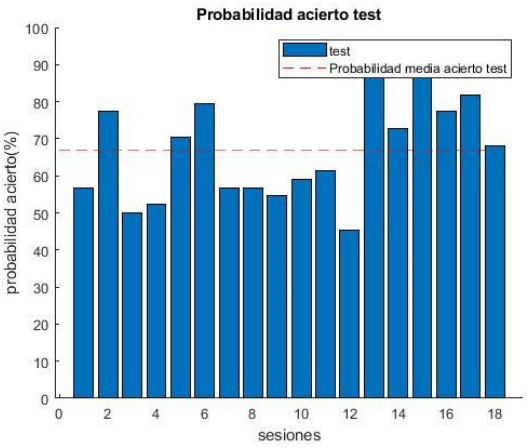












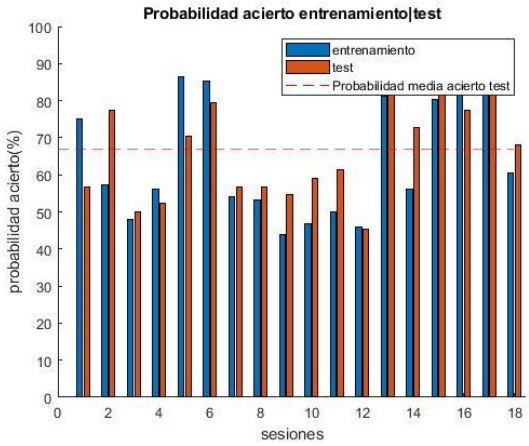


Figura 6-101. Gráficas resultados Kernel Gaussiano Naive Bayes varias sesiones y usuarios.

6.2.3.2.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 64.6 Â± 3.9

Tiempo max medio entrenamiento: 1.272795 Â± 0.884139

Porcentaje medio de acierto test: 66.8 Â± 3.3

Tiempo máx medio test: 0.113330 Â± 0.019622

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 64.6 Â± 4.8

Tiempo máx medio entrenamiento, teniendo en cuenta media de cada usuario individual: 1.272795 Â± 0.840090

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 66.8 Â± 4.0

Tiempo máx medio test, teniendo en cuenta media de cada usuario individual: 0.113330 Â± 0.018014

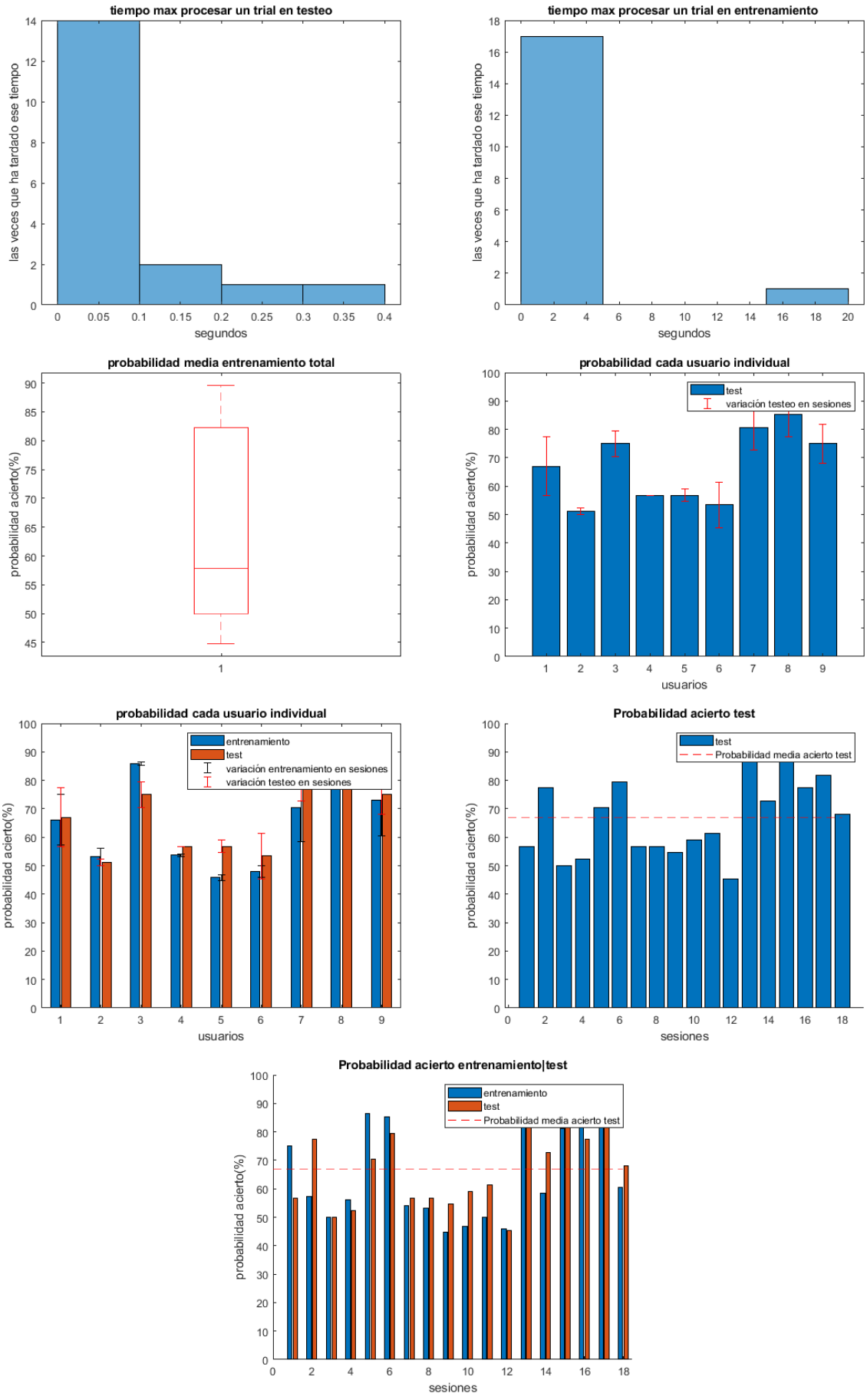


Figura 6-102. Gráficas resultados Kernel Gaussiano Naive Bayes varias sesiones y usuarios.

6.2.3.2.1.3 Prueba sin CSP

Porcentaje medio de acierto entrenamiento: 55.9 Â± 2.3

Tiempo max medio entrenamiento: 0.860568 Â± 0.177658

Porcentaje medio de acierto test: 59.6 Â± 2.7

Tiempo máx medio test: 0.202993 Â± 0.025041

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 55.9 Â± 3.0

Tiempo máx medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.860568 Â± 0.169646

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 59.6 Â± 3.4

Tiempo máx medio test, teniendo en cuenta media de cada usuario individual: 0.202993 Â± 0.024099

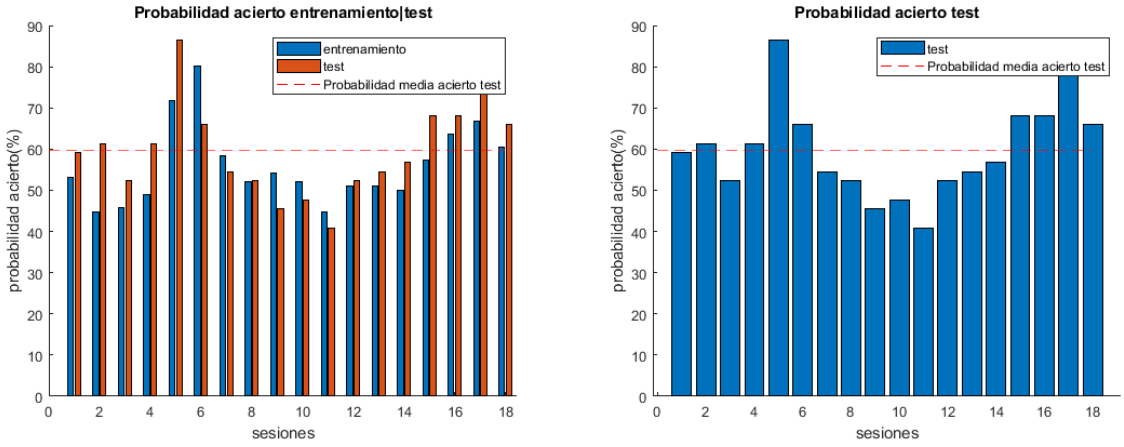


Figura 6-103. Gráficas resultados Kernel Gaussiano Naive Bayes varias sesiones y usuarios sin CSP.

6.2.4 KNN

Se usa **mi_script_Isabel_BUFFER_knn_usuarios.m**.

6.2.4.1 Usando 100 de la sesión como entrenamiento, 44 como testeo y k=7.

6.2.4.1.1 Aprovechando propiedades de MATLAB

6.2.4.1.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 65.3 Â± 4.6

Tiempo máx. medio entrenamiento: 0.215885 Â± 0.070719

Porcentaje medio de acierto test: 66.5 Â± 3.4

Tiempo máx. medio test: 0.051536 Â± 0.009301

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 65.3 Â± 5.8

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.215885 Â± 0.069019

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 65.3 Â± 5.8

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.215885 Â± 0.069019

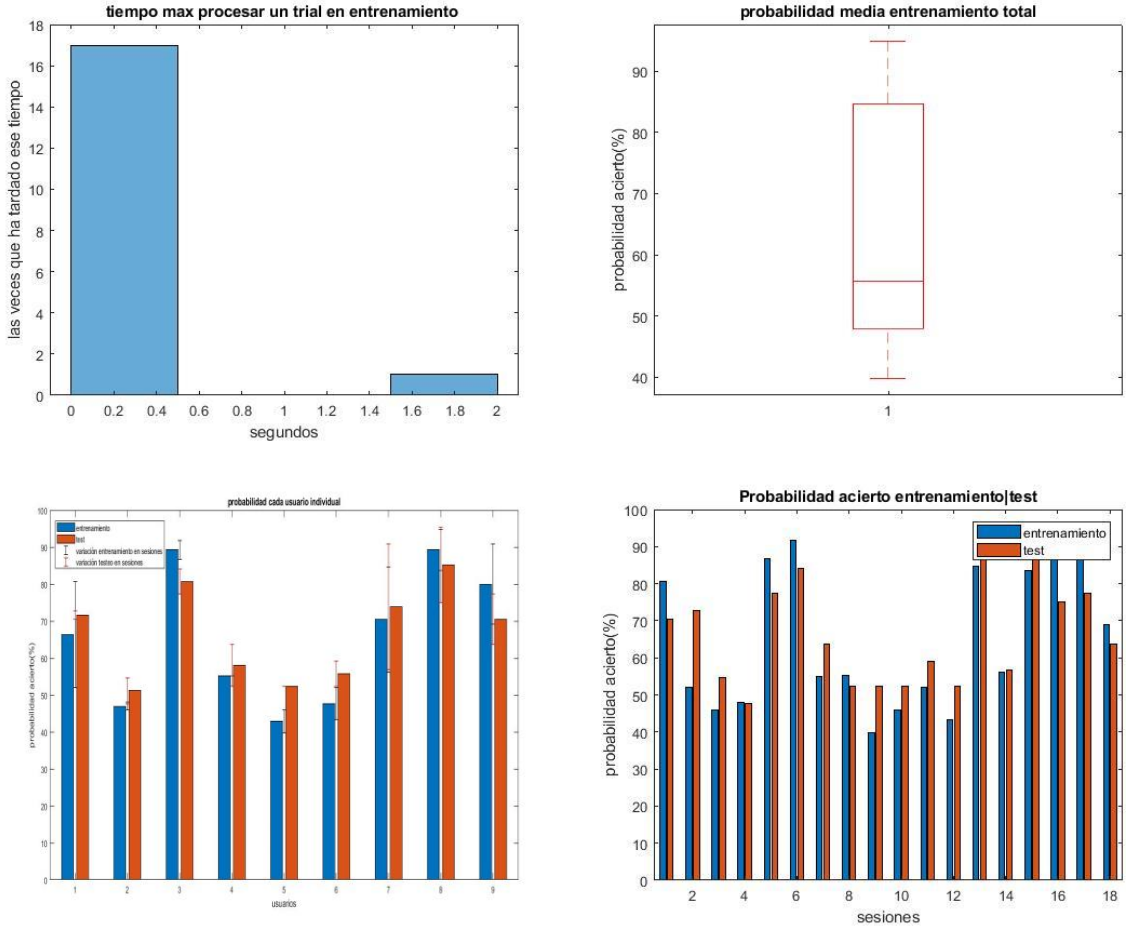


Figura 6-104. Gráficas resultado de varios usuarios con KNN con k=7.

6.2.4.1.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 65.3 Â± 4.6

Tiempo máx. medio entrenamiento: 0.398724 Â± 0.200858

Porcentaje medio de acierto test: 66.5 Â± 3.4

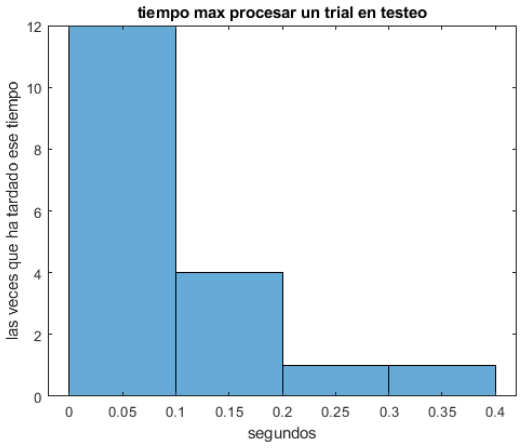
Tiempo máx. medio test: 0.089278 Â± 0.022224

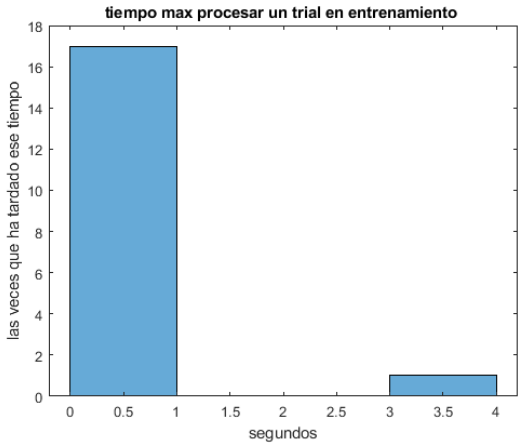
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 65.3 Â± 5.8

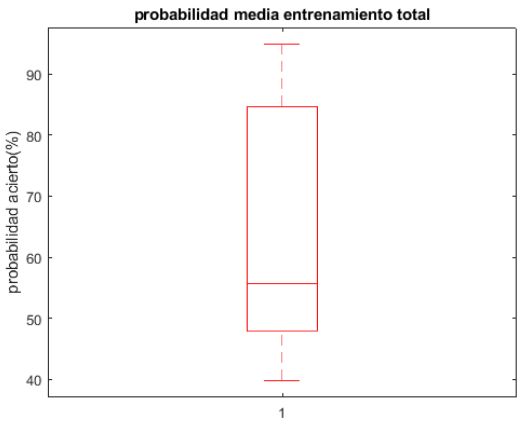
Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.398724 Â± 0.201370

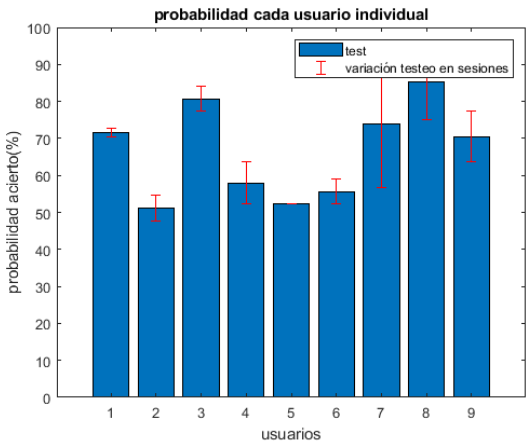
Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 66.5 Â± 4.0

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.089278 Â± 0.025638









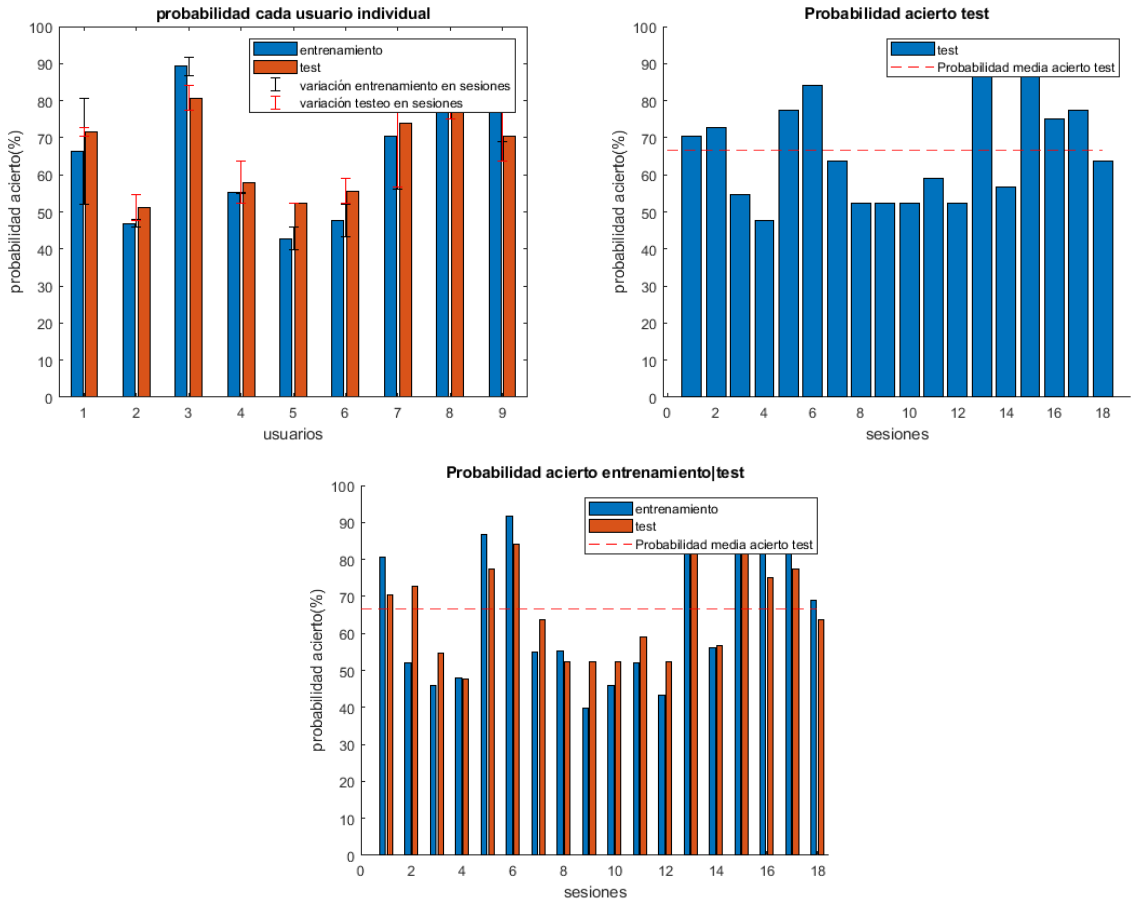


Figura 6-105. Gráficas resultado de varios usuarios con KNN con k=7.

6.2.4.1.1.3 Prueba sin CSP

Porcentaje medio de acierto entrenamiento: 56.4 Â± 2.4

Tiempo máx. medio entrenamiento: 0.441712 Â± 0.139515

Porcentaje medio de acierto test: 62.5 Â± 2.9

Tiempo máx. medio test: 0.111675 Â± 0.030523

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 56.4 Â± 2.6

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.441712 Â± 0.136033

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 62.5 Â± 3.1

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.111675 Â± 0.028080

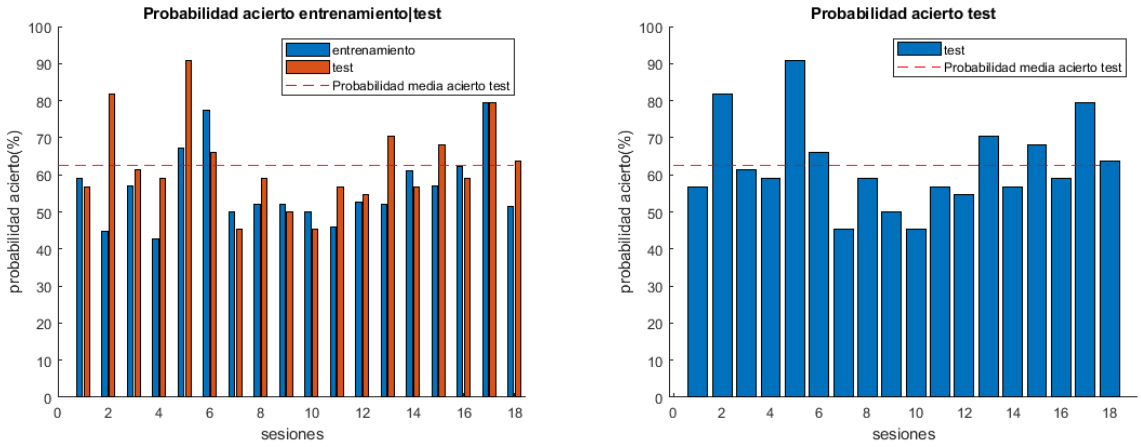


Figura 6-106. Gráficas resultado de varios usuarios con KNN con k=7.

6.2.5 Decision trees

6.2.5.1 Usando 100 de la sesión como entrenamiento y 44 como testeo.

6.2.5.1.1 Aprovechando propiedades de MATLAB

Se usa **mi_script_Isabel_BUFFER_tree_usuarios.m**.

6.2.5.1.1.1 Decision tree nivel 1

6.2.5.1.1.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 60.0 Â± 3.6

Tiempo máx. medio entrenamiento: 0.270174 Â± 0.095786

Porcentaje medio de acierto test: 66.9 Â± 3.5

Tiempo máx. medio test: 0.071405 Â± 0.016564

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 60.0 Â± 4.3

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.270174 Â± 0.090945

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 60.0 Â± 4.3

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.270174 Â± 0.090945

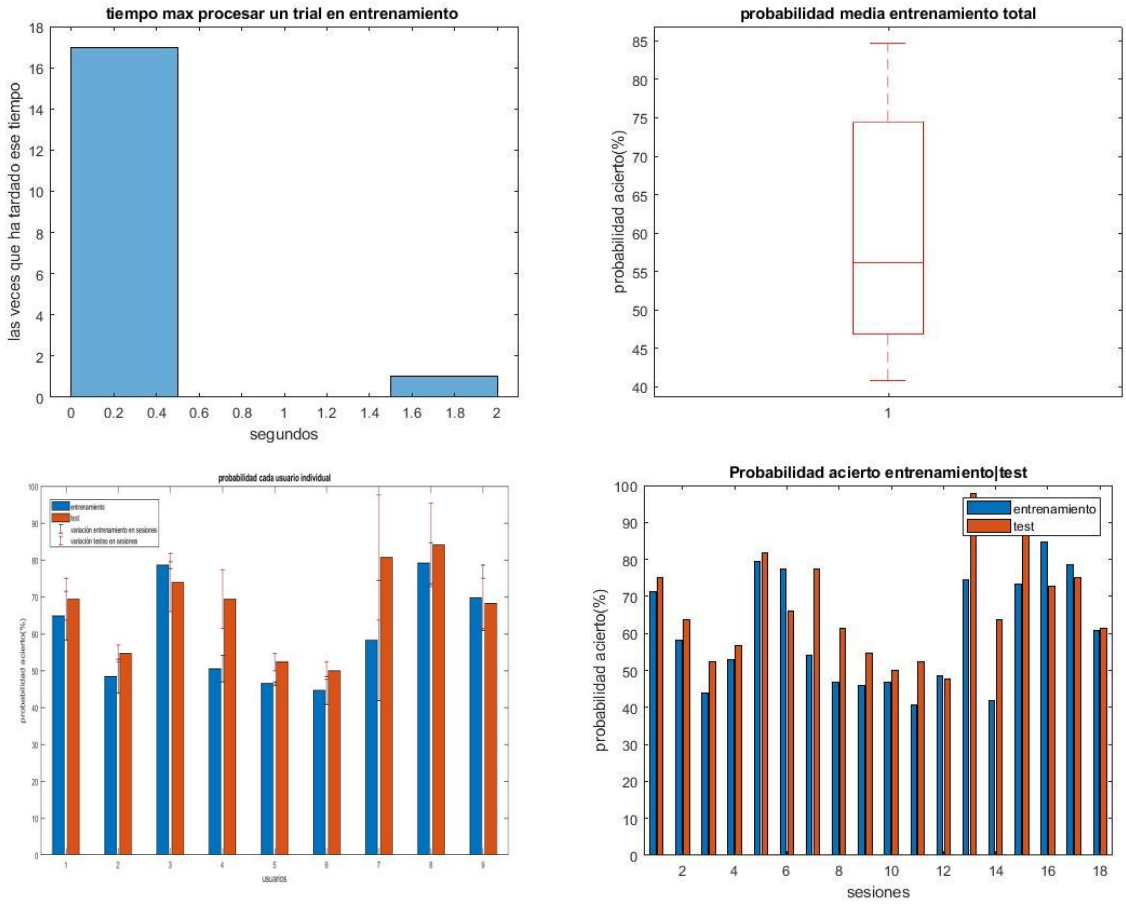


Figura 6-107. Gráficas resultado varios usuarios Decision tree nivel 1.

6.2.5.1.1.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 60.0 Â± 3.6

Tiempo máx. medio entrenamiento: 0.485970 Â± 0.247452

Porcentaje medio de acierto test: 66.9 Â± 3.5

Tiempo máx. medio test: 0.046387 Â± 0.015647

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 60.0 Â± 4.3

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.485970 Â± 0.233129

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 66.9 Â± 3.9

Tiempo máx medio test, teniendo en cuenta media de cada usuario individual: 0.046387 Â± 0.020802

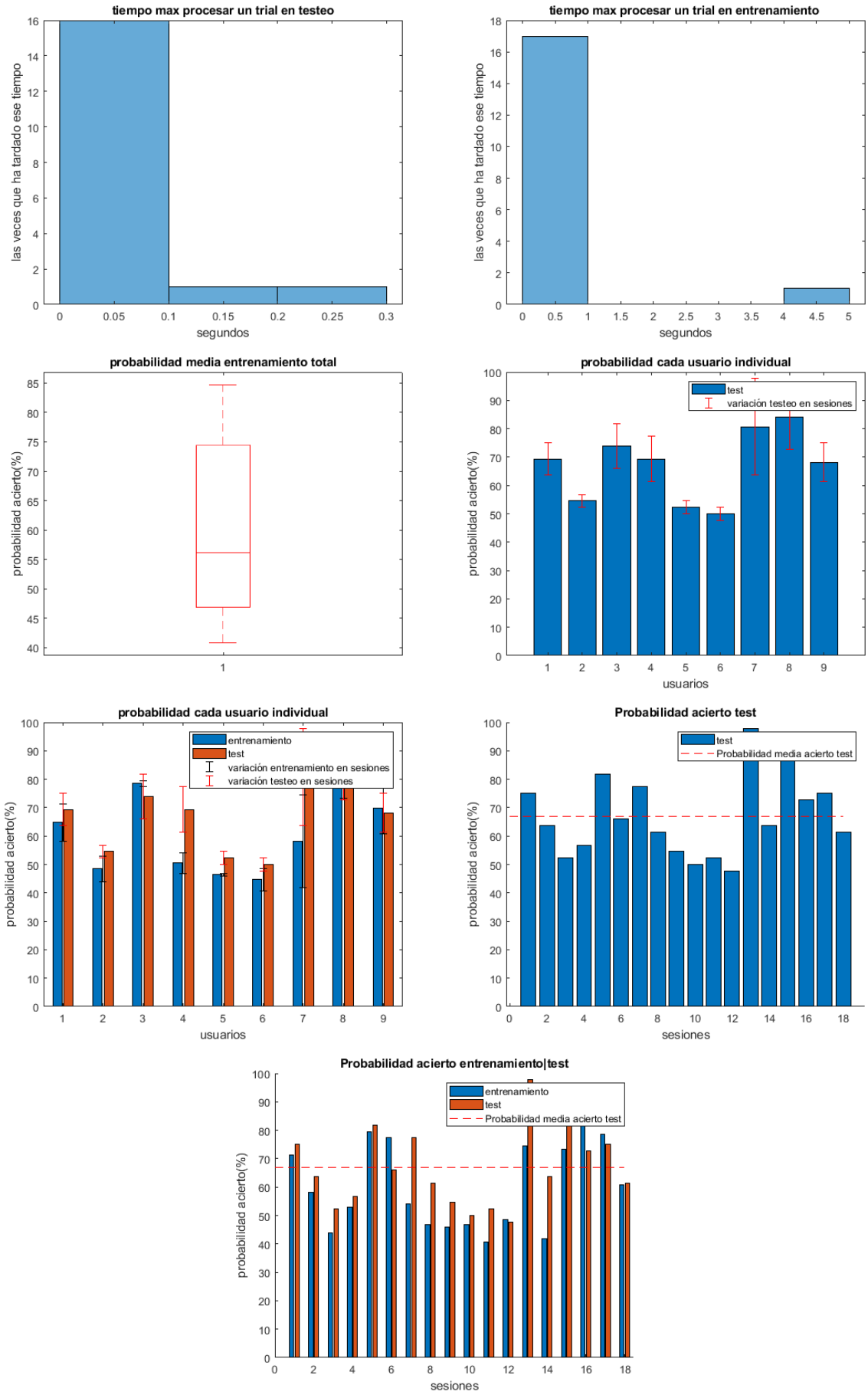


Figura 6-108. Gráficas resultado varios usuarios Decision tree nivel 1.

6.2.5.1.2 Usando App de MATLAB

6.2.5.1.2.1 Coarse Tree

Se usa **mi_script_Isabel_BUFFER_coarse_tree_usuarios.m**.

6.2.5.1.2.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 61.7 Â± 3.7

Tiempo máx. medio entrenamiento: 1.072672 Â± 0.620078

Porcentaje medio de acierto test: 66.4 Â± 3.7

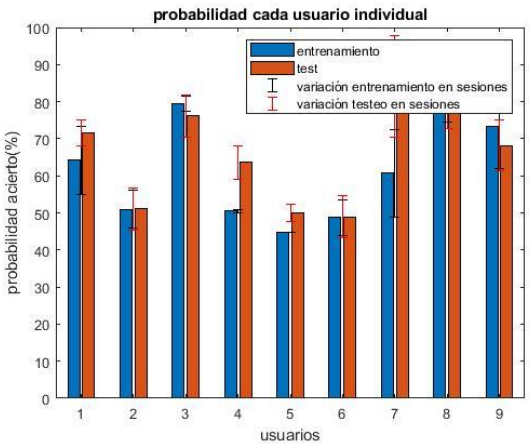
Tiempo máx. medio test: 0.077746 Â± 0.013702

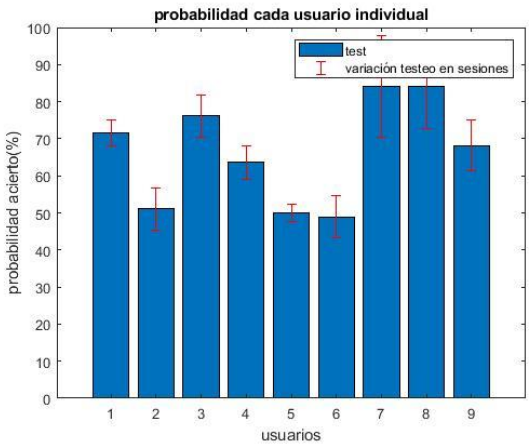
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 61.7 Â± 4.5

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 1.072672 Â± 0.589944

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 66.4 Â± 4.4

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.077746 Â± 0.014975





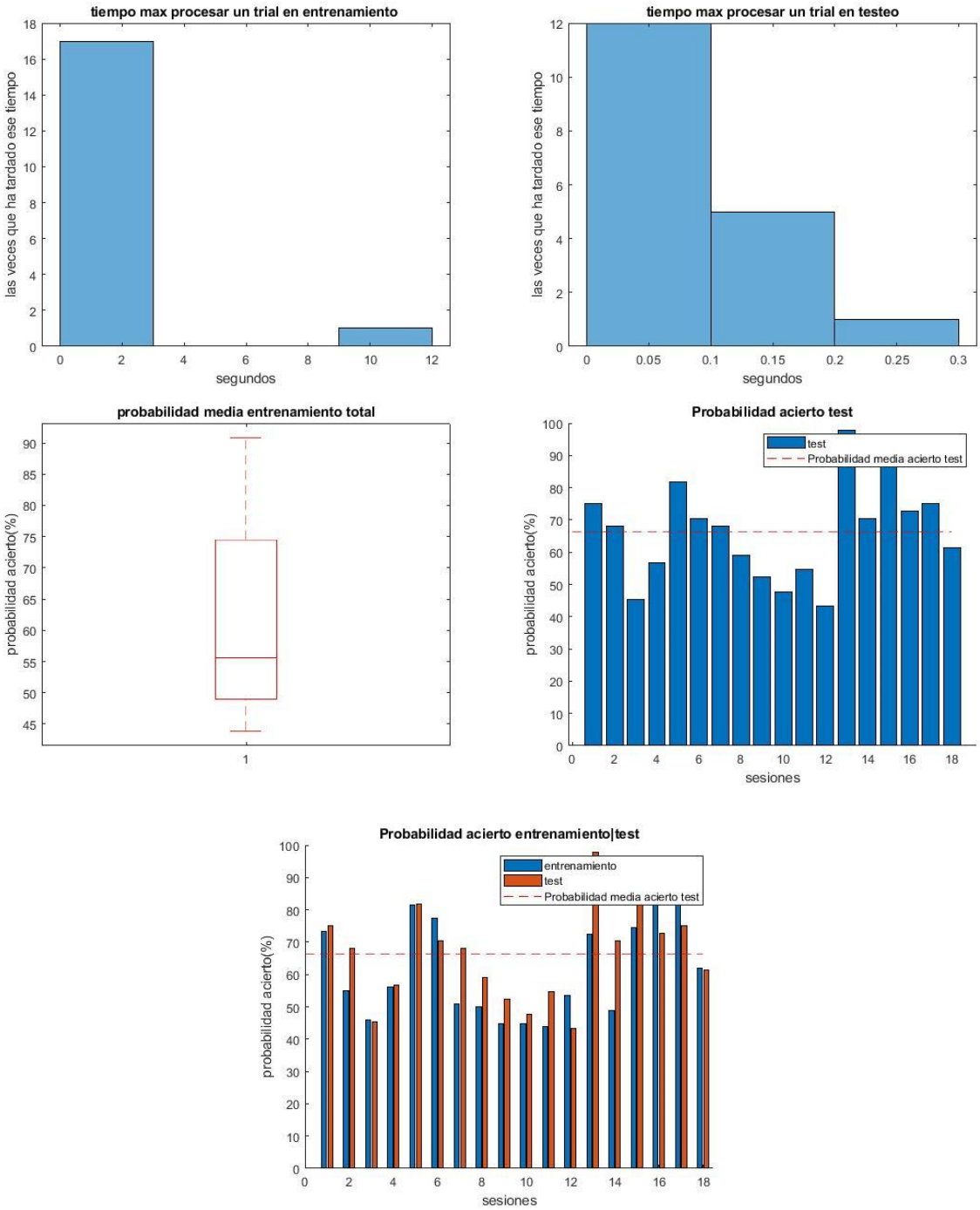


Figura 6-109. Gráficas resultado varios usuarios Coarse Tree.

6.2.5.1.2.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 61.7 Â± 3.7

Tiempo máx. medio entrenamiento: 0.939339 Â± 0.470973

Porcentaje medio de acierto test: 66.4 Â± 3.7

Tiempo máx. medio test: 0.075777 Â± 0.013944

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 61.7 Â± 4.5

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.939339 Â± 0.463388

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 66.4 Â± 4.4

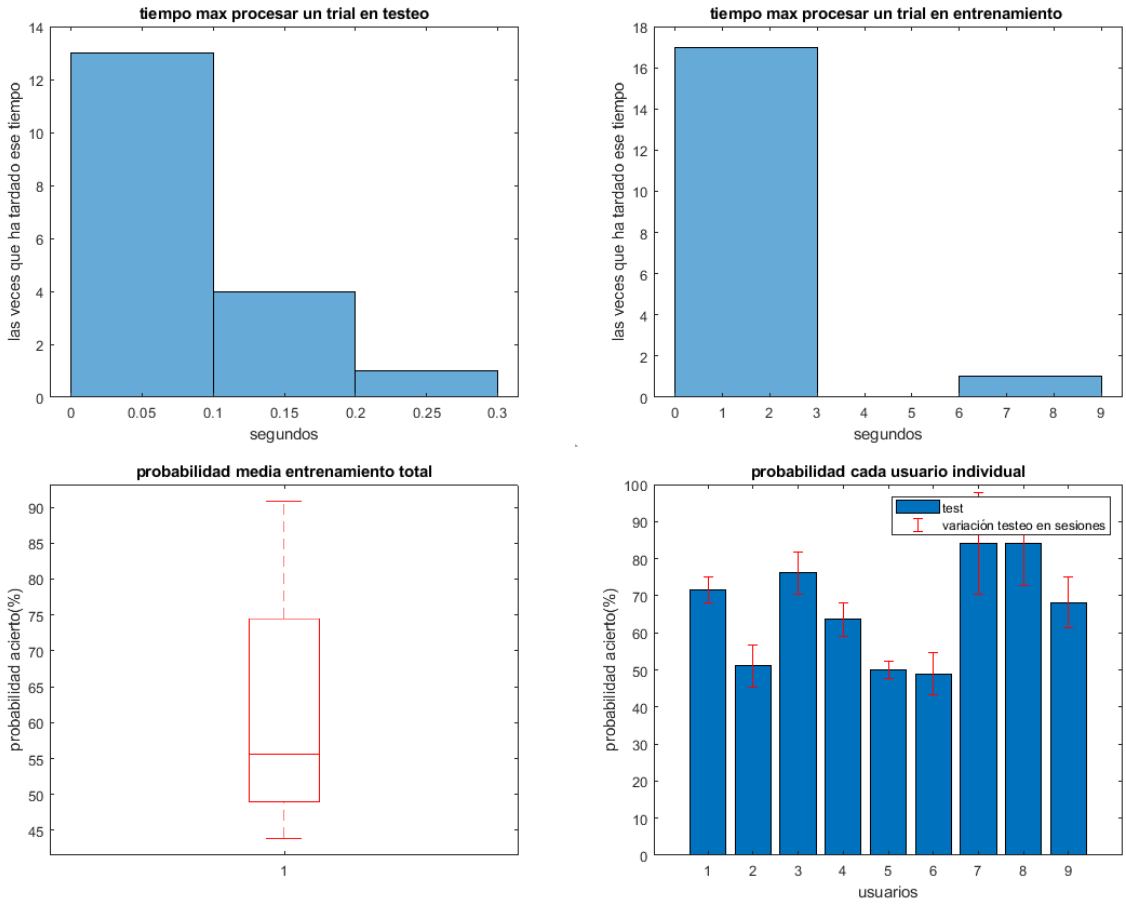
Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.075777 Â± 0.012720

Precisión clasificador con cross-validation: 0.7 (pu)± 0.040888

Se comprobó también, el resultado de probabilidad de acierto media (es una media de lo obtenido en todas las sesiones) que daría usando cross-validation de 5 kfolds con los 100 ***trials*** asignados para el entrenamiento (la que se obtiene con el clasificador exportado de la App):

Precisión clasificador: 0.7 (pu)± 0.040888

Da un resultado de probabilidad mayor por el método de validación de cross-validation, en comparación con el método usado de probar el clasificador en un grupo fijo de datos de testeo, es decir, coger los 44 últimos de la sesión. No obstante, el valor no es muy lejano al anteriormente obtenido.



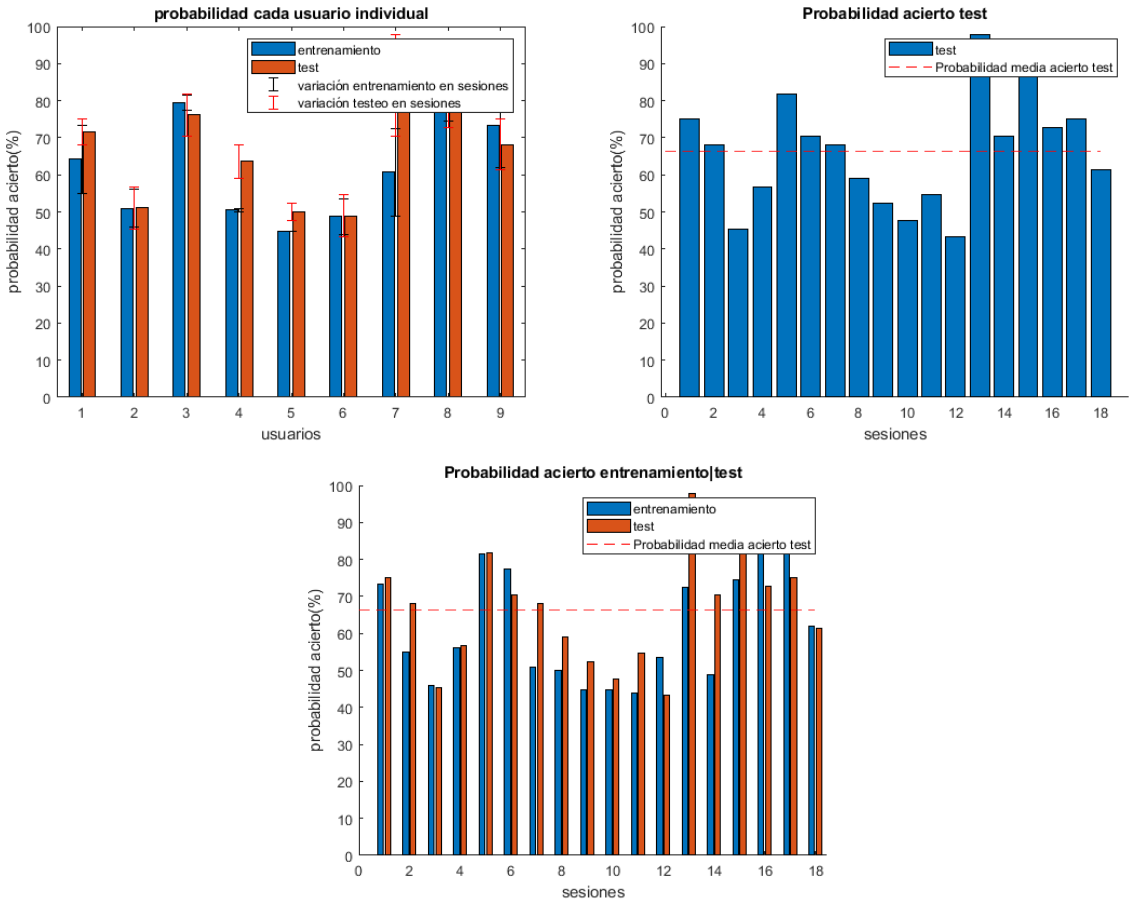


Figura 6-110. Gráficas resultado varios usuarios Coarse Tree.

6.2.6 SVM

Se usa **mi_script_Isabel_BUFFER_svm_usuarios.m.**

6.2.6.1 Usando 100 de la sesión como entrenamiento y 44 como testeo.

6.2.6.1.1 Aprovechando propiedades de MATLAB

6.2.6.1.1.1 función kernel gaussiana

6.2.6.1.1.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 65.3 Â± 4.7

Tiempo máx. medio entrenamiento: 0.436677 Â± 0.252706

Porcentaje medio de acierto test: 67.6 Â± 3.5

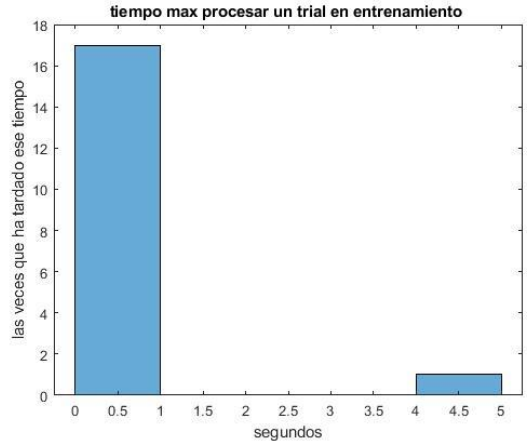
Tiempo máx. medio test: 0.051468 Â± 0.012786

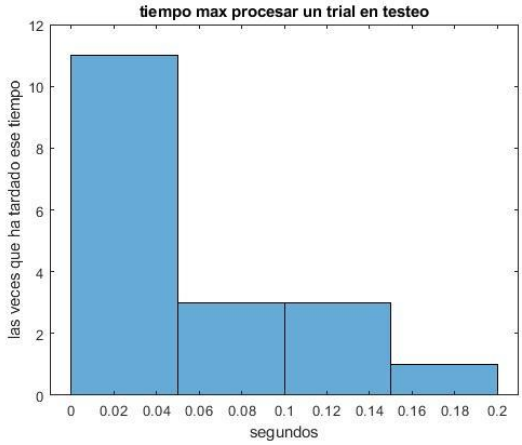
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 65.3 Â± 5.9

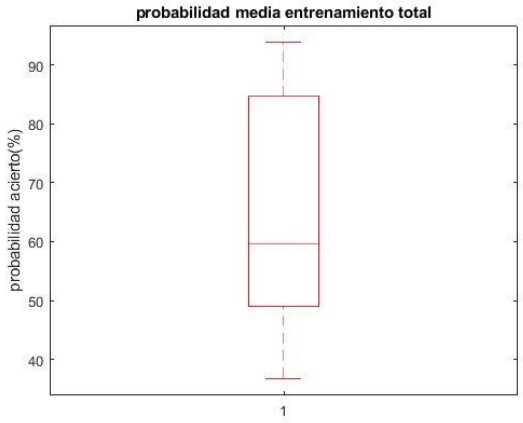
Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.385007 Â± 0.178381

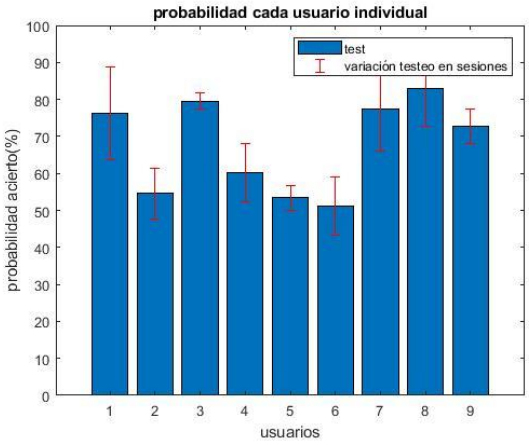
Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 67.6 Â± 4.0

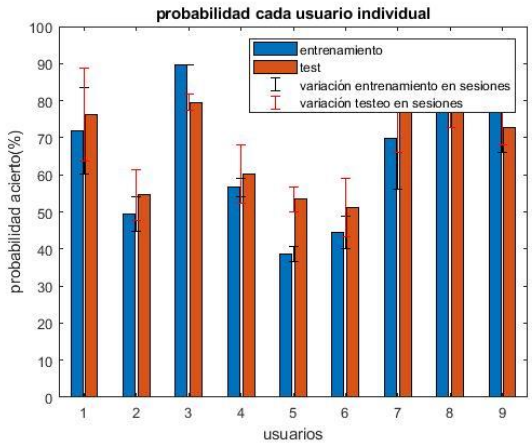
Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.065874 Â± 0.014555

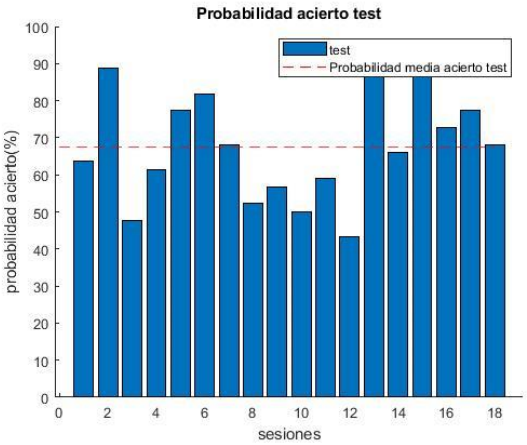












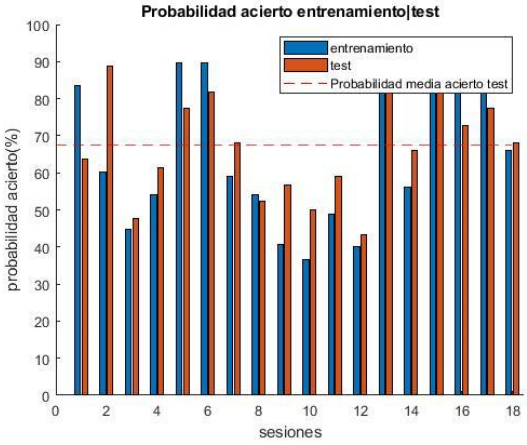


Figura 6-111. Gráficas resultado varios usuarios SVM kernel gaussiano.

6.2.6.1.1.1.1 Prueba de aplicar testeo a los datos de entrenamiento

Porcentaje medio de acierto test: 79.7 Â± 3.8

Tiempo máx. medio test: 0.091308 Â± 0.016089

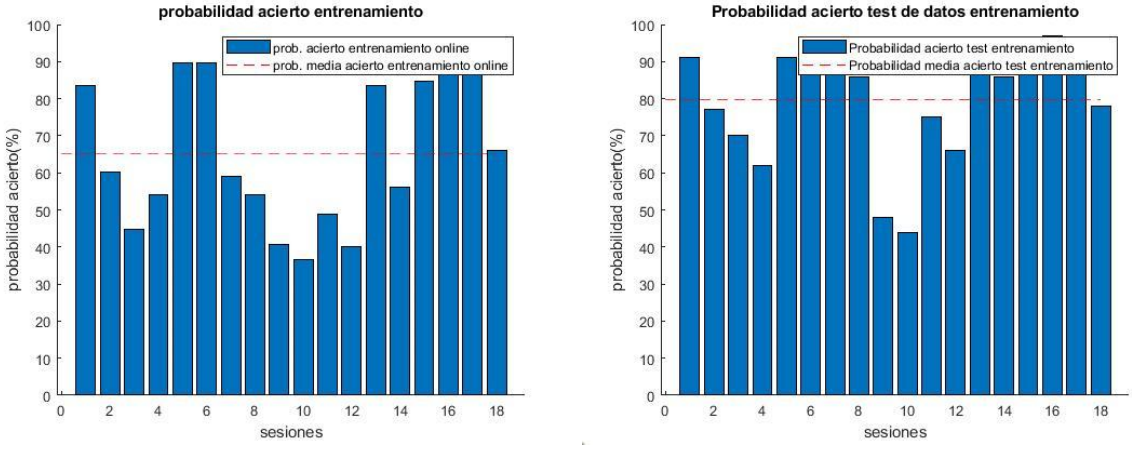


Figura 6-112. Probabilidad de acierto testeando datos de entrenamiento y probabilidad de acierto de entrenamiento online.

6.2.6.1.1.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 65.1 Â± 4.7

Tiempo máx. medio entrenamiento: 0.478102 Â± 0.232843

Porcentaje medio de acierto test: 67.4 Â± 3.5

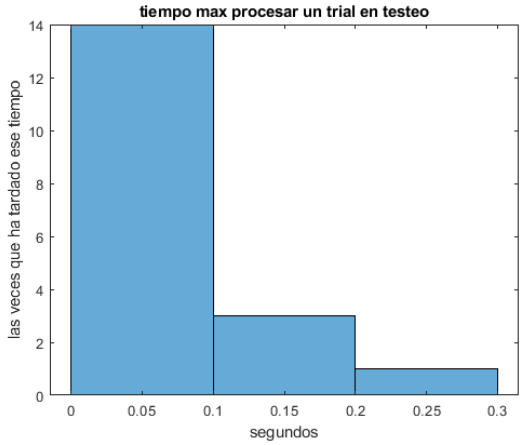
Tiempo máx. medio test: 0.054099 Â± 0.014283

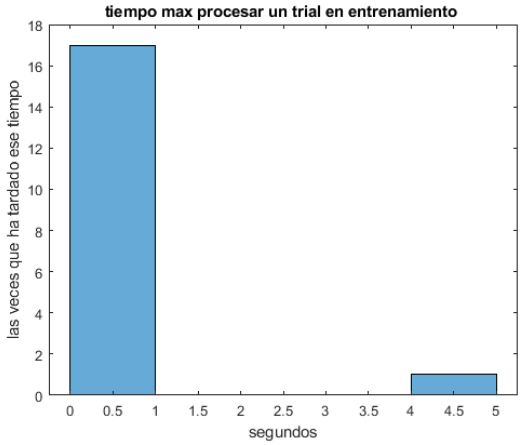
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 65.1 Â± 6.0

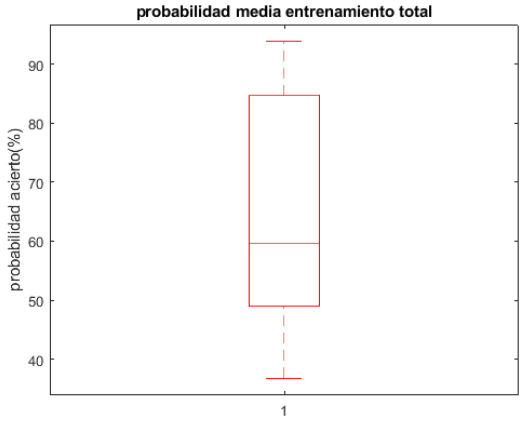
Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.478102 Â± 0.229553

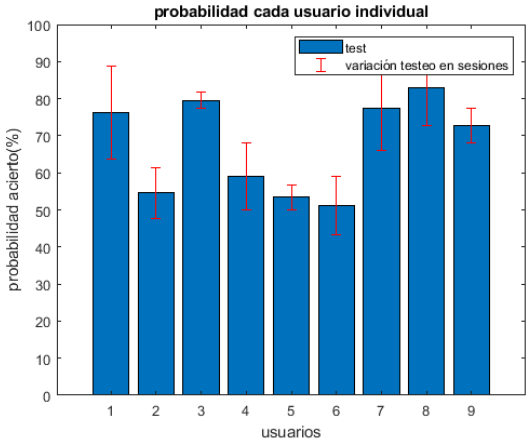
Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 67.4 Â± 4.0

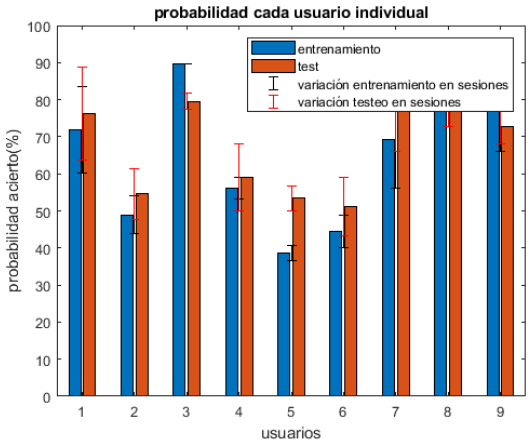
Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.054099 Â± 0.018344

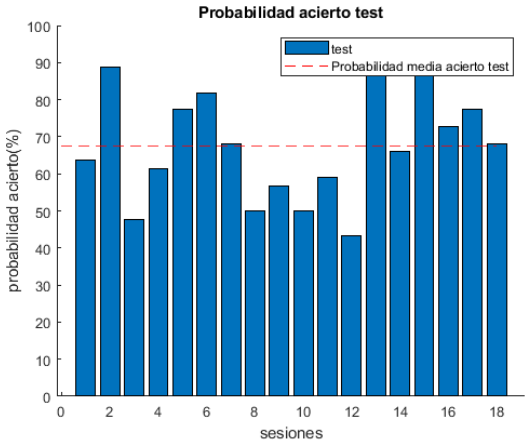












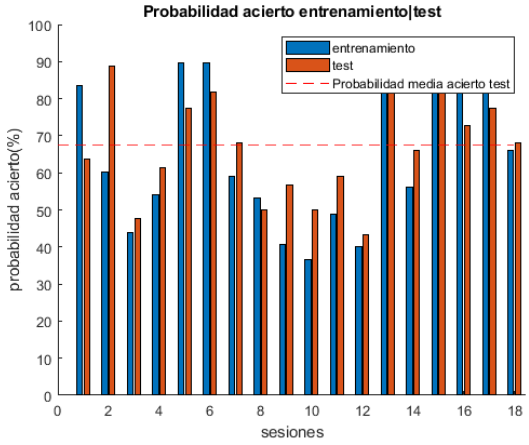


Figura 6-113. Gráficas resultado varios usuarios SVM kernel gaussiano.

6.2.6.1.1.1.2.1 Prueba de aplicar testeo a los datos de entrenamiento

Porcentaje medio de acierto test: 79.6 Â± 3.8

Tiempo máx. medio test: 0.041759 Â± 0.005192

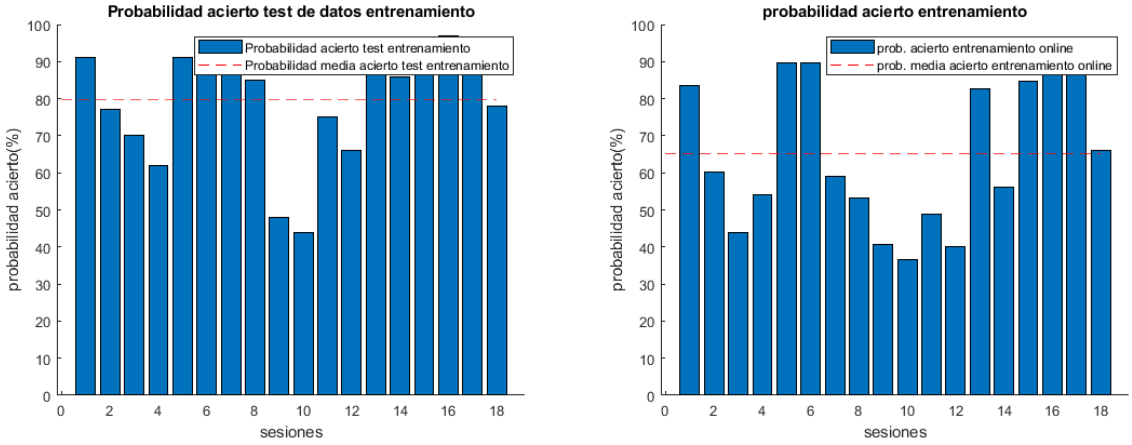


Figura 6-114. Probabilidad de acierto testeando datos de entrenamiento y probabilidad de acierto de entrenamiento online.

6.2.6.1.1.1.3 Prueba realizada sin aplicar CSP

Porcentaje medio de acierto entrenamiento: 56.1 Â± 2.9

Tiempo máx. medio entrenamiento: 0.238363 Â± 0.094190

Porcentaje medio de acierto test: 62.0 Â± 2.7

Tiempo máx. medio test: 0.035930 Â± 0.009025

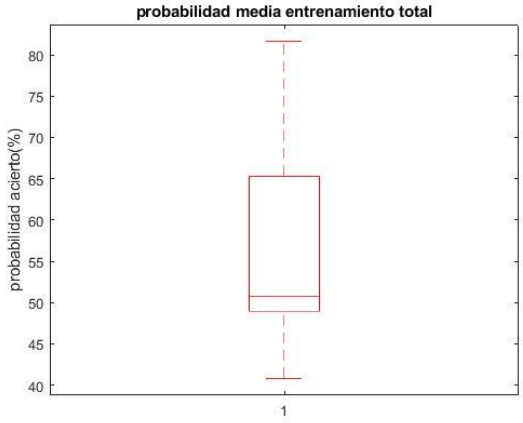
Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 56.1 Â± 3.4

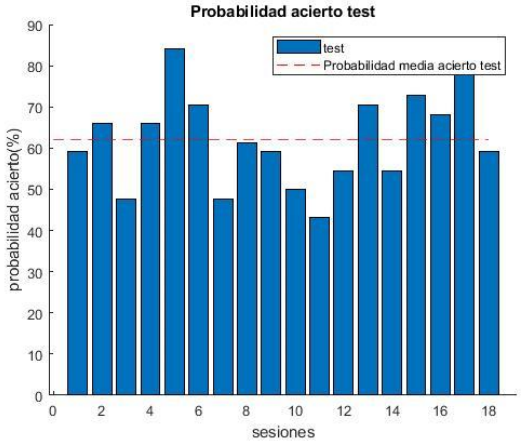
Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 0.238363 Â± 0.090509

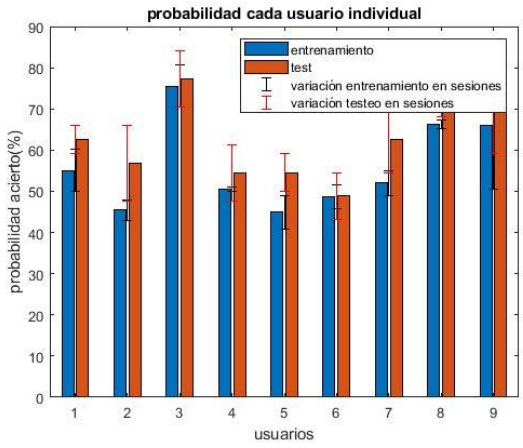
Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 62.0 Â± 2.9

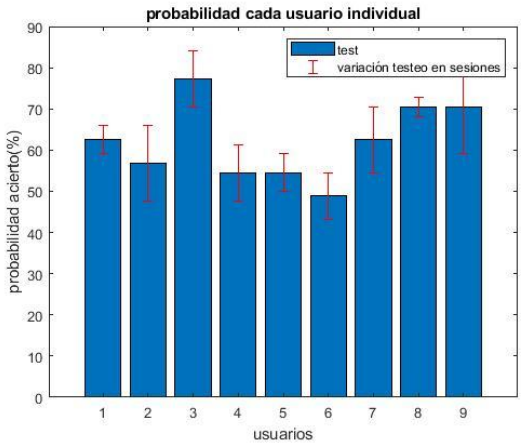
Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 0.035930 Â± 0.008477

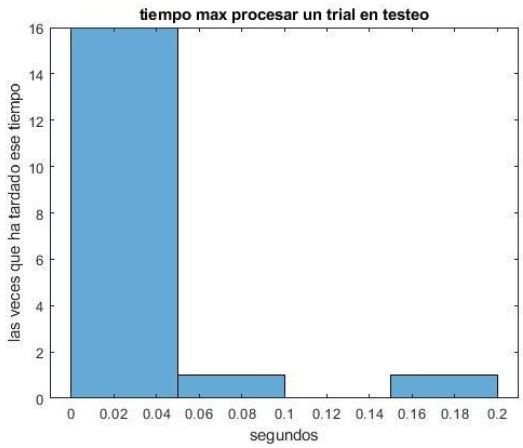
La probabilidad de acierto ha bajado con respecto al caso en que sí se usa CSP.

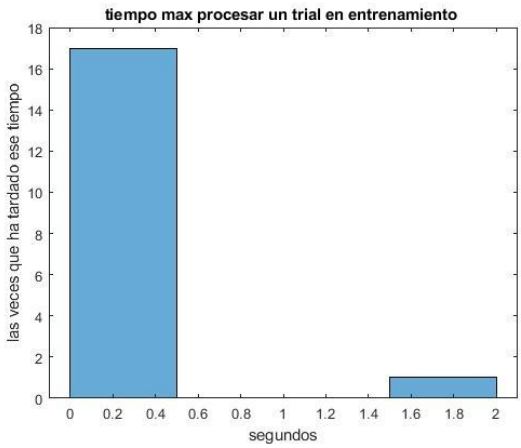












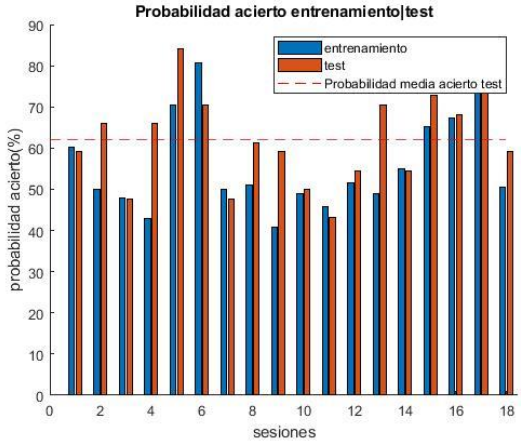


Figura 6-115. Gráficas resultado varios usuarios SVM kernel gaussiano sin CSP.

6.2.7 Logistic Regression

Se usa **mi_script_Isabel_BUFFER_logisticregression_usuarios.m**

6.2.7.1 Usando 100 de la sesión como entrenamiento y 44 como testeo.

6.2.7.1.1 Usando App de MATLAB

6.2.7.1.1.1 Pruebas sin ordenar matriz de filtros W

Porcentaje medio de acierto entrenamiento: 67.7 Â± 3.6

Tiempo máx. medio entrenamiento: 1.513249 Â± 0.247530

Porcentaje medio de acierto test: 67.0 Â± 3.2

Tiempo máx. medio test: 0.067492 Â± 0.011917

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 67.7 Â± 4.4

Tiempo máx. medio entrenamiento, teniendo en cuenta media de cada usuario individual: 1.513249 Â± 0.256368

Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 67.7 Â± 4.4

Tiempo máx. medio test, teniendo en cuenta media de cada usuario individual: 1.513249 Â± 0.256368

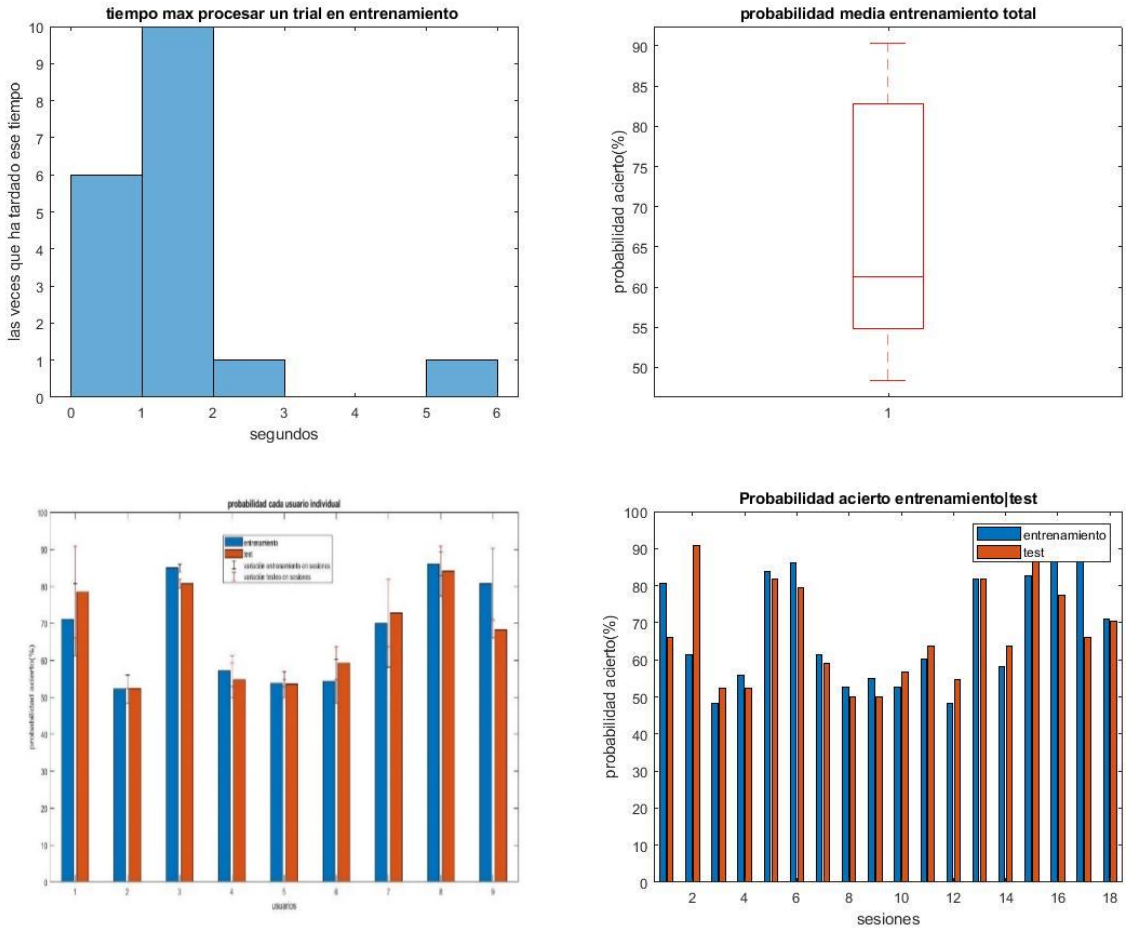


Figura 6-116. Gráficas resultado varios usuarios Logistic Regression.

6.2.7.1.1.2 Pruebas con ordenación de matriz de filtros W

Porcentaje medio de acierto entrenamiento: 67.6 Â± 3.6

Tiempo max medio entrenamiento: 3.356350 Â± 1.130937

Porcentaje medio de acierto test: 67.2 Â± 3.3

Tiempo máx medio test: 0.130856 Â± 0.020168

Porcentaje medio de acierto entrenamiento, teniendo en cuenta media de cada usuario individual: 67.6 Â± 4.5

Tiempo máx medio entrenamiento, teniendo en cuenta media de cada usuario individual: 3.356350 Â± 1.092263

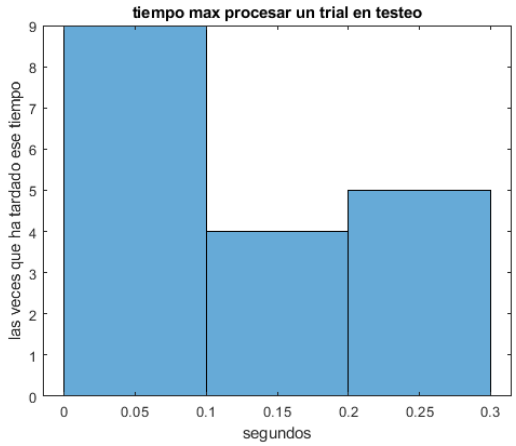
Porcentaje medio de acierto test, teniendo en cuenta media de cada usuario individual: 67.2 Â± 4.0

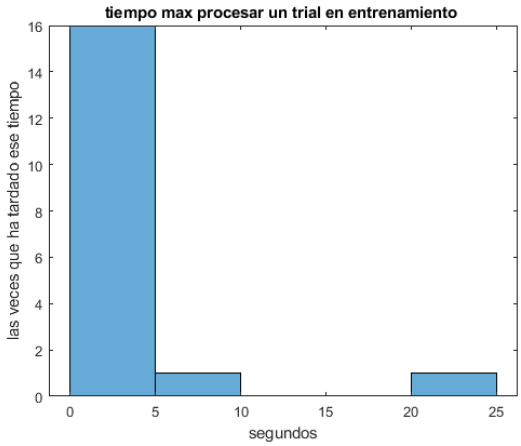
Tiempo máx medio test, teniendo en cuenta media de cada usuario individual: 0.130856 Â± 0.025469

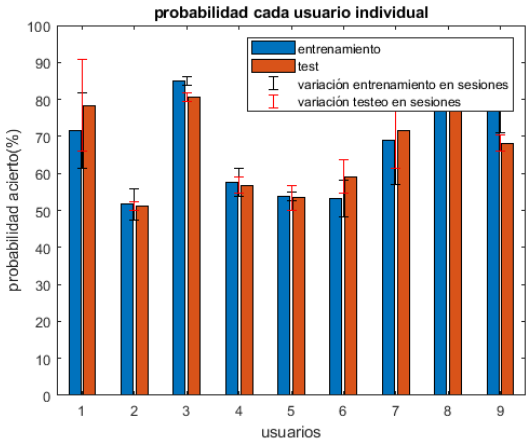
Se comprobó el resultado de probabilidad de acierto media que daría usando cross-validation de 5 kfolds:

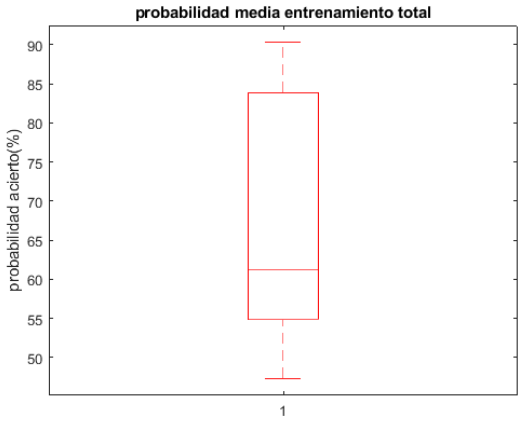
Precisión clasificador: 0.7 (pu)± 0.040329

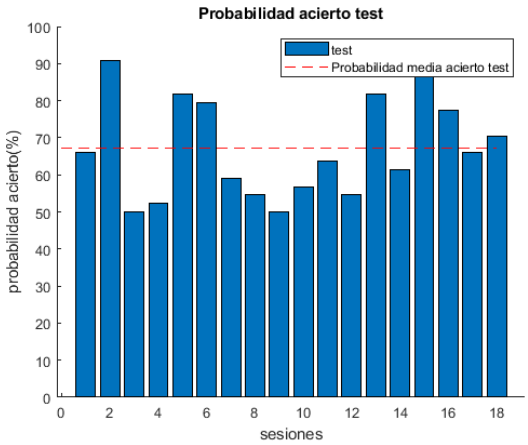
Da un resultado de probabilidad mayor por el método de validación de cross-validation, en comparación con el otro método.

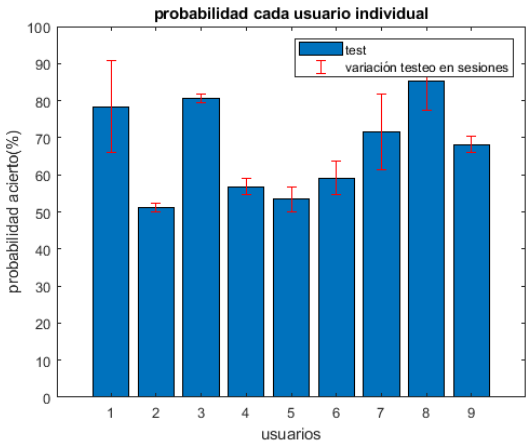












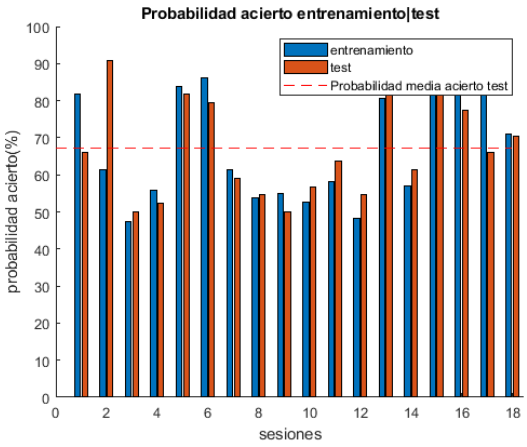


Figura 6-117. Gráficas resultado varios usuarios Logistic Regression.

6.2.8 Conclusiones: Comparando los clasificadores con varios usuarios

Los resultados obtenidos con varios usuarios y sesiones dan una idea más genérica de cómo resultaría usar estos algoritmos en MI-BCI.

La mayoría de las conclusiones que se dedujeron con un usuario, son aplicables al caso de varios usuarios. No obstante, se ha apreciado que el hecho de tener o no CSP afectaba a la media de probabilidad de acierto de manera positiva o negativa, según el clasificador usado. Por otra parte, hay que mencionar que en el caso de un usuario daba mayor probabilidad de acierto con CSP en la mayoría de los casos, mientras que cuando se hallaba la probabilidad media teniendo en cuenta varios usuarios, se vio que podía cambiar este hecho en algunos casos. Por ejemplo, en LDA empeoraba el resultado al aplicar CSP cuando eran varios usuarios, mientras que con un usuario mejoraba. Además, con respecto al método de validación del algoritmo, se apreció que, de media, daba mejor resultado usar validación cruzada (cross-validation) en diversos algoritmos, en comparación con el otro método de validación usado.

Igual que en el caso de una sesión de un usuario, se muestran a continuación unas tablas resumen de las probabilidades de acierto obtenidas con 100 ensayos para entrenamiento online y 44 para el testeo. Siendo los resultados obtenidos con el CSP forzando la ordenación del filtro y sin forzarla. Se consideran de más interés los resultados con la ordenación, ya que uno de los requisitos iniciales de este proyecto era aplicar el clasificador con el CSP explicado anteriormente.

- Pruebas sin ordenar matriz de filtros **W**:

Clasificador	P.A. Entr. Online (%)	Desviación estándar media	P.A. Test.(%)	Desviación estándar media	T.max medio Ent. (s)	T.max. medio Test (s)
LDA lineal paso	65.8	3.7	65.5	3.2	0.06	0.015
LDA función linear	66.7	4.0	64.8	3.4	0.298	0.049
LDA función diaglinear	59.5	2.9	66.4	3.9	0.23	0.06
LDA función Pseudolinear	67.0	4.1	64.8	3.4	0.3265	0.086
QDA función	64.6	3.5	65.3	3.7	0.23	0.05
Bayes paso	55.5	1.8	59.7	3.4	0.22	0.015

Kernel Naive Bayes Gaussiano func.	64.2	3.9	66.8	3.3	0.79	0.13
KNN func. k=7	65.3	4.6	66.5	3.4	0.216	0.05
Tree func. Nivel 1	60.0	3.6	66.9	3.5	0.27	0.07
SVM func. Con kernel gaussiana	65.3	4.7	67.6	3.5	0.437	0.05
Logistic Regression App	67.7	3.6	67.0	3.2	1.51	0.06

Tabla 6-6. Resultados con varios usuarios sin forzar ordenación de la matriz de filtros.

Con respecto a la probabilidad de acierto en entrenamiento online, ha dado mejor resultado el Logistic Regression, siendo el segundo el LDA pseudolineal realizado con funciones de MATLAB. En el proceso de testeo, en el cual los resultados son más significativos, se ha concluido que el SVM con kernel gaussiana es el que da mejor resultado.

Con respecto a tiempos máximos medios de procesamiento en el entrenamiento, el que ha dado menor valor ha sido el LDA, sin usar funciones especiales de MATLAB. El mayor ha sido con el Logistic Regression. En el testeo, igualmente, el LDA ha sido el de menor valor. El de Naive Bayes también obtuvo un valor de tiempo pequeño, pero en ése se obtiene una probabilidad de acierto que es casi aleatoria. Tener en cuenta que los resultados de tiempo pueden variar levemente, incluso realizando la misma simulación.

En definitiva, el que se consideraría con mejores resultados en probabilidad de acierto de testeo, sin forzar ordenación, sería el SVM con Kernel gaussiano.

- Pruebas con ordenación de matriz de filtros **W**:

Clasificador	**P.A. Entr. Online (%)**	**Desviación estándar media**	**P.A. Test. (%)**	**Desviación estándar media**	**T.max medio Ent (s)**	**T.max medio Test (s)**
LDA paso	65.9	3.7	64.6	3.4	0.11	0.04
LDA función linear	66.1	4.1	64.8	3.4	0.26	0.62
LDA función diaglinear	59.7	2.9	66.7	3.8	0.4	0.08
LDA función Pseudolinear	66.8	4.0	64.8	3.4	0.46	0.06
QDA función	65.1	3.4	65.8	3.6	0.46	0.02
Bayes paso	55.4	1.9	59.8	3.5	0.23	0.04
Kernel Naive Bayes Gaussiano func.	64.6	3.9	66.8	3.3	1.2	0.11
KNN func. k=7	65.3	4.6	66.5	3.4	0.4	0.09
Tree func. Nivel 1	60.0	3.6	66.9	3.5	0.48	0.04

Tree App Coarse	61.7	3.7	66.4	3.7	0.94	0.08
SVM func. Con kernel gaussiana	65.1	4.7	67.4	3.5	0.48	0.05
Logistic Regression App	67.6	3.6	67.2	3.3	3.35	0.13

Tabla 6-7. Resultados con varios usuarios con ordenación de la matriz de filtros.

Los resultados varían un poco cuando se fuerza a que esté ordenada la matriz de filtros, pero en algunos clasificadores prácticamente no varía el resultado.

Los resultados de tiempo máximo medio de los ensayos, como ya se mencionó, son una estimación aproximada de cuánto podría tardar como máximo, pudiendo variar levemente el resultado ejecutándose la misma sesión. El que parece que tarda menos en el entrenamiento es el clasificador LDA, el que se escribió el código paso por paso. El LDA es el más rápido y sencillo de implementar. En cuanto al tiempo de testeo, en general son tiempos pequeños, inferiores a un segundo.

Los resultados de probabilidad media de acierto están todos por debajo del 70%, pero cercanos a este valor. Este resultado no solo depende del clasificador, se recuerda que también afecta el número de ensayos utilizados para entrenar y para validar, así como el método utilizado para la validación, entre otros posibles factores. En algunos clasificadores realizados con la App, se quiso comprobar qué resultado daban con cross-validation, como ya se mencionó, resultando en los experimentados una probabilidad de acierto media del 70% aproximadamente.

Finalmente, se establece que el **SVM con Kernel gaussiano** es el que ha dado la mayor probabilidad media de acierto. Al haberse probado en varias sesiones, este resultado es más importante que el obtenido con un usuario.

6.3 Conclusiones

A continuación, se resume lo más importante de lo mencionado en este capítulo:

Se han probado en algunos clasificadores tres posibles maneras de programarlos (paso a paso, funciones especiales de MATLAB o con la aplicación de MATLAB Classification Learner) y en otros casos dos o una de esas maneras, tanto para un usuario como para varios usuarios.

Se hicieron diferentes pruebas con algunos clasificadores, como cambiar el número de filtros; cambiar el número de elementos para entrenamiento y test; cambiar cuántos ensayos hay que almacenar antes de empezar a clasificar; hacerlo con o sin CSP; hacerlo con o sin forzar la ordenación de los filtros; probar diferentes métodos de validación; usar programación sin o con rutinas internas de MATLAB; probar diferentes niveles de podado en los Decision Tree; probar diferentes valores de *k* en el KNN; y qué pasaría si fuera el entrenamiento offline en lugar de online.

Los resultados de probabilidad de acierto y tiempos de procesamiento del entrenamiento corresponden al caso de realizarlo online, no obstante, también se hizo una comprobación de qué resultados darían en caso de hacerse offline en algunos clasificadores. El entrenamiento online se adapta de forma dinámica al número de ensayos (***trials***) disponible en cada iteración. Al no disponer de todos los ensayos, éste tiende a infravalorar los resultados de acierto en entrenamiento (***training***) sobre el conjunto completo de datos. Por esta razón, la probabilidad de acierto en test (que se calcula tras el entrenamiento, sobre un conjunto de datos distinto), en general, puede llegar

a ser superior a la del entrenamiento online. Pero, como se ha explicado, eso no supone una contradicción teórica en el funcionamiento de los algoritmos. Cuando se realizaron pruebas con entrenamiento offline, se verificó que la probabilidad de acierto en entrenamiento era mayor a la de testeo en ese caso.

En el LDA la covarianza media total se probó a calcularla de dos formas. La primera forma era igualándola a la covarianza de la matriz de características. La segunda forma era igualándola a la suma ponderada de las covarianzas de cada clase, ponderándolas con las probabilidades de cada una. La probabilidad de acierto en el entrenamiento online difirió, pero la de testeo permaneció igual. Se podría suponer que no afecta mucho realizar el cálculo de una manera o de la otra.

Se comprobó también que, si se quitaba la ordenación de la matriz de filtros antes de aplicar entrenamiento del LDA, y solo se establecía al principio, cuando se obtiene la matriz de filtros con los ensayos en los que hay que esperar que haya un mínimo de uno de cada clase, se obtenían mismos resultados. Se comprobó que se mantenían ordenados en las actualizaciones de la matriz de filtros. Por otra parte, se vio que, si se quitaba también la ordenación en los primeros ensayos, empeoraba resultados del clasificador LDA. Se podría considerar que se reduce el efecto del CSP.

Algunos clasificadores daban mismos resultados de probabilidad de acierto en las pruebas de con o sin comando de forzar la ordenación de la matriz de filtros, no se veían afectados por esta modificación, podría deberse a que prácticamente se acaban obteniendo los mismos valores de los parámetros del algoritmo.

Por otra parte, se verificó que, al aumentar el número de filtros del CSP en el LDA, la probabilidad de acierto en el testeo bajaba. Además, si no se aplicaba CSP en los experimentos realizados con una sesión de un usuario, la probabilidad de acierto bajaba.

Se comprobó cómo varía la probabilidad de acierto en entrenamiento online y test según el número de ensayos que se cojan para cada grupo. Cuantos más ensayos se cojan para el entrenamiento online, mayor será la probabilidad de éste y de la de testeo. También hay que considerar que el resultado de probabilidad de acierto, en testeo, no será fiable en caso de haber pocos ensayos para testear el algoritmo.

No es muy significativo comprobar resultados con una única sesión, por ello se realizaron, además, pruebas con diferentes sesiones y usuarios.

La mayoría de las conclusiones que se dedujeron con un usuario, son aplicables al caso de varios usuarios. No obstante, se ha apreciado que el hecho de tener o no CSP afectaba a la media de probabilidad de acierto, de manera negativa o positiva según el algoritmo usado. El CSP suele provocar mejoras cuando se usa una base de datos que proviene de muchos sensores. Cuando el número de sensores utilizados no es relativamente grande, la mejoría depende en gran medida del tipo de clasificador que se utilice. Por otra parte, respecto a el método de validación del algoritmo, se considera relevante señalar que en varios clasificadores se apreció una mejora de resultados de probabilidad de acierto de testeo, de media, al usar validación cruzada (cross-validation).

Los resultados de tiempo máximo medio de los ensayos son una estimación aproximada de cuánto podría tardar el proceso como máximo, pudiendo variar levemente el resultado ejecutándose la misma sesión. El tiempo depende de las condiciones del ordenador en que se ejecute. El que parece que tarda menos en el entrenamiento es el clasificador LDA, el que se escribió el código paso por paso. El LDA es el más rápido y sencillo de implementar. En cuanto al tiempo de testeo, en general son tiempos pequeños, inferiores a un segundo.

Los resultados de probabilidad media de acierto están todos por debajo del 70%, pero cercano a este valor. Este resultado no solo depende del clasificador, también depende de otros factores, como el número de ensayos utilizados para entrenar y para validar, así como el método utilizado para la validación, entre otros.

Finalmente, se puede concluir que el **SVM con Kernel gaussiano** es el que ha dado la mayor probabilidad media de acierto. Al haberse probado en varias sesiones, este resultado es más importante que el obtenido

con un usuario. En general, los algoritmos de tipo SVM suelen ser más robustos que otros algoritmos y posibilitan trabajar con más características o sin CSP, al tiempo que no permiten el sobreajuste.

7 CÓDIGO EN MATLAB

"Inteligencia artificial, aprendizaje profundo, aprendizaje automático… te dediques a lo que te dediques, si no lo comprendes tienes que ponerte con ello y aprender qué es. Porque de lo contrario serás un dinosaurio dentro de 3 años."

Mark Cuban

Se realizará una breve explicación de los diferentes códigos realizados en MATLAB, es decir, los códigos de los clasificadores explicados anteriormente. Se expondrá con más profundidad el LDA, mostrándose los diagramas de flujos del programa correspondiente a la realización de los cálculos paso por paso. Las explicaciones hacen referencia a los códigos realizados para las pruebas con una única sesión, para varias sesiones y usuarios son los mismos con pequeñas modificaciones.

En lo que respecta a la aplicación Classification Learner, para comprobar el rendimiento de los clasificadores con ella, usando cross-validation folds, había que introducirle los vectores de características, siendo el último elemento del vector la etiqueta de la clase. Para esto se ejecutó primero **mi_script_Isabel_BUFFER_3.m**, obteniendo la matriz con vectores de características y el vector con las etiquetas reales. Luego se creó una nueva matriz, que era la de características con la fila de las etiquetas añadida al final de ésta. A continuación, se le indicó a la App que usase esa matriz y cogiese la última fila como clases. Después de hallar las probabilidades de acierto con la App, se exportaron los clasificadores que daban resultados interesantes para probarlos en un entrenamiento en tiempo real, procediendo como con el resto de los clasificadores programados sin la App.

7.1 CSP+LDA

7.1.1 Código realizado paso por paso

El código completo se encuentra en **mi_script_Isabel_BUFFER_3.m.** En este archivo se realiza la clasificación en tiempo real usando CSP y LDA. Primero se carga los datos de entrenamiento de mano izquierda y derecha de un usuario. Luego se especifica la longitud del ensayo (***trial***), el número de canales, el número de ***trials*** y el número de filtros al reducir dimensión; y se inicializan las variables a usar. Tener presente que, para cargar los datos, hay que indicar en el programa la ruta de la carpeta en la cual se hayan almacenado.

```
Ruta = 'C:\Users\isabe\Desktop\tfg\tfg matlab\Isabel\'

load([Ruta, 'c42a_7_class12_test.mat'])

x = DATA.x{1,1}{1}; %Tensor con todos los trial de la competición 500x22x144
```

```
long_data = size(x,1); %longitud de cada trial; 500

active_channels = size(x,2); %numero de canales; 22

n_Trials = size(x,3); %numero total de trials; 144

TensorData = zeros(active_channels,long_data,n_Trials); %Tensor de trials

C = zeros( active_channels, active_channels, n_Trials); %Covarianza de cada trial

TrLabels = zeros(n_Trials,1); %Clase de cada trial

nFilters = 8; %Numero de filtros

f = 0.1; %Constante de olvido

pac=0.5; % prob. acierto inicial

n_c1=0; %Numero de trials clase 1

n_c2=0; %Numero de trials clase 2

TeLabelEst = 0; %Clase estimada del trial

p_matrix = zeros(n_Trials-2,1); %Matriz que contiene las probabilidades de acierto

acierto_matrix = rand(n_Trials-2,1); %Matriz binaria que contiene el acierto de
cada trial

acierto_matrixAUX = rand(n_Trials-2,1); %Matriz auxiliar de acierto

j = 1; %variable para procesar primero medio trial y luego completo

trial_completo = 0;%variable para indicar si se ha procesado el nuevo trial entero

cc=0;

limit=2;%limit y cc es para imponer un número determinado de trial de obtenida
características antes de clasificar

sumacx1=zeros(active_channels,active_channels);

sumacx2=zeros(active_channels,active_channels);

sigmax_c1=0;

sigmax_c2=0;

sumafeatures1=0;

sumafeatures2=0;

clasificado=1;%como clasifico medio trial y luego entero, necesito esta variable
para que no recorra dos veces sumatorio características

T=1;

Nx=active_channels;

p=nFilters;

ini_class=0; %número trial que inicia clasificación
```

```
limi=0;%como limit, por si se quiere imponer restricciones de una clase

act_csp_2=0;%para activar obtención de sumacx1 y sumacx2 para una vez estemos
usando clasificador, para poder quitar el bucle for
```

Se estableció un factor de olvido inicial de 0.1. El factor de olvido es una manera de pesar de igual manera las medidas muy viejas y las nuevas. Las muestras se ponderan dándoles más o menos relevancia a sus valores pasados con respecto al último valor, según el parámetro de factor de olvido, ***f*** [217].

Se asumió que inicialmente había una probabilidad de 50% de darse cada clase. También se estableció que la probabilidad de acierto inicial era del 50%, luego ésta se iría actualizando con el uso del clasificador, con cada ensayo de la sesión.

Se recorren los ***trials*** que se tienen disponibles de uno en uno. Lo que se pretende es simular cómo sería si se estuviese entrenando el clasificador en tiempo real. Si se hiciese con un casco, el código sería similar. En lugar de cogerse los datos del amplificador, se cogen de un archivo '***.mat***'.

Se establece una cantidad de ***trials*** para realizar el entrenamiento y otra para verificar el buen funcionamiento del clasificador, para testeo. En el código hay primero un bucle ***for*** para entrenamiento, y luego otro para datos de testeo.

```
num=44;%numero de elementos que tomo como test

for i=1:n_Trials-num %entrenamiento

...

end

tiempo_max_c=max(tiempo);

if num==0

    return;

end
```

```
ini_class=n_Trials-num+1;

for i=ini_class:n_Trials %testeo

...

end
```

Al finalizar los bucles, se mostrará la probabilidad de acierto en entrenamiento online y en testeo, así como la tasa clasificación errónea en ambos casos. También se mostrarán las matrices de confusión y el tiempo máximo en que se tardó en procesar un ***trial***. Para los cálculos de probabilidad de acierto, se sabe cuál es la clase real de cada ***trial***.

El diagrama de flujo del algoritmo de entrenamiento del clasificador es:

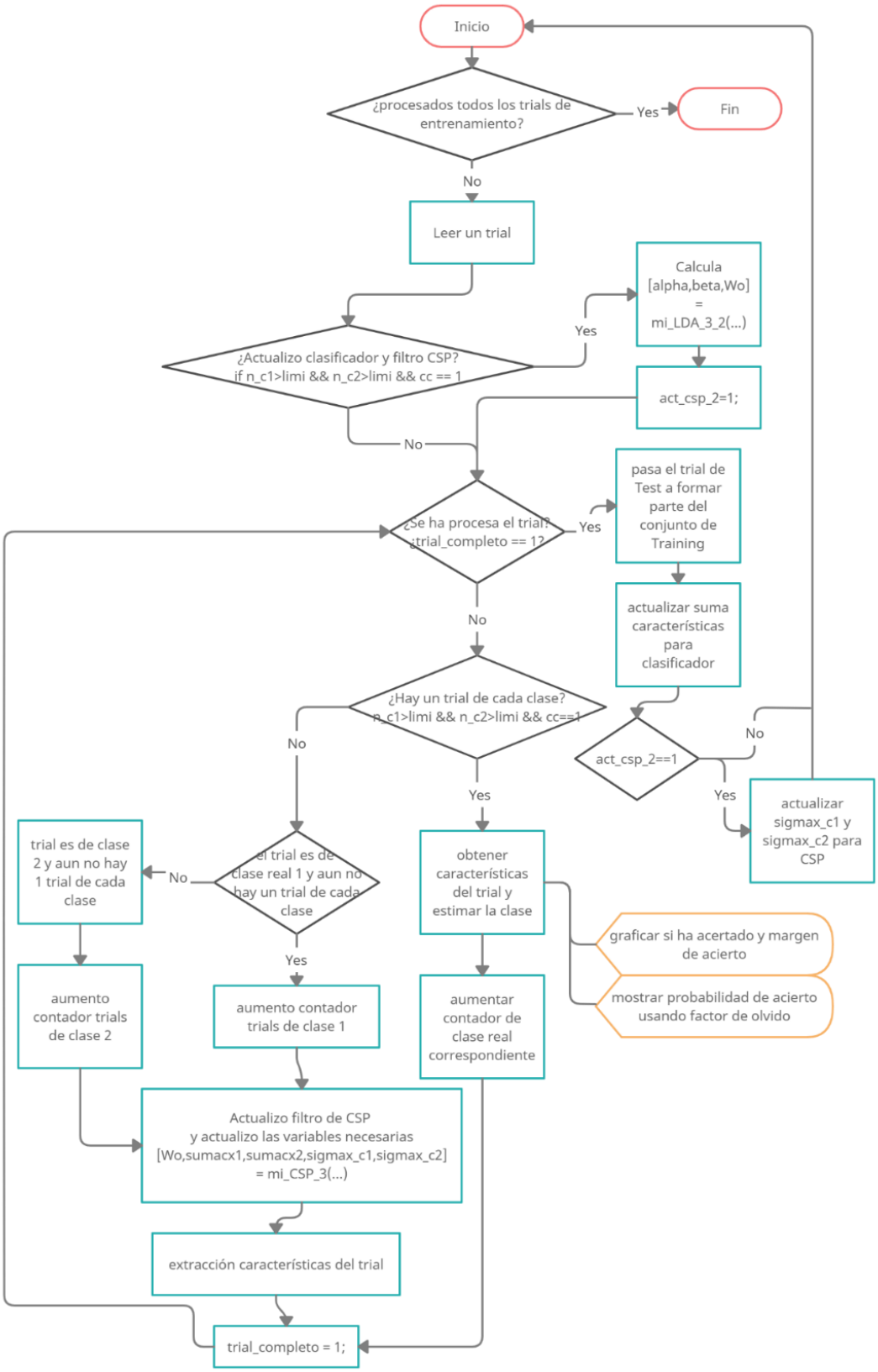


Figura 7-1. Diagrama flujo MATLAB CSP+LDA. Realizado con creately.com.

A continuación, se profundizará en la explicación del algoritmo. Las funciones usadas son:

- Función ***[alpha,beta,Wo] = mi_LDA_3_2(TrLabels,xTest,TrTensor, TrFeatures, ind1, ind2,active_channels,nFilters,i,sumacx1,sumacx2,sumafeatures1,sumafeatures2,sigmax_c1,sigmax_c2)*** :

En esta función se actualizan los parámetros del clasificador (alpha(α), beta(β)) y la matriz de filtros (***Wo***) de CSP. Las variables ***sigmax_c1, sigmax_c2***, para calcular ***Wo***, corresponden a $\hat{\Sigma}^{(1)}_{x|c_1}, \hat{\Sigma}^{(1)}_{x|c_2}$ de la explicación del CSP, el apartado 4.4. Con respecto a su cálculo, se programó de forma que se reutilicen las halladas en la iteración anterior, la del ensayo anterior de la misma sesión. También se reutiliza de una iteración a la siguiente las variables ***sumafeatures1*** y ***sumafeatures2*** del cálculo para LDA, siendo éstas el sumatorio de características de cada clase ($\sum_{\tau:c(\tau)=c_k} f_\tau$). Las actualizaciones de estas variables se realizan al final de procesar cada ***trial***.

```
n_c1=numel(ind1); %número trials de clase 1

n_c2=numel(ind2); %número trials de clase 2

Nx=active_channels; %número de canales; 22

p=nFilters; %%nFilters = 8; %Número de filtros

 %% CSP :hallar Wo

[V,D]=eig(sigmax_c1,sigmax_c2);

[B,indice]=sort(diag(D));

 V=V(:,indice);

Wo=[V(:,1:p/2),V(:,Nx-p/2+1:Nx)];

%% obtención clasificador con características de entrenamiento

mu1=(1/n_c1) *sumafeatures1;

mu2=(1/n_c2) *sumafeatures2;

sigma=cov(TrFeatures');%covarianza global (media global), tiene en cuenta todas las clases %     size(sigma)%8     8

is=inv(sigma);

alpha=is*(mu1-mu2); %vector de 8 filas, 8x1

% beta=0.5*(mu1+mu2)- ((log(pc1)-log(pc2)) *(mu1-mu2)/( (mu1-mu2)'*is*(mu1-mu2)));%vector de 8 filas, 8x1

beta=0.5*(mu1+mu2); %se comprobó que la probabilidad prácticamente no varía si le quito lo otro

% norm(beta-beta2) es casi cero
```

Se comprobó que, si al cálculo de *beta* se le quita el segundo término de la resta, no varía el resultado, $\frac{\log p(c_1) - \log p(c_2)}{(\mu_1-\mu_2)^T \hat{\Sigma}_f^{-1}(\mu_1-\mu_2)}(\mu_1-\mu_2)$ es prácticamente cero. La probabilidad de cada clase es aproximadamente igual. También es relevante tener presente que los datos (las señales) tienen de media cero, se les restó la media en la etapa de preprocesamiento.

- Función ***[Wo, sumacx1,sumacx2,sigmax_c1,sigmax_c2] = mi_CSP_3 (nFilters,TrTensor,n_c1,n_c2,sumacx1,sumacx2,TrLabels,active_channels,i)***:

Se utiliza para calcular el CSP cuando aún no se está actualizando el clasificador, aún no se tienen suficientes ***trials*** de cada clase. Es relevante mencionar que todos los clasificadores programados usan la función **mi_CSP_3** para los primeros ensayos, antes de empezar a clasificar.

```
 k=i;%número de trial en procesamiento

 T=1;%intervalo de tiempo para covarianza muestral

 Nx=active_channels;%número de canales(n° sensores); 22

 p=nFilters;%%   nFilters-->Numero de filtros,   nFilters = 8, "el número de
características que queremos extraer de las señales que se corresponde con el número
de filtros espectrales que se obtienen"

 X=TrTensor(:,:,k);%Se carga el trial i en X, siendo i el número del trial.%size
22   500

 Cx_0(:,:,k)=(X*X')*1/T;%matriz de covarianza muestral de X  % size    22    22

 Cx_1(:,:,k)=Cx_0(:,:,k)/(trace(Cx_0(:,:,k))/Nx);%matriz de covarianza muestreada
versión refinada%size   22    22

 if TrLabels(k)==1   %se hace suma de covarianzas de cada clase

     sumacx1=sumacx1+Cx_1(:,:,k);

 else

     sumacx2=sumacx2+Cx_1(:,:,k);

 end

  %% obtener sigma_c1 y sigma_c2 (covarianza media de cada clase):

 if n_c1>0 && n_c2>0

     sigmax_c1=sumacx1*1/n_c1;

     sigmax_c2=sumacx2*1/n_c2;

 elseif n_c1>0

     sigmax_c1=sumacx1*1/n_c1;

     sigmax_c2=sumacx2;

 elseif n_c2>0

     sigmax_c2=sumacx2*1/n_c2;

     sigmax_c1=sumacx1;

 else

     sigmax_c2=sumacx2;

     sigmax_c1=sumacx1;%le estoy diciendo que sea una matriz de ceros 22x22 al
inicio
```

```
 end

 %% Finalmente se obtiene Wo

 [V,D]=eig(sigmax_c1,sigmax_c2);

 [B,ind]=sort(diag(D));

 V=V(:,ind);

 Wo=[V(:,1:p/2),V(:,Nx-p/2+1:Nx)];%dato de salida:Wo--> filtro espacial a
optimizar, conjunto de N pesos de canal que constituyen un filtro espacial
particular
```

A la matriz de características del ***trial*** actual, se le calcula la covarianza muestral refinada de (4.14). Luego se realiza la suma de covarianzas muestrales y se obtiene la covarianza media de cada clase. Finalmente, se obtiene la matriz de filtros. Al principio, al no haber un ***trial*** de cada clase, se inicializa como una matriz de ceros siendo de tamaño $nº\,canales\,x\,nº\,canales$.

Una vez explicadas las funciones usadas, se mencionarán algunos detalles relevantes del código:

Para que el usuario sepa si está pensando de manera correcta la acción, se clasificará primero solo medio ***trial*** del ***trial*** que se esté procesando en ese momento, mostrándosele luego en pantalla si se ha clasificado correctamente o no, siendo el resultado complementado, además, con el margen de error. Después de eso, el clasificador vuelve a clasificar el mismo ***trial***, pero completo. También se verá mostrado en una gráfica si el completo lo clasifica bien, mostrando su margen de error. Esto da además una idea de cómo le afecta al clasificador tener solo medio ***trial***, se ve si es capaz de clasificarlo. La gráfica en cuestión es la siguiente:

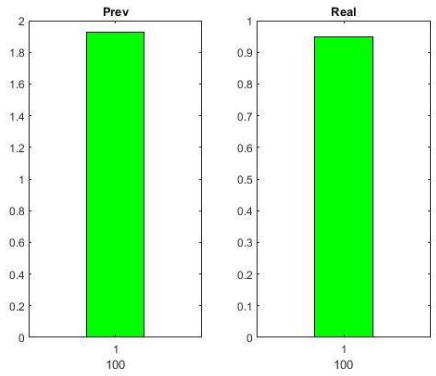


Figura 7-3. Ejemplo de clasificación correcta de medio ***trial*** y del ***trial*** completo, actuales. El número de debajo de cada barra es el número de iteración, el número de ***trial.***

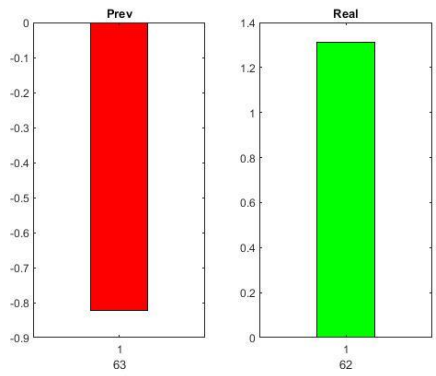


Figura 7-2. Ejemplo de tener incorrecto el medio trial de la itteración actual y tener correcto el ***trial*** completo de la iteración anterior.

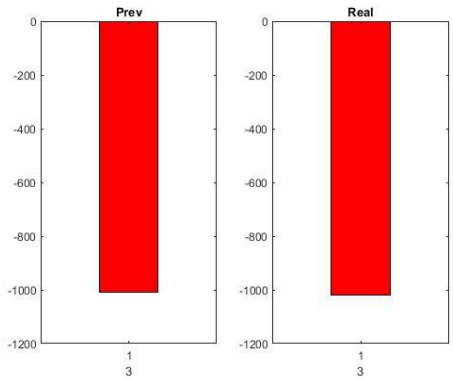


Figura 7-4. Ejemplo de tener incorrecto el medio ***trial*** y el ***trial*** entero de la iteración actual.

El tamaño de cada barra indica el margen de error, este margen es $\alpha^T(f_\tau - \beta)$ de la ecuación (5.9).

```
margen=alpha'*(TeFeature-beta);
```

La gráfica representada a la izquierda corresponde al resultado del medio ***trial*** y la de la derecha al del ***trial*** completo. Como se conocen las clases verdaderas, se comparan con las estimadas. Si es correcto la barra se colorea en verde, sino en rojo. Además, debajo de cada gráfica se muestra el número del ***trial*** que se está procesando, es decir, indica a cuál corresponde de los ***trials*** de una sesión. Como se ve en Figura 7-4, al principio habrá mucho margen de error porque se tienen pocas observaciones.

Cada vez que se actualiza el gráfico, se inserta una pausa en la ejecución del código, para que el usuario le dé tiempo a visualizarlo, y para que le dé tiempo a pensar. Para ello se usa el comando ***pause(),*** introduciéndole el número de segundos entre los paréntesis.

Otra forma que tiene el usuario de comprobar el correcto funcionamiento del clasificador es mirando la ventana de comandos, donde se mostrará la probabilidad de acierto teniendo en cuenta el factor de olvido.

```
 acierto =  (TeLabelEst==TeLabel); %comprobación acierto de clase estimada

 pac = (1 - f)*pac + f * acierto; %Calculo probabilidad de acierto usando factor
de olvido

 acierto_matrix(i-(ini_class-1)) = acierto;

 p_matrix(i-(ini_class-1)) = pac;

 fprintf('p_acierto: %3.1f | acierto = %1d | acierto_previo = %1d \n', pac, acierto,
aciertoAux);
```

Otro asunto que hay que comentar, es sobre la obtención de las características, pues hay dos formas equivalentes de obtenerlas. Una no es mejor que la otra. Se ha querido usar una de las formas para cuando no se está clasificando, al principio, y la otra para cuando sí.

La primera es:

```
 yTest = Wo'*xTest;%reducción de dimensión

 TeFeature = log(var(yTest,0,2));%extracción características
```

La segunda es:

```
cov_tes= Wo'*covt_test*Wo; %reducción de dimensión

dr=diag(cov_tes);

TeFeature=log(dr./sum(dr));%extracción características
```

Por otra parte, el cálculo de la covarianza se puede hacer de la manera explicada anteriormente o con el comando ***cov()*** de MATLAB. Sin embargo, la estimación de la covarianza que se ha explicado no es la que realiza MATLAB, usa una estimación distinta. Es válida, pero las matrices de covarianza no serán iguales a las obtenidas con el otro método.

El algoritmo de testeo difiere del de entrenamiento en que no se actualiza el clasificador ni el filtro del CSP, se usan los últimos parámetros que se consiguieron en el entrenamiento. Por tanto, no es necesario esperar a que haya un ***trial*** de cada clase como en el código de entrenamiento. El diagrama de flujo del algoritmo de testeo es:

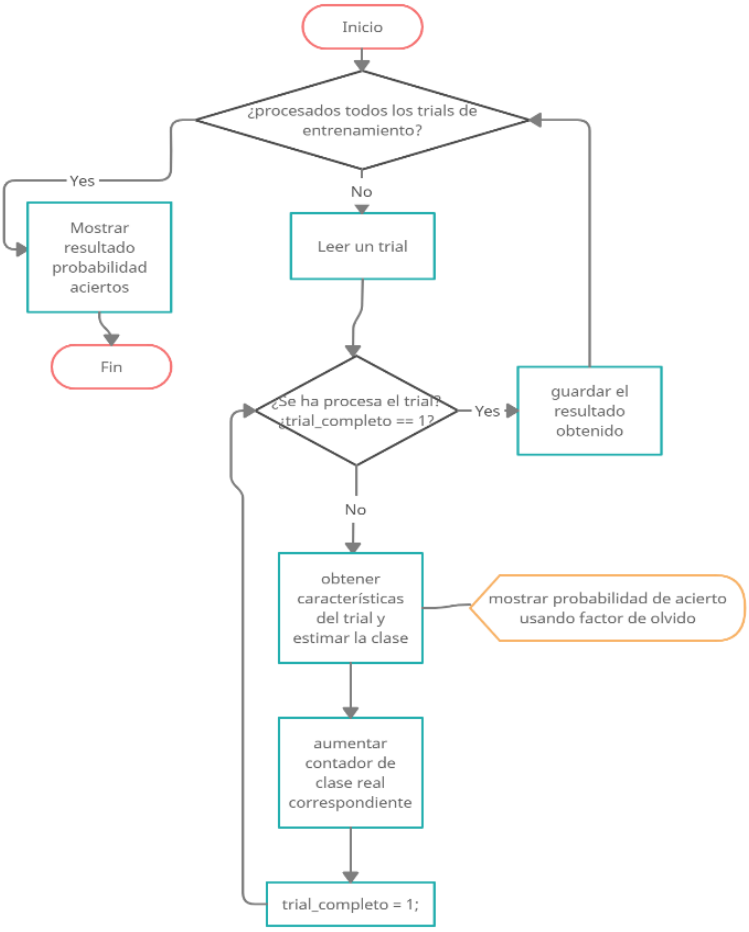


Figura 7-5.Diagrama flujo testeo. Realizado con creately.com.

Una vez realizado el entrenamiento y la validación, se procede con el cálculo de la tasa de predicción de cada uno:

```
 p_acert_c=sum(TrLabels(ini_class:n_Trials-num)'==
T_estLabels(ini_class:n_Trials-num))/numel(T_estLabels(ini_class:n_Trials-
num))*100;
```

```
 p_acert=sum(TrLabels(ini_class:n_Trials)'==T_estLabels(ini_class:n_Trials)
)/numel(T_estLabels(ini_class:n_Trials))*100;
```

Y la tasa clasificación errónea es:

```
 vector_incorrectas=TrLabels(limit+1:ini_class-1)'~=
T_estLabels(limit+1:ini_class-1);

misclassrate=sum(vector_incorrectas )/numel(T_estLabels(limit+1:ini_class-1))*100;
```

```
 vector_incorrectas_test=TrLabels(ini_class:n_Trials)'~=
T_estLabels(ini_class:n_Trials) ;

 misclassrate=sum(vector_incorrectas_test)/numel(T_estLabels(ini_class:n_Trials))
*100;
```

Para medir el tiempo, se usaron los comandos ***tic*** y ***toc***:

```
 tic;

 ...

 tiempo(i)= toc;
```

Variables	Utilidad
n_Trials	Número de *trials* disponibles en una sesión:144
x	Tensor con todos los *trial* de la competición 500x22x144
i	Número de iteración, qué *trial* se está estudiando
xTest, X, xAux2	Se carga tensor del *trial* número **i**.
xAux1	Se carga la mitad del *trial* número **i**
TeLabel	Clase real del *trial* i
long_data	Longitud de cada *trial*: 500
f	Constante de olvido
pac	Probabilidad de acierto del clasificador, probabilidad de acierto usando factor de olvido.
n_c1, n_c2	Número de *trials* de cada clase (real) que se han almacenado.
TeLabelEst	Clase estimada de un *trial*
TeLabelEstAux	Clase estimada de medio *trial*

j	Variable para procesar primero medio *trial* y luego completo
trial_completo	Variable para indicar si se ha procesado el nuevo *trial* entero
limit	Número determinado de *trial* de obtención de características antes de entrenar clasificador
cc	Ayuda a establecer el límite de *trials* almacenados junto a **limit**. Indica cuándo se empieza a clasificar, valiendo 1 en ese momento.
sumacx1, sumacx2	Suma covarianzas refinadas de cada clase respectivamente
sigmax_c1, sigmax_c2	Parámetros sigma del CSP, la covarianza media de cada clase respectivamente
sumafeatures1, sumafeatures2	Sumatorio de vectores de características de clase 1 y clase 2 respectivamente. Se usa para clasificación LDA
clasificado	Como se clasifica medio *trial* y luego entero, necesito esta variable para que no recorra dos veces sumatorio características
T	Periodo entre muestras, para cálculo de la covarianza
Nx, active_channels	Número de canales: 22
p, nFilters	Numero de filtros, número de características para un ensayo: 8
ini_class	En entrenamiento es número de *trial* en que se empieza a clasificar. En testeo es número de *trial* en que empieza el testeo.
limi	Como **limit**, por si se quiere imponer restricciones de número de *trials* solo de una clase
act_csp_2	Para activar obtención de **sumacx1** y **sumacx2** una vez empiece a usar clasificador
num	Número de elementos que se toman como test
cov_tes	Covarianza de *trial* completo con reducción de dimensionalidad **Wo** aplicada
ce	Covarianza de medio *trial* con reducción de dimensionalidad **Wo** aplicada
covt_aux1	Covarianza de medio *trial* para hallar características
covt_test	Covarianza de trial completo (**xAux2**) para hallar características
TeFeature	Vector características del *trial* actual en estudio
margen	Margen de error. Como de cerca está de una clase u otra, ver si se aleja mucho del valor 0, positiva o negativamente.
aciertoAux	Comprobación acierto de mitad del *trial* actual
acierto	Comprobación acierto de *trial* actual
yTest	El **xTest** reduciendo dimensionalidad con **Wo**.
TrLabels	Etiqueta de clases de entrenamiento

Wo	Matriz de filtros de reducción de dimensionalidad, CSP
Cx_0	Covarianza de **TrTensor**
Cx_1	Covarianza refinada de **TrTensor**
T_estLabels	Vector con clases estimadas
TrFeatures	Matriz con vectores de características de entrenamiento
ind1, ind2	Índice *trials* clase 1 y 2 respectivamente, en **TrLabels**
TrTensor	Tensor *trials* entrenamiento almacenados
tiempo_max_c	Tiempo máximo que se ha tardado en entrenamiento en procesar un *trial*
p_acert_c	Probabilidad acierto entrenamiento
Alpha	Parámetro LDA, para realizar proyección.
Beta	Parámetro LDA, el umbral de separación entre clases.
tiempo_max	Tiempo máximo que se ha tardado en testeo en procesar un *trial*
p_acert	Probabilidad acierto testeo
misclassrate	Tasa clasificación errónea entrenamiento o test

Tabla 7-1. Variables más importantes usadas en el programa con LDA

7.1.2 Aprovechando propiedades de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_analisis_discriminante_lda.m.**

Para poder usarse el modelo, se tuvo que aumentar el mínimo de observaciones a esperar antes de clasificar. Se estableció que tenía que haber un mínimo de 3 observaciones para poder empezar a clasificar.

```
limit=3;
```

En este caso, se usa las funciones de MATLAB que permiten crear modelos de clasificación. La creación del modelo quedaría así:

```
mdl= fitcdiscr(TrFeatures' ,TrLabels(1:i-1));%lineal
```

Por defecto el tipo de discriminación es lineal, pero hay más tipos. Se especifica con la propiedad "***DiscrimType***", la cual puede ser ***lineal, diaglinear, pseudolinear, quadratic, diagquadratic*** y ***pseudoquadratic.*** Para más información sobre la función se puede consultar: [218].

Para la predicción, en caso de una la función de MATLAB, siempre se usa:

```
TeLabelEst= predict(mdl,TeFeature');
```

Cuando se usen estas funciones de MATLAB, se va a calcular, aparte de lo mencionado en clasificadores anteriores, la pérdida por resustitución en el modelo:

```
errTrain = resubLoss(mdl);
```

Y la pérdida basada en la distribución de datos:

```
mdlLoss=loss(mdl,TrFeatures(:,ini_class:n_Trials)',TrLabels(ini_class:n_Trials))
;%the classification lost

errTest = mdlLoss;
```

7.1.3 Usando App de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_app_lda.**

En este caso, se tiene que establecer que haya un mínimo de 4 observaciones para poder empezar a clasificar.

Se utilizó la aplicación ***Classification Learner*** de MATLAB:

```
  [trainedClassifier,validationAccuracy]=
trainClassifier_linear_dis([TrFeatures;TrLabels(1:i-1)']);
```

La función de entrenamiento del clasificador se exportó de la App.

```
TeLabelEst=trainedClassifier.predictFcn(TeFeature); %estimación de la clase del
trial %numero 1 o 2
```

- ***trainedClassifier***: una estructura que contiene el clasificador entrenado. La estructura contiene varios campos con información sobre el clasificador entrenado.

- ***trainedClassifier.predictFcn***: una función para hacer predicciones sobre nuevos datos.

- ***validationAccuracy***: un doble que contiene la precisión en porcentaje. En la aplicación, la lista del historial muestra esta puntuación de precisión global para cada modelo.

7.2 CSP+QDA

7.2.1 Aprovechando propiedades de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_analisis_discriminante_qda.m .**

En este caso, se tiene que establecer que haya un mínimo de 20 observaciones para poder empezar a clasificar.

```
limit=20;
```

```
mdl=fitcdiscr(TrFeatures',TrLabels(1:i-1),"DiscrimType","quadratic");%cuadratico
```

7.2.2 Usando App de MATLAB

En este caso, se tiene que establecer que haya un mínimo de 23 observaciones para poder empezar a clasificar.

```
[trainedClassifier,validationAccuracy]=trainClassifier_qda(([TrFeatures;TrLabels
(1:i-1)']));
```

7.3 CSP+BAYES

7.3.1 Código realizado paso por paso

El código completo se encuentra en **mi_script_Isabel_BUFFER_4_bayesiano.m.** El número de ensayos a esperar se mantuvo en 2.

Se establece la misma probabilidad de la clase para mano derecha e izquierda, un 50%.

```
pC1 = pac;

pC2 = pC1;
```

La obtención de los parámetros del clasificador Naive-Bayes durante el entrenamiento, se codifica de la siguiente manera:

```
Nprot1 = n_c1;

Nprot2 = n_c2;

%% Cálculo de media y varianza de ambas clases:

mu1  = mean(Z1')';

V1 = zeros(8,8);

for ii = 1:Nprot1

z = Z1(:,ii);

V1 = V1 + (z-mu1)*(z-mu1)';

end

V1 = V1/Nprot1;

mu2  = mean(Z2')';

V2 = zeros(8,8);

for ii = 1:Nprot2

z = Z2(:,ii);

V2 = V2 + (z-mu2)*(z-mu2)';

end

V2 = V2/Nprot2;

%% Precalculamos algunas cosas:

% El segundo término de la función de decisión de cada clase:

% fd_i(x) = -1/2*r_i^2(x) + f_i

f1 = log(pC1) - 1/2 * log(det(V1));

f2 = log(pC2) - 1/2 * log(det(V2));

% La inversa de la matriz de covarianza:
```

```
V1inv = inv(V1);

V2inv = inv(V2);
```

Una vez actualizados los parámetros, la asignación de la clase sería:

```
% Distancia de Mahalanobis a cada una de las clases:

r1Cuad = (TeFeature-mu1)' * V1inv * (TeFeature-mu1);

r2Cuad = (TeFeature-mu2)' * V2inv * (TeFeature-mu2);

% Cálculo de las funciones de decisión:

fd1 = -1/2 * r1Cuad + f1;

fd2 = -1/2 * r2Cuad + f2;

[fdMax, clase] = max ([fd1,fd2]);
```

En ***Z1*** y ***Z2*** se van almacenando los patrones de cada clase correspondiente durante el entrenamiento.

Los parámetros que se usan en el testeo son el último valor de la media y la matriz de covarianza inversa obtenidas.

En este clasificador se ha querido representar la distancia de Mahalanobis en cada iteración, para medio ***trial*** y ***trial*** completo, siendo ésta la altura de la barra:

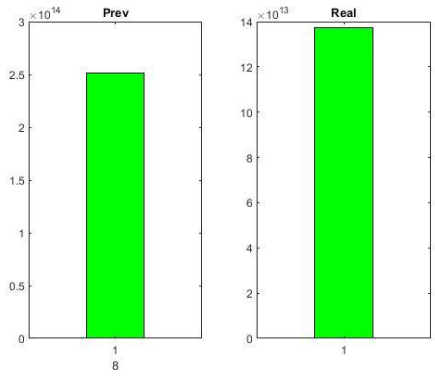


Figura 7-7. Distancia Mahalanobis al inicio.

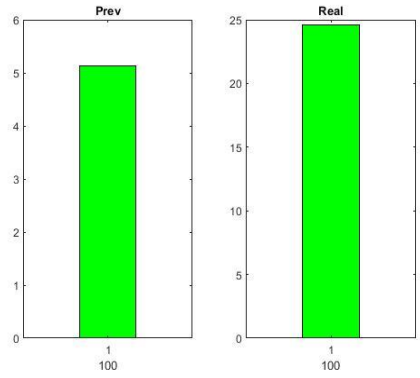


Figura 7-6. Distancia Mahalanobis al final.

Como se ve en Figura 7-7 y Figura 7-6, al principio la distancia es muy grande con pocas observaciones, luego disminuye.

7.3.2 Aprovechando propiedades de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_naive_bayes.m**. Para poder usarse el modelo, se tuvo que establecer que esperase que hubiese 4 patrones antes de clasificar.

```
mdl=fitcnb(TrFeatures' ,TrLabels(1:i-1));
```

Una forma de mejorar el resultado sería usando Kernel Naive Bayes:

```
mdl = fitcnb(TrFeatures' ,TrLabels(1:i-1), "DistributionNames","kernel"); %con
esto mejora resultado
```

Lo denominado Distribuciones de predictores (***DistributionNames***) se usa para modelar los datos. En el caso de ***kernel*** sería la "*estimación de la densidad de suavizado del núcleo. Entrena el clasificador utilizando el tipo más suave del* ***kernel*** *en el elemento. El software ignora elementos que no corresponden a un predictor cuya distribución es* ***kernel***" [219]. Si no se especifica, el tipo de ***kernel*** por defecto es el Gaussiano. También podría ser de tipo Caja, Epanechnikov o Triangular. Para más información sobre la función se puede consultar: [220].

7.4 CSP+KNN

7.4.1 Aprovechando propiedades de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_knn.m**. El mínimo de ensayos que tiene que haber antes de clasificar, se mantuvo en 2.

```
mdl = fitcknn(TrFeatures' ,TrLabels(1:i-1) ,"NumNeighbors",10);

mdl.DistanceWeight = "squaredinverse";

mdl.BreakTies="nearest";
```

La *k* se establece con la propiedad ***"NumNeighbors".*** Si no se le indica nada al comando de MATLAB, se ajusta a un modelo KNN con *k*=1. Esto quiere decir que el modelo utiliza solo el ejemplo más cercano conocido para clasificar una observación dada. Provocando que el modelo sea sensible en los casos que hay valores extraños en los datos de entrenamiento. Existe la posibilidad que los nuevos patrones cerca de los valores atípicos se clasifiquen mal [146]. Se puede hacer que el modelo sea menos sensible a los patrones aumentando la *k*, se utiliza la clase más común de muchos vecinos. La *k* que se escoja depende de cómo sean los patrones para clasificar.

La propiedad ***DistanceWeight*** es la función de ponderación de distancia, "*puede ser* ***("equal")*** *sin ponderación,* ***("inverse")*** *el peso es de 1/distancia,* ***("squaredinverse")*** *el peso es de 1/distancia*^{2} *o usar una función que acepta una matriz de distancias no negativas y devuelve una matriz del mismo tamaño que contiene pesos de distancia no negativos*" [221].

El ***BreakTies*** es el algoritmo de desempate "*cuando varias clases tienen el mismo costo más pequeño, utilizado por el método de predicción. Puede ser que utilice el índice más pequeño entre los grupos atados ('****smallest****'); utilice la clase con el vecino más cercano entre los grupos empatados ('****nearest****'); o utilice un desempate aleatorio entre los grupos empatados ('****random****')*" [221].

7.5 CSP+Decision Trees

7.5.1 Aprovechando propiedades de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_tree.m**. El número de ensayos a esperar se mantuvo en 2.

```
mdl = fitctree (TrFeatures' ,TrLabels(1:i-1));

mdl = prune(mdl,"Level",3);
```

La función devuelve "*un árbol de decisión de clasificación binaria ajustado basado en las variables de entrada*". Para más información sobre la función se puede consultar: [222].

La función ***'prune'*** sirve para podar un árbol entrenado, devuelve el árbol podado hasta el nivel que se le indique. La propiedad "***Level***" es un escalar numérico que puede ser desde 0 (sin poda) hasta el mayor nivel de poda de este árbol (***max(tree.PruneList***)).

Si no se le indica nada, devuelve el árbol de decisión completo, sin podar, pero con la información de poda óptima añadida. Esto es útil sólo si se ha creado el árbol podando otro árbol, o utilizando la función ***fitctree*** con la poda desactivada. Si se planea podar un árbol varias veces a lo largo de la secuencia, es más eficiente crear primero la secuencia de poda óptima. Para más información sobre la función consulte: [223].

Para visualizar el árbol obtenido se usa:

```
view(mdl,'Mode','graph');
```

7.5.2 Usando App de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_app_tree.m.** El número de ensayos a esperar se mantuvo en 2.

Si se usa Coarse Tree:

```
 [trainedClassifier, validationAccuracy] =
trainClassifier_tree_coarse_tree([TrFeatures;TrLabels(1:i-1)']);
```

7.6 CSP+SVM

7.6.1 Aprovechando propiedades de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_svm.m.** El número de ensayos a esperar se mantuvo en 2.

```
mdl = fitcsvm(TrFeatures' ,TrLabels(1:i-1));
```

Existe posibilidad de usar funciones ***kernel*** y de diferentes tipos.

```
mdl = fitcsvm(TrFeatures',TrLabels(1:i-1),"KernelFunction","gaussian");
```

```
mdl = fitcsvm(TrFeatures',TrLabels(1:i-1),"KernelFunction","polynomial");
```

El SVM "*admite la asignación de datos predictores mediante funciones de **kernel** y admite la optimización mínima secuencial (SMO), el algoritmo iterativo de datos únicos (ISDA) o 1 minimización de margen flexible a través de la programación cuadrática para la minimización de la función objetiva*" [224].

"*Para entrenar un modelo SVM lineal para la clasificación binaria en un conjunto de datos de alta dimensión, es decir, un conjunto de datos que incluye muchas observaciones se puede usar en su lugar la función '**fitclinear**'*" [224].

La ***KernelFunction*** puede ser "*lineal (predeterminado para el aprendizaje de dos clases), gaussiana (por defecto para el aprendizaje de una clase) o polinomial. Permiten convertir lo que sería un problema de clasificación no lineal en el espacio dimensional original, a un sencillo problema de clasificación lineal en un espacio dimensional mayor*" [225].

Para más información sobre la función se puede consultar: [226].

7.7 CSP+ Logistic Regression

7.7.1 Usando App de MATLAB

El código completo se encuentra en **mi_script_Isabel_BUFFER_machine_app_logistic_regression.m.** En este caso se tiene que establecer que haya un mínimo de 7 observaciones para poder empezar a clasificar.

Se programó con la aplicación de MATLAB:

```
 [trainedClassifier,validationAccuracy]=
trainClassifier_logistic_regression([TrFeatures;TrLabels(1:i-1)']) ;
```

La función de entrenamiento del clasificador exportado de la App:

```
 TeLabelEst=trainedClassifier.predictFcn(TeFeature);%estimación de la clase del
trial %numero 1 o 2
```

7.8 CSP+ Conjunto de clasificadores (Ensemble Classifiers)

7.8.1 Usando App de MATLAB

7.8.1.1 Subspace KNN

El código completo se encuentra en mi**_script_Isabel_BUFFER_machine_app_ensemble_subspace_knn.** En este caso se tiene que establecer que haya un mínimo de 4 observaciones para poder empezar a clasificar.

```
 [trainedClassifier,validationAccuracy]=
trainClassifier_subspace_knn([TrFeatures;TrLabels(1:i-1)']);
```

Para la implementación del otro tipo de clasificador Ensemble probado, Subspace Discriminant, se utilizó un procedimiento análogo, entrenando el correspondiente con la App de MATLAB.

7.9 Varios usuarios

Para probar los clasificadores para varias sesiones y varios usuarios, se utiliza **my_repit_sesiones.m.** En él se indicará qué clasificador se quiere usar, comentando y descomentando las posibles opciones.

Se creó un nuevo archivo para cada clasificador, adaptado en forma de función, pudiéndose probar con diferentes usuarios. Las funciones son:

```
[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo,alpha,beta]=mi_script_Isabel_BU
FFER_LDA_usuarios(ruta,archivo,num);

[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo,mdl]=mi_script_Isabel_BUFFER_fu
ncion_lda_lineal_usuarios(ruta,archivo,num);


[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo,mdl]=mi_script_Isabel_BUFFER_QD
A_usuarios(ruta,archivo,num);

[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo]=mi_script_Isabel_BUFFER_bayes_
usuarios(ruta,archivo,num);
```

```
[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo,mdl]=mi_script_Isabel_BUFFER_kn
n_usuarios(ruta,archivo,num);

[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo,mdl]=mi_script_Isabel_BUFFER_tr
ee_usuarios(ruta,archivo,num);

[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo]=mi_script_Isabel_BUFFER_coarse
_tree_usuarios(ruta,archivo,num);


[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo,mdl]=mi_script_Isabel_BUFFER_sv
m_usuarios(ruta,archivo,num);

[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo]=mi_script_Isabel_BUFFER_logist
icregression_usuarios(ruta,archivo,num);

[p_acert_c,p_acert,tiempo_max_c,tiempo_max,Wo]=mi_script_Isabel_BUFFER_naiveB
ayesfuncion_usuarios(ruta,archivo,num);
```

Dentro de las funciones, se podrán seleccionar las propiedades del clasificador, modificando líneas de código. Por ejemplo, en SVM se puede seleccionar si usar Kernel polinomial o gaussiano.

También se creó un programa para probar un clasificador con 144 de testeo y 144 de entrenamiento, **my_repit_usuarios.m**. Este solo se probó con LDA de paso a paso, creándose un programa de testeo aparte, denominado **mi_script_Isabel_BUFFER_LDA_usuarios_test.m**.

7.10 Conclusiones

"Se ha realizado la programación de la interfaz en MATLAB. Se ha probado en algunos clasificadores unas tres posibles maneras de programarlos (paso a paso, funciones especiales o con la aplicación) y en otros casos dos o una de esas maneras". Se ha comentado como se ha programado cada uno y los archivos **.m** a los que hace referencia cada código. Se hicieron códigos para el caso de una sesión y para el caso de varias sesiones.

En la programación, lo que se hizo fue imitar como sería extraer los datos y clasificar en tiempo real. Se iban cogiendo de uno en uno los ensayos de los datos de competición, acumulándose a los que se consideran los datos de entrenamiento, hasta llegar a un determinado número y los siguientes serían de testeo. El clasificador se inicia cuando tiene al menos una observación de cada clase, siendo dos clases las posibles. En algunos casos se esperó a que hubiera más ensayos de cada clase, porque así lo requerían algunos clasificadores para su correcto funcionamiento.

En el caso del LDA realizado paso a paso, se incluyó que apareciese una gráfica del margen de error durante el entrenamiento y si lo estaba haciendo bien, para que el usuario pudiese corregir su pensamiento si estaba haciendo mal la imaginación motora. En Naive Bayes realizado paso a paso se incluyó gráfica de la distancia de Mahalanobis y si lo estaba haciendo bien o no.

8 CONCLUSIONES Y LÍNEAS FUTURAS

"Siempre he estado convencido de que la única forma de hacer que funcione la inteligencia artificial es hacer el cálculo de manera similar al cerebro humano. Ese es el objetivo que he estado persiguiendo. Estamos progresando, aunque todavía tenemos mucho que aprender sobre cómo funciona realmente el cerebro."

Geoffrey Hinton

Se procede a hacer un repaso de lo visto en el documento sacando algunas conclusiones al respecto. También se comentará brevemente posibles ampliaciones de este proyecto.

8.1 Conclusiones

Los sistemas BCI están todavía en desarrollo, pero ha habido muchos avances en investigación en este campo en los últimos años. La importancia de esta tecnología reside principalmente en lograr que las personas puedan controlar dispositivos solo con sus pensamientos, facilitando la vida de muchas personas, especialmente las discapacitadas.

En este trabajo se ha investigado sobre el funcionamiento de los MI-BCI, algunos métodos de obtención de características y reducción de dimensión de las señales de entrada, así como diferentes algoritmos de Machine Learning. Después de realizar estas averiguaciones, se procedió a realizar el objetivo del trabajo: analizar la eficiencia de diferentes clasificadores para un MI-BCI, simulando el entrenamiento del clasificador en tiempo real a partir de una base de datos. Se realiza la comprobación de resultados de probabilidad de acierto en entrenamiento y testeo, así como del tiempo en clasificar. Las señales cerebrales registradas correspondían a un conjunto de usuarios que imaginaron el movimiento de la mano derecha o izquierda. Los datos se dividían en sesiones y estos a su vez en ensayos.

De los diferentes métodos de reducción de dimensionalidad y obtención de características que hay, se han explicado en este documento de manera simple los siguientes: FFT, AR, WT, CSP, FBCSP. Se ha estudiado de

manera más exhaustiva el CSP por ser el más popular en usarse en MI-BCI. Es el único que se llegó a implementar en MATLAB.

Hay multitud de algoritmos de aprendizaje automático. En este proyecto se tuvo como misión investigar aquellos que puedan ser eficientes en la clasificación de dos clases y sean sencillos de implementar, aunque también se han mencionado otros que están pensados para casos multiclase. Los clasificadores sobre los que se ha buscado información son: LDA, QDA, RDA, SVM, Naive Bayes, KNN, k-Means, Decision Trees, Logistic Regression, Ensemble Classifiers. Todos ellos se han implementado en MATLAB excepto el de K-Means y el de RDA. De los Ensemble Classifiers existen muchos tipos, se sacaron resultados de los que se denominan Subspace Discriminant y Subspace KNN.

Se implementaron y probaron en MATLAB los diferentes algoritmos de aprendizaje automático supervisado, para diferenciar cuándo se pensaba en mano derecha o mano izquierda. El entrenamiento, como ya se mencionó, se realizó simulando el procedimiento que se llevaría a cabo en caso de realizarse de forma online. El entrenamiento online se adapta de forma dinámica al número de ensayos disponible en cada iteración. Al no disponer de todos los ensayos, éste tiende a infravalorar los resultados de acierto en entrenamiento sobre el conjunto completo de datos. Por esta razón, la probabilidad de acierto en testeo, la cual se calcula tras el entrenamiento sobre un conjunto de datos distinto, en general, puede llegar a ser superior a la del entrenamiento online. Ahora bien, esto no supone una contradicción teórica en el funcionamiento de los algoritmos. En caso de realizar los mismos procesos de manera offline, se comprueba que en efecto se obtiene lo teóricamente esperado, la probabilidad de acierto del entrenamiento es mayor que la de testeo.

De los clasificadores probados, en general, el que dio mejor resultado en probabilidad de acierto en validación fue el SVM con Kernel gaussiano. Sin embargo, el resultado de probabilidad de acierto medio de los usuarios era en torno a un 67.4%, no está muy lejos del 50%, que sería una clasificación aleatoria. No se llegaron a implementar clasificadores no supervisados, pero darían resultados peores que los supervisados [38]. Una probabilidad del 67.4% no es suficiente para implementarlo en un sistema de uso real. Para obtener una comunicación fiable se debe lograr una tasa de acierto de como mínimo un 70% [5], aunque no está muy lejos de este valor. Lo deseable sería conseguir una probabilidad media de acierto mayor del 90%, pero esto actualmente es bastante complicado.

No obstante, como se ilustró en el apartado 6.3, la probabilidad de acierto depende de cada usuario, el número de ensayos usados durante el entrenamiento y testeo, el número de filtros en el CSP o si no se usa, el método con el que se realiza la verificación del clasificador, si el entrenamiento se realiza online u offline, el método para calcular las covarianzas, entre otros factores posibles.

Un hecho a resaltar es que había usuarios que obtenían mejor probabilidad que otros. En el apartado 6.1.8, se pudo apreciar que había un usuario que logró una probabilidad de más del 80%. Además, se pudo percibir que lograba probabilidades altas con varios de los clasificadores probados, siendo mayor en el Decision Tree de nivel 1 con 97.73%. Por consiguiente, se podría afirmar que el rendimiento de este sistema depende en gran medida de la habilidad del usuario para realizar la imaginación motora. Los usuarios podrían aprender a mejorar su rendimiento, se podría realizar más ensayos con el mismo usuario hasta conseguir que aprenda a imaginar los movimientos. El inconveniente principal del sistema es que no es compatible de un usuario a otro una vez realizado el entrenamiento, hay que volver a entrenar con el nuevo usuario si se quiere conseguir una probabilidad alta en la mayoría de los casos. Cada cerebro es distinto, habrá personas que deban entrenar más para lograr un mejor rendimiento.

El usuario deberá aprender a utilizar la interfaz y acostumbrarse a ella, para así lograr alta probabilidad de acierto. Un ejemplo sería cuando se juega por primera vez a un videojuego, hay que aprender a usar los controles, una vez te acostumbras te cuesta menos manejarlo que la primera vez que jugaste. Sin embargo, la imaginación motora resulta mucho más complicada de enseñar, debido a la complejidad del pensamiento humano y lo abstracto que

puede resultar. Tener en cuenta que no es lo mismo para el usuario mover físicamente el brazo, que imaginar que mueve el brazo sin llegar a moverlo.

8.2 Líneas futuras

El campo de las BCI sigue actualmente en desarrollo y todavía hay que mejorar muchos aspectos para que su uso esté más extendido.

Uno de los principales asuntos a considerar son los beneficios y desventajas de los métodos invasivos. El segundo asunto sería la forma de detectar y caracterizar las señales cerebrales necesarias. *"La mayoría de los estudios de BCI han tratado el dominio del tiempo, la frecuencia y el espacio de las señales cerebrales de forma independiente. Estudios sobre las interdependencias entre las diferentes dimensiones de las señales pueden conducir a una mejora significativa en el rendimiento de BCI"* [6] . El tercer asunto es la tasa con la que se adquiere información para convertirla en comandos, en los BCI no es lo suficientemente rápida en algunas aplicaciones. El cuarto asunto es que el BCI debe adaptarse a cualquier usuario que lo use, lo cual requiere de mucho entrenamiento en la mayoría de estos sistemas, sin lograrse probabilidades de acierto muy altas. El quinto asunto sería la comodidad y colocación de los electrodos, es dificultoso e incómodo, en concreto los no invasivos.

Como posibles líneas de investigación futuras de este proyecto, se podría probar otros diferentes métodos de obtención de características, otras formas de clasificación o mejorar la obtención de la señal a la que se extraen características. Adicionalmente, se podría hacer que el algoritmo se adapte cada vez que un usuario lo usa partiendo de lo ya aprendido en sesiones anteriores, en lugar de empezar desde el principio el procedimiento.

También se probó a realizar el aprendizaje con una red neuronal (**red_neu_mi_script_Isabel_BUFFER.m**), se decidió descartarla debido a que el proceso de entrenamiento de una red neuronal es lento y requiere de muchos datos, no es recomendable para el objetivo de este proyecto. Ya que se quiere que el aprendizaje sea en tiempo real (online), en cada iteración se volvía a reentrenar desde el principio la red con el nuevo dato, haciéndolo un proceso muy pausado. Además, para usar una red neuronal se necesita tener muchos datos almacenados al inicio antes de usarla. Se probó a usar el clasificador LDA hasta un cierto número de ensayos y luego cambiar a realizar el entrenamiento de la red neuronal de manera online. Lo recomendable es entrenar la red de forma offline. Por otra parte, se podría intentar hacer una adaptación del algoritmo para que partiese de lo ya aprendido antes, sería una posible línea futura de este trabajo. Existen diferentes tipos de redes neuronales y dependen del número de neuronas que se use. Se podría comprobar que tal resulta con otras redes neuronales. Un ejemplo de red neuronal en MATLAB se puede consultar en el documento [227].

En este trabajo solo se ha usado el método CSP para la reducción de dimensionalidad y extracción de características, por ser el más simple y comúnmente usado en el BCI, sin embargo, se podría probar otros tipos de algoritmos, como los mencionados en el capítulo 4 u otras variantes del CSP. Por ejemplo, el método CSP adaptativo (ACSP) para analizar datos de EEG de ensayo único de sujetos únicos y múltiples. La señal del EEG puede exhibir una variación significativa entre sujetos e interindividuales, el ACSP está pensado para arreglar esto [228].

Otra posible ampliación, sería programar el algoritmo que convierte la señal identificada en un comando para una determinada aplicación. Por ejemplo, en el caso de un videojuego hacer que un coche se mueva de izquierda

a derecha y viceversa. Por otra parte, se podría aumentar la complejidad del sistema BCI para detectar de qué manera se quiere mover el brazo derecho o izquierdo, para así dar órdenes a los motores de un exoesqueleto de parte superior, los brazos. Para conseguir esto, sería más fácil con un BCI invasivo que con uno no invasivo.

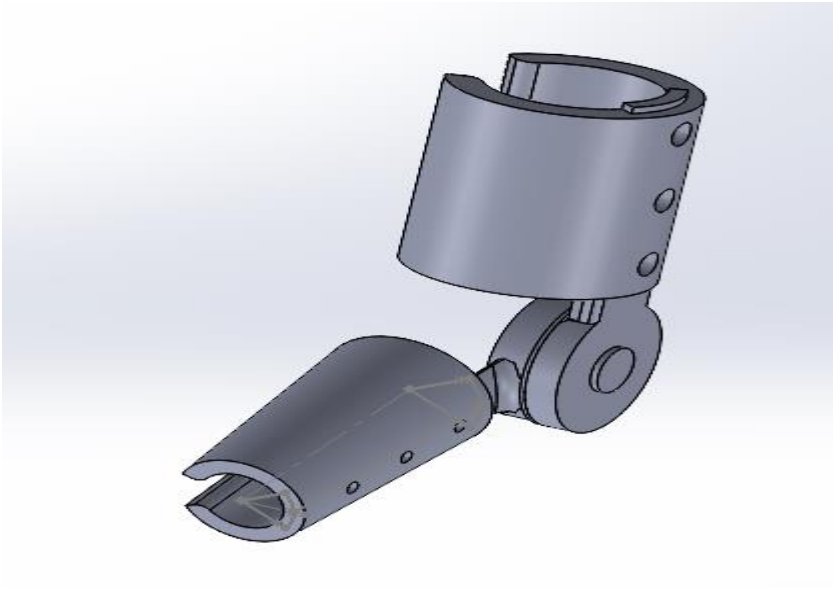


Figura 8-1. Ejemplo de prototipo de exoesqueleto de brazo, realizado con el programa SolidWorks.

Tal vez algún día los sistemas BCI puedan llegar a ser una manera de interactuar persona-máquina tan relevante como otras que existen en el presente.

REFERENCIAS

[1] I. Gil Montañés, «EL ROL DEL TERAPEUTA OCUPACIONAL EN EL USO DE LA INTERFAZ,» Universidad de Zaragoza , 2019.

[2] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Neuralink. [Último acceso: 2021].

[3] «wikipedia: Interfaz_cerebro-computadora,» [En línea]. Available: https://es.wikipedia.org/wiki/Interfaz_cerebro-computadora. [Último acceso: 2021].

[4] C. N. Henríquez Muñoz, «Estudio de Técnicas de análisis y clasificación de señales EEG en el contexto de Sistemas BCI (Brain Computer Interface),» Madrid, 2014.

[5] F. J. Olías Sánchez, «Estudio del método Common Spatial Patterns y sus variantes en interfaces cerebro-ordenador,» Sevilla, 2016.

[6] L. F. Nicolás Alonso, «Clasificación de características de electroencefalogramas en sistemas Brain Computer Interface basados en ritmos sensoriomotores».

[7] M. Á. López Gordo, «Interfaz BCI de altas prestaciones basada en la detección y procesamiento de la actividad cerebral (BCI-Depracap),» Granada, 2009.

[8] J. E. MUÑOZ CARDONA, «CLASIFICACION DE PATRONES DE IMAGINACIÓN MOTORA EN UNA INTERFAZ CEREBRO COMPUTADOR DE BAJO COSTO USANDO SOFTWARE LIBRE,» Pereira, 2014.

[9] C. Ferrin, H. Loaiza-Correa y J. &. V. A. P. Díaz-Paz, «Evaluación del aporte de la covarianza de las señales electroencefalográficas a las interfaces cerebro-computador de imaginación motora para pacientes con lesiones de médula espinal,» *TecnoLógicas*, vol. 22, nº 46, pp. 213-231, 2019.

[10] D. F. D'Croz Barón, «Reconocimiento de Imaginación Motora de Señales EEG en el Dominio Temporal aplicando Modelos Paramétricos,» Tonantzintla, Puebla, 2011.

[11] D. Fuentes Hitos, «Técnicas de Procesado de Señales Cerebrales,» Sevilla, 2015.

[12] Á. Morán García, «DISEÑO DE INTERFACES CEREBRO-MÁQUINA CONTROLADOS MEDIANTE REGISTROS DE EEG,» Madrid, 2015.

[13] M. E. RENGIFO BEDOYA y J. L. PORRAS GALINDO, «INTERFAZ CEREBRO COMPUTADOR BASADA EN POTENCIALES DE ESTADO ESTABLE EVOCADOS

VISUALMENTE PARA EL CONTROL DE MOVIMIENTO DE UN DISPOSITIVO VIRTUAL,» SANTIAGO DE CALI, 2016.

[14] C. Arboleda, E. García, A. Posada y R. Torres, «Diseño y construcción de un prototipo de interfaz cerebro-computador para facilitar la comunicación de personas con discapacidad motora,» *EIA,* nº 11, pp. 105-115, 2009.

[15] F. Parra, Estadística y Machine Learning con R, Universidad Nacional de Educación a Distancia, 2019.

[16] A. Pérez-Muelas Noguera, «Sistema de Software de Adquisición y Procesado de EEG,» Cartagena, 2017.

[17] V. Carmen, H. Stefan, J. Tania, M. Klaus-Robert y N. V. V., «Sensorimotor Functional Connectivity: A Neurophysiological Factor Related to BCI Performance,» *Frontiers in Neuroscience,* vol. 14, p. 1278, 2020.

[18] S. Lee, S. W. Younghak Shin, K. Kim y H.-N. Lee, «Review of Wireless Brain-Computer Interface Systems,» 2013.

[19] N. Tibrewal, N. Leeuwis y M. Alimardani, «The Promise of Deep Learning for BCIs: Classification of Motor Imagery EEG using Convolutional Neural Network,» *bioRxiv,* 2021.

[20] D. Rodríguez Cassolà, «Medición, procesado y clasificación de señales electroencefalográficas,» Sevilla, 2018.

[21] J. Gutiérrez-Martínez, J. Cantillo-Negrete, D. Elías-Viñas y R. I. Cariño-Escobar, «Los sistemas de interfaz cerebro-computadora:una herramienta para apoyar la rehabilitación de pacientes con discapacidad motora,» *Investigación en discapacidad,* vol. 2, nº 2, pp. 62-69, 2013.

[22] R. Leeb, D. Friedman, G. R. M. ¨uller-Putz, R. Scherer, M. Slater y G. Pfurtscheller, «Self-paced (Asynchronous) BCI control of a wheelchair in virtual environments: A case study with a tetraplegic. Comput. Intell,» 2007.

[23] L. Sailema, «lsailema.blogspot,» 2011. [En línea]. Available: http://lsailema.blogspot.com/2011/09/comunicacion-telepatica-brain-computer.html.

[24] «rehabilitacionblog,» [En línea]. Available: http://www.rehabilitacionblog.com/2011/03/bci-brain-computer-interfaces-en-ensayo.html. [Último acceso: 2021].

[25] K. C. Arana y A. V. Albán, «Prótesis de mano virtual movida por señales encefalograficas – EEG,» *Prospect,* vol. 14, nº 2, pp. 99-110, 2016.

[26] «gtec,» [En línea]. Available: https://www.gtec.at/product/g-pangolin/. [Último acceso: 2021].

[27] «spectrum.ieee.org,» [En línea]. Available: https://spectrum.ieee.org/the-human-os/biomedical/bionics/the-tech-behind-a-mind-reading-dress-could-lead-to-wireless-batteryless-exoskeleton-control. [Último acceso: 2021].

[28] «fgcsic.es,» [En línea]. Available: http://www.fgcsic.es/lychnos/es_ES/articulos/Brain-Computer-Interface-aplicado-al-entrenamiento-cognitivo. [Último acceso: 2021].

[29] A. F. Castro Gutiérrez y J. F. Ortega Hernández, «Implementación de una interfaz cerebro-computador (BCI) que permita controlar el movimiento de elevación y avance durante el vuelo de un cuadricóptero (dron).,» Santiago de Cali, Colombia, 2017.

[30] J. A. V. Dávila, J. V. Macías y M. V. Lamas, «Videojuegos basados en BCI (Interface cerebro computadora): Revisión Sistemática Literaria,» 2017.

[31] A. Curtin, H. Ayaz, Y. Liu, P. Shewokis y B. Onaral, « A P300-based EEG-BCI for spatial navigation control,» de *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2012.

[32] «gtec,» [En línea]. Available: https://www.gtec.at/product/webinars/.

[33] F. J. Olías Sánchez, «Eeg signal processing in motor imagery brain computer interfaces with improved covariance,» Sevilla, 2020.

[34] «pinimg,» [En línea]. Available: https://i.pinimg.com/originals/a8/a4/94/a8a49484b985a3dd1689877b1e0b79ad.jpg. [Último acceso: 2021].

[35] «metodotodoesposible,» [En línea]. Available: https://www.metodotodoesposible.es/uncategorized/las-frecuencias-cerebrales-que-son-y-por-que-necesitas-conocerlas/. [Último acceso: 2021].

[36] «psicologiayempresa,» [En línea]. Available: https://psicologiayempresa.com/fisica-cerebral-senales-electricas-del-cerebro.html. [Último acceso: 2021].

[37] «um,» [En línea]. Available: https://www.um.es/lafem/DivulgacionCientifica/CienciaySalud/Portalyblog/cienciaysalud.laverdad.es/8_3_3.html. [Último acceso: 2021].

[38] D. García Ramos, «TFG EEG signal classification for MI-BCI,» Grado en Ingeniería de las Tecnologías de Telecomunicación (GITT) Universidad de Sevilla, Sevilla, 2020.

[39] «youtube,» [En línea]. Available: https://i.ytimg.com/vi/drCJVIKgvqs/maxresdefault.jpg. [Último acceso: 2021].

[40] «mayoclinic.org,» [En línea]. Available: https://www.mayoclinic.org/es-es/tests-procedures/eeg/about/pac-20393875. [Último acceso: 7 Julio 2021].

[41] C. Toxtli Hernández, «interfaz-cerebro-maquina,» [En línea]. Available: https://es.slideshare.net/carlostoxtli/interfaz-cerebro-maquina. [Último acceso: 2021].

[42] «Google imágenes,» [En línea]. Available: . [Último acceso: 2021].

[43] C. Escolano, I. Iturrate, J. Antelis y J. Minguez, «Dispositivos robóticos de rehabilitación basados en Interfaces CerebroOrdenador: silla de ruedas y robot para teleoperación,» Zaragoza.

[44] S. René Vivar Vera y M. A. Abud Figueroa, «Desarrollo de una BCI utilizando el potencial».

[45] V. Peterson, Y. Atum, F. Jauregui y I. Gareis, «Detección de potenciales evocados relacionados,» *Ingeniería Biomédica*, vol. 7, nº 14, pp. 50-58, 2013.

[46] J. Fernandez-Vargas, H. U. Pfaff, F. B. Rodríguez y P. Varona, «Assisted closed-loop optimization of SSVEP-BCI efficiency,» Madrid, Spain, 2013.

[47] S. Fernandez-Fraga, M. Aceves-Fernandez y J. Rodríguez-Resendíz, «Steady-state visual evoked potential (SSEVP) from EEG signal modeling based upon recurrence plots,» de *Evolving Systems*, vol. 10, Springer, 2019, p. 97–109.

[48] S. H. Patel y P. N. Azzam, «Characterization of N200 and P300: Selected Studies of the Event-Related Potential,» vol. 2, nº 4, p. 147–154, 2005.

[49] R. Mendes Duarte, «Low cost Brain Computer Interface system for AR.Drone Control,» 2017.

[50] F. Cuenca Martínez, «neurorehabnews,» 13 Febrero 2019. [En línea]. Available: https://neurorehabnews.com/aprendizaje-motor/capacidad-de-generar-imagenes-mentales-motoras-y-su-relacion-con-los-niveles-de-actividad-fisica.html.

[51] Y. Jeon, C. S. Nam, Y.-J. Ki y M. C. Whang, «Event-related (De)synchronization (ERD/ERS) during motor imagery tasks:Implications for brainecomputer interfaces,» *International Journal of Industrial Ergonomics*, vol. 41, nº 5, pp. 428-436, 2011.

[52] E.Monge-Pereira, F.Molina-Rueda, F.M.Rivas-Montero, J.Ibáñezc, J.I.Serranod, I.M.Alguacil-Diegoa y J.C.Miangolarra-Page, «Electroencefalografía como método de evaluación tras un ictus. Una revisión actualizada,» *NEUROLOGÍA*, vol. 32, nº 1, pp. 40-49, 2017.

[53] «brainloop,» [En línea]. Available: https://aksioma.org/brainloop/m_imagery_dynamics.html. [Último acceso: 2021].

[54] B. M. Villegas Méndez y M. G. Rojas Fernández, «Interfaz cerebro ordenador BCI mediante el uso de Emotiv Insight,» *RevActaNova*, vol. 9, nº 1, 2019.

[55] A. Ahangi, M. Karamnejad, N. Mohammadi, R. Ebrahimpour y N. Bagheri, «Multiple classifier system for EEG signal classification with application to brain–computer interfaces,» *Neural Computing and Applications*, 2012.

[56] P. Wierzgała, D. Zapała, G. Wójcik y M. Jolanta, «Most Popular Signal Processing Methods in Motor-Imagery BCI: A Review and Meta-Analysis,» *Frontiers in Neuroinformatics*, p. 78, 2018.

[57] I. Gareis, G. Gentiletti, R. Acevedo y L. Rufiner, «Feature Extraction on Brain Computer Interfaces using Discrete Dyadic Wavelet Transform: Preliminary Results,» *Journal of Physics Conference Series*, 2011.

[58] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Interfaz_cerebro-computadora. [Último acceso: 2021].

[59] B. García, «blogthinkbig,» 2018. [En línea]. Available: https://blogthinkbig.com/el-futuro-de-la-tecnologia-brain-computer. [Último acceso: 2021].

[60] S. Cruces, «Motor Imagery Brain Computer Interfaces».

[61] R. H. Sánchez, «Brain Computer Interface for cognitive training and domotic assistance against the effects of ageing (BCIAgeing),» 2013.

[62] M.-H. Lee, S.-W. Lee, S. Fazli, J. Williamson, Y.-E. Lee y H.-K. Kim, «EEG Dataset and OpenBMI Toolbox for Three BCIParadigms: An Investigation into BCI Illiteracy,» Qabanbay Batyr Ave 53, Astana 010000, Kazakhstan & 145 Anam-ro, Seongbuk-gu,Seoul, 02841, Korea, 2019.

[63] D. Maison y T. Oleksy, «Validation of EEG as an Advertising Research Method: Relation Between EEG Reaction Toward Advertising and Attitude Toward Advertised Issue (Related to Political and Ideological Beliefs),» de *Neuroeconomic and Behavioral Aspects of Decision Making*, 2017.

[64] «gtec,» [En línea]. Available: https://www.gtec.at/shop/. [Último acceso: 2021].

[65] C. L. Marcos Rojas, J. D. Chailloux Peguero y E. Alba Blanco, «Real time identification of motor imagery actions on EEG signals.,» *EAC*, vol. 41, nº 1, 2020.

[66] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada. [Último acceso: 2021].

[67] C. Brunner, R. Leeb, G. R. Müller-Putz, A. Schlög y G.Pfurtscheller, «Bci competition 2008 – graz data set a,» 2008.

[68] G. Pfurtscheiler, C. Guger y H. Ramoser , «EEG-based brain-computer interface using subject-specific spatial filters,» de *Engineering Applications of Bio-Inspired Artificial Neural Networks*, 2006, pp. 248-254.

[69] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Electrooculograma. [Último acceso: 2021].

[70] O. J. O. Murillo, G. R. Fuentes y F. J. López, «Diseño e Implementación de un Sistema de Control de Movimientos para una plataforma Móvil usando ElectroOculografía,» Tunja, Colombia.

[71] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Artefacto_(error_de_observaci%C3%B3n). [Último acceso: 2021].

[72] R. A. Alagia Gimeno, «Procesamiento de artefactos en EEG para aplicaciones,» Valencia, 2018.

[73] P. Górski, «Common Spatial Patterns in a few channel BCI interface,» *Computer Science,* vol. 8, pp. 56-63, 2014.

[74] «mathworks,» [En línea]. Available: https://es.mathworks.com/products/matlab.html. [Último acceso: 2021].

[75] «mathworks,» [En línea]. Available: https://es.mathworks.com/products/connections/product_detail/g-needaccess.html. [Último acceso: 2021].

[76] L. Korczowski, «louis-korczowski,» 14 Marzo 2018. [En línea]. Available: https://louis-korczowski.org/tribute-to-stephen-hawking. [Último acceso: 9 Julio 2021].

[77] S. Cruces, J. Olias, R. Martín-Clemente y M. A. Sarmiento-Vega, «Eeg signal processing in mi-bci applications with improved covariance matrix estimators,» *IEEE Transactions on Neural Systems and,* nº 5, p. 895–904, 2019.

[78] «mathworks,» [En línea]. Available: https://matlabacademy.mathworks.com/R2020b/es/portal.html?course=machinelearning. [Último acceso: 2021].

[79] B. MEDINA, J. E. SIERRA y A. Barrios ULLOA, «Técnicas de extracción de características de señales EEG en la imaginación de movimiento para sistemas BCI,» *ESPACIOS,* vol. 39, nº 22, p. 36, 2018.

[80] F. M. RAMÍREZ DIEZ, «USO DE CARACTERÍSTICAS ESPECTRALES Y TEMPORALES PARA,» Santiago, Chile, 2017.

[81] B. Zhou, Z. L. Xiaopei Wu, L. Zhang y X. Guo, «A Fully Automated Trial Selection Method for Optimization of Motor Imagery Based Brain-Computer Interface,» *plosone,* 2016.

[82] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Transformada_r%C3%A1pida_de_Fourier. [Último acceso: Marzo 2021].

[83] N. C. Swati Aggarwal, «Signal processing techniques for motor imagery brain computer interface: A review,» *Array,* pp. 3-6, 2019.

[84] . J. Lin y C. Hsieh , «A Wireless BCI-Controlled Integration System in Smart Living Space for Patients. Wireless Pers Commun,» vol. 88, pp. 395-412, 2016.

[85] D. J. Krusienski, . D. J. McFarland y . J. R. Wolpaw, «An Evaluation of Autoregressive Spectral Estimation Model Order for Brain-Computer Interface Applications,» de *International Conference of the IEEE Engineering in Medicine and Biology Society*, New York City, USA, 2006.

[86] H. Kyu Lee y Y.-S. Choi, «Application of Continuous Wavelet Transform and Convolutional Neural Network in Decoding Motor Imagery Brain-Computer Interface,» *Entropy*, vol. 21, nº 12, 2019.

[87] V. Peterson, Y. Atum, F. Jauregui y I. Gareis, «Detección de potenciales evocados relacionados a eventos en interfaces cerebro-computadora mediante transformada wavelet,» *Revista Ingeniería Biomédica, Escuela de Ingeniería de Antioquia-Universidad CES*, vol. 7, nº 14, pp. 51-59, 2013.

[88] «Wikipedia:Análisis de componentes principales,» [En línea]. Available: https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales#:~:text=En%20estad%C3%ADstica%2C%20el%20an%C3%A1lisis%20de,%C2%ABcomponentes%C2%BB)%20no%20correlacionadas.. [Último acceso: 2021].

[89] D. Barber, Bayesian Reasoning and Machine Learning, DRAFT, 2015.

[90] «yarpiz,» [En línea]. Available: https://yarpiz.com/category/machine-learning. [Último acceso: 2021].

[91] A. Barachant, S. Bonnet, M. Congedo y C. Jutten, «Common Spatial Pattern revisited by Riemannian Geometry,» *IEEE*, pp. 472-476, 2010.

[92] «wikipedia: Geometría de Riemann,» [En línea]. Available: https://es.wikipedia.org/wiki/Geometr%C3%ADa_de_Riemann. [Último acceso: 2021].

[93] M. Congedo, A. Barachant y R. Bhatia, «Riemannian geometry for EEG-based brain-computer interfaces; a primer and a review,» *Brain-Computer Interfaces*, vol. 4, pp. 1-20, 2017.

[94] A. S. M. Miah, M. A. Rahim y J. Shin, «Motor-Imagery Classification Using Riemannian,» *electronics*, 2020.

[95] M. Joadder, S. Siuly, E. Kabir, H. Wang y Y. Zhang, «A New Design of Mental State Classification for Subject Independent BCI Systems,» *IRBM*, vol. 40, nº 5, pp. 297-305, 2019.

[96] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Correlaci%C3%B3n#:~:text=En%20probabilidad%20y%20estad

%C3%ADstica%2C%20la,proporcionalidad%20entre%20dos%20variables%20estad%C3%ADsticas.. [Último acceso: 2021].

[97] «wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Covariance_matrix. [Último acceso: 2021].

[98] «wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Estimation_of_covariance_matrices. [Último acceso: 2021].

[99] S. Cruces, «EEG PROCESSING,» Sevilla.

[100] «wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Rayleigh_quotient. [Último acceso: 2021].

[101] «mathworks: eig,» [En línea]. Available: https://es.mathworks.com/help/matlab/ref/eig.html. [Último acceso: 2021].

[102] «wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Generalized_eigenvector. [Último acceso: 2021].

[103] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Vector_propio_y_valor_propio. [Último acceso: 2021].

[104] Departamento de Matemática Aplicada II, «GIA_Sevilla_ESI,» Universidad de Sevilla, 2013. [En línea]. Available: https://neblan.files.wordpress.com/2012/10/tema-5.pdf. [Último acceso: 2021].

[105] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Funci%C3%B3n_mon%C3%B3tona. [Último acceso: 2021].

[106] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Logaritmo. [Último acceso: 2021].

[107] «wikipedia: Common_spatial_pattern,» [En línea]. Available: https://en.wikipedia.org/wiki/Common_spatial_pattern. [Último acceso: 2021].

[108] H. Cho, M. Ahn, K. Kim y S. Jun, «Increasing session-to-session transfer in a brain–computer interface with on-site background noise acquisition,» *Journal of Neural Engineering*, vol. 12, 2015.

[109] B. Blankertz, T. Ryota , S. Lemm, M. Kawanabe y K.-R. Müller, «Optimizing Spatial Filters for Robust EEG Single-Trial Analysis,» *IEEE SIGNAL PROCESSING MAGAZINE*, pp. 41-56, 2008.

[110] F. Lotte y C. Guan, «Regularizing Common Spatial Patterns to Improve BCI Designs: Theory and Algorithms,» 2010.

[111] K. Ang, Z. Chin, H. Zhang y C. Guan, «Filter Bank Common Spatial Pattern (FBCSP) in brain-computer interface,» de *IEEE International Joint Conference on Neural Networks*, Hong Kong, 2008.

[112] F. Lotte, «A Tutorial on EEG Signal Processing Techniques forMental State Recognition in Brain-Computer Interfaces,» de *Guide to Brain-ComputerMusic Interfacing*, E. R. Miranda y J. Caste, Edits., Springer, 2014.

[113] A. K. Keng, C. Z. Yang, W. Chuanchu, G. Cuntai y Z. Haihong, «Filter bank common spatial pattern algorithm on BCI competition IV Datasets 2a and 2b,» *Frontiers in Neuroscience,* vol. 6, p. 39, 2012.

[114] S. Sánchez Reinoso, «Identificación de objetos en tiempo real utilizando tecnicas y clasificadores de visión artificial para el reconocimiento de patrones,» Ecuador, 2011.

[115] F. Sancho Caparrini, «Aprendizaje Supervisado y No Supervisado,» 14 Diciembre 2020. [En línea]. Available: http://www.cs.us.es/~fsancho/?e=77.

[116] M. Döring, «datascienceblog,» [En línea]. Available: https://www.datascienceblog.net/post/machine-learning/linear-discriminant-analysis/. [Último acceso: 2021].

[117] «programmerclick.com,» [En línea]. Available: https://programmerclick.com/article/5388728976/. [Último acceso: 4 Julio 2021].

[118] L. F. Nicolas-Alonso, «Brain computer interfaces, a review.,» *Sensors,* vol. 12, nº 2, pp. 1211-1279, 2012.

[119] «geeksforgeeks,» 6 Mayo 2019. [En línea]. Available: https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/. [Último acceso: 7 Julio 2021].

[120] S. Raschka, «sebastianraschka,» 3 Agosto 2014. [En línea]. Available: https://sebastianraschka.com/Articles/2014_python_lda.html. [Último acceso: 7 Julio 2021].

[121] J. A. Ordaz Sanz, M. d. C. Melgar Hiraldo y C. M. Rubio Castaño, «MÉTODOS ESTADÍSTICOS Y ECONOMÉTRICOS EN LA EMPRESA Y PARA FINANZAS».

[122] «minitab,» [En línea]. Available: https://support.minitab.com/es-mx/minitab/18/help-and-how-to/modeling-statistics/multivariate/supporting-topics/discriminant-analysis/what-is-linear-discriminant-analysis/. [Último acceso: 2021].

[123] «github,» [En línea]. Available: http://uc-r.github.io/discriminant_analysis. [Último acceso: 2021].

[124] P. Gaur, R. B. Pachori, H. Wang y G. Prasad, «An empirical mode decomposition based filtering method for classification of motor-imagery EEG signals for enhancing brain-computer interface,» de *int Conference on Neural Networks (IJCNN)*, 2015.

[125] «Tema 6: Análisis Discriminante,» [En línea]. Available: http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/AMult/tema6am.pdf.

[126] J. Amat Rodrigo, «cienciadedatos,» 2016. [En línea]. Available: https://www.cienciadedatos.net/documentos/29_comparacion_regresion_logistica_linear_discriminant_analisis__lda_knn.

[127] «scikit-learn.org,» [En línea]. Available: https://scikit-learn.org/stable/modules/lda_qda.html. [Último acceso: 2021].

[128] J. M. Marín Diazaraque, «Tema 1: Análisis Discriminante,» [En línea]. Available: http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/tema1dm.pdf. [Último acceso: 2021].

[129] «Quadratic Discriminant Analysis,» [En línea]. Available: https://online.stat.psu.edu/stat508/book/export/html/696.

[130] O. Trigui, M. b. messaoud y W. Zouch, «Brain-computer interface: Frequency domain approach using the linear and the quadratic discriminant analysis,» de *Advanced Technologies for Signal and Image Processing (ATSIP), 2014 1st International Conference on*, Sousse, Tunisia, 2014.

[131] «wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Quadratic_classifier. [Último acceso: 2021].

[132] Y. Guo, T. Hastie y R. Tibshirani, «Regularized linear discriminant analysis and its application in microarrays,» *Biostatistics*, vol. 8, nº 1, pp. 86-100, 2006.

[133] «rapidminer,» [En línea]. Available: https://docs.rapidminer.com/latest/studio/operators/modeling/predictive/discriminant_analysis/quadratic_discriminant_analysis.html. [Último acceso: 2021].

[134] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/classificationdiscriminant.cvshrink.html#bs8_n55. [Último acceso: 20 Mayo 2021].

[135] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/regularize-discriminant-analysis-classifier.html. [Último acceso: 2021].

[136] S. Van Vaerenbergh y I. Santamaría, «Métodos kernel para clasificación,» 2018.

[137] J. Amat Rodrigo, «cienciadedatos.net,» Abril 2017. [En línea]. Available: https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines#Introducci%C3%B3n. [Último acceso: 2021].

[138] T. N. Do, «Automatic Learning Algorithms for Local Support Vector Machines,» *SN Computer Science*, vol. 1, nº 2, 2020.

[139] «coursera,» [En línea]. Available: https://www.coursera.org/learn/deteccion-objetos/lecture/JXyER/l4-6-support-vector-machines-svm-desarrollo-matematico. [Último acceso: 2021].

[140] L. A. Moctezuma Pascual, «Distinción de estados de actividad e inactividad lingüística para interfaces cerebro computadora,» Puebla, 2017.

[141] «Teorema de Cover,» [En línea]. Available: https://en.wikipedia.org/wiki/Cover%27s_theorem. [Último acceso: 2021].

[142] «coursera.org,» Universitat Autònoma de Barcelona, [En línea]. Available: https://es.coursera.org/lecture/deteccion-objetos/l4-6-support-vector-machines-svm-desarrollo-matematico-JXyER. [Último acceso: Julio 2021].

[143] R. M. Villanueva, «DETECCIÓN DE OBJETOS POR COMPUTADOR,» [En línea]. Available: https://rdu.iua.edu.ar/bitstream/123456789/886/1/Versi%C3%B3n%2032%20del%20TFG%20Rodolfo%20Villanueva%20Detecci%C3%B3n%20de%20objetos%20por%20computador.pdf.

[144] «Códigos de salida de corrección de errores (ECOC) para el aprendizaje automático,» [En línea]. Available: https://topbigdata.es/codigos-de-salida-de-correccion-de-errores-ecoc-para-el-aprendizaje-automatico/. [Último acceso: 2021].

[145] «coursera,» [En línea]. Available: https://coursera.org/share/c744858c81f2b2d1baf1055fe239a278. [Último acceso: 2021].

[146] «mathworks,» [En línea]. Available: https://matlabacademy.mathworks.com/es. [Último acceso: 2021].

[147] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/choose-a-classifier.html. [Último acceso: 2021].

[148] «ichi.pro,» [En línea]. Available: https://ichi.pro/es/13-algoritmos-de-clasificacion-de-aprendizaje-automatico-para-ciencia-de-datos-y-su-codigo-247965255771399. [Último acceso: 2021].

[149] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Clasificador_bayesiano_ingenuo. [Último acceso: 2021].

[150] T. Joseph, «The Monty Hall Problem: Naive Bayes explained!,» 3 Julio 2020. [En línea]. Available: https://towardsdatascience.com/the-monty-hall-problem-naive-bayes-explained-fadd991edeb2. [Último acceso: 2021].

[151] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Distribuci%C3%B3n_normal. [Último acceso: 2021].

[152] W. Gómez Flores, «Análisis de Datos: Clasificación Bayesiana».

[153] «codingninjas,» 4 Septiembre 2020. [En línea]. Available: https://www.codingninjas.com/blog/2020/09/04/classification-decision-trees-naive-bayes-gaussian-bayes-classifier/. [Último acceso: 2021].

[154] M. Vargas, «Reconocimiento de formas,» Sevilla, 2019.

[155] «wikipedia: Distancia_de_Mahalanobis,» [En línea]. Available: https://es.wikipedia.org/wiki/Distancia_de_Mahalanobis. [Último acceso: 2021].

[156] E. Quirós Rosado y A. M. Felicisimo, «Clasificación de imágenes multiespectrales ASTER mediante funciones adaptativas,» 2009.

[157] S. Lemm, C. Schafer y G. Curio, «BCI competition 2003-data set III: probabilistic modeling of sensorimotor /spl mu/ rhythms for classification of imaginary hand movements,» *IEEE Transactions on Biomedical Engineering*, vol. 51, nº 6, pp. 1077-1080, Junio 2004.

[158] D. Cavouras, K. Koutroumbas, A. Pikrakis y S. Theodoridis, «Classifiers Based on Bayes,» de *Introduction to Pattern Recognition*, 1 ed., Academic Press, 2010.

[159] K. Koutroumbas y S. Theodoridis, Pattern Recognition, Academic Press, 2008.

[160] «aprendemachinelearning.com,» 2018. [En línea]. Available: https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/#:~:text=K%2DNearest%2DNeighbor%20es%20un,el%20mundo%20del%20Aprendizaje%20Autom%C3%A1tico.. [Último acceso: 2021].

[161] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/K_vecinos_m%C3%A1s_pr%C3%B3ximos. [Último acceso: 2021].

[162] «aprendeia.com,» [En línea]. Available: https://aprendeia.com/k-vecinos-mas-cercanos-teoria-machine-learning/. [Último acceso: 2021].

[163] N. E'zzati Md Isa, A. Amir, M. Zaizu Ilyas y M. Shahrazel Razalli, «The Performance Analysis of K-Nearest Neighbors (K-NN) Algorithm for Motor Imagery Classification Based on EEG Signal,» de *MATEC Web of Conferences*, Perlis , 2017.

[164] «aprendemachinelearning,» 2018. [En línea]. Available: https://www.aprendemachinelearning.com/k-means-en-python-paso-a-paso/.

[165] «aprendemachinelearning,» 2018. [En línea]. Available: https://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/. [Último acceso: 2021].

[166] J. Amat Rodrigo, «cienciadedatos.net,» octubre 2020. [En línea]. Available: https://www.cienciadedatos.net/documentos/33_arboles_decision_random_forest_gradient_boosting_c50.

[167] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_basado_en_%C3%A1rboles_de_decisi%C3%B3n. [Último acceso: 2021].

[168] «sitiobigdata,» 2019. [En línea]. Available: https://sitiobigdata.com/2019/12/14/arbol-de-decision-en-machine-learning-parte-1/#.

[169] J. Amat Rodrigo, «cienciadedatos,» Octubre 2020. [En línea]. Available: https://www.cienciadedatos.net/documentos/py07_arboles_decision_python.html.

[170] «aprendeia.com,» [En línea]. Available: https://aprendeia.com/arboles-de-decision-clasificacion-teoria-machine-learning/.

[171] R. Ferrero, J. L. Lopez y P. Merayo, «maximaformacion,» [En línea]. Available: https://www.maximaformacion.es/blog-dat/que-son-los-arboles-de-decision-y-para-que-sirven/.

[172] P.-N. Tan, M. Steinbach, A. Karpatne y V. Kumar, «Basic concepts, decision trees and model evaluation,» de *Introduction to Data Mining*, 2005.

[173] D. ForBusiness, «Algoritmos Machine Learning: Arboles Decision para Data Science,» 23 Mayo 2020. [En línea]. Available: https://www.youtube.com/watch?v=LZkIfA5kgl0. [Último acceso: 4 Julio 2021].

[174] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_basado_en_%C3%A1rboles_de_decisi%C3%B3n#Impureza_de_Gini. [Último acceso: 2021].

[175] «Aprendizaje Supervisado: Decision Tree Classification,» [En línea]. Available: https://aprendeia.com/aprendizaje-supervisado-decision-tree-classification/. [Último acceso: 2021].

[176] «bookdown,» [En línea]. Available: https://bookdown.org/content/2031/arboles-de-decision-parte-i.html. [Último acceso: 2021].

[177] J. Martinez Heras, «iartificial,» 19 Septiembre 2020. [En línea]. Available: https://www.iartificial.net/arboles-de-decision-con-ejemplos-en-python/.

[178] dataEvo, «Machine Learning: del arbol de decisión al bosque de árboles con boosting (XGBoost),» 24 Febrero 2017. [En línea]. Available: https://www.youtube.com/watch?v=PN-8dTXh3DU.

[179] J. J. Espinosa-Zúñiga, «Aplicación de algoritmos Random Forest y XGBoost en una base de solicitudes de tarjetas de crédito,» *Ingeniería, investigación y tecnología*, vol. 21, nº 3, 2020.

[180] M. Á. Lopez Gordo, «Interfaz bci de altas prestaciones en deteccion y procesamiento de la actividad cerebral,» Universidad de Granada, Granada, 2009.

[181] J. Martinez Heras, «iartificial,» 21 Septiembre 2020. [En línea]. Available: https://www.iartificial.net/regresion-logistica-para-clasificacion/. [Último acceso: 2021].

[182] «aprendeia,» 28 Junio 2019. [En línea]. Available: https://aprendeia.com/regresion-logistica-multiple-machine-learning-teoria/. [Último acceso: 2021].

[183] J. Brownlee, «machinelearningmastery,» 1 Abril 2016. [En línea]. Available: https://machinelearningmastery.com/logistic-regression-for-machine-learning/.

[184] «datahacker,» [En línea]. Available: http://datahacker.rs/005-pytorch-logistic-regression-in-pytorch/. [Último acceso: 9 Julio 2021].

[185] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Regresi%C3%B3n_log%C3%ADstica. [Último acceso: 2021].

[186] N. E. M. Isa, A. Amir, M. Z. Ilyas y M. S. Razalli, «Motor imagery classification in Brain computer interface (BCI) based on EEG signal by using machine learning technique,» *Bulletin of Electrical Engineering and Informatics,* vol. 8, nº 1, pp. 269-275, Marzo 2019.

[187] J. Martinez Heras, «iartificial,» 20 Septiembre 2020. [En línea]. Available: https://www.iartificial.net/gradiente-descendiente-para-aprendizaje-automatico/.

[188] CII_IA, «Entropía cruzada,» 7 Marzo 2020. [En línea]. Available: https://www.youtube.com/watch?v=sedrzwgC-1E. [Último acceso: 2021].

[189] «coursera,» [En línea]. Available: https://www.coursera.org/learn/deteccion-objetos/lecture/TCacZ/l2-4-a-regresion-logistica-aprendizaje-i. [Último acceso: 2021].

[190] «coursera,» [En línea]. Available: https://www.coursera.org/learn/deteccion-objetos/lecture/b68uj/l2-4-b-regresion-logistica-aprendizaje-ii. [Último acceso: 2021].

[191] Datapolítica, «REGRESIÓN LOGÍSTICA BINARIA (1): Teoría [FÁCIL],» 28 Abril 2021. [En línea]. Available: https://www.youtube.com/watch?v=qGpn_Ycw6T8. [Último acceso: 5 Julio 2021].

[192] S. Dreiseitl y L. Ohno-Machado, «Logistic regression and artificial neural network classification models: a methodology review,» *Journal of Biomedical Informatics,* vol. 35, nº 5-6, pp. 352-359, 2002.

[193] E. C. Gracía, «Introducción a la Regresion Logistica,» 27 Julio 2017. [En línea]. Available: https://www.youtube.com/watch?v=HFswrM68yPU. [Último acceso: 5 julio 2021].

[194] CII_IA, «Regresión Logística,» 7 Marzo 2020. [En línea]. Available: https://www.youtube.com/watch?v=SeM4Rtoa4EU. [Último acceso: 5 Julio 2021].

[195] I. R. R. Castro, «Regresión logística (Aspectos teóricos),» 25 Septiembre 2014. [En línea]. Available: https://www.youtube.com/watch?v=Ry4_P1TrtYI. [Último acceso: 5 Julio 2021].

[196] A. M. Lopez, «coursera,» Universitat Autònoma de Barcelona, [En línea]. Available: https://es.coursera.org/lecture/deteccion-objetos/l2-4-b-regresion-logistica-aprendizaje-ii-b68uj. [Último acceso: 5 Julio 2021].

[197] «coursera,» [En línea]. Available: https://coursera.org/share/e7e564399c913ca52308d3d22d039610. [Último acceso: 2021].

[198] J. Martinez Heras, «iartificial,» 19 Septiembre 2020. [En línea]. Available: https://www.iartificial.net/regularizacion-lasso-l1-ridge-l2-y-elasticnet/.

[199] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/choose-a-classifier.html#bunt0rb-1. [Último acceso: 11 Mayo 2021].

[200] V. Smolyakov, «Ensemble Learning to Improve Machine Learning Results,» 22 Agosto 2017. [En línea]. Available: https://blog.statsbot.co/ensemble-learning-d1dcd548e936. [Último acceso: 28 Julio 2021].

[201] T. G. Dietterich, «Ensemble Methods in Machine Learning,» de *Lecture Notes in Computer Science*, vol. 1857, Oregon State University, Corvallis, Oregon, USA, Springer, Berlin, Heidelberg, 2000.

[202] E. Lutins, «towardsdatascience.com,» 2 Agosto 2017. [En línea]. Available: https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f. [Último acceso: Julio 2021].

[203] S. Paul, «Ensemble Learning in Python,» 6 Septiembre 2018. [En línea]. Available: https://www.datacamp.com/community/tutorials/ensemble-learning-python. [Último acceso: 28 Julio 2021].

[204] «wikipedia: Ensemble_learning,» [En línea]. Available: https://en.wikipedia.org/wiki/Ensemble_learning. [Último acceso: 2021].

[205] «geeksforgeeks: ensemble-classifier-data-mining,» 30 Mayo 2019. [En línea]. Available: https://www.geeksforgeeks.org/ensemble-classifier-data-mining/.

[206] N. Demir, «toptal.com: Ensemble Methods: Elegant Techniques to Produce Improved Machine Learning Results,» [En línea]. Available: https://www.toptal.com/machine-learning/ensemble-methods-machine-learning. [Último acceso: 2021].

[207] A. SINGH, «analyticsvidhya.com: A Comprehensive Guide to Ensemble Learning (with Python codes),» 18 Junio 2018. [En línea]. Available: https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/. [Último acceso: 28 Julio 2021].

[208] «mathworks: classification-ensembles,» [En línea]. Available: https://es.mathworks.com/help/stats/classification-ensembles.html?s_tid=CRUX_lftnav. [Último acceso: 2021].

[209] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/feature-selection-and-feature-transformation.html. [Último acceso: 2021].

[210] «mathworks: classification-learner-app,» [En línea]. Available: https://es.mathworks.com/help/stats/classification-learner-app.html?s_tid=CRUX_lftnav. [Último acceso: 2021].

[211] «mathworks: classificationlearner-app,» [En línea]. Available: https://www.mathworks.com/help/stats/classificationlearner-app.html. [Último acceso: 2021].

[212] K. Fukumizu, L. Song y A. Gretton, *Kernel Bayes' rule,* Cornell University, 2018.

[213] «RapidMiner,» [En línea]. Available: https://docs.rapidminer.com/latest/studio/operators/modeling/predictive/bayesian/naive_bayes_kernel.html. [Último acceso: 2021].

[214] F. Raulf, «datasciencecentral,» 3 Enero 2020. [En línea]. Available: https://www.datasciencecentral.com/profiles/blogs/naiv-bayes-classifier-using-kernel-density-estimation-with.

[215] «mathworks: train-decision-trees-in-classification-learner-app,» [En línea]. Available: https://es.mathworks.com/help/stats/train-decision-trees-in-classification-learner-app.html. [Último acceso: 2021].

[216] «mathworks: choose-a-classifier,» [En línea]. Available: https://es.mathworks.com/help/stats/choose-a-classifier.html?searchHighlight=fine%20tree&s_tid=srchtitle#bunt0ky. [Último acceso: 2021].

[217] «docplayer.es,» [En línea]. Available: https://docplayer.es/51900928-1-1-introduccion-metodo-de-identificacion-por-minimos-cuadrados-forma-recursiva-inclusion-del-factor-de-olvido.html. [Último acceso: 2021].

[218] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/fitcdiscr.html?searchHighlight=fitcdiscr&s_tid=srchtitle. [Último acceso: 2021].

[219] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/classificationnaivebayes-class.html?searchHighlight=fitcnb&s_tid=srchtitle. [Último acceso: 2021].

[220] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/fitcnb.html?searchHighlight=fitcnb&s_tid=srchtitle. [Último acceso: 2021].

[221] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/fitcknn.html?searchHighlight=fitcknn&s_tid=srchtitle. [Último acceso: 2021].

[222] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/fitctree.html. [Último acceso: 2021].

[223] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/classificationtree.prune.html. [Último acceso: 2021].

[224] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/fitclinear.html. [Último acceso: 2021].

[225] «ecured,» [En línea]. Available: https://www.ecured.cu/Funci%C3%B3n_Kernel. [Último acceso: 2021].

[226] «mathworks,» [En línea]. Available: https://es.mathworks.com/help/stats/fitcsvm.html?searchHighlight=fitcsvm&s_tid=srchtitle. [Último acceso: 2021].

[227] N. Peirotén López de Arbina, «Diseño de una red neuronal en Matlab para análisis de señales de electroencefalograma,» Sevilla, 2018.

[228] X. Song y S. Yoon, «Improving brain-computer interface classification using adaptive common spatial patterns,» *Comput Biol Med*, vol. 61, pp. 150-60, 2015.

[229] R. Martín-Clemente, J. Olias, . M. A. Sarmiento-Vega y S. Cruces, «EEG Signal Processing in MI-BCI Applications,» *IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING*, vol. 27, nº 5, pp. 895-904, Mayo 2019.

[230] H. Ramoser, . J. Müller-Gerking y . G. Pfurtscheller, «Optimal Spatial Filtering of Single Trial EEG During,» *IEEE TRANSACTIONS ON REHABILITATION ENGINEERING*, vol. 8, nº 4, pp. 441-445, Diciembre 2000.

[231] P. E. Hart, R. O. Duda y D. G. Stork, Pattern Classification, www, 1973.

[232] C. M. Bishop, Pattern Recognition and Machine Learning, 2006.

[233] H. Ramoser, J. Müller-Gerking y G. Pfurtscheller, «Optimal Spatial Filtering of Single Trial EEG During,» *IEEE TRANSACTIONS ON REHABILITATION ENGINEERING*, vol. 8, nº 4, december 2000.

[234] V. Zarzoso, S. Cruces y R. Martín-Clemente, «Unsupervised Common Spatial Patterns,» *IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING*, vol. 27, nº 10, pp. 2135-2144, 2019.

[235] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Onda_estacionaria. [Último acceso: 2021].

[236] N. Acevedo, «nataliaacevedo.com,» 23 abril 2020. [En línea]. Available: https://nataliaacevedo.com/arboles-de-decisiones-las-bases-para-entenderlos/. [Último acceso: 2021].

[237] «niponageek.com,» [En línea]. Available: https://niponageek.com/2016/10/19/neurowear-biotecnologia-nipona-para-el-cerebro/. [Último acceso: 2021].

[238] «ingenieriabiomedica,» 2021. [En línea]. Available: https://www.ingenieriabiomedica.org/post/primer-dispositivo-con-interfaz-cerebro-computador-para-la-rehabilitaci%C3%B3n.

[239] J. Martinez Heras, «iartificial,» 2 Octubre 2020. [En línea]. Available: https://www.iartificial.net/regresion-lineal-con-ejemplos-en-python/. [Último acceso: 2021].

[240] E. R. Miranda y J. Caste, Edits., de *Guide to Brain-ComputerMusic Interfacing*, Springer, 2014.

[241] C. Martínez Ruedas, «DETECTOR MULTIUSUARIO PARA DS-CDMA BASADO EN SVM,» Sevilla.

[242] «Sesgo estadístico,» [En línea]. Available: https://es.wikipedia.org/wiki/Sesgo_estad%C3%ADstico. [Último acceso: 2021].

[243] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Hiperplano. [Último acceso: 2021].

[244] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte. [Último acceso: 2021].

[245] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Sobreajuste. [Último acceso: Junio 2021].

[246] «chi-square-goodness-of-fit-test,» [En línea]. Available: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/chi-square-goodness-of-fit-test/. [Último acceso: 2021].

[247] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Filtro_elimina_banda. [Último acceso: 2021].

[248] «superprof,» [En línea]. Available: https://www.superprof.es/apuntes/escolar/matematicas/estadistica/disbidimension/correlacion.html. [Último acceso: 2021].

[249] «topdoctors,» [En línea]. Available: https://www.topdoctors.es/diccionario-medico/electromiografia. [Último acceso: 2021].

[250] «languages.oup,» [En línea]. Available: https://languages.oup.com/google-dictionary-es/. [Último acceso: 2021].

[251] «programmerclick.com,» [En línea]. Available: https://programmerclick.com/article/89251664901/. [Último acceso: 2021].

[252] «Random_forest,» [En línea]. Available: https://en.wikipedia.org/wiki/Random_forest. [Último acceso: 4 Julio 2021].

[253] «Boosting en Machine Learning con Python,» 10 Junio 2017. [En línea]. Available: https://relopezbriega.github.io/blog/2017/06/10/boosting-en-machine-learning-con-python/. [Último acceso: 5 Julio 2021].

[254] P. -. G. Izquierdo, «Machine Learning - Regresión Logística,» 27 Enero 2019. [En línea]. Available: https://www.youtube.com/watch?v=_bm2gPdAbS8. [Último acceso: 5 Julio 2021].

[255] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Raz%C3%B3n_de_momios. [Último acceso: 2021].

[256] J. Cerda, C. Vera y G. Rada, «Odds ratio: aspectos teóricos y prácticos,» *Revista médica de Chile,* vol. 141, nº 10, pp. 1329-1335, 2013.

[257] «wisc.edu,» [En línea]. Available: http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/lossfunctions/SquaredLoss.pdf. [Último acceso: 5 abril 2021].

[258] «wikipedia: Error cuadrático medio,» [En línea]. Available: https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio. [Último acceso: 2021].

[259] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Multiplicadores_de_Lagrange. [Último acceso: 2021].

[260] «lexico,» [En línea]. Available: https://www.lexico.com/es/definicion/encefalo. [Último acceso: 2021].

[261] «concepto.de,» [En línea]. Available: https://concepto.de/habilidades-cognitivas/. [Último acceso: 2021].

[262] «psicoactiva,» [En línea]. Available: https://www.psicoactiva.com/blog/arousal-o-activacion-cortical-y-la-ley-de-yerkes-dodson/.

[263] «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/An%C3%A1lisis_exploratorio_de_datos. [Último acceso: 2021].

[264] J. Treviño y R. Landa, «Direcciones Conjugadas con Estimación de Gradiente en Algoritmos Evolutivos,» 2017.

[265] «mathworks: assess-classifier-performance,» [En línea]. Available: https://es.mathworks.com/help/stats/assess-classifier-performance.html. [Último acceso: 2021].

[266] «mathworks: sort,» [En línea]. Available: https://es.mathworks.com/help/matlab/ref/sort.html. [Último acceso: 2021].

[267] «google-dictionary-es,» [En línea]. Available: https://languages.oup.com/google-dictionary-es/. [Último acceso: 2021].

[268] M. d. P. Gómez Gil, «Procesamiento digital de señales,» 2017. [En línea]. Available: http://ccc.inaoep.mx/~pgomez/cursos/pds/slides/S1B-Est.pdf. [Último acceso: 2021].

[269] P. Alcala, «SEÑALES ALEATORIAS Y RUIDO,» [En línea]. Available: https://slideplayer.es/slide/4046340/. [Último acceso: 2021].

[270] J. Carcamo, «Tema 5: Esperanza y momentos,» [En línea]. Available: http://verso.mat.uam.es/~pablo.fernandez/Tema-PREST-5.pdf. [Último acceso: 2021].

[271] «wikipedia: Covarianza,» [En línea]. Available: https://es.wikipedia.org/wiki/Covarianza. [Último acceso: 2021].

[272] «wikipedia: correlation,» [En línea]. Available: https://en.wikipedia.org/wiki/Correlation. [Último acceso: 2021].

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

GLOSARIO

A

AAR
modelos autorregresivos adaptativos, 54
ACSP
adaptive CSP, 258
ANN
Redes Neurales Artificiales, 72
app
Aplicación, 138, 175, 249, 253, 254
AR
Modelos Autorregresivos, 51, 53, 54, 55, 56

B

BCI
Brain Computer Interface, ix, x, xi, 21, 22, 23, 24, 25, 26, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 44, 45, 47, 50, 51, 52, 53, 58, 69, 72, 73, 74, 80, 85, 89, 94, 96, 98, 103, 113, 125, 178, 258, 259

C

CSP
Common Spatial Pattern, x, 51, 52, 57, 59, 61, 67, 68, 69, 114, 237, 240, 241, 242, 245, 247, 248, 249, 250, 252, 253, 254, 258
CWT
Continuous Wavelet Transform, 55

D

DDWT
transformada wavelet diádica discreta, 56

E

ECG
Electrocardiograma, 43, 44
ECOC
Códigos de Salida con Corrección de Errores, 86
ECoG
Electrocorticografía, 37, 44
EEG
Electroencefalograma, 23, 25, 30, 31, 32, 33, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 51, 52, 53, 54, 56, 57, 59, 61, 63, 64, 65, 67, 68, 69, 74, 96
EMG
electromiografía, 44
EOG
Electrooculograma, 42, 43, 44
ERD
Event Related Desynchronization, 34, 49, 50
ERS
Event Related Synchronization, 34, 35, 36, 45, 47, 49, 50, 64

F

FBCSP
Filter Bank Common Spatial Patterns, 68, 69
FBRCSP
CSP regularizado de banco de filtros, 68
FFT
Fast fourier transform, 51, 52, 53, 54, 55, 56, 69
fMRI
functional Magnetic Resonance Imaging, 37
FT
Transformada de Fourier, 55

K

KNN
K-nearest neighbor, 72, 95, 96, 110, 151, 181, 185, 213, 233, 252, 254

L

LDA
Análisis Discriminante Lineal, x, 72, 73, 78, 79, 80, 81, 82, 85, 91, 106, 109, 111, 117, 119, 120, 125, 132, 133, 134,

M

P

Q

R

S

T

W