SISTEMA AMBULATORIO DE BAJO COSTE PARA EL DIAGNÓSTICO AUTOMÁTICO DE EPILEPSIA

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Sistema ambulatorio de bajo coste para el diagnóstico automático de epilepsia

Javier Marchán Loro

Septiembre, 2014

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

Sistema ambulatorio de bajo coste para el diagnóstico automático de epilepsia

Autor: Javier Marchán Loro

Director: Dra. María José Santofimia Romero

Septiembre, 2014

Javier Marchán Loro

Ciudad Real – Spain

E-mail: javier.marchan1@alu.uclm.es; j.marchan.loro@gmail.com
Teléfono: 627 00 11 72
Web site:

TRIBUNAL:

Presidente:

Secretario:

Vocal:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	**SECRETARIO**	**VOCAL**
Fdo.:	Fdo.:	Fdo.:

Resumen

El electroencefalograma (EEG) tiene muchas aplicaciones, entre las que destaca su papel en el diagnóstico de epilepsia. Interpretar a través de una inspección visual la señal recogida es una tarea ardua y principalmente manual y por lo tanto sujeta a errores.

La automatización de esta tarea no es sin embargo sencilla, pues en este tipo de pruebas también se registran artefactos o ruidos que dificultan un análisis automático. En este sentido, nuevas líneas de investigación se han abierto para encontrar mecanismos que sean robustos a este tipo de ruido. Este proyecto implementa un algoritmo siguiendo un modelo Bag-of-Words (BoW), que pretende dar respuesta a la necesidad de un mecanismo eficiente para el diagnóstico automático de epilepsia. El algoritmo descompone la señal EEG usando una transformada wavelet. Después, se extrae un conjunto de vectores de características que resumen la información más importante de la señal. Se obtiene un histograma que caracteriza la señal de los canales EEG haciendo *clustering* y se clasifica con una SVM, obteniendo un modelo con el cual predecir nuevas muestras.

Este trabajo pretende además ofrecer una solución completa, a modo de prototipo totalmente funcional, que permita llevar a cabo esa diagnosis sin tener que depender de costosos equipos de electroencefalografía y personal altamente cualificado (neurofisiólogos) que interprete los resultados. Este trabajo por lo tanto abre una vía alternativa para que países en vías de desarrollo no dependan de la falta de equipamiento o personal entrenado para la diagnosis de este tipo de enfermedad.

Abstract

Electroencephalograms (EEG) have numerous applications, among which stand out epilepsy diagnosis. The signal can be interpreted by visual inspection but this is tedious and mainly manual task hence error prone.

The automatization of this task is therefore not easy. EEG also records artefacts or noise that made more difficult an automatic analysis. New research lines have been opened to find robust mechanics to this kind of noise. This work implements a Bag-of-Words (BoW) algorithm that hope satisfy necessity for automaic epilepsy diagnosis. The algorithm decomposes a EEG signals using wavelet transform analysis. From the decomposed signal, a feature vector set is computed in which salient features of the signal are captured. An histogram is therefore obtained by clustering these features and then used to compute a model using SVM. This modelis used to classify new signals.

This work is also intended to offer a holistic solution, in terms of a comprehensive functional prototype that supports the diagnosis of this illness without having to depend on expensive electroencephalography equipment and high qualified health workers (neurophysiologist). Therefore, this work opens an alternative way for developing countries to overcome the the absence of equipment or workers that is delaying or preventing epileptic patients from being correctly diagnosed.

Índice general

Índice de cuadros

Índice de figuras

Índice de listados

Listado de acrónimos

OMS Organización Mundial de la Salud

TAC Tomografía Axial Computarizada

IRM Imagen por Resonancia Magnética

GNU GNU is Not Unix

PFC Proyecto Fin de Carrera

NUI Natural User Interface

BoW Bag of Words

EEG Electroencefalograma

WT Wavelet Transfrom (Transformada Wavelet)

FT Fourier Transform (Transformada de Fourier)

DWT Discrete Wavelet Transfrom (Transformada Wavelet Discreta)

ARCO Arquitectura y Redes de Computadores

API Application Programming Interface

EM Expectation-Maximization

HRUM Hospital Regional Universitario de Málaga

SDK Software Development Kit

SVM Support Vector Machine

ANSI American National Standars Institute

UML Unified Modeling Language

RISC Reduced Instruction Set Computing

ARM Acorn RISC Machine

SCB Single Board Computer

RBF Radial Basic Function

MRA Análisis Multirresolución

Agradecimientos

A mi familia.
A los compañeros con los que me he cruzado, desde el primer curso al último, con los que tan bien he trabajado diariamente.
A María José: por tu amabilidad, por tu disponibilidad y por todo el tiempo que has dedicado al proyecto, incluida esta memoria.
A mis amigos y a mis amigas, por tantos buenos momentos.

A todos, un agradecimiento sincero.

Javier Marchán Loro

A mi familia

“

Creced y multiplicaos, dijimos, y las máquinas crecieron y se multiplicaron.
Nos habían prometido que trabajarían para nosotros.
Ahora nosotros trabajamos para ellas.
Multiplican el hambre las máquinas que inventamos para multiplicar la comida.
Nos matan las armas que inventamos para defendernos.
Nos paralizan los autos que inventamos para movernos.
Nos desencuentran las ciudades que inventamos para encontrarnos.
Los grandes medios, que inventamos para comunicarnos, no nos escuchan ni nos ven.
Somos máquinas de nuestras máquinas.
Ellas alegan inocencia.
Y tienen razón.

”

Eduardo Galeano, *Espejos: una historia casi universal*, 2008

Introducción

1

La explosión del conocimiento científico y tecnológico de los últimos tiempos ha tenido como resultado la transformación de todos los aspectos de la vida.

La ingeniería, entendida como el desarrollo de tecnologías que permitan dar solución a problemas que se plantean en el conjunto de la sociedad, debiera tener vocación social y la ingeniería informática en particular lleva muchos años aportando soluciones en este sentido.

Sin caer en un fetichismo de la tecnología, sin atribuirle un poder mágico o sobrenatural por el que pueda resolver problemas de índole muy distinta (ya sea político, social, etc.), es de vital importancia comprender su potencial transformador y su papel en la mejora objetiva de las condiciones de vida de la población.

El trabajo que se presenta aquí ha sido concebido con esta ilusión.

1.1 ¿Qué es la epilepsia?

La epilepsia es un trastorno cerebral caracterizado principalmente por recurrentes e impredecibles[1] interrupciones de la actividad cerebral normal. Estas interrupciones se denominan ataques epilépticos [FvEBb+05]. Están causados por descargas anormales de actividad eléctrica neuronal, que impiden el control normal de diferentes funciones cerebrales como la conciencia, la memoria, la atención, el movimiento, la sensibilidad, la vista, el oído, el gusto o el olfato. Estas descargas anormales se propagan a otras neuronas y se extienden por la corteza cerebral. A grandes rasgos, se pueden diferenciar dos tipos de crisis según su extensión en el cerebro: crisis focales o parciales en las que la zona que se activa al inicio de la crisis está limitada a un grupo de neuronas y crisis generalizadas en las que se activan simultáneamente extensos grupos de neuronas en ambos hemisferios cerebrales.

Es un problema global que afecta a todas las edades, clases sociales, familias y países, imponiendo una enorme carga física, psicológica, social y económica sobre ellos [JFM+12] [Org05]. Según la Organización Mundial de la Salud (OMS)[2] hay aproximadamente 50 millo-

[1] El término epilepsia deriva del griego 'epilambaneim', que significa 'coger por sorpresa'.

[2] Organización Mundial de la Salud. Es la autoridad directiva y coordinadora de la acción sanitaria en el sistema de las Naciones Unidas.

nes de personas con epilepsia en todo el mundo; 65 millones según la Epilepsy Foundation[3], siendo uno de las mayores trastornos cerebrales del mundo; 360.000 casos se darían en España. Se estima, además, que el 5-10 % de la población padece alguna crisis epiléptica a lo largo de su vida.

Estudios recientes han revelado que un 70 % de personas con epilepsia pueden llevar una vida normal si son diagnosticadas y tratadas correctamente, tanto en países en desarrollo como desarrollados [Bra04].

La incidencia (el número de nuevos casos por año) de epilepsia es de 24 a 53 por cada 100.000 personas en países desarrollados [AD90] y de 49.3 a 190 por cada 100.000 personas en países en desarrollo [P02]. En España la incidencia anual es de 31 a 57 por cada 100.000, lo que da una cifra de 12.400 a 22.000 casos nuevos cada año.

Hay evidencias considerables de que el número de afectados por desórdenes neurológicos y mentales se incrementará, aumentando a su vez el gasto sanitario.

Para una gestión apropiada de los enfermos de epilepsia, los servicios de diagnóstico constituyen un recurso importante. La Tomografía Axial Computarizada (TAC), la Imagen por Resonancia Magnética (IRM), el EEG y el vídeo-EEG están disponibles para los profesionales de la salud en un 85 %, 69 %, 87 % y 50 % de los países estudiados en [Org05], respectivamente[4].

Sin embargo, hay una disponibilidad desigual de los servicios de diagnóstico entre los diferentes grupos estudiados según sus ingresos[5]. Se estima que el 80 % de personas con epilepsia reside en países en desarrollo y no reciben tratamiento moderno o incluso no han sido diagnosticados [P97].

La mayoría de las nuevas tecnologías están disponibles en muchos países, aunque pueden estar concentradas en unos pocos centros o en las mayores ciudades. En países con bajos ingresos donde hay pocos recursos, el alto coste recurrente que implica el mantenimiento del equipamiento puede limitar su disponibilidad. La aspiración, como marca la OMS, debería ser mejorar la cobertura para servir a toda la población.

Pero no sólo los países empobrecidos tienen problemas para contar con suficientes recursos en el tema que nos ocupa. La Unidad de Epilepsia del Hospital Regional Universitario de Málaga (HRUM), con la que se ha colaborado activamente en el planteamiento de este proyecto tiene que lidiar con listas de espera en torno al año y medio o dos años para realizar

[3]Epilepsy Foundation: About Epilepsy. Recuperado el 16 de julio de 2014: `http://www.epilepsy.com/learn/epilepsy-101/what-epilepsy`

[4]Todos los datos e información contenidos en [Org05] han sido recogidos en un gran estudio internacional llevado a cabo en el periodo de 2002 a 2004 y que incluyó a 160 países, áreas o territorios.

[5]Los países fueron agrupados en cuatro categorías según el producto interior bruto (PIB) *per cápita* de 2003 [Ban05]. Los grupos son los siguientes: ingresos bajos (US$ 765 o menos), ingresos medio-bajos (US$ 766–3035), ingresos medio-altos (US$ 3036–9385) e ingresos altos (US$ 9386 o más).

algunas pruebas diagnósticas como un EEG de sueño [6].

1.2 El papel del EEG en el diagnóstico de la epilepsia

Según la Epilepsy Foundation, además del historial clínico y el examen neurológico, el EEG es la herramienta más determinante en el diagnóstico de la epilepsia. Una máquina capaz de realizar electroencefalogramas es un dispositivo que registra la actividad eléctrica del cerebro a través de electrodos colocados en la cabeza del paciente recogiendo las señales producidas por las descargas eléctricas de las neuronas en determinadas partes del cerebro.

El EEG también ayuda a determinar el tipo de ataque y de síndrome epiléptico, dando información tanto para el diagnóstico, clasificación, pronóstico, evolución y decisión sobre el tratamiento. Sin embargo, a pesar de su importancia esta tarea sigue siendo principalmente manual y requiere de profesionales altamente entrenados que interpreten los registros del EEG a través de una inspección visual.

Interpretar un registro EEG no es una tarea sencilla. Como se ha dicho requiere de entrenamiento y de una gran formación. No existen patrones determinados que puedan indicar con seguridad la existencia de una crisis (aunque hay algunos tipos de epilepsia asociados a algunos patrones típicos EEG). Cuando alguien tiene una crisis, causa un cambio en el EEG. Esto significa que aunque sea descrito como no normal, no prueba que la persona tenga epilepsia. Un EEG normal tampoco excluye la existencia de epilepsia (alrededor de un 10 % de los pacientes con epilepsia nunca muestran descargas epileptiformes [Smi05]).

A pesar de las limitaciones tradicionales de esta herramienta, que se trasladan al intento de automatizar la tarea del diagnóstico, existen multitud de trabajos que han abordado este reto, como se verá en el capítulo 3.

1.3 Estructura del documento

Capítulo 2: Objetivos

Donde se enuncia el objetivo general que se busca y se concretan los objetivos específicos marcados para conseguirlo.

Capítulo 3: Antecedentes

Donde se exponen los conocimientos elementales para comprender el resto del documento. Se han omitido explicaciones tediosas o enrevesadas de los conceptos matemáticos, presentándolos de forma intuitiva. Se describen también los principales trabajos hasta la fecha sobre el tema.

Capítulo 4: Método de trabajo

Donde se describe y se da una introducción de la metodología utilizada para resolver el

[6] El paciente permanece sin dormir la noche anterior a la realización de la prueba y duerme durante la realización de la misma. Suelen ser más largos de lo habitual.

problema, los pasos seguidos y algunas de las cosas más destacables de la implementación. Se listan al final del capítulo las herramientas hardware y software implicadas.

Capítulo 5: Arquitectura

Donde se describe la arquitectura del sistema desarrollado y algunos de los patrones software utilizados. Al final del capítulo se incluye el diagrama de clases resultante.

Capítulo 6: Resultados

Donde se incluye información sobre los *datasets* utilizados y el repositorio donde se aloja el trabajo. Se hace una análisis de los resultados y se comparan con trabajos previos y por último, se muestran los listados de código para entrenar el sistema y usarlo para predecir.

Capítulo 7: Conclusiones y propuestas

Donde se extraen conclusiones de los resultados analizados y se proponen futuras líneas de trabajo o mejoras para continuar el Proyecto Fin de Carrera (PFC).

Objetivos

2

A continuación se explica el objetivo general del PFC y el alcance del mismo. Para poder alcanzarlo se marcan una serie de subojetivos u objetivos específicos que son los que guiarán todo el desarrollo.

2.1 Objetivo general

Una vez que se ha comprendido la carga que supone la epilepsia en la sociedad, así como la importancia del EEG para su diagnóstico, resulta conveniente mitigar los problemas e inconvenientes que se presentan tanto en los países desarrollados como en desarrollo por su falta de accesibilidad, incomodidad para los pacientes, largas listas de espera, etc.

Es por eso que este PFC se marca como objetivo buscar alternativas a los servicios de diagnóstico que complementen a los actuales o sirvan como base para futuros trabajos que acaben proporcionando soluciones completamente eficaces y funcionales. Para ello, se desarrollará un *sistema ambulatorio de bajo coste para el diagnóstico automático de la epilepsia*. Ambulatorio por permitir al paciente cierta movilidad y no obligarlo a estar en cama. Automático porque el sistema será capaz de clasificar la actividad cerebral recogida como una señal *sana* o *epiléptica*. La solución, a su vez, debe suponer una diferencia de precio respecto a los dispositivos médicos actuales tal que se pueda considerar de bajo coste.

En ningún caso se pretende sustituir al personal sanitario sino facilitar su trabajo y mejorar la calidad de la atención médica, favoreciendo la accesibilidad a este recurso básico para el diagnóstico de la epilepsia redundando en beneficio de toda la sociedad.

2.2 Objetivos específicos

Para poder cumplir el objetivo general marcado y obtener un sistema como el descrito, se definen una serie de objetivos específicos funcionales.

2.2.1 Implementación de un algoritmo BoW

La implementación de un algoritmo BoW es el pilar fundamental del PFC. Será capaz de procesar la señal eléctrica obtenida a través de un EEG o de un *dataset* y ofrecer una predicción.

Para llevar a cabo esta implementación será necesario:

- Un componente que lea de un *dataset* una señal EEG, formando tramas de *s* segundos llamadas «ventanas» para su procesamiento posterior.
- Un componente que realice una Discrete Wavelet Transfrom (Transformada Wavelet Discreta) (DWT) a una señal.
- Un componente que agrupe o haga *clustering* sobre un conjunto de vectores. Estos vectores serán resultado de un proceso de caracterización de la señal. En capítulos posteriores se explicará en detalle.
- Un componente que clasifique unos datos de entrada, a partir de un modelo obtenido previamente entrenando el clasificador.

Se puede considerar como un objetivo inherente el estudio de los conceptos mínimos necesarios para implementar el algoritmo. Esto permitirá comprender el algoritmo y conocer el funcionamiento de las librerías utilizadas.

2.2.2 Captación de señal de un dispositivo EEG

Además de poder aplicar el algoritmo a muestras de datos almacenadas previamente, el sistema buscado debe ser capaz de realizar un EEG. Para ello, se evaluarán los dispositivos de EEG disponibles en el mercado atendiendo a dos criterios:

- El sistema está concebido para ser de bajo coste.
- El sistema está concebido para ser ambulatorio, es decir, debe permitir cierta movilidad al paciente.

2.2.3 Estudio de librerías disponibles

Para llevar a cabo satisfactoriamente los objetivos específicos anteriores será necesario una revisión de las librerías disponibles en el lenguaje de programación escogido que provean de la funcionalidad requerida en cada momento. Se buscará en cualquier caso el uso de las librerías más extendidas en la literatura y que estén suficientemente probadas y documentadas.

2.2.4 Integración en otras plataformas hardware

Una vez que el sistema esté desarrollado y funcionando en la plataforma habitual de desarrollo, se estudiará la posibilidad de integrarlo en un Single Board Computer (SCB) como Raspberry Pi. No hay preferencia por ninguna tecnología ni ningún producto específico del mercado. La idea es integrarlo en una plataforma más pequeña que un computador de escritorio o portátil que sea capaz de hacer el trabajo de procesamiento. Si bien la funcionalidad sería la misma, se tendría de esta forma un sistema más manejable.

Antecedentes

ESTE capítulo proporciona los conocimientos previos necesarios para abordar el Proyecto Fin de Carrera. Pretende mostrar todos aquellos conceptos y herramientas utilizados durante su realización. Algunas partes se tratarán de forma resumida pero suficiente, especialmente las relativas a la componente matemática que subyace en todos ellos.

Por último, se exploran los principales trabajos realizados sobre el tema.

3.1 Electroencefalografía

La Electroencefalografía (EEG) es la captura y registro de la actividad eléctrica del cerebro a través de electrodos colocados en la cabeza del paciente, bien sobre la superficie del cuero cabelludo (electrodos superficiales) o bien sobre la base del cráneo sin necesidad de procedimiento quirúrgico (electrodos basales).

La electroencefalografía es usada en una amplia variedad de situaciones clínicas, pero la gran mayoría de EEGs son obtenidos como parte de la evaluación de ataques o epilepsia. El EEG también es útil separando enfermedades psiquiátricas de desórdenes orgánicos.

Como ya se ha dicho en el capítulo 1: Introducción, para la *Epilepsy Foundation*[1], además del historial clínico y el examen neurológico, el EEG es la herramienta más determinante en el diagnóstico de la epilepsia.

Además ayuda a determinar el tipo de ataque y de síndrome epiléptico, dando información relevante para los profesionales sanitarios sobre la clasificación, evolución o para decidir sobre el tratamiento.

A pesar de que las interferencias electromagnéticas o los artefactos de movimiento generan ruido en los registros de biopotenciales[2], presenta algunas ventajas resumidas en [AG13]:

- Los costes hardware son significativamente menores que otros métodos.
- Los sensores EEG pueden ser colocados en mayor número de lugares que otras técnicas y permiten la movilidad del sujeto.

[1] `http://www.epilepsy.com/`.

[2] Potencial eléctrico que puede medirse entre dos puntos en células vivas, tejidos y organismos y que es consecuencia de sus procesos bioquímicos.

- La frecuencia de muestreo es superior a otros métodos.
- La Electroencefalografía es silenciosa, lo que posibilita un mejor estudio cuando el sujeto debe experimentar estímulos sensoriales.
- La Electroencefalografía también puede detectar procesamiento de información encubierto (que no requiere una respuesta).
- Puede ser usada en sujetos incapaces de dar respuestas motrices.
- Es no invasiva.

La amplitud de la señal de un electroencefalograma común está determinada por el voltaje y suele situarse entre -100 y +100 μV. La frecuencia rara vez excede los 40 Hz, en la onda Gamma. Esto hace que se puedan eliminar las bandas con frecuencias superiores a 50 Hz aplicando un filtro digital sin una pérdida importante de información. Así, se limpiaría la señal de ruido e interferencias, que supone un primer paso imprescindible para el procesamiento posterior de la señal, sea cual sea la aplicación. La Figura 3.1 muestra un ejemplo de una señal EEG normal[3].

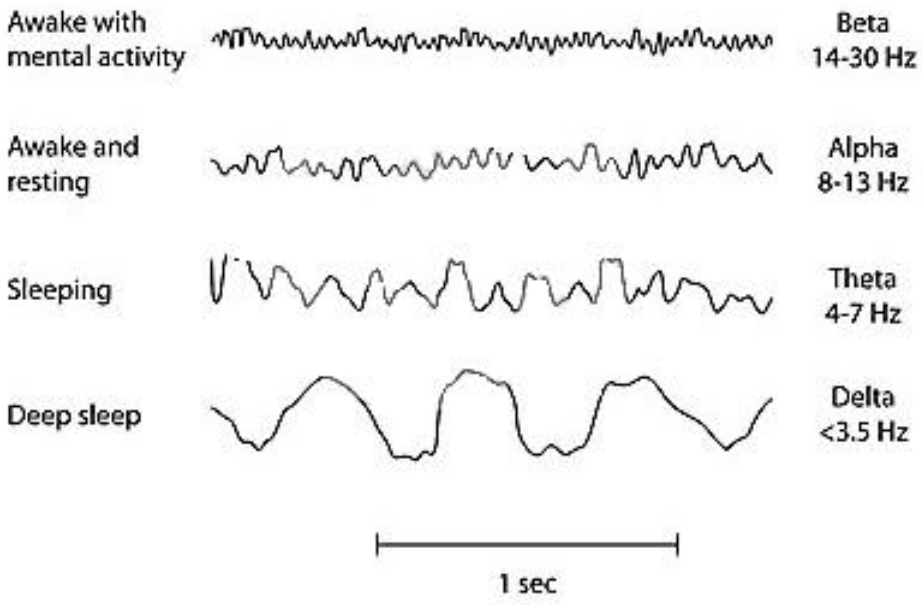

Figura 3.1: Ejemplo de ondas cerebrales normales

La señal EEG es no estacionaria (ver 3.2). Una señal EEG registrada por un electrodo está influenciada por diferentes estructuras cerebrales profundas, cada una «transmitiendo» con una intensidad diferente y cambiante y en una fracción de segundo la fuente principal de la señal registrada a menudo cambia de una estructura a otra [Klo09].

[3]El uso del término normal es problemático aquí. Sólo se puede comparar un EEG de una misma persona en diferentes períodos de tiempo.

3.1.1 Otras aplicaciones

En el contexto de este PFC, la electroencefalografía se usa como herramienta clínica para el diagnóstico de epilepsia. Pero cada vez su uso es más extendido y se construyen lo que se denomina interfaces naturales (Natural User Interface (NUI)) que permiten la interacción persona-computador a partir de la actividad eléctrica del cerebro.

Para mostrar su potencial y el auge de esta tecnología se exponen algunos ejemplos recientes de aplicaciones aparecidos en los medios de comunicación, lo que invita a pensar que el uso de la electroencefalografía no es una anécdota y su avance seguirá en los próximos años.

- BrainWriterr[4]. Proyecto de código abierto presentado en la exposición del *Barbican's "Digital Revolution"* que aúna la tecnología de reconocimiento ocular con un EEG que permite a personas con parálisis comunicarse con los demás e interactuar con un computador.

- Exoesqueletos controlados con la mente.[5] Tuvo un impacto mediático enorme en la inauguración de la Copa Mundial de Fútbol en Sao Paulo el 12 de junio de 2014. Este sistema procesa las señales EEG y el exoesqueleto ayuda a moverse a gente con parálisis.

- Emotiv Insight[6] [7]. Se trata de la nueva versión del producto utilizado en este trabajo (se verá más adelante) que proporciona la capacidad de interactuar con el mundo que nos rodea a personas con necesidades especiales. Es llamativo el caso de una joven en un estado semi-vegetativo que fue capaz de controlar directamente objetos virtuales en una pantalla a través del *headset*, el «casco» que se pone en la cabeza y realiza el EEG.

- En [LCD+13] se lleva a cabo un experimento para controlar un robot cuadricóptero en las tres dimensiones del espacio usando un EEG no invasivo. Demuestra por primera vez la habilidad para controlar un robot volador usando un EEG como interfaz cerebro-computador.

En todos estos ejemplos, el denominador común es utilizar un equipo que realice un EEG, generalmente uno no profesional y de bajo coste, y posteriormente procesar la información obtenida de forma que un sistema sea capaz de «aprender» a interpretarla.

[4] `http://blogthinkbig.com/dispositivo-brainwriter/` Último acceso: 14 de agosto de 2014.

[5] `http://www.bbc.co.uk/mundo/noticias/2014/05/140507_mundial_exoesqueleto_brasil2014_am.shtml` Último acceso: 14 de agosto de 2014.

[6] `http://emotiv.com/`, Último acceso: 14 de agosto de 2014.

[7] `http://iq.intel.com/mind-reading-headset-helps-people-with-special-needs/?linkId=7565917` , Último acceso: 14 de agosto de 2014.

3.2 Análisis de señal

Esta sección tiene como objetivo dar una visión general e introducir los conceptos básicos, simplificando todo lo posible la componente matemática, que son necesarios para la comprensión de la Transformada Wavelet; esto es, la diferencia entre señales estacionarias y no estacionarias, la Transformada de Fourier, la Transformada Rápida (STFT), que intenta corregir las deficiencias de las Transformada de Fourier y por último, la Transformada Wavelet.

3.2.1 Fundamentos básicos

Muchos fenómenos físicos se pueden describir mediante una señal en el dominio del tiempo. Esta señal se puede dibujar como una función tiempo-amplitud, donde el tiempo es la variable independiente y la amplitud la variable dependiente.

Señales estacionarias y no estacionarias

Se habla de señales estacionarias cuando mantienen constante en el tiempo algún parámetro temporal significativo, en este caso particular, la frecuencia. La figura 3.2 muestra una señal estacionaria de 5 Hz. El componente de frecuencia se encuentra presente en todo instante de tiempo. La figura 3.3 muestra la siguiente señal:

$$x(t) = sin(2\pi 10t) + sin(2\pi 20t) + sin(2\pi 50t) \tag{3.1}$$

Es una señal estacionaria con tres componentes de frecuencias de 10, 20 y 50 Hz, que están presentes en cualquier instante.

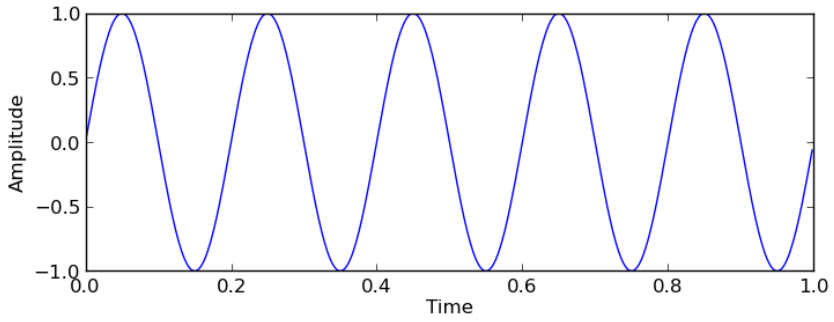

Figura 3.2: Señal de 5 Hz

Por el contrario, se habla de señales no estacionarias cuando existen componentes de frecuencia distintos para intervalos de tiempo determinados. La figura 3.4 muestra una señal no estacionaria que tiene tres componentes de frecuencias distintas para tres intervalos de tiempo: de 0 a a 0.4 segundos una sinusoide de 10 Hz; de 0.4 a 0.8 segundos una de 20 Hz y de 0.8 a 1 segundo otra de 50 Hz .

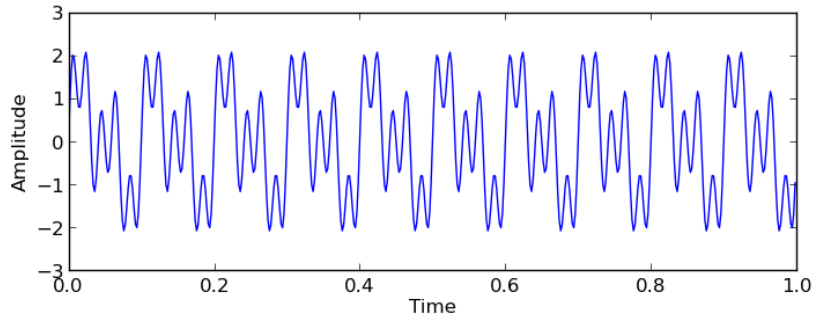

Figura 3.3: Señal con contenido de 10, 20 y 50 Hz

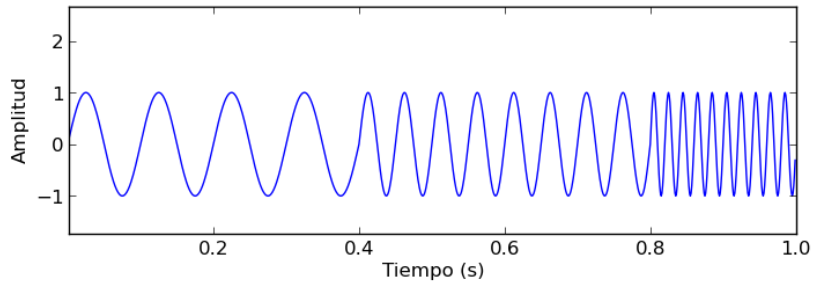

Figura 3.4: Señal con contenido de 5, 10 y 25 Hz

Transformada de Fourier

Al dibujar una señal como una función tiempo-amplitud, no siempre se muestra la información más apropiada. En muchos casos, resulta más útil conocer la información que contiene la señal en el dominio de la frecuencia, es decir, representarla como una función del tiempo y de la frecuencia.

La Transformada de Fourier

$$F(w) = \int_{-\infty}^{\infty} f(t)e^{-jwt}dt \tag{3.2}$$

muestra el contenido de frecuencia a partir del dominio del tiempo. La Figura 3.5 muestra una señal de 5 Hz y su espectro de frecuencias, donde se comprueba que la señal tiene una sola componente de frecuencia de 5 Hz en todo el espectro.

Las figuras 3.6 y 3.7 muestran ahora las señales anteriores (3.3 y 3.4) con sus respectivos espectros de frecuencia.

Aunque las dos señales son muy diferentes en el dominio del tiempo, si se comparan los espectros se observa que son muy parecidos. Es fácil darse cuenta de que la Transformada de Fourier no aporta información respecto al instante de tiempo en el que aparece dicha frecuencia, algo que es crucial para las señales no estacionarias, sino solo respecto al contenido

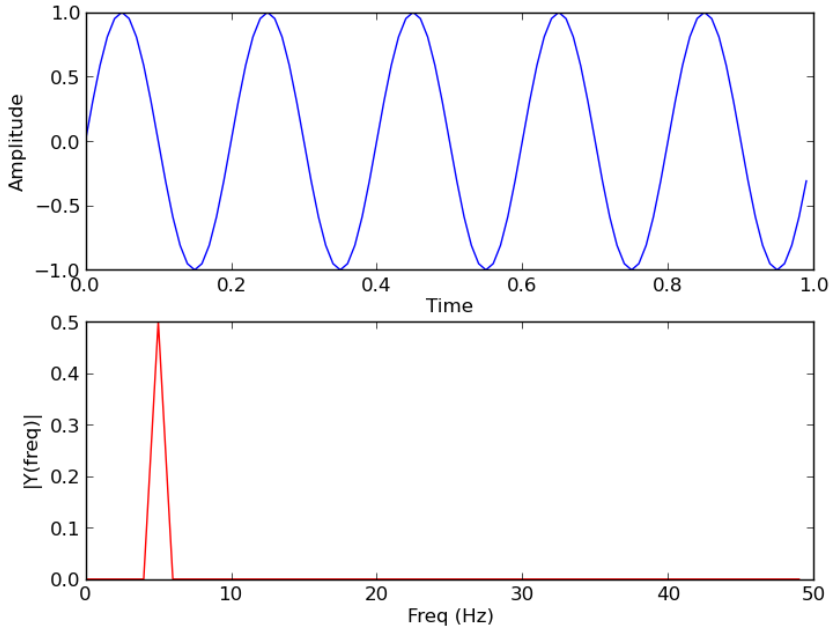

Figura 3.5: Señal de 5 Hz y su espectro de frecuencias

espectral de la señal.

Transformada de Fourier de tiempo corto (STFT)

Resuelve el problema del análisis de señales no estacionarias mediante la transformada de Fourier. Es una adaptación utilizando un procedimiento llamado segmentación. Consiste en dividir una señal *x(t)*, mediante una función tiempo-ventana, en segmentos a través del tiempo de manera que se pueda asumir que cada segmento es una señal estacionaria. Se calcula la Transfomada de Fourier a cada uno de estos segmentos y se traslada la función ventana hasta que no se sobrepone con la anterior. Este proceso continúa hasta que se cubre toda la señal.

Sin embargo, no se resuelve el problema de la representación tiempo-frecuencia. Como resultado de el principio de incertidumbre de Heisenberg, no es posible conocer las representación exacta tiempo-frecuencia, sino sólo los intervalos de tiempo en los cuales existen determinadas bandas de frecuencia. Aparece un problema de resolución.

La Transformada de Fourier de Tiempo Corto tiene, por tanto, una limitación importante debido a que las dimensiones de las ventanas son constantes. Ventanas estrechas proporcionan una resolución pobre en el dominio de la frecuencia pero buena en el dominio del tiempo; ventanas anchas proporcionan buena resolución en el dominio de la frecuencia y mala en el dominio del tiempo.

Para resolver estas limitaciones nace la Trasformada Wavelet y el Análisis Multirresolución (MRA). El MRA analiza una señal para diferentes frecuencias con diferentes resolucio-

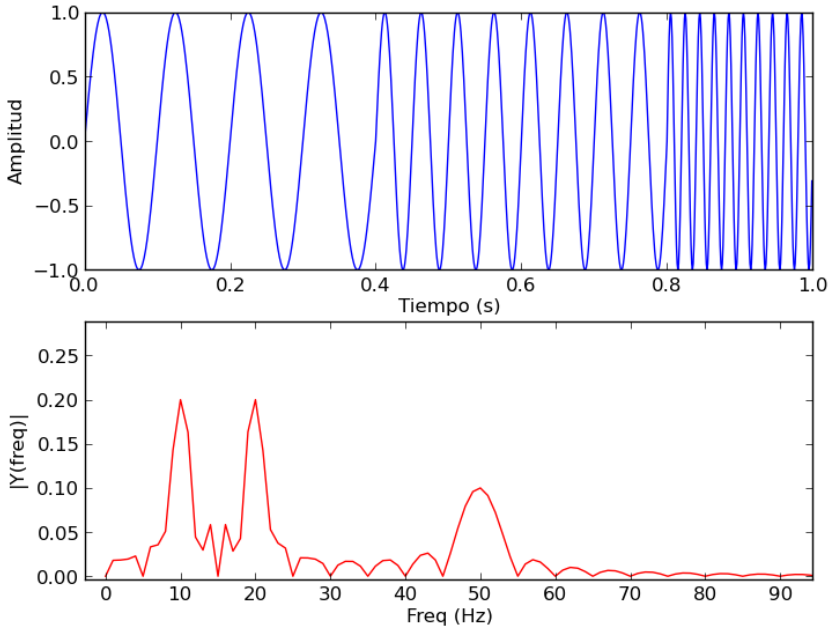

Figura 3.6: Señal no estacionaria con contenido de 10, 20 y 50 Hz y su espectro de frecuencias.

nes, por lo tanto cada componente espectral no se resuelve de idéntica forma como sucede en la STFT.

3.2.2 Transformada Wavelet

La Transformada Wavelet (a veces llamada transformada de ondícula) es una herramienta matemática desarrollada a mediados de los años 80. Resuelve los problemas de la STFT, haciendo posible una buena representación de una señal tanto en tiempo como en frecuencia y es eficiente para el análisis de señales no estacionarias y de rápida transitoriedad.

De forma simplificada, la Transformada Wavelet de una función $f(t)$ es la descomposición de $f(t)$ en un conjunto de funciones $\Psi_{s,\tau}(t)$ que son llamadas *Wavelets*. Son funciones bases de la Transformada Wavelet, generadas a partir de la traslación y dilatación o cambio de escala de una función Wavelet básica, llamada *Wavelet madre* que se define como:

$$\Psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\Psi\left(\frac{t-\tau}{s}\right) \tag{3.3}$$

La Transformada Wavelet se define como:

$$W_f(s,\tau) = \int f(t)\Psi_{s,\tau}(t)dt \tag{3.4}$$

Las funciones *Wavelets* $\Psi_{s,\tau}$ que se generan de la *Wavelet madre* $\Psi(t)$ tienen diferente

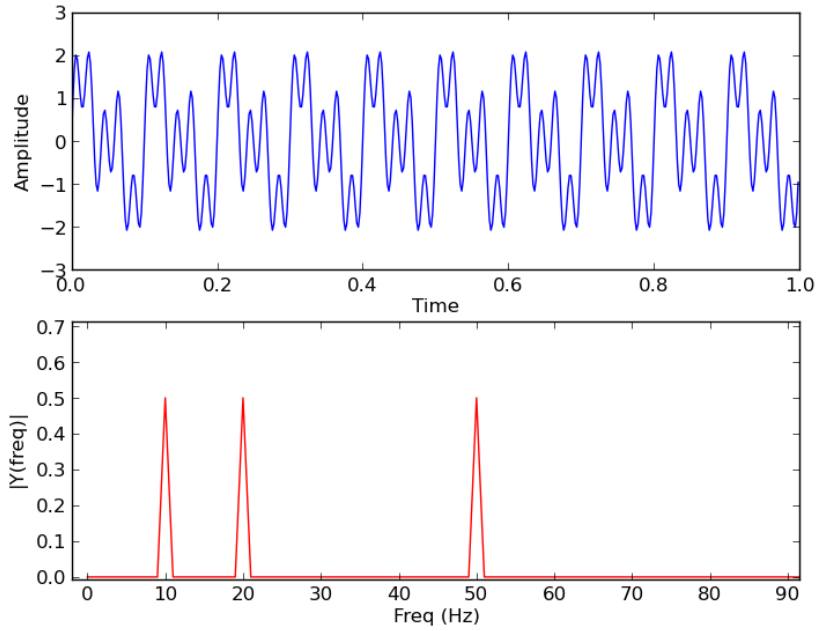

Figura 3.7: Señal no estacionaria con contenido de 10, 20 y 50 Hz y su espectro de frecuencias.

escala s y ubicación τ, pero la misma forma. Así, cambiando el valor de s se cubren rangos diferentes de frecuencia.

Esta transformada tiene unas propiedades excelentes para muchas clases de señales del mundo real y es muy eficiente computacionalmente. Ha sido aplicado en campos como la compresión y procesamiento de imágenes, reconocimiento de patrones, análisis de señales de electrocardiogramas y electroencefalogramas, de sonido o de radar.

3.2.3 Transformada Discreta Wavelets (DWT)

Actualmente todos los procesos numéricos son realizados por computadores. Por este motivo se utiliza la transformada wavelet discreta (DWT). Es capaz de entregar suficiente información tanto para el análisis como para la reconstrucción de una señal con una significativa reducción del tiempo de procesamiento.

Las frecuencias que son más dominantes en la señal original aparecerán como amplitudes altas en la región de la DWT que incluya esas frecuencias. La diferencia con la Fourier Transform (Transformada de Fourier) (FT) es que con la DWT no se pierde la localización en el tiempo de estas frecuencias.

El cálculo de la Transformada Discreta Wavelet se hace esencialmente a partir de los Bancos de Filtros (FB). Un banco de filtros es un conjunto de filtros pasabajos, pasa-altos y pasabanda, cada uno de los cuales cubre una banda en el espectro de frecuencia. Cada filtro separa la función dada en dos sub-bandas no superpuestas independientes, de alta y baja

frecuencia. A este proceso se le denomina análisis. La señal original puede ser reconstruida con la transformada inversa. Cuando los filtros se aplican de forma continua se obtiene un árbol de filtros, donde la salida de uno es la entrada del siguiente. Es decir, para obtener dicho árbol:

- Se descompone la señal en dos componentes: uno de baja frecuencia (*approximation*) y otro de alta frecuencia (*detail*).
- Se vuelve a aplicar un filtro a el componente de baja frecuencia.
- El proceso continúa hasta que la señal se ha descompuesto en un cierto número de niveles predefinidos.

La figura 3.8 muestra un ejemplo de descomposición de una señal.

Existen diferentes familias de Transformadas Wavelet: Haar, Daubechies, Symlets, Coiflest son algunas de ellas. Escoger la Wavelet correcta para una aplicación específica es crucial y debe basarse en la forma de la señal a analizar.

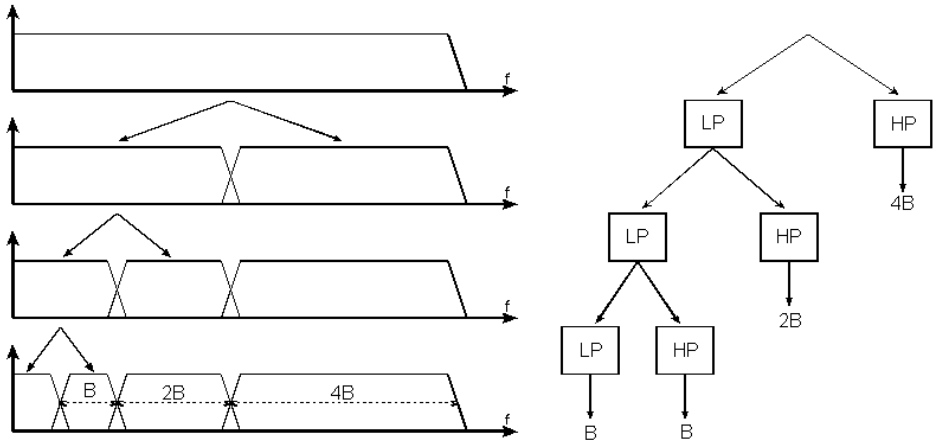

Figura 3.8: Descomposición de la señal usando DWT

En [MdCF02] se explica con más detalle. También se puede consultar [Mal89], y [Dau92] para un tratamiento teórico del análisis wavelet por Ingrid Daubechies, matemática y física que da nombre a una de las familias Wavelets más extendidas.

NOTA: En el anexo A se encuentran los scripts hechos para obtener las imágenes de las señales de esta sección.

3.3 Algoritmo de agrupamiento o *Clustering*

Antes de abordar los algoritmos de agrupamiento o *Clustering* se presentan brevemente dos conceptos necesarios para su comprensión.

3.3.1 Datos

Las observaciones o los datos son el objeto del *clustering*. Consisten en una serie de atributos que están en un orden. Un punto *x* con *m* atributos es un vector de dimensión *m*.

$$x = (x_1, x_2, ..., x_m) \tag{3.5}$$

N puntos x_i forman un *set* o conjunto llamado conjunto de datos o *dataset*.

$$D = (x_1, x_2, ..., x_N) \subset \mathbb{R}^m \tag{3.6}$$

que puede representarse también como una matriz $N \times m$

3.3.2 Distancia

Desde un punto de vista formal, la distancia o métrica se define como cualquier función real *d*, tal que para cualquier punto *x, y, z*:

$$d(x, y) \geq 0,\ y\ (x, y) = 0,\ si\ y\ solo\ si\ x = y \tag{3.7}$$

$$d(x, y) = d(y, x) \tag{3.8}$$

$$d(x, z) \leq d(x, y) + d(y, z) \tag{3.9}$$

La primera propiedad asegura que la distancia es siempre no negativa, y la única forma de que pueda ser cero es para puntos iguales. La segunda propiedad indica la naturaleza simétrica de la distancia. La tercera propiedad es la desigualdad triangular.

A continuación se discuten tres de las métricas más conocidas: Euclídea, Manhattan y Mahalanobis

Distancia Euclídea

También conocida como la norma L_2, es la métrica más usada y se deriva del Teorema de Pitágoras. Es la manera más común en la que se mide la distancia en el mundo real.

$$d_{euclidean}(x, y) = \sqrt{\sum_{i=0}^{m} (x_i - y_i)^2} \tag{3.10}$$

donde *x* e *y* son vectores de dimensión *m*. Funciona mejor con dimensiones pequeñas ya que ignora la similitud entre atributos.

Distancia Manhattan

También conocida como la norma L_1. Es una métrica de uso común que obtiene su nombre del diseño de la mayoría de las calles de Manhattan. Para dos puntos, se define como la suma de la diferencia de sus componentes.

$$d_{manhattan}(x, y) = \sum_{i=0}^{m} |x_i - y_i| \tag{3.11}$$

Distancia de Mahalanobis

Es una función de distancia estadística. Mide la distancia entre dos puntos en el espacio definidos por dos o más variables correlacionadas. Es decir, la distancia de Mahalanobis, al contrario que la distancia euclídea, tiene en cuenta la correlación entre las variables. Si dos variables no tienen correlación ambas distancias son iguales.

$$d_{mahalanobis}(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)} \tag{3.12}$$

donde Σ^{-1} es la matriz de covarianza.

3.3.3 *Clustering*

Consiste en dividir o agrupar, de una forma no supervisada, un conjunto de vectores en grupos de tal manera que los vectores de un mismo grupo, al que se llama *cluster*[8], sean más similares entre ellos que a los que están en otros grupos o *clusters*.

Los algoritmos de agrupamiento y otras muchas técnicas están basadas en la medida de la similitud entre objetos. Generalmente la similitud se define en términos de la distancia y bajo este criterio es por el que se agrupan los vectores.

Sin embargo, el clustering no es un algoritmo en sí mismo; simplemente define el objetivo que se pretende lograr. Para lograrlo, es decir, encontrar los *clusters* en los cuáles se agrupan el conjunto de datos a analizar, hay que aplicar algún algoritmo de *clustering*. Hay una gran cantidad de algoritmos y se pueden categorizar por su modelo o su noción de *cluster*. Fraley y Raftery (1998) [CA98] sugirieron dividirlos en dos grupos principales:

- *Clustering* jerárquico. Busca una jerarquía de *clusters*. Conecta objetos para formar *clusters* según su distancia de forma recursiva. El resultado es un dendograma que representa los grupos de objetos anidados y los niveles de similitud (Figura 3.9).
- *Clustering* basado en particiones. Coloca las instancias de un *cluster* a otro partiendo de un particionamiento inicial.

[8] Aunque no hay un criterio general aceptado para definir cluster, se puede asumir que es un grupo de muestras mutuamente similares.

Posteriormente, Han y Kamber (2001) [HK01] sugirieron categorizar los métodos en tres categorías adicionales:

- Métodos *density-based*. Asumen que los puntos que pertenecen a cada cluster se han obtenido a partir de una distribución probabilística específica.
- *Clustering model-based*. Estos métodos intentan optimizar el proceso de adaptación de los datos a determinados modelos matemáticos.
- Métodos *grid-based*. Estos métodos particionan el espacio en un número finito de celdas que forman una estructura *grid* sobre las cuales son llevadas a cabo todas las operaciones para *clustering*.

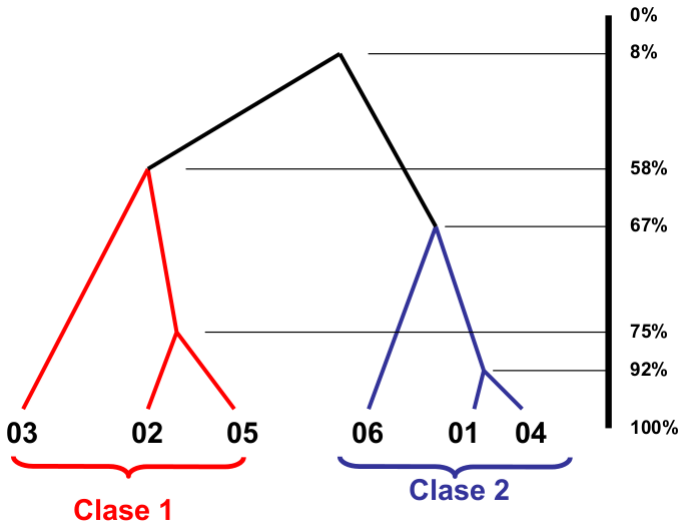

Figura 3.9: Ejemplo de un resultado de análisis por *clustering* jerárquico. Imagen obtenida de [Oli]

La calidad se consigue maximizando la distancia *inter-clusters* y minimizando la distancia *intra-clusters* y depende del algoritmo, de la función distancia y de la aplicación. Más información sobre *clustering* y sus métodos en [RM] y [Dur10].

Algoritmo k-means

Es el método de particionamiento más conocido. K-means es un método iterativo que busca la mejor separación de los datos en un predeterminado número de clusters. K puntos aleatorios son escogidos para ser los centros de los k clusters (centroides) en la primera iteración y cada instancia es asignada al cluster con el centro más cercano. El siguiente paso es recalcular los centros de cada cluster como la media de las instancias que pertenecen al cluster:

$$\mu_k = \frac{1}{N_k} \sum_{q=1}^{N_k} x_q \tag{3.13}$$

donde N_k es el número de las instancias que pertenecen al *cluster k*.

Todas las instancias son reasignadas de nuevo al cluster cuyo centro es el más cercano. Este paso es repetido hasta que no se producen más reasignaciones y los centroides quedan establecidos. En cada iteración, el error cuadrático ϵ de la distancia de cada instancia $x_i, \forall i \in dataset$ al centroide c_k es minimizado.

$$\varepsilon = \sum_{k=1}^{K} \sum_{i \in c_j} \parallel x_i - c_k \parallel^2 \tag{3.14}$$

Como todos los algoritmos de clustering, se necesita escoger una función distancia para medir la cercanía de las instancias a los centros de los clusters. Para un valor dado del parámetro k, el algoritmo k-means encuentra k clusters. Esto hace que la elección de este parámetro sea muy importante. El Listado 3.1 contiene un pseudocódigo del algoritmo y la Figura 3.10 un ejemplo de una ejecución del algoritmo sobre un conjunto cualquiera de puntos [9].

```
1 Entrada: S (dataset), K (numero de clusters)
2 Salida: clusters
3    1: Inicializar K centroides
4    2: mientras no se cumpla la condicion de parada hacer:
5    3:     Asignar instancias al cluster cuyo centroide es ↩
        mas cercano
6    4:     Actualizar centroides con las nuevas ↩
        asignaciones
```

Listado 3.1: Algoritmo K-means

Algoritmo Expectation-Maximization (EM)

Pertenece a la categoría de métodos *density-based*. La idea es continuar ampliando un cluster mientras que la densidad (el número de puntos o datos) en el «vecindario» excede un umbral.

EM es una metodología iterativa que permite encontrar estimadores de máxima verosimilitud en modelos probabilísticos. Alterna entre un paso de «esperanza» (E-expectation), que crea una función de la «esperanza» de la función de verosimilitud evaluada usando la estimación actual del parámetro *k*, y un paso de maximización (M-maximization), que recalcula los parámetros *k* maximizando la verosimilitud esperada en el paso E.

Su mayor virtud es que es capaz de obtener clusters elípticos, llamados «gaussianos», en lugar de los esféricos estimados por el algoritmo k-means, siendo más versátil y adaptable a problemas de *clustering* complejos.

[9] `http://www.practicaldb.com/data-visualization-consulting/cluster-analysis/` Último acceso: 23 de agosto de 2014.

3.4 Support Vector Machine

Las Support Vector Machine (SVM) ([CS14] [Dur10]) pertenecen a la categoría de los clasificadores lineales. Calculan un hiperplano[10] de separación que equidista de los ejemplos más cercanos de cada clase (Figura 3.11) y consiguen lo que se denomina un margen máximo a ambos lados del hiperplano. Este criterio permite definir un hiperplano de separación como óptimo, cuando el margen τ, que es la mínima distancia entre dicho hiperplano y el ejemplo más cercano de cualquiera de las dos clases, es de tamaño máximo (Figura 3.12).

Los ejemplos de cada clase que se encuentran justo en la frontera se denominan vectores soporte.

Los datos lineales se pueden separar con un hiperplano. Sin embargo, asumir la linealidad puede ser un error en muchas ocasiones. En estos casos, los datos no se pueden separar en un espacio lineal y la clasificación falla. Para resolver esto los vectores de entrenamiento x_i son *mapeados* a un espacio dimensional mayor por una función *kernel*. En este nuevo espacio los datos sí son separables linealmente y SVM encuentra un hiperplano de margen máximo. La Figura 3.13 ilustra esta situación. El hiperplano de separación en el espacio transformado puede representar una frontera no lineal en el espacio original.

El objetivo de SVM es producir un modelo, basado en los datos de entrenamiento, que prediga si un dato nuevo pertenece a una categoría o a otra.

3.4.1 Funciones *kernel*

Una función *kernel* se puede interpretar como una especie de medida de similitud entre objetos de entrada. Aunque se han propuesto nuevos *kernels*, los cuatro básicos son:

- Lineal: $K(x_i, x_j) = x_i^T x_j$.
- Polinomial: $K(x_i, x_j) = (x_i^T x_j + r)^d, \gamma > 0$
- RBF: $K(x_i, x_j) = exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$
- Sigmoide: $K(x_i, x_j) = tanh(\gamma x_i^T x_j + r)$

Computacionalmente puede ser muy costoso transformar los datos, por eso se hace uso de lo que se conoce como «*kernel* trick» que hace innecesario calcular las coordenadas en el espacio explícitamente. Se puede encontrar más información sobre el «*kernel* trick» y de la base matemática subyacente en [Dur10], [Jor04] y [Hof06]

Escoger la función *kernel* correcta para un problema determinado es probablemente la parte más delicada de SVM. En la práctica, un *kernel* polinomial de grado pequeño o un *kernel* RBF es un buen comienzo para muchos problemas.

[10] Un hiperplano es un análogo de muchas dimensiones al plano de dos dimensiones en el espacio tridimensional. Separa el espacio en dos semiespacios.

3.5 El modelo Bag of Words

Hasta ahora se han presentado diversos conceptos de forma aislada, sin saber qué papel juegan ni la relación entre ellos. En esta sección se describe el modelo BoW que es la esencia de este PFC y relaciona y da sentido a todo lo expuesto anteriormente.

El modelo BoW fue propuesto originalmente en el campo del procesamiento del lenguaje natural, pero no sólo no se ha limitado a él sino que ha sido aplicado con éxito en otros muchos como el campo de la visión por computador, reconocimiento de imágenes o reconocimiento de patrones.

En primer lugar, se expone en líneas generales en qué consiste el modelo BoW para explicar a continuación cómo se ha adaptado en [SRDTB$^+$12] para el problema del análisis de EEG, donde se trabaja con la hipótesis de que la misma aproximación aplicada al procesamiento del lenguaje natural y visión por computador puede ser aplicada para el diagnóstico de epilepsia.

3.5.1 Descripción general

En el campo del procesamiento de lenguaje natural y búsqueda y recuperación de información, donde apareció por primera vez, el modelo BoW consiste en una representación simple de documentos, cada uno de los cuales está formado por muchas palabras y es descrito por un histograma de palabras, con la suma total de la aparición de cada palabra sin importar el orden en el que aparecen. En general, se trata de calcular un valor que represente un determinado atributo. Identificando el histograma de palabras se puede analizar fácilmente el contenido del documento. BoW permite un modelado basado en diccionario. Es conocido como una técnica potente para el análisis de documentos.

3.5.2 Adaptación del modelo al análisis EEG

En el contexto del análisis EEG, las palabras son «características» y cada documento es representado por tanto como un vector de características o *feature vector*. Las señales de los canales EEG juegan el papel de documentos mientras que los segmentos de la señal son las palabras del documento. Sin embargo, usar los segmentos enteros es una representación pobre de las palabras debido a la gran cantidad de información redundante que compone cada segmento. Para mejorar esa representación se hace uso del análisis Wavelet. Además de minimizar la cantidad de información requerida para caracterizar un segmento, magnifica los elementos que están relacionados con la presencia de actividad epileptiforme.

El proceso de generación del diccionario de palabras (se puede encontrar en la literatura como *codebook generation*) consiste en identificar las diferentes palabras que aparecen en un documento. Desde el punto de vista del diagnóstico de epilepsia, consiste en identificar los diferentes vocabularios que aparecen en los registros de los canales EEG. Además, pertenecen a *clusters* diferentes en los cuales los vectores de características EEG pueden ser

agrupados. Después de haber generado el vocabulario, el siguiente paso consiste en obtener el histograma que caracteriza la señal de los canales EEG haciendo *clustering* a los vectores de características con el número óptimo de *clusters* en los cuales estos datos pueden ser agrupados. Es decir, se medirá el número de ocurrencias que de cada palabra del vocabulario aparece en ese segmento de señal. Este número se obtuvo en [SRDTB+12] de forma empírica, como muestra la Figura 3.14, evaluando una función de optimización para un rango de 1 a 8 *clusters* y dando como resultado un número óptimo de *clusters* de 2.

Establecido el número de *clusters* y realizado el *clustering*, el histograma se obtiene midiendo la distancia de cada vector de características a cada *cluster*. El último paso consiste en entrenar el sistema con ejemplos de segmentos que correspondan a actividad normal y a actividad epileptiforme. Para el aprendizaje se usa un clasificador SVM.

La metodología seguida se resume en la Figura 3.15. Estos pasos se agrupan en dos procesos: uno dedicado al entrenamiento del sistema y otro dedicado a la clasificación de datos nuevos.

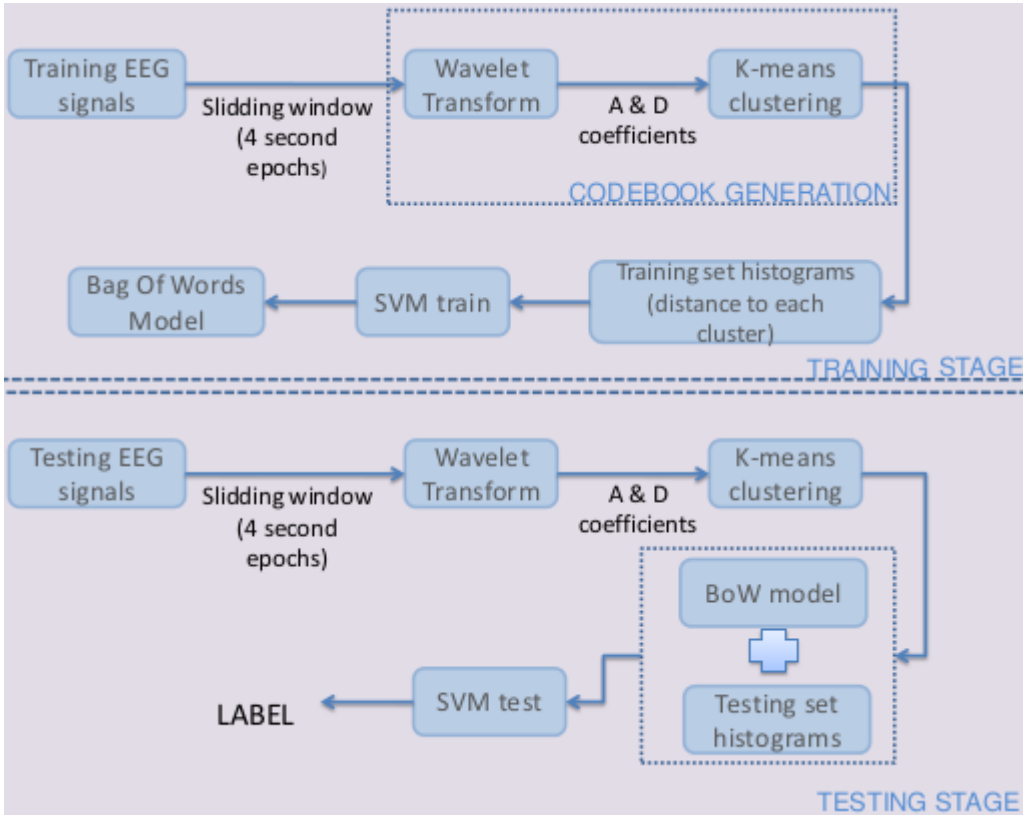

Figura 3.15: BoW para análisis EEG [SRDTB+12]

3.6 Trabajos previos

El diagnóstico de epilepsia es una tarea susceptible de ser automatizada y se han propuesto muchos trabajos buscando una aproximación a este objetivo. El primer trabajo conocido basado en el análisis EEG fue propuesto en [Got82] [Got90] y consiste en medir cuantitativamente la «novedad» de la señal EEG. Además, se lleva a cabo un análisis temporal continuo, comparando cada segmento de señal con un historial.

En [WSEG04], el algoritmo *Reveal* que se implementa también está basado en el análisis de EEG y en un sistema basado en reglas para identificar escenarios de potenciales ataques.

Los métodos que combinan el análisis EEG en el dominio del tiempo y en el de la frecuencia han demostrado mejorar la probabilidad de acierto en la detección de ataques frente a los que se centran en un solo dominio. En este sentido, la Wavelet Transfrom (Transformada Wavelet) (WT) supera las limitaciones de la FT [KGDA05] y se convierte en uno de los algoritmos de procesamiento de la señal para el análisis EEG más usado: [OAND10] [Jan10] [HR12] [OHO11] [PM08] [SE08] [FBFK11]. Como alternativa a la FT destaca también [TTF07], que usa una distribución *Wigner-Ville* para obtener la distribución de energía de una señal EEG.

Después de caracterizar la señal, una práctica común a todos es tomar una decisión para clasificar la señal EEG como sana o epiléptica. La entrada del clasificador consiste en un resumen de la señal, o un conjunto de características que la describan. Algunas de las formas utilizadas para extraer esta información son: a través de la distribución de la energía [OAND10] [OHO11] [PM08] [TTF07] [GKVL06]; con medidas estadísticas como la media, la desviación estándar, la varianza, etc. [Jan10] [PM08] [FBFK11] o regresión logística [SE08].

El tipo y el número de las características tiene un impacto directo en el comportamiento del sistema, por eso es necesario seleccionar la técnica más apropiada para maximizar sus resultados. Los algoritmos de clasificación necesitan que los datos estén etiquetados con la categoría a la que pertenecen y que se intenta predecir. Esto puede ser una desventaja respecto a los que se han mencionado al principio de la sección: [Got82] y [WSEG04].

En cuanto a las estrategias de clasificación, predominan dos tendencias en el análisis EEG para detección automática de ataques:

- Redes neuronales [OAND10] [HR12] [OHO11] [PM08] [TTF07] [BLZ08] [ST09].
- Clasificadores lineales [FBFK11] [YZLW12] como SVM [GKVL06] [GVSC03] [EWW+08] o *clustering k-means* [Jan10].

  Se han propuesto otros algoritmos de aprendizaje máquina como la programación genética [SFV07].

> Para una revisión completa de los algoritmos usados para diseñar sistemas con interfaces Cerebro-Computador basados en EEG se puede acudir a [LCL+07].

Por último, cabe mencionar el proyecto SMART (*Seizure Monitoring and Response Transducer*) llevado a cabo por estudiantes de la Universidad de Rice[11] que consiste en un cinturón que alerta sobre crisis epilépticas a través del teléfono móvil. En lugar de usar un EEG, miden la frecuencia de respiración y la conductancia eléctrica de la piel mediante unos electrodos. Cuando se sobrepasan unos valores predefinidos la información se envía mediante Buletooth a un teléfono móvil que lleva la persona y envía una alerta a quien corresponda.

[11] `news.rice.edu/2013/05/13/rice-has-help-for-people-with-epilepsy/`. Último acceso: 21 de agosto de 2014.

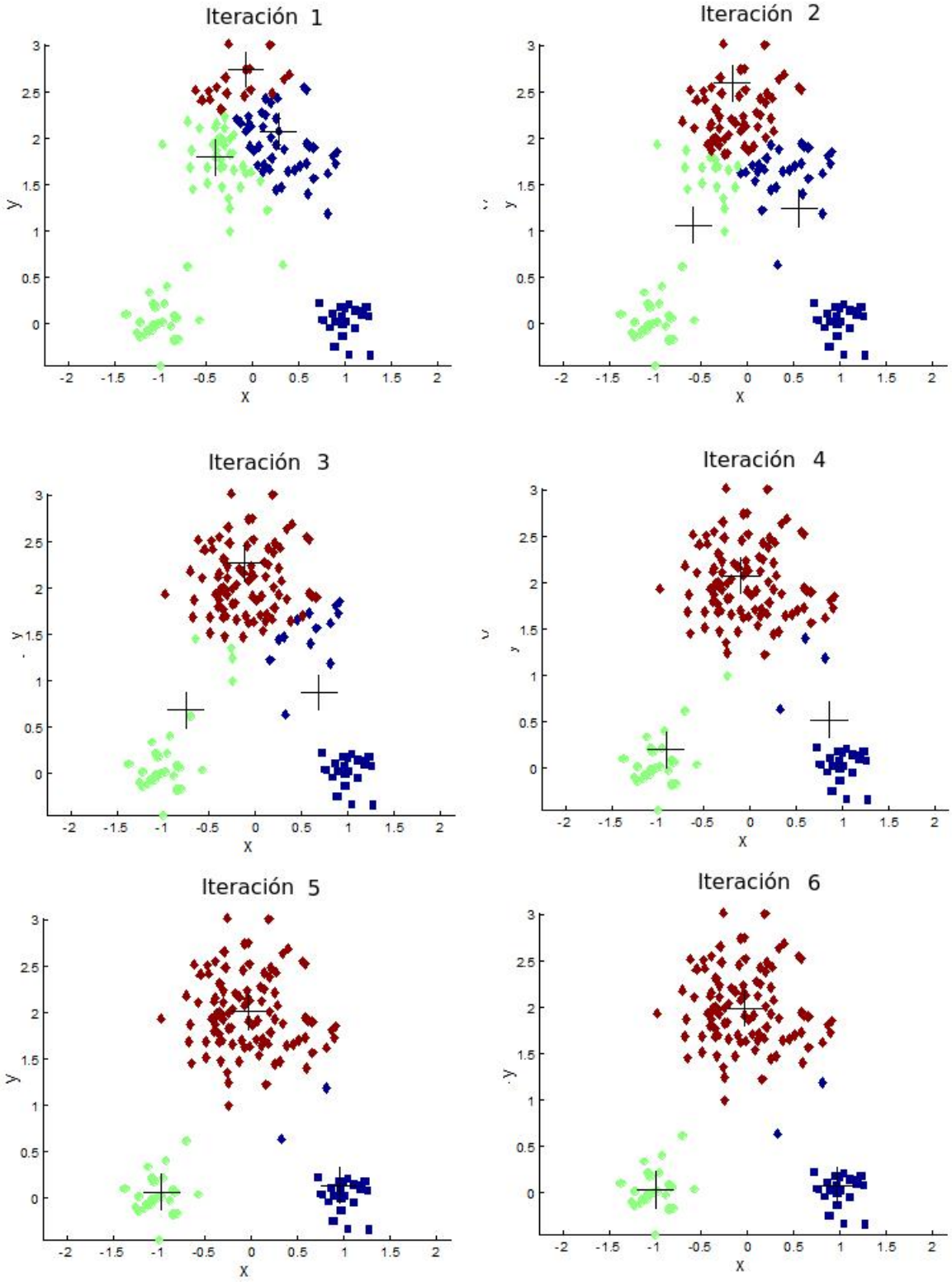

Figura 3.10: Ejemplo de ejecución del algoritmo k-means sobre un conjunto cualquiera de puntos

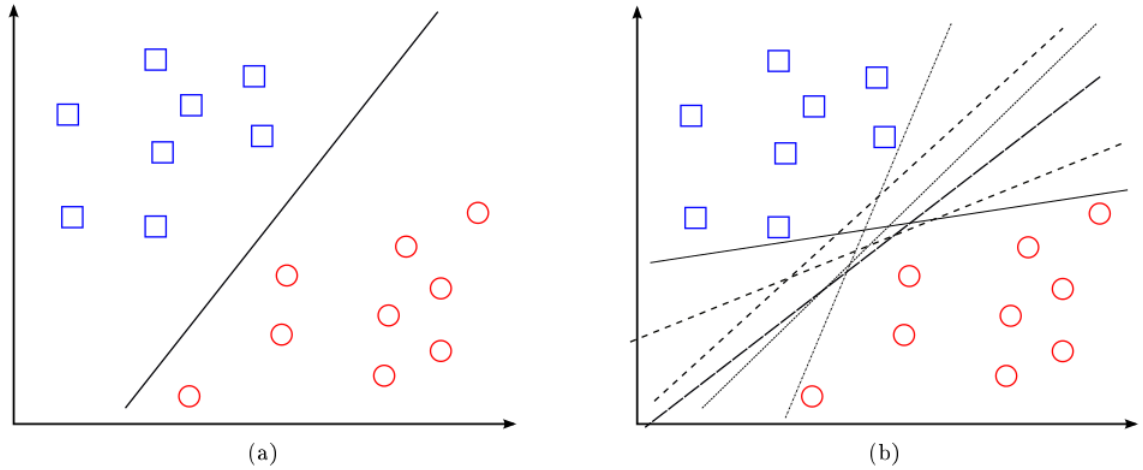

Figura 3.11: Hiperplanos de separación en un espacio bidimensional de un conjunto de ejemplos separables en dos clases: (a) ejemplo de hiperplano de separación (b) otros ejemplos de hiperplanos de separación (no óptimos), de los infinitos posibles [CS14]

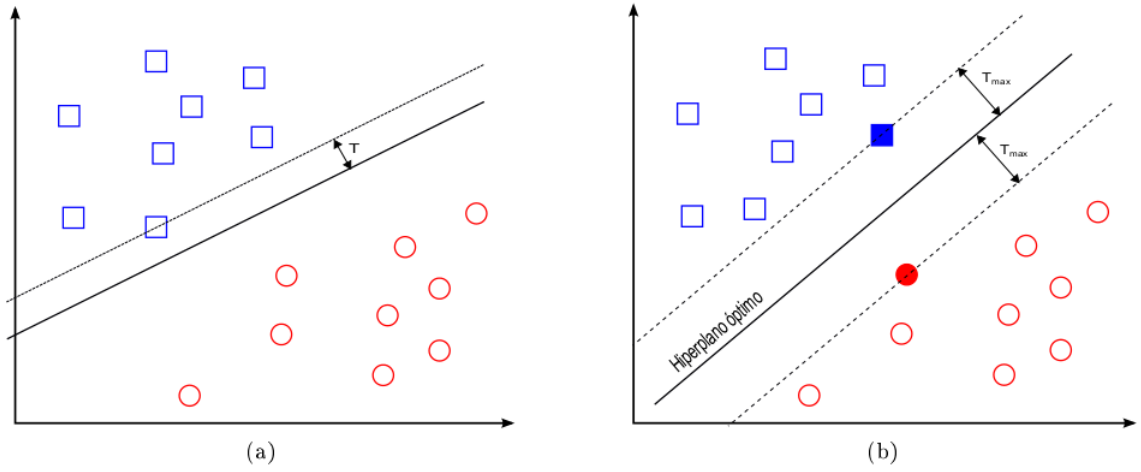

Figura 3.12: Margen de un hiperplanos de separación: (a) hiperplano de separación no óptimo y su margen asociado (b) hiperplano de separación óptimo y su margen (máximo) asociado. [CS14]

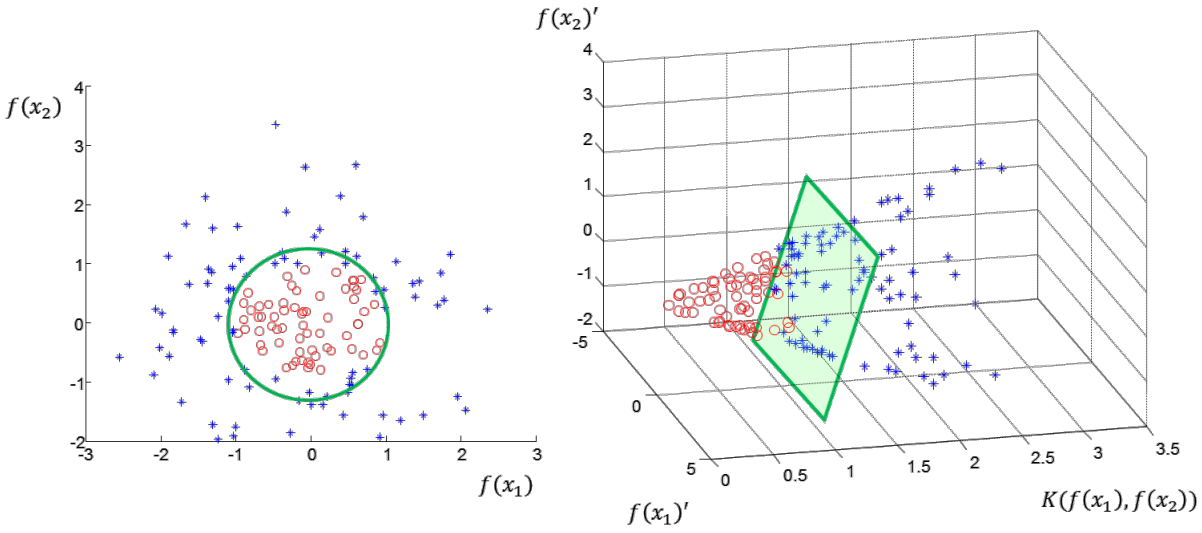

Figura 3.13: A la izquierda, datos que no se pueden separar linealmente. A la derecha, esos datos son «mapeados» a un espacio por un *kernel* no lineal [SRDTB+12]

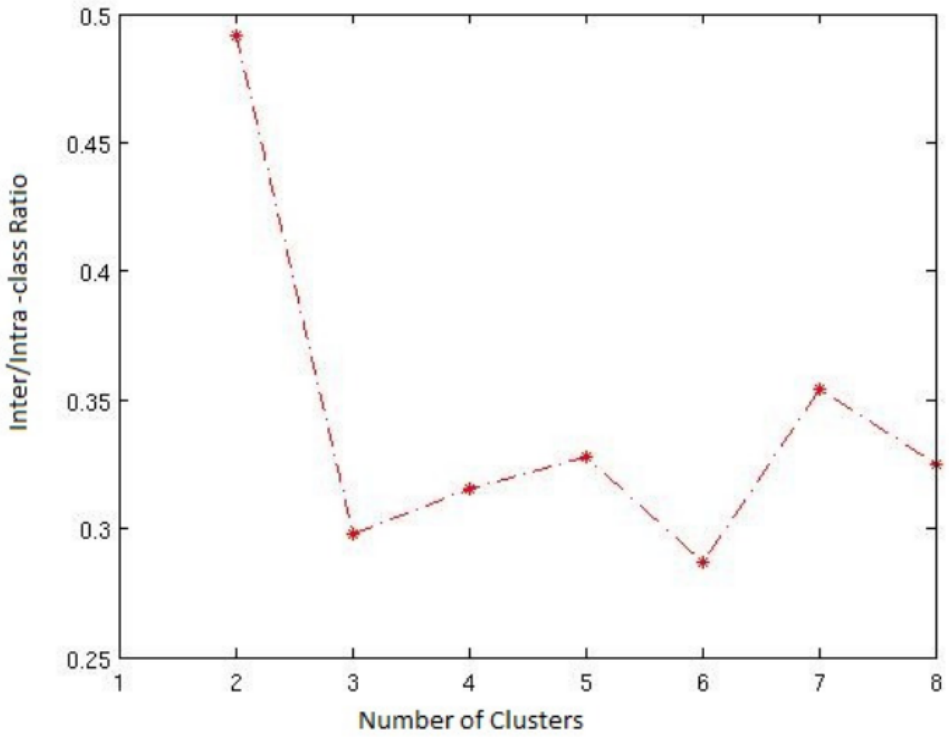

Figura 3.14: Función de optimización para determinar el número óptimo de *clusters*

Método de trabajo

4

A lo largo de este capítulo se explica cómo se ha resuelto el problema planteado en este PFC, describiendo la primera aproximación al problema; cómo se dividió el trabajo y la metodología de desarrollo utilizada. A su vez, se justifica cada una de las decisiones más relevantes adoptadas en cada momento. Al final del capítulo se listan las herramientas utilizadas, tanto hardware como software.

Este capítulo y el siguiente están estrechamente relacionados.

Una vez estudiado y comprendido el problema que se propone resolver, se dividió conceptualmente en tres bloques principales:

1. Implementación del algoritmo Bag Of Words.
   1.1. Training
   1.2. Testing
2. Captación y preparación de la señal proveniente del EEG.
3. Integración del sistema BoW sobre algún SCB como Raspberry Pi.

Para abordar los retos planteados por cada uno de estos bloques funcionales, se requería:

- Un estudio teórico sobre cada una de las herramientas matemáticas utilizadas. El objetivo no era adquirir un conocimiento profundo (lo que en la mayoría de los casos excedía los límites del PFC), sino un conocimiento suficiente para comprender el papel de tal herramienta.
- Elección de una librería en C++ atendiendo a criterios como facilidad de uso, rendimiento y documentación adecuada.
- Usar la librería elegida para obtener la funcionalidad esperada de cada módulo.

Estos tres puntos son transversales a prácticamente todo el desarrollo.

4.1 Metodología

Debido a la considerable complejidad que presenta un algoritmo de estas características, la práctica recomienda seguir una proceso de desarrollo iterativo e incremental, de forma que

se divida todo el trabajo en unidades más pequeñas.

4.1.1 Scrum

Se ha optado por una metodología de desarrollo ágil [Poo09] para minimizar los riesgos desarrollando software en periodos de tiempo cortos, y en particular, por Scrum, que emplea el enfoque iterativo e incremental buscado. Es idónea en trabajos donde el problema no puede ser entendido completamente desde el inicio y se adapta en la aparición de nuevos requisitos.

Hay que tener en cuenta que es una metodología ágil para la gestión de proyectos, «por lo que queda fuera de su ámbito el tratamiento explícito de, por ejemplo, el testing, el control de versiones, el diseño, arquitectura, etc.» [1]

Se ha buscado un equilibrio entre el compromiso que supone seguir una metodología y la flexibilidad requerida durante el desarrollo del proyecto. Por ser el PFC un trabajo personal, presenta importantes diferencias respecto a un proyecto desarrollado en el marco de una empresa (generalmente en equipo). Por eso, se han adaptado cuantos elementos han sido necesarios, bajo el criterio del autor y de la directora.

Se ha escrito mucho sobre Scrum en los últimos tiempos y existen multitud de manuales y guías para aprender cómo funciona y cómo aplicarlo. Por eso se ofrece a continuación simplemente una breve introducción a los conceptos más importantes extraídos de [yJS13]. Se remite al lector a la abundante bibliografía sobre el tema para más información [MDD09] [PAW13] [Kni07].

El Equipo Scrum (Scrum Team)

En los Equipos Scrum se asignan una serie de roles y responsabilidades. Los más importantes se listan a continuación:

El Equipo de Desarrollo (Development Team)

Los que desempeñan el trabajo de entregar un incremento de producto. El tamaño óptimo es lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para completar una cantidad de trabajo significativa.

El Dueño del Producto (Product Owner)

Es el responsable de maximizar el valor de producto y del trabajo del Equipo de Desarrollo. Las decisiones del Dueño del Producto se reflejan en el contenido y en la priorización de la Lista del Producto (Product Backlog), de la que es el único responsable.

El Scrum Master

Es el responsable de que el trabajo se ajuste a la teoría, prácticas y reglas de Scrum y de que el proyecto avanza según los previsto.

[1] Javier Garzás. Scrum para Dummies. `http://www.javiergarzas.com/2012/05/scrum-dummies-1.html`. Último acceso: 4 de julio de 2014.

Eventos de Scrum

Todos los eventos son bloques de tiempo con una duración máxima:

El Sprint

Es el corazón de Scrum. Abarca un periodo de tiempo de dos a cuatro semanas. Acaba con un incremento de producto, utilizable y potencialmente desplegable. Es un contenedor del resto de eventos. Durante el Sprint, el alcance puede ser clarificado y renegociado entre el Dueño del Producto y el Equipo de Desarrollo a medida que se va aprendiendo más. Cada Sprint tiene una definición de qué se va a construir, un diseño y un plan flexible que guiará la construcción.

Reunión de Planificación de Sprint

Se planifica el trabajo a realizar durante el Sprint y se lleva a cabo de forma conjunta por todo el equipo. Responde a las siguientes preguntas:

- ¿Qué puede entregarse en el incremento resultante del Sprint que comienza?
- ¿Cómo se conseguirá hacer el trabajo necesario para entregar el incremento?

El número de elementos de la Lista de Producto seleccionados para el Sprint depende únicamente del Equipo de Desarrollo. También decide cómo se construirá esta funcionalidad para formar un incremento. Como resultado se obtiene la Lista de Pendientes del Sprint (Sprint Backlog).

Objetivo del Sprint

Es la meta establecida para el Sprint en la Reunión de Planificación de Sprint.

Scrum Diario

Reunión diaria breve para evaluar el progreso hacia el Objetivo del sprint. El Scrum diario optimiza las posibilidades de que el Equipo de Desarrollo cumpla el Objetivo del Sprint.

Revisión de Sprint

Tiene lugar al final del Sprint para inspeccionar el incremento. El Equipo de Desarrollo hablará acerca de qué fue bien durante el Sprint, qué problemas aparecieron y cómo fueron resueltos.

Artefactos de Scrum

Pretenden maximizar la transparencia de la información clave:

Lista de Producto (Product Backlog)

Es una lista ordenada de todo lo que podría ser necesario en el producto, y es la única fuente de requisitos. Es dinámica; cambia constantemente para identificar lo que el producto necesita. El desarrollo más temprano de la misma solo refleja los requisitos conocidos y mejor entendidos al principio.

Lista de Pendientes del Sprint (Sprint Backlog)

Es el conjunto de elementos de la Lista de Producto seleccionados para el Sprint más un plan para entregar el incremento del producto.

Figura 4.1: Vista general de Scrum. `www.scrumalliance.org`

4.2 Evolución

Aquí se va a presentar una visión lo más exacta posible del desarrollo. Como es lógico, no se van a mostrar todos los artefactos ni el estado de los elementos durante todo el ciclo de vida, sino que se ofrecerá una instantánea del estado del desarrollo en determinados momentos. La propia naturaleza de las metodologías ágiles hacen imposible documentar todo el proceso llevado a cabo como si de un hilo temporal continuo se tratase. Además, el objetivo de este capítulo es dar una idea al lector de cómo evolucionó el desarrollo sin sumergirlo en un mar de páginas que acabaría ignorando. A continuación, se muestran los Sprints llevados a cabo y lo más destacable de cada una de las tareas. Como ya se ha comentado, los artefactos se han modificado para adaptarlos a las peculiaridades de un PFC y no se ajustan a la perfección a lo que dice la teoría.

4.2.1 Fase Pregame y Product Backlog

Se define el sistema que se construirá. En este estado tan temprano del desarrollo, los requisitos conocidos y entendidos coinciden prácticamente con los módulos mostrados en la Figura 3.15, que también proporciona el diseño de alto nivel, por lo que no se dedica mucho más esfuerzo a esta cuestión. Tema aparte es la definición de la arquitectura, que se fue perfilando a lo largo de cada sprint, con el diseño de cada tarea y con cada trozo de código escrito; por esta razón no existe ninguna tarea específica para ello, ya que es inseparable de cada una de las tareas que se describirán a continuación. Como ya se ha comentado, esta lista se va actualizando constantemente y el Cuadro 4.1 muestra una instantánea de los requisitos en el inicio. En él se estiman los días y se establece una prioridad para establecer únicamente qué tareas deben de ir obligatoriamente después de haber realizado otras. En los cuadros de

los *Sprint Backlogs* se vuelve a hacer una estimación de los días y se indica a qué requisito pertenece cada tarea.

4.2.2 Compra del hardware EEG

Entre los meses de febrero y marzo de 2013, cuando se empezó a perfilar este PFC, se procedió a la compra del equipo EEG por parte del grupo Arquitectura y Redes de Computadores (ARCO). No se ha considerado como un Sprint la búsqueda y la elección ya que por aquel entonces no se había establecido ni la metodología a seguir, ni requisitos. Era, simplemente, la compra de un material necesario para la realización del PFC.

Se pidió presupuesto de tres productos, cuyas características se detallan a continuación:

- *actiChamp*. Puede ser usado con 32, 64, 96, 128 y 160 canales con una tasa de muestreo de hasta 100 KHz.
- *TREA*. Equipo EEG ambulatorio de 25 canales. Con una frecuencia de muestreo de 200 Hz a 1600 Hz tiene una autonomía de 72 horas.
- *Emotiv*. Sistema EEG inalámbrico con 14 canales y una frecuencia de muestreo de 128 Hz y una autonomía de 12 horas.

En el anexo B se muestra el presupuesto de los distintos equipos.

El producto elegido fue el EEG Emotiv (Figura 4.2). Este equipo cuenta con un precio que se ajusta al término «bajo coste» si se compara con el resto de alternativas. Además, tiene un tamaño reducido, perfecto para un sistema ambulatorio y cuenta con una licencia para investigación con pleno acceso a los datos «en crudo» de los sensores.

Figura 4.2: EEG Emotiv. El *headset* se comunica a través de un sistema bluetooth con el ordenador

ID	Requisito	Prioridad	Días estimados
1	Ventana deslizante	1	1
2	Leer señal de fichero	1	3
3	Búsqueda de librerías DWT	1	1
4	Caracterización señal	2	1
5	Integración 1, 2, 3	3	2
6	Búsqueda de librerías para Clustering	1	2
7	Clustering	2	5
8	Hacer histograma	3	2
9	Búsqueda de librerías para SVM	1	1
10	Clasificador	2	4
11	Acceder a los datos del headset	2	5
12	Integración con un SCB	4	5

Cuadro 4.1: Product Backlog

ID	Tarea	Días	Requisito (ID)
1	Búsqueda de librerías DWT	1	3
2	Prueba de la librería	2	3
3	Ventana deslizante	3	1
4	Leer señal de fichero	2	2
5	Caracterización señal	2	4
6	Integración 2,3 y 4	2	5
7	Bug cálculo DWT	4	4

Cuadro 4.2: Sprint Backlog 1

4.2.3 Sprint 1

ID.1: Búsqueda de librerías DWT. 26 de diciembre

Después de la búsqueda se cuenta con los siguientes candidatos:

- Wavelets Library [SJN07]
- Wavelet1d [Hus11]
- GSL - GNU Scientific Library [Fou13]

ID.2: Prueba de la librería. 2-5 de enero

Se prueban las librerías seleccionadas. Viendo la documentación disponible, la facilidad de instalación y la facilidad de uso de la API se escoge Wavelets Library [SJN07]. Se implementa un primer ejemplo sencillo y se tiene que los coeficientes obtenidos como resultado de aplicar la DWT se escriben en ficheros. A dichos coeficientes habrá que seguir aplicando transformaciones en el resto de los módulos, por lo que hay dos opciones:

1. Modificar la librería para que devuelva los resultados en forma de vectores o cualquier otra estructura de datos útil.
2. Leer los ficheros inmediatamente después de que sean generados por la librería y almacenarlos en vectores o cualquier otra estructura de datos útil.

Se implementa la segunda opción.

En cuanto a la DWT, hay que destacar que se utiliza una Daubechies con nivel 5 de descomposición. Si la Figura 3.8 mostraba gráficamente el proceso general de descomposición, la Figura 4.3 muestra cómo se realiza la descomposición para este caso en concreto.

ID.1: Ventana deslizante. 6,7 y 8 de enero.

La ventana deslizante es una abstracción por la cual se obtienen ventanas o fragmentos de la señal de tamaño s con un solapamiento entre ellas de o (Ver Figura 4.4). Las ventanas se obtienen mientras se recorre de forma continua en el tiempo la señal. Las Figura 4.5 y 4.6 muestran el diseño realizado.

ID.5: Integración 2, 3 y 4. 12 y 13 de enero

Con un pequeño ejemplo se prueba en conjunto la lectura de la señal desde un fichero (por ventanas) y el cálculo de la DWT de cada ventana.

ID.5: Caracterización de la señal. 15 de enero

Al aplicar la DWT se obtiene una gran cantidad de valores correspondientes a los coeficientes de cada nivel en los que se descompone la señal. Por esta razón, la señal se «caracteriza», es decir, se resume y se reduce seleccionando un número pequeño de valores que sean representativos. Estos valores son los siguientes:

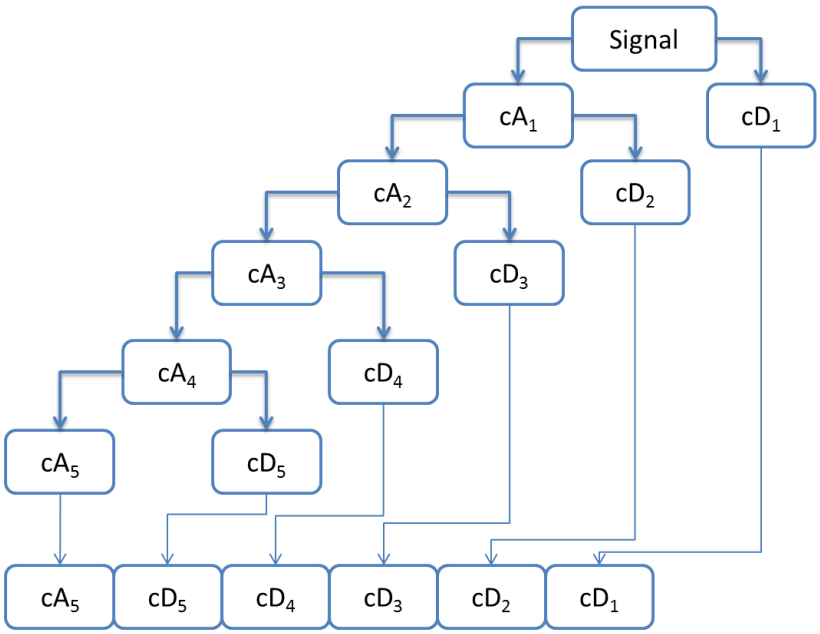

Figura 4.3: DWT de 5 niveles

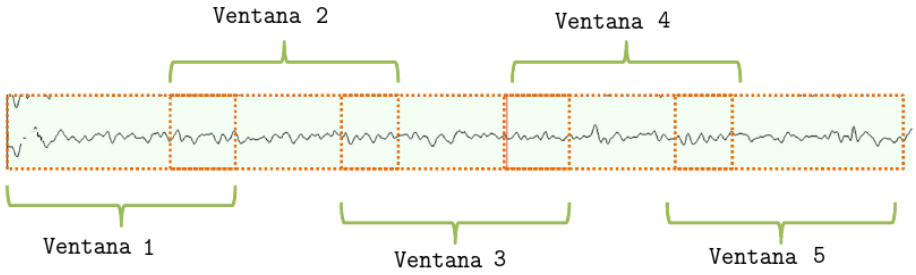

Figura 4.4: Segmentación de la señal en ventanas de tamaño *s* y un *overlap o*

- El máximo.
- El mínimo.
- La media.
- La desviación estándar.

Así, se obtienen estos cuatro valores de cada uno de los coeficientes que resultan de cada nivel de descomposición; en total 24 valores (cuatro medidas estadísticas por cada uno de los seis coeficientes). En la Figura 4.7 se puede ver gráficamente.

Hay que resaltar que este proceso se realiza por cada ventana.

4.2.4 Sprint 2

ID.1: Búsqueda de una librería para clustering. 25 de enero.

Se barajaron dos posibilidades

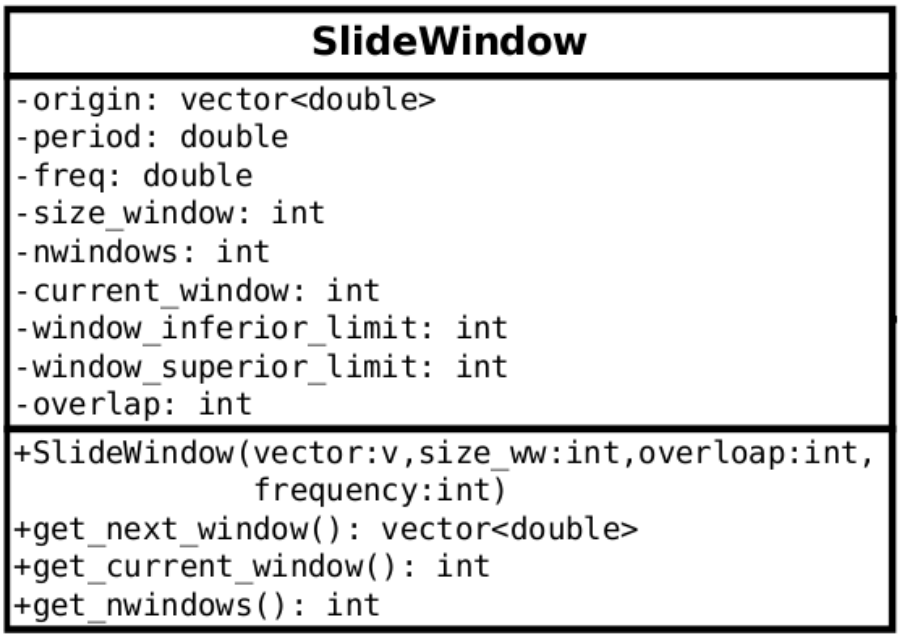

Figura 4.5: Diseño de la ventana deslizante

- OpenCV [Bra00]
  - BSD license.
  - Para visión artificial, implementa más de 2000 algoritmos y abarca una multitud de áreas.
  - 1999.
  - De uso extendido y con buena documentación.
- mlpack [CCS+13]
  - GNU LGPL version 3 license.
  - Librería para *machine learning* con énfasis en la escalabilidad, velocidad y facilidad de uso.
  - 2013.
  - Buena documentación oficial.

ID	Tarea	Días	Requisito
1	Búsqueda librerías para clustering	2	6
2	Instalación y pruebas	2	6
3	Clustering	5	7
4	Hacer histograma	1	8

Cuadro 4.3: Sprint Backlog 2

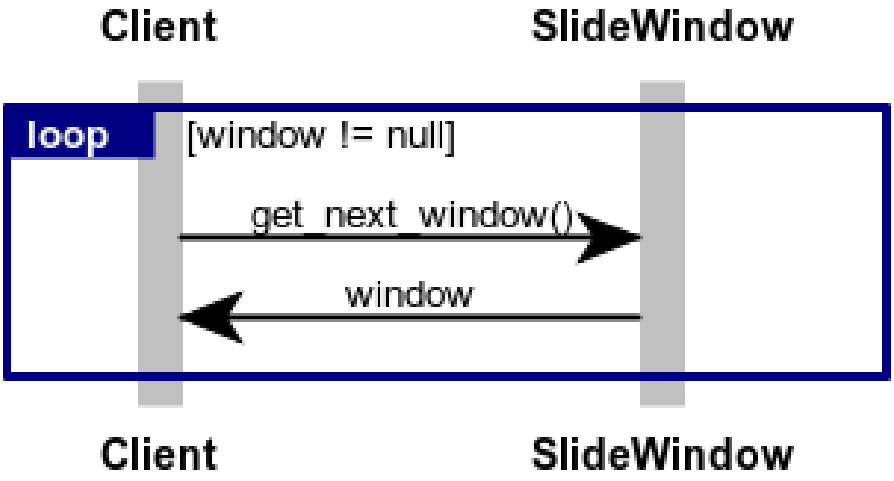

Figura 4.6: Diagrama de secuencia de la Ventana Deslizante

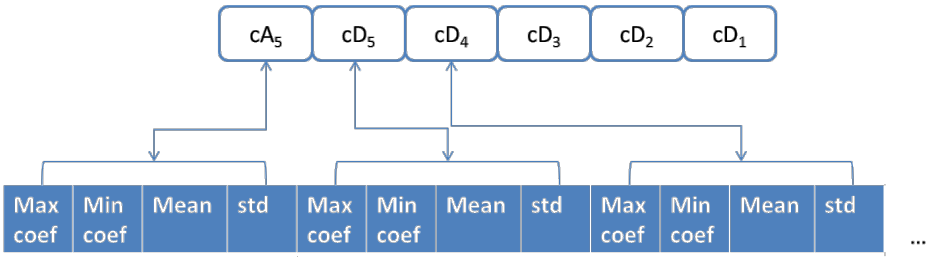

Figura 4.7: Caracterización de la señal

Eligiendo la librería *mlpack* por ser únicamente una librería de *machine learning* y no tan general como OpenCV.

ID.2: Instalación y pruebas. 3 y 4 de febrero.

Después de resolver todas las dependencias e instalar la librería se comprueba que esté instalada correctamente en el sistema. Para comprobarlo de una manera fácil se procedió con un ejemplo sencillo. Se utilizaron los datos representados en la Figura 4.8 de forma que a simple vista se supiese si la agrupación en *clusters* era correcta. Al ejecutar en un terminal:

```
2   \$ kmeans -c 3 -i data.csv -v -o assignments.csv -C ↩
      centroids.csv
```

Se obtiene en los ficheros *assignments.csv* y *centroids.csv* el número del *cluster* al que pertenece cada punto de *data.csv* y las coordenadas de los centroides, respectivamente. En la Figura 4.9 se muestra una representación de estos resultados. El punto rojo es el centroide de cada *cluster*.

Con este resultado se puede decir que la instalación y la prueba han sido satisfactorias.

Llegados a este punto, surgen nuevas necesidades y preguntas:

- ¿Realizar el clustering utilizando el ejecutable o utilizando la API? mlpack provee varios ejecutables para que los métodos puedan ser usados sin ningún conocimiento de C++ y también una interfaz simple para usar mlpack en C++.
- ¿Cómo preparar los datos para ser cargados? mlpack usa *Armadillo*[2] ([San10]) para trabajar con matrices, por tanto, hay que trabajar con las estructuras de datos adecuadas. Mientras que en el ejemplo anterior basta con indicar el archivo donde se encuentran los datos, ahora será necesario prepararlos de forma que se puedan interpretar internamente por la librería.

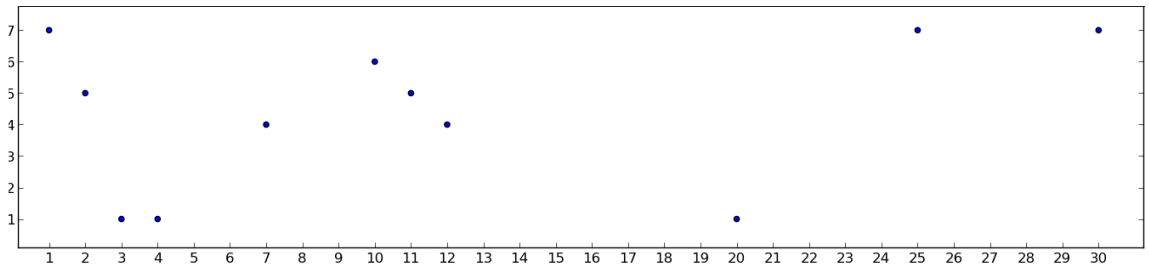

Figura 4.8: 11 puntos en el espacio

ID.3: Clustering. 25 de enero - 22 de febrero

Una vez hecha una prueba básica, fue necesario profundizar en la API así como en la teoría expuesta en el Capítulo 3 para realizar el módulo de *clustering*.

Qué se agrupa

Los datos de entrada serán los vectores de características o *feature vector* a partir de ahora (ver Figura 4.7). En lugar de tener puntos en dos dimensiones como en el ejemplo de la Figura 4.8, los puntos son ahora de 24 dimensiones, una por cada variable, y habrá tantos puntos como ventanas en las que se segmente la señal.

[2] Armadillo es una librería en C++ que usa técnicas avanzadas para operar con matrices de la forma más rápida posible. La documentación de Armadillo se puede encontrar en `http://arma.sourceforge.net/docs.html`.

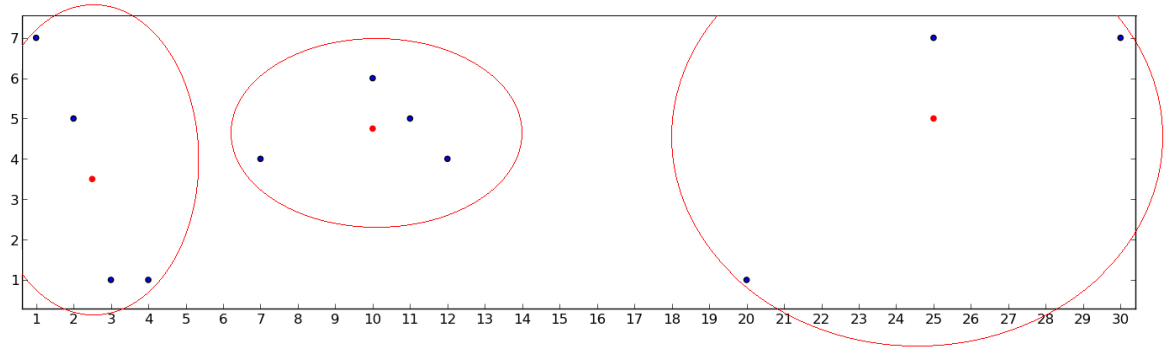

Figura 4.9: Agrupamiento en 3 *clusters* de 11 puntos

Cómo se agrupa

Una de las cosas más importantes es la elección del número de *clusters*. Por los resultados obtenidos en [SRDTB+12] se tiene que el número óptimo es dos. Cogiendo estos resultados, en el sistema que se está construyendo el número fijo de *clusters* será también de dos.

Algoritmo

En un principio se decide usar un algoritmo *k-means*, dejando abierta la posibilidad de usar posteriormente un algoritmo EM. *K-means*, como ya se expuso en el Capítulo 3, es una solución buena para la gran mayoría de los problemas y es más sencilla que el algoritmo EM, con un trasfondo matemático que se sale de los límites del PFC.

A lo largo de esta tarea se dan respuesta a las preguntas surgidas al final de la tarea anterior: se decide usar la interfaz que proporciona *mlpack* para usar los métodos como *k-means* y así tener un mayor control del flujo de datos. Hay que tener en cuenta que los resultados que esta etapa produzca serán utilizados por la siguiente. También se crea una estructura de datos consistente en una matriz de tipo *Armadillo* para almacenar los *feature vector* de 24 dimensiones o variables.

ID.4: Hacer histograma. 23 y 24 de febrero.

Siguiendo el modelo BoW, es necesaria la generación del conjunto de *palabras* que pueden aparecer o *diccionario*. Aunque se ha obtenido un conjunto de *feature vectors* desde la señal EEG, esto no puede ser considerado todavía como el equivalente a palabras en un documento. El objetivo de este paso consiste precisamente en redefinir el *feature vector* como una función del vocabulario. El vocabulario del sistema BoW será el número óptimo de *clusters* establecidos previamente en dos, identificados por sus centroides (ver Figura 4.10).

Este proceso genera un nuevo descriptor, a partir de la distancia de cada *feature vector* a cada uno de los centroides.

En (4.1) se puede ver el histograma generado, donde $d_{m,[1,2]}$ es el resultado de una función

distancia entre fv_m y $Ck_{[1,2]}$

$$Hist_{m,2} = \begin{pmatrix} d_{1,1} & d_{1,2} \\ d_{2,1} & d_{2,2} \\ \vdots & \vdots \\ d_{m,1} & d_{m,2} \end{pmatrix} \tag{4.1}$$

$$FV_{m,24} = \begin{pmatrix} fv_{1,1} & fv_{1,2} & \cdots & fv_{1,24} \\ fv_{2,1} & fv_{2,2} & \cdots & fv_{2,24} \\ \vdots & \vdots & \ddots & \vdots \\ fva_{m,1} & fv_{m,2} & \cdots & fv_{m,24} \end{pmatrix} \begin{pmatrix} Ck_{1,1} & fv_{1,2} & \cdots & fv_{1,24} \\ Ck_{2,1} & fv_{2,2} & \cdots & fv_{2,24} \end{pmatrix} = Ck_{2,24}$$

Figura 4.10: Generación del histograma

4.2.5 Sprint 3

ID	Tarea	Días	Requisito
1	Búsqueda librerías para SVM	1	9
2	Instalación y pruebas	1	9
3	Clasificador	5	10
4	Formatear datos	2	10

Cuadro 4.4: Sprint Backlog 3

ID.1: Búsqueda librerías para SVM. 25 de febrero

Se barajaron dos posibilidades que se encontraban en el *Top 3* de la web `www.svms.org`: libSVM [CL11] y SVMLight [Joa98]. Ya que ambas tenían características similares y buena documentación, no había muchos más elementos para tomar una decisión y finalmente se optó por la primera de ellas.

ID.2: Instalación y prueba. 28 de febrero.

Una vez descargado y compilado el software, puede ser utilizado a partir de los ejecutables o haciendo uso de la interfaz en C++. Conviene realizar unas pruebas antes con los ejecutables para familiarizarse con el entorno y el formato específico de los datos que impone libSVM:

```
1  <etiqueta> <indice_1>:<valor> ... <indice_n><valor>
```

Cada línea contiene una instancia y termina con un carácter de fin de línea. <label>es un número entero que indica la etiqueta de la clase. El par <indice_n>:<valor>indica cada atributo: <indice_n>es un número entero que empieza en 1 y <value>es un número real. Si los datos son para *testing* el campo <label>se usa solo para calcular la precisión, y si se desconoce se puede rellenar con cualquier valor. El Listado 4.1 muestra un ejemplo de un fichero de texto plano de datos con el formato correcto.

```
+1 1:0.7 2:1 3:1 4:-0.3 5:-0.1
-1 1:0.5 2:-1 3:0 4:-0.6 5:1 6:-1 7:1
+1 1:0.16:1 3:-0.33 4:-0.49 5:-0.38 6:-1

...

-1 1:0.45 2:1 3:1 4:-0.3 5:-0.3() 6:-1 7:-1
```

Listado 4.1: Ejemplo del formato de los datos

ID.2: Clasificador. 15-16 de marzo

Al igual que ocurrió con la fase de *clustering* y la librería *mlpack*, en esta tarea se diseña y se construye un clasificador haciendo uso de la librería libSVM y la interfaz que provee.

Como se contó en la introducción del capítulo, la división lógica de la implementación del algoritmo *BoW* es entre la parte de *training* y *testing*. Aquí es donde esa división se hace efectiva por primera vez. Así, este módulo, será el encargado de llevar a cabo estas tareas, como se ve de forma muy sencilla en la Figura 4.11.

Figura 4.11: Clase Classifier

Posteriormente, cuando se integren todos los módulos usando como guía la arquitectura, se irá definiendo claramente esa división, esas dos partes que conforman el sistema.

En este punto se produce una modificación sustancial del Sprint Backlog 3 (ver Cuadro 4.4) debido a los nuevos requisitos que aparecen como consecuencia del avance en el desarrollo. El nuevo Sprint Backlog se puede ver en el Cuadro 4.5.

ID.2: Clasificador:training y Clasificador:testing. 30 de marzo - 23 de abril.

Estas dos tareas, implementadas como dos métodos, se encargan de obtener un modelo entrenando con los datos del histograma de la etapa anterior, y de realizar una predicción

ID	Tarea	Días	Requisito
1	Búsqueda librerías para SVM	1	9
2	Instalación y pruebas	1	9
3	Clasificador:Training	5	10
4	Clasificador:Testing	3	10
5	Formatear datos	5	3

Cuadro 4.5: Sprint Backlog 4, segunda versión

en base a ese modelo, respectivamente. El modelo creado por el software libSVM tiene que ser almacenado (es un fichero de texto plano) pues tiene que ser cargado en una ejecución diferente. Es decir, esto impone que antes de usar el método que hace *testing*, previamente se haya obtenido un modelo. Sin embargo, para poder hacer uso de la interfaz de este software, requiere que los datos con los que se va a realizar el entrenamiento estén en un formato determinado, al igual que debían estarlo los datos si se usaba el ejecutable directamente. La etapa de entrenamiento solo tendrá que realizarse una vez. Como resultado se obtendrá el modelo de clasificación que podrá utilizarse en las sucesivas etapas de clasificación.

Se detalla a continuación cómo se transforman los datos.

ID.4: Formatear datos. 30 de marzo - 23 de abril.

En el archivo de cabecera *svm.h* se declaran las funciones y las estructuras que se necesitan. Como ya se ha dicho, antes de clasificar los datos de *testing* se necesita construir un modelo SVM usando datos de *training* y la función que se muestra en el Listado 4.2

```
1  struct svm_model *svm_train(
2          const struct svm_problem *prob,
3          const struct svm_parameter *param
4                                  );
```

Listado 4.2: Función svm-train

El segundo argumento es una estructura con todos los parámetros necesarios para construir el clasificador. Para comprender mejor cada uno de ellos, se puede acudir al Capítulo 3 y consultar la documentación de la librería. En el Listado 4.3 se puede ver un extracto de la clase *Classifier.cpp* que corresponde al método *stablish_parameters*, el cual establece unos parámetros por defecto utilizados durante el entrenamiento. Son los mismos valores por defecto que aporta la librería y pueden funcionar razonablemente bien para la mayoría de los problemas. Sin embargo, después de los primeros intentos fallidos de clasificación se ajustó el parámetro γ (*gamma*) a 0.5. Este es uno de los más críticos y es dependiente de la estructura particular de los datos. Una forma aceptada para calcularlo es $\frac{1}{num\_atributos}$. Ya se ha visto que el número de atributos de cada vector de entrenamiento es de dos.

Esto refleja la importancia de tener una comprensión teórica suficiente de todas las herramientas, como ya se ha insistido a lo largo de este documento.

```
void Classifier::stablish_parameters()
{
    /* Use default values of library example*/
    param.svm_type = C_SVC;
    param.kernel_type = RBF;
    param.degree = 3;
    param.gamma = 0.5;  // 1/num_features
    param.coef0 = 0;
    param.nu = 0.5;
    param.cache_size = 100;
    param.C = 1;
    param.eps = 1e-3;
    param.p = 0.1;
    param.shrinking = 1;
    param.probability = 0;
    param.nr_weight = 0;
    param.weight_label = NULL;
    param.weight = NULL;

}
```

Listado 4.3: Parámetros por defecto para el clasificador

El primer argumento es el que tiene más interés. Es la estructura que describe el problema. En el Listado 4.4 se puede ver la forma que tiene. La variable «l» es el número de datos de entrenamiento. «y» es un array con las etiquetas de cada vector de entrenamiento y «x» es un array de punteros, cada uno de los cuales apunta a una representación dispersa de un vector de entrenamiento, representado como un array de svm node (ver Listado 4.5).

Hay que recordar que los datos a esta etapa llegan en forma de matriz (el histograma generado por la etapa de *clustering*) y habrá que transformar esos datos conociendo estas estructuras. Para ello se construye el método *create_problem*. La Figura 4.12 es una representación de la estructura *svm_problem* cuando se va rellenando al ejecutar dicho método.

```
struct svm_problem
{
        int l;
        double *y;
        struct svm_node **x;
};
```

Listado 4.4: Estructura svm problem que describe un problema

4.2.6 Sprint 4

Este Sprint consistió en la unión de todos los módulos desarrollados. Hasta aquí, cada uno de ellos se hacía funcionar de forma independiente, dedicando a ello un esfuerzo pequeño.

```
1  struct svm_node
2  {
3          int index;
4          double value;
5  };
```

Listado 4.5: Estructura svm node que describe un nodo o atributo

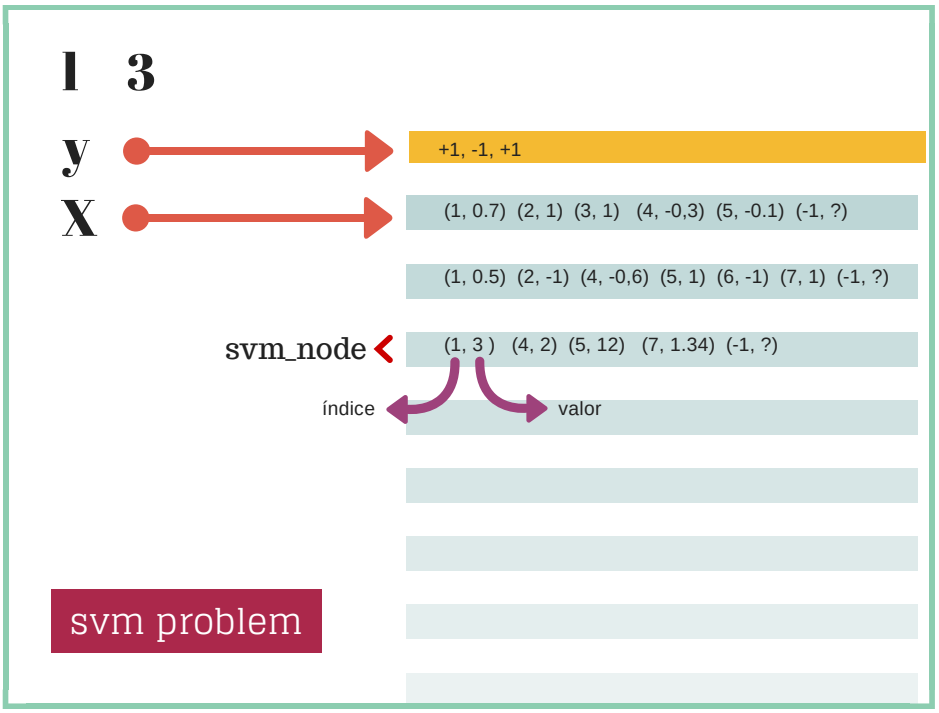

Figura 4.12: La estructura *svm_problem* contiene la variable «l» con el total de vectores de entranamiento; la variable «y» que apunta a un array con las etiquetas de cada vector de entrenamiento y la variable «x» que es un array de punteros, cada uno de los cuales es un array de *svm_node* (líneas verdes) que representan los vectores de entrenamiento. Como se puede ver, cada vector de entrenamiento termina con el índice -1.

Es una idea similar al prototipado evolutivo, donde el prototipo desechable se realiza sin consumir muchos recursos. Es decir, hasta ahora cada módulo funcionaba con su propio programa principal (*main*).

Es el Sprint más relacionado con la arquitectura, que se detallará en el Capítulo 5.

Como ya se ha comentado, este Sprint no se trata del diseño y construcción de la arquitectura del sistema, ya que es algo presente durante todo el desarrollo, y si bien los módulos eran independientes unos de otros, su diseño e implementación ya estaban orientados y guiados por la arquitectura.

También se definió la interfaz del sistema para el *training* y el *testing*. Así, el programa principal puede escribirse en unas pocas líneas y sin necesidad de conocer grandes detalles de la implementación. De nuevo, en el Capítulo 5 se ofrecen más detalles y en el Capítulo 6

los resultados, con algunos listados del código que ejemplifican el uso del sistema BoW.

Por último, se dedicó parte del Sprint a corregir un bug detectado en el cálculo de la DWT. Después de comparar los coeficientes obtenidos con los arrojados por el prototipo en Matlab, se decidió cambiar la librería escogida. Se probó con otra de las seleccionadas en el Sprint correspondiente y al volver a tener el mismo problema se optó por usar una librería bien conocida de Python. Para ello, se hizo un script que se ejecuta desde el código C++, utilizando una librería para empotrar código Python.

4.2.7 Sprint 5

ID.1: API Emotiv: Acceso a los datos. 9-11 de junio.

ID	Tarea	Días	Requisito
1	API Emotiv. Acceso a los datos	5	11
2	Engine	2	11

Cuadro 4.6: Sprint Backlog 5

La API Emotiv se presenta como una interfaz en ANSI C que está declarada en tres archivos de cabecera (edk.h, EmoStateDLL.h y edkErrorCode.h) e implementada en dos librerías (libedk.so y edk_utils.so). Las aplicaciones C o C++ que usen la API Emotiv tienen que incluir *edk.h* y enlazar con *libedk.so*.

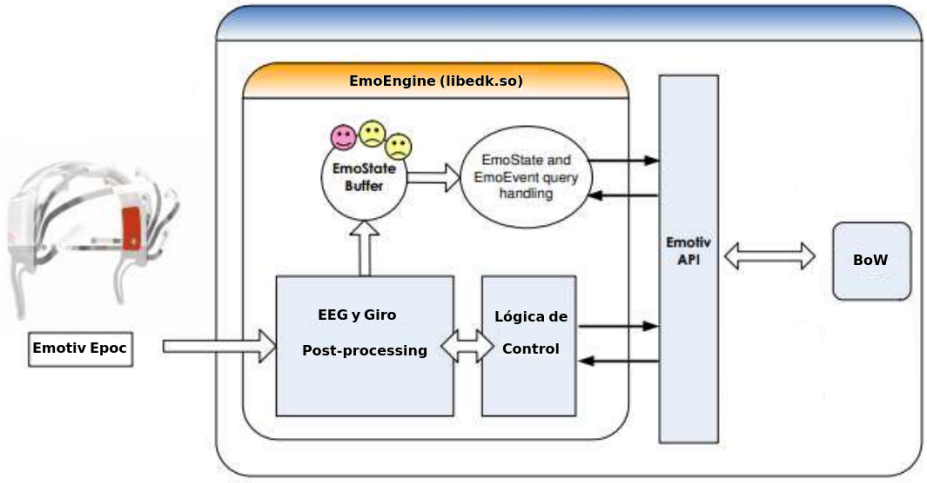

Figura 4.13: Esquema extraido del manual

El Emotiv EmoEngine es la abstracción lógica de la funcionalidad que Emotiv provee en *libedk.so*. Se comunica con el Emotiv headset y recibe los datos preprocesados del EEG, entre otros. En el Anexo C se muestra un diagrama de flujo de alto nivel (extraído del manual de usuario) para conectar una aplicación con el EmoEngine.

El acceso a las mediciones del EEG requiere la creación de una estructura *DataHandle*, un *handle* que provee acceso a los datos subyacentes. Es inicializado con una llamada a *EE_*-

DataCreate. Durante el proceso de medición, EmoEngine mantendrá un buffer de los datos muestreados, medido en segundos, que debe ser inicializado con una llamada a *DataSetBuffersizeInsec(int secdonds)* antes de recoger cualquier dato. Para habilitar la adquisición de datos se realiza una llamada a *DataAcquisitionEnable*. Ahora, EmoEngine empezará a recoger los datos, almacenándolos en un buffer interno. Hay que tener en cuenta que la aplicación debe acceder a los datos antes de que el buffer se sobrescriba.

Para iniciar la recuperación de los datos del buffer, se llama a *DataUpdateHandle*, que preparará los últimos datos almacenados en una estructura pasada como parámetro a esta función. Todos los datos capturados desde la última llamada serán recuperados. Por último, para transferir los datos a algún buffer de la aplicación, hay que llamar a la función *EE_DataGet*.

Después de varias pruebas, se decidió establecer el tamaño del buffer en 4 segundos y recuperar los datos cada 3 segundos. De esta forma, se tiene la seguridad de que no se va a perder ningún dato al sobrescribirse el buffer (ver imagen 4.14).

El Listado 4.6 muestra un pseudocódigo que resume esta explicación.

```
1   // Establish a connection to the EmoEngine
2   if (EE_EngineConnect() == EDK_OK){
3           // A handle to provide access to the EEG ↩
               measurements
4           DataHandle datah = EE_DataCreate();
5           EE_DataSetBufferSizeInSec(4); //seconds
6           EE_DataAcquisitionEnable();
7           while(true){
8                   sleep(3); //wait 3 seconds
9                    // Initiate retrieval of the latest EEG ↩
                        buffered data.
10                   r = EE_DataUpdateHandle(datah);
11                   // transfer the data into a buffer in our↩
                         application
12                   EE_DataGet(datah, channel, my_buffer_data↩
                        );
13           } //while
14  }
15   EE_DataFree(datah);
16   EE_EngineDisconnect();
17   EE_EmoEngineEventFree(eEvent);
```

Listado 4.6: Pseudocódigo para acceder a los datos del headset

ID.2: Engine. 16-24 de junio.

Hasta ahora lo que se ha hecho ha sido acceder a los datos del headset. Esta tarea encapsula la funcionalidad en una clase. Siguiendo la línea de buscar un diseño que ponga énfasis en la modificabilidad y mantenibilidad, esta clase tiene que implementar una interfaz llamada *IEngine*, como muestra la Figura 4.15. De esta forma, si cambiase el dispositivo

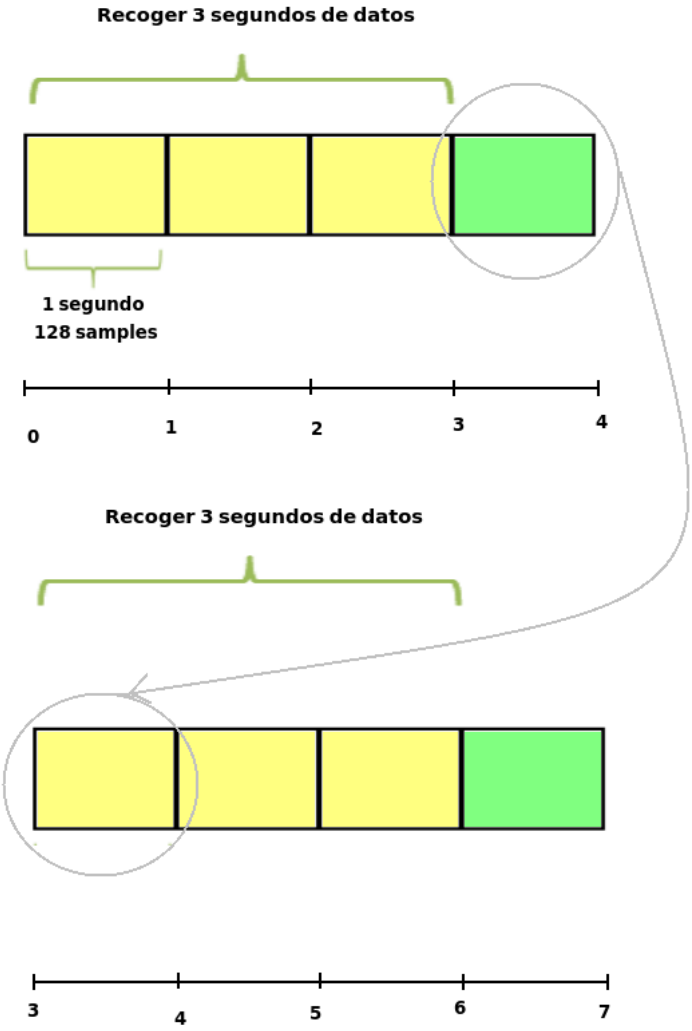

Figura 4.14: Buffer de adquisición de datos

hardware utilizado, sólo habría que implementar una clase nueva que hiciese uso de la API correspondiente, pero no habría que modificar el resto de clases del sistema.

Se presentaron dos alternativas una vez que se disponía de las mediciones del EEG.

1. Almacenarlas hasta que el usuario decidiese parar la prueba y ejecutar el algoritmo BoW al final.

2. Ejecutar el algoritmo BoW mientras que se recogían las mediciones.

Después de evaluar las dos opciones, se concluyó que la segunda opción, a pesar de ser la más atractiva a priori, no aportaba nada sustancial respecto a la primera, y la complejidad y consumo computacional que esto introducía en el sistema no estaba justificado. Esta alternativa requería una coordinación entre la parte *Engine*, que recoge los datos, y la parte *BoW*, que procesa los datos, para lo que sería necesario la implementación del problema

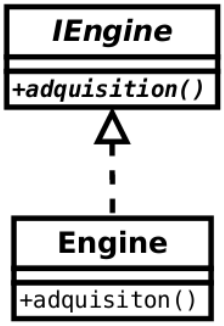

Figura 4.15: La clase Engine implementa la interfaz IEngine

productor-consumidor (o similar). Además, introducía un mayor acoplamiento.

Por último, la primera opción no supone una merma en la funcionalidad. Muy al contrario, funciona como se espera que funcione un sistema de estas características, es decir, recogiendo los datos y aplicarles las transformaciones necesarias para obtener unos resultados.

4.2.8 Sprint 6

Durante este Sprint se estudió la posibilidad de utilizar una Raspberry Pi[3] como plataforma donde ejecutar el algoritmo construido e integrarlo con el hardware de EEG.

Sin embargo, pronto aparecieron los primeros problemas. Como es conocido, Raspberry Pi cuenta con un procesador ARM con el que las librerías provistas por el fabricante del hardware EEG Emotiv, eran incompatibles. Debido a la popularización de los SCB como Raspberry Pi, el fabricante (con el que se mantuvo contacto a través de correo electrónico) aseguró el lanzamiento próximo de un nuevo SDK para ARM dentro del tiempo planificado para el PFC. Sin embargo, este lanzamiento nunca llegó a producirse, por lo que se abandonó esta opción.

Siguiendo esta línea, se estudió a continuación la posibilidad sustituir la Raspberry Pi por un «mini PC[4]». En primer lugar, se utilizó un «mini PC» del grupo ARCO, reutilizado de otro proyecto. Se trataba de un producto antiguo y presentó diversos problemas con las versiones de las librerías necesarias para satisfacer todas las dependencias del software.

Después de varios días, no se pudo hacer funcionar el hardware EEG Emotiv, por lo que se decidió comprar uno nuevo, atendiendo principalmente a dos elementos: El procesador y el precio. Se barajaron los modelos que se recogen en el Anexo D y finalmente se procedió a la compra del Intel NUC DC3217BY.

[3]Es un computador del tamaño de una tarjeta de crédito que puede ser usado en proyectos de electrónica, y para muchas de las tareas de un PC de escritorio `www.raspberrypi.org`.

[4]Computadores de pequeñas dimensiones, generalmente comprimidos en una caja que cabe en una mano.

4.3 Herramientas

En esta sección se describen las herramientas software y los recursos hardware empleados.

4.3.1 Hardware

- **Emotiv Epoc Model 1.0**. Se trata de un producto con el que cuenta el grupo de investigación ARCO. Ofrece un EEG multicanal, inalámbrico y basado en el sistema internacional 10-20 de posicionamiento de los electrodos superficiales[5].
- **Un computador Dell XPS430** del grupo de investigación ARCO con las siguientes prestaciones:
  - Intel Core 2 Duo, 3 GHz
  - 2 GiB RAM
- **Un computador portátil Asus K53SD** con las siguientes prestaciones:
  - Intel Core i5-2450, 2.5 GHz con dos núcleos
  - 8 GiB RAM
- **Raspberry Pi**. Computador del tamaño de una tarjeta de crédito que puede ser usado en proyectos de electrónica y para muchas de las tareas de un PC de escritorio.
- **mini PC Intel NUC DC3217BY**. Los mini PC son computadores de pequeño tamaño, comprimidos en una caja y con un peso de unos cientos de gramos.

4.3.2 Software

Sistema operativo

- Sistema Operativo Debian GNU/Linux Wheezy. Distribución del sistema operativo GNU con núcleo Linux.
- Sistema Operativo Ubuntu GNU/Linux 12.04. Es una de las distribuciones GNU con núcleo Linux más extendidas. Basada en Debian, está enfocado a la facilidad de uso.

Lenguajes de programación

- MATLAB. Es un framework con el que es posible analizar datos, desarrollar algoritmos y crear modelos o aplicaciones.
- C++. Lenguaje de programación de alto nivel que proporciona mecanismos de orientación a objetos. Se utiliza como lenguaje de programación principal.

[5] `http://www.emotiv.com/epoc/`.

- Python. Lenguaje de alto nivel, interpretado, multiparadigma y orientado a objetos.

Software de desarrollo

- *Emotiv Software Development Kit (*SDK*) Research Edition*. SDK del EEG Emotiv Epoc Model que permite la adquisición de datos *en crudo*.
- GCC. Compilador de GNU para C y C++.
- GDB. Depurador para el compilador de GNU.
- GNU Make. Herramienta para la generación automática de ejecutables .
- QtCreator. IDE para C++.
- KDevelop. IDE para C++.
- Git. Software de control de versiones. Registra todos los cambios hechos en un proyecto durante todas sus fases de desarrollo.
- Bitbucket. Servicio de alojamiento basado en web para proyectos que utilizan el sistema de control de versiones Mercurial o Git.

Librerías

- Python API. API para C y C++ que permite empotrar código Python.
- mlpack. Librería en C++ para *machine learning*. Entre los algoritmos que incluye permite realizar *clustering*.
- armadillo. Librería en C++ que usa técnicas avanzadas para operar con matrices de la forma más rápida posible.
- libsvm. Librería con interfaz C++ para SVM.

Documentación

- LaTeX. Lenguaje de marcado de documentos de carácter técnico.
- BibTeX. Herramienta para la descripción de referencias para documentos escritos con LaTeX.
- Kile. Editor TeX/LaTeXpara entornos de escritorio KDE.
- Dia. Software con licencia GPL que permite la creación de diagramas de flujo, Unified Modeling Language (UML), etc, y permite exportarlos a multitud de formatos.
- Canvas. Plataforma online gratuita para crear todo tipo de diseños.
- KolourPaint. Sencilla herramienta libre de edición gráfica para GNU/Linux.

Arquitectura

5

En este capítulo se detalla la arquitectura del sistema desarrollado. Se definirá de manera abstracta cada uno de los componentes y la relación entre ellos. En todo momento, desde el diseño preliminar hasta la implementación, se ha tenido en mente favorecer la reutilización, pero sobre todo la ampliación posterior del sistema y la fácil modificabilidad ante cualquier cambio en los requisitos.

El algoritmo consta de dos partes claramente diferenciadas. Por un lado, la parte dedicada a entrenar (*training*) el sistema y por otra la parte dedicada a la predicción (*predict*), cuya precisión se da en función de la configuración de los módulos que intervienen en el entrenamiento del sistema.

Este capítulo y el anterior están estrechamente relacionados.

5.1 Arquitectura *pipes and filters*

La definición de la arquitectura no fue una tarea específica sino un resultado que se iba obteniendo con el avance del sistema.

La complejidad del software y su tamaño, así como la reutilización de estructuras utilizadas en problemas similares son factores que motivan el uso de una arquitectura software que incluya la descripción de los componentes que forman el sistema, de la interacción entre ellos y de los patrones que guían su composición. El objetivo es obtener un diseño preliminar de alto nivel y una visión general de la organización del sistema.

Los estilos arquitectónicos generalmente se clasifican en:

- Sistemas basados en flujos de datos
- Sistemas Call/Return
- Componentes independientes
- Máquinas virtuales
- Sistemas centrados en datos

El primero de ellos está caracterizado por tratar al sistema como un conjunto de transformaciones sobre flujos de entrada de datos y tiene como ejemplo típico la arquitectura *pipe*

and filter.

En la arquitectura *pipe and filter* cada componente lee un flujo de datos de su entrada y produce un flujo de datos a su salida aplicando una transformación local. Estos componentes se denominan *filters*. Los componentes denominados *pipes* son los encargados de conectar los flujos de datos, desde la salida de un filtro a la entrada de otro.

Una característica importante de este estilo es que los filtros deben ser entidades independientes y no deben compartir su estado con otros filtros. Los mejores ejemplos de arquitecturas *pipe and filter* son los programas escritos en Unix shell, los compiladores, y los que se dan en el dominio del procesamiento de la señal, entre otros.

En este caso, se ha optado por una especialización de la arquitectura *pipe and filter* que restringe la topología a una secuencia lineal, cuyo esquema se puede apreciar en la Figura 3.15. Algunas de las ventajas que aporta es la facilidad de mantenimiento, crecimiento y de análisis de rendimiento.

5.2 Descripción general

A continuación se describen los diferentes módulos o elementos que conforman el sistema a alto nivel:

- **Módulo de obtención de señal**: Se encarga de leer una señal de EEG.
  - **Submódulo Leer desde fichero**: Obtiene una señal desde uno o varios ficheros.
  - **Submódulo Leer desde EEG**: Obtiene una señal desde un dispositivo de EEG.
- **Módulo de Bag of Words**: Se encarga de realizar las transformaciones a la señal necesarias y que ya se han descrito. Es el módulo principal. A su vez, se puede descomponer en:
  - **Submódulo de DWT**: Aplica una Discrete Wavelet Transform a una señal.
  - **Submódulo de Clustering**: Hace clustering sobre un conjunto de vectores que describen una señal.
  - **Submódulo de Clasificación**:
    - Entrena el clasificador con un conjunto de vectores que describen una señal.
    - Clasifica un conjunto de vectores que describen una señal.

La Figura 5.2 muestra los distintos módulos que conforman el sistema.

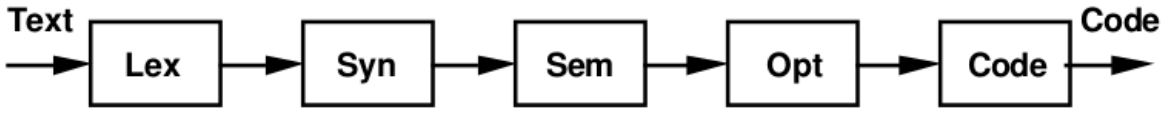

Figura 5.1: Modelo tradicional de un compilador. Imagen obtenida de [GS94]

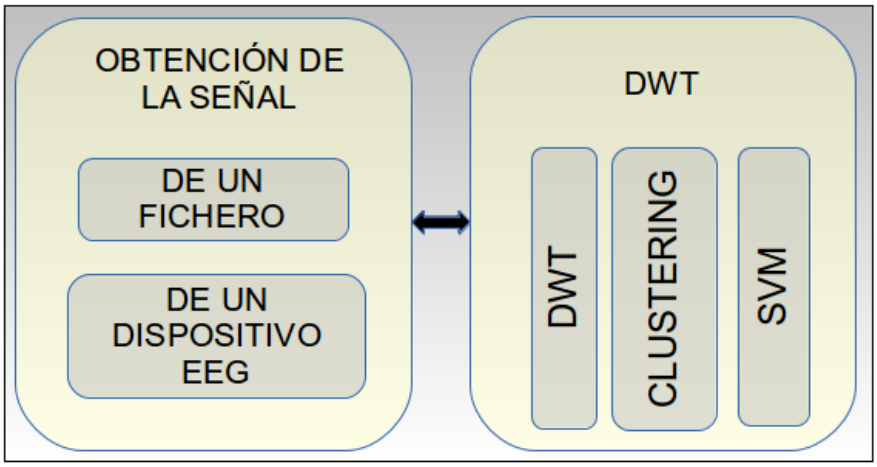

Figura 5.2: Módulos

Se exponen ahora algunos de los principales patrones aplicados en el diseño del software [GHJV96]. En principio, el software que implementa el algoritmo BoW partía del prototipo Matlab, pero se ha realizado un diseño que procura un mejor mantenimiento y una fácil adaptabilidad a posibles cambios y mejoras. Aunque la simplicidad aparente de la arquitectura *pipe and filters* invita a la escritura de un código poco cuidado, se ha realizado un esfuerzo importante para obtener un resultado que se ajuste a las prácticas de la ingeniería del software.

Patrón Método Plantilla

Es un patrón de comportamiento que define el esqueleto de un algoritmo en forma de operaciones abstractas. El método plantilla fija la ordenación de las operaciones, pero permite que las subclases modifiquen su comportamiento para adaptarse a la necesidades determinadas de cada momento. Es una técnica fundamental de reutilización de código ya que permite implementar las partes de un algoritmo que no cambian y dejar que sean las subclases quienes implementen el comportamiento que puede variar.

La solución propuesta busca reutilizar código y establecer la ordenación de las operaciones llevadas a cabo. Para ello, la clase abstracta *BoWSystem* fija la estructura del algoritmo propuesto e implementa algunas de las funciones comunes a la parte de training y de testing. Esta clase es la clase raíz de la jerarquía de clases y la columna vertebral que sostiene la arquitectura *pipes and filters* descrita. Su interfaz pública (el método *execute*) como se puede ver en el Listado 5.1 y 5.2, ejecuta cada uno de los pasos a seguir.

```
1   void BoWSystem::execute()
2   {
3    {
4      obtain_signal();
5      obtain_feature_vectors_matrix();
```

```
1  class BoWSystem{
2
3  protected:
4   void virtual obtain_signal();
5   void virtual obtain_feature_vectors_matrix();
6   void virtual obtain_feature_vectors_reduct() = 0;
7   void virtual classify() = 0;
8
9  public:
10  /*
11   * Skeleton of the BoW algorithm
12   */
13  void execute();
14
15
16 };
```

Listado 5.1: BoWSystem.hpp

```
6       obtain_feature_vectors_reduct();
7       classify();
8     }
9    }
```

Listado 5.2: Método *execute*

De esta clase derivan dos: *BoWTraining* y *BoWPredict* que implementan los métodos que las diferencian, completando el modelo propuesto de Bag of Words:

- Método *classify*. La parte de training crea el modelo SVM; la parte de testing obtiene una predicción a partir del modelo creado.
- Método *obtain_feature_vectors_reduct*. Para obtener el vector de características reducido la parte de training en primer lugar realiza el *clustering*. En segundo lugar obtiene el histograma. La parte de testing sólo necesita obtener el vector de características reducido con los datos obtenidos durante el *clustering*, esto es, la información relativa a los centroides.

Tomando como modelo esta implementación resultaría sencillo, por ejemplo, ampliar el sistema para leer los datos de la señal de ficheros o repositorios con otra estructura a los utilizados ahora sin necesidad de profundizar en el código. El módulo que se encarga de ello, aunque está implementado en la clase raíz, puede ser sobre escrito.

Patrón Strategy

Es un patrón de comportamiento que encapsula distintos algoritmos de una familia y los hace intercambiables (ver Figura 5.3). Permite al algoritmo variar independientemente de los clientes que lo usan. Captura la abstracción en una interfaz y oculta los detalles de implementación en las clases derivadas de la interfaz.

Se puede usar este patrón cuando se necesitan distintas variantes de un algoritmo o

muchas clases relacionadas difieren sólo en su comportamiento.

Este patrón se ha aplicado para construir el módulo de *clustering*. Así, se permite modificar el algoritmo usado si se desea añadiendo un nuevo método. Para el sistema desarrollado se ha implementado el algoritmo *k-means*.

Patrón Fachada

La clase *BoWSystem* sirve a su vez de fachada al proporcionar una interfaz unificada para el conjunto de interfaces, haciendo más fácil de usar el sistema.

De forma similar a una clase *Compilador*, ejemplo típico de un patrón fachada, que facilita la tarea de compilar sin ocultar la funcionalidad de más bajo nivel, la clase *BoWSystem* abstrae de las peculiaridades del sistema si no se quiere modificarlo para adaptarlo a nuevas necesidades.

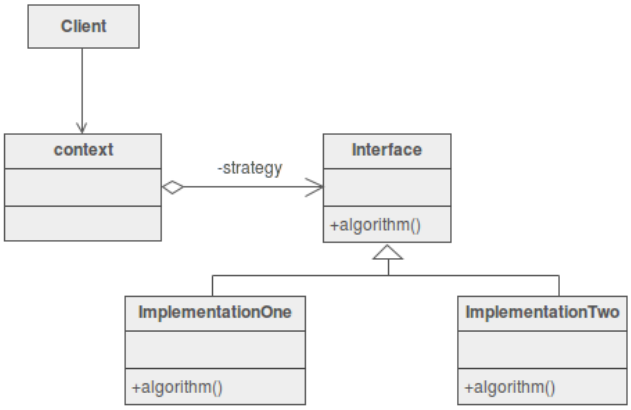

Figura 5.3: Ejemplo de patrón estrategia extraído del libro *Dessign Patterns Explained Simply*

Todo ello da como resultado el diagrama de clases de la Figura 5.4

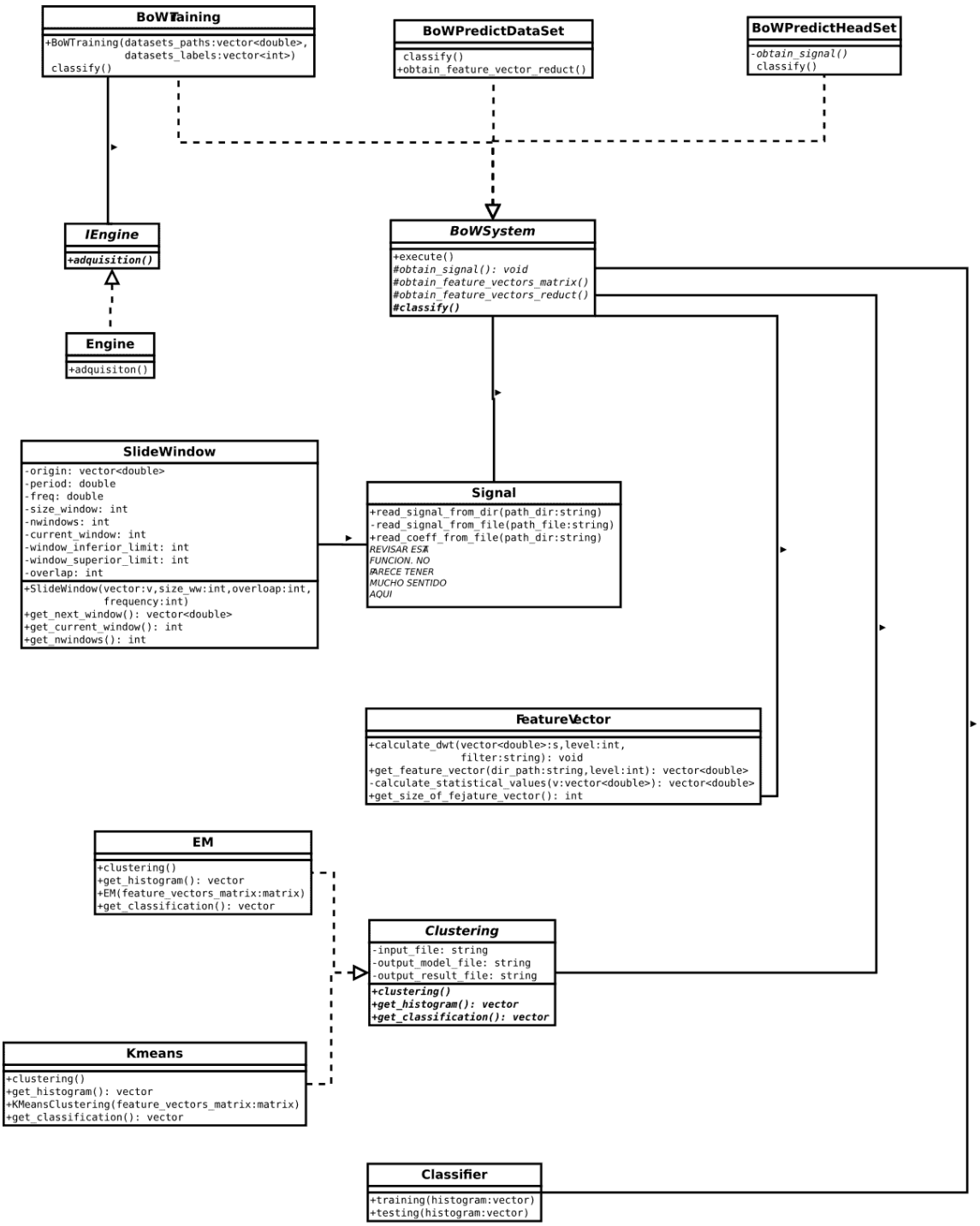

Figura 5.4: Diagrama de clases.

Resultados

6

Se ha desarrollado un sistema para el diagnóstico automático de epilepsia. A partir de una señal EEG, y después un entrenamiento supervisado, el sistema clasifica la señal como *sana* o *epiléptica*.

A continuación se dan más detalles de los datos usados tanto para el entrenamiento como para el *testing* y un análisis de los resultados y del tiempo consumido.

6.1 *Datasets*

Tan importante como el desarrollo del sistema es su fase posterior de entrenamiento y de *testing*. Se han usado un total de 9 *datasets* con distintas características que corresponden a tres fuentes diferentes. La gran mayoría de ellos se pueden encontrar en otros estudios o investigaciones lo que ha permitido una comparativa de métodos directa, basada en los resultados obtenidos del proceso de clasificación. Hay que destacar que no todos los *datasets* se encontraban en las mismas unidades de medida, al no haber sido construidos con la misma instrumentación, por lo que hubo que ajustar los datos al hacer el *testing* con las unidades de los datos con los que se había hecho el entrenamiento.

6.1.1 *Training*

El sistema ha sido entrenado con los *datasets* públicos y usados en multitud de estudios de la Universidad de Bonn [ALR+01]. Se trata de cinco conjunto de datos (A, B, C, D y E), sin embargo solo se ha hecho uso de los conjuntos A y E que contienen segmentos de 23.6 segundos de 100 canales distintos. Estos segmentos fueron cuidadosamente seleccionados después de una inspección visual para eliminar artefactos[1]. El conjunto A y B consisten en segmentos tomados de un EEG superficial practicado sobre cinco voluntarios sanos usando el sistema internacional 10-20 de posicionamiento de los electrodos superficiales (Figura 6.1). Los voluntarios estaban despiertos y con los ojos abiertos (A) y cerrados (B).

El conjunto E contiene segmentos con actividad ictal de cinco pacientes que consiguieron un control completo de los ataques después de la extirpación de una de las formaciones

[1] Actividad muscular del usuario como movimientos de los ojos, parpadeo o movimientos de las extremidades.

hipocampales, que fue diagnosticada correctamente como la zona epileptogénica.

Todas las señales EEG fueron registradas con una frecuencia de muestreo de 173.61 Hz.

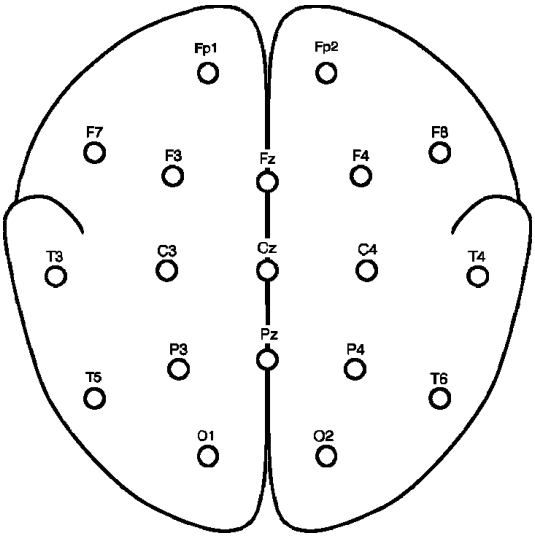

Figura 6.1: Esquema de localización de los electrodos superficiales acorde al sistema internacional 10-20

6.1.2 *Testing*

Para el *testing* se han usado *datasets* de tres fuentes distintas. En primer lugar se han usado los mismos datos de la Universidad de Bonn usados para el entrenamiento. Además, se han usado otros dos que no han sido considerados durante la etapa de entrenamiento: Uno de ellos, hecho público por el «Epilepsy Center of the University Hospital of Freiburg», con dos conjuntos nombrados como I y J. Ambos contienen datos extraídos de un ataque, libre de artefactos y muestreados a una frecuencia de 256 Hz.

Por último, un *dataset* con registros que provienen de un entorno no ideal como los anteriores, con numerosos artefactos y atenuación. Proviene de el HRUM y está compuesto de cinco conjuntos, denominados F, G, H y K para mantener la misma nomenclatura. El F registra la actividad de un paciente sano y contiene multitud de artefactos. El G registra actividad inter-ictal y contiene también numerosos artefactos. El H contiene registros de un ataque parcial en el lóbulo temporal izquierdo del cerebro. Contiene registros de la parte sana y epiléptica. El K registra una tónico-clónica[2]. El F y G están muestreados a una frecuencia de 511.99 Hz. y el H y K a una frecuencia de 200 Hz. [oFG13]

6.2 Repositorio

Tanto el código como la documentación del proyecto se encuentran alojados en *bitbucket.org* [3] [bit] y se puede descargar ejecutando en un terminal:

[2]Es el tipo de crisis más conocido. En una primera fase, llamada tónica, aparece rigidez muscular en el tronco y las extremidades. En una segunda fase, llamada clónica, aparecen espasmos rítmicos de todo el cuerpo.

[3]url del proyecto: `https://bitbucket.org/JavierMarchan/pfc`.

```
1   git git@bitbucket.org:JavierMarchan/pfc.git
```

Cuando se ha descargado el repositorio, se obtiene la estructura del proyecto preparada para compilar. El *Makefile* (ver Apéndice E) hace una compilación modular, permitiendo ahorrar una gran cantidad de tiempo al crear los distintos ejecutables o cuando se modifican uno o varios ficheros. La estructura es la siguiente:

- doc/ Contiene la documentación del proyecto preparada para compilar ejecutando *make* en un terminal.
- main/ Contiene el código del proyecto
  - ▷ bin/ Directorio donde se guardan los ejecutables.
  - ▷ data/ Directorio donde se almacenan los *datasets* y otros archivos generados durante la ejecución.
  - ▷ include/ Contiene los archivos de cabecera.
  - ▷ lib/ Contiene las librerías necesarias para compilar la parte que conecta con el hardware EEG emotiv. Forma parte del SDK y se obtienen al adquirir el producto.
  - ▷ obj/ Contiene los ficheros *objeto* generados durante la compilación.

6.3 Resultados

Los distintos ejecutables que se crean son tres ejemplos que muestran el uso del sistema:

- **main-train** (Listado 6.1). Se crea al ejecutar *make main-train*. Se trata de un programa que hace uso de la parte de *training*. En él hay que indicar qué *datasets* se van a usar para el entrenamiento, la *etiqueta* de cada uno[4] y la frecuencia a la que fueron registrados los datos. Aunque la ruta de los *datasets* puede ser cualquiera, se ha creado un directorio *data* especialmente para guardarlos en él. En este directorio se crean dos ficheros como resultado de la ejecución del *training*:
  - `centroids.csv`. Como su nombre indica, contiene las coordenadas de los centroides producidos en el *training*. Es un fichero en formato .csv que se puede ver con cualquier editor de texto.
  - `model`. Es el modelo generado por el clasificador en el *training*. Al ser texto plano se puede ver con cualquier editor de texto.

  Los dos ficheros serán utilizados posteriormente en la fase de *testing*
- **main-predict** (Listado 6.2). Se crea al ejecutar *make main-predict*. Se trata de un programa que hace uso de la parte de *testing*. En el ejemplo mostrado se ejecuta con el

[4] Se utiliza una característica del lenguaje, los enumerados, para crear dos etiquetas: HEALTHY y EPILEPTIC que representan a los *datsets* con una señal sana y epiléptica, respectivamente.

dataset I, que fue creado tomando registros a una frecuencia de 256 Hz. También se observa que se incluye la variable *dataset_label* y se le asigna el valor *EPILEPTIC*. Esto se hace exclusivamente para obtener un porcentaje de aciertos en la predicción, ya que conocemos que el *dataset* que estamos probando le corresponde dicha etiqueta. Por supuesto, en un entorno real no sólo careceríamos de esta información, sino que el propio objetivo sería obtener la predicción.

- **main-headset** (Listado 6.3). Se crea al ejecutar *make main-headset*. Se trata de un programa que hace uso de la parte de *testing*, pero conectando con el hardware EEG *emotiv* descrito. Mientras se está ejecutando, se registran las mediciones procedentes de los 16 canales. Cuando el usuario decide terminar de recoger datos, se usan estos como entrada al sistema. Además, se crea un nuevo *dataset* en el directorio *data* llamado *Headset* que almacena los datos.

```
1  #include "include/BoWTraining.hpp"
2  #include "include/BoWSystem.hpp"
3
4  #include <iostream>
5  #include <vector>
6  #include <string>
7  using namespace std;
8
9  enum condition{
10   HEALTHY = 0,
11   EPILEPTIC
12 };
13
14 int main(int argc, char** argv)
15 {
16   vector<string> datasets_paths  = {"data/Aset/",
17          "data/Eset/"};
18   vector<int>    datasets_labels = {  HEALTHY    ,
19            EPILEPTIC};
20   vector<double> freqs           = {   173.61     ,
21            173.61};
22
23   BoWSystem *BoWBonn = new BoWTraining(datasets_paths, ↩
       datasets_labels, freqs);
24   BoWBonn->execute();
25 }
```

Listado 6.1: main-train

6.4 Análisis de los resultados

La Tabla 6.1 muestra los resultados que se obtienen realizando el *training* y el *testing* como se describe en las secciones 6.1.1 y 6.1.2, respectivamente.

Se pueden observar varias cosas interesantes de la tabla. En primer lugar, los resultados

```
#include "include/BoWPredictDataSet.hpp"
#include "include/BoWSystem.hpp"

#include <iostream>
#include <vector>
#include <string>
using namespace std;

enum condition{
  HEALTHY = 0,
  EPILEPTIC
};

int main(int argc, char** argv)
{
  vector<string> datasets_paths  = {"data/Iset/"};
  vector<int>    datasets_labels = {  EPILEPTIC   };
  vector<double> freq            = {      256     };
  BoWSystem *BoWTestBonn = new BoWPredictDataSet(
                datasets_paths, datasets_labels, freq);
  BoWTestBonn->execute();
}
```

Listado 6.2: main-predict

```
#include "include/BoWPredictHeadSet.hpp"
#include "include/BoWSystem.hpp"

#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main(int argc, char** argv)
{
  vector<double> freq            = {      128     };
  BoWSystem *BoWTestBonn = new BoWPredictHeadSet(freq);
  BoWTestBonn->execute();
}
```

Listado 6.3: main-predict

	HEALTHY		**EPILEPTIC**	
	Set	%	Set	%
Bonn	A	100	E	100
HRUM	F	11,3	G	93,33
			H	96,74
			K	98,09
Freiburg			I	95,48
			J	93,89

Cuadro 6.1: Resultados

de la clasificación de los *datasets* de la Universidad de Bonn [ALR+01] son de un **100 % de aciertos**. En la Tabla 6.2 se puede ver una comparativa con los resultados obtenidos en 9 trabajos que usan los mismos *datasets* de la Universidad de Bonn pero con diferentes métodos que van desde las redes neuronales hasta el análisis Wavelet.

Referencia	**%**
Kannathal et al. [JFM+12]	95
Polat and Gulness [oF13]	98.72
Guo et al. [WLSN12]	99.6
Wang et al. [SEC+04]	99.5
Janjarasjitt et al. [Org12]	99
Husain et al. [Jan10]	98.2
Fathima et al. [ST09]	99.8
Übeyli et al. [FBFK11]	94.93
Fathima et al. [SBS99]	96.9
PFC	100

Cuadro 6.2: Comparativa con otros trabajos

En el trabajo actual se mejoran todos los anteriores.

En segundo lugar hay que destacar la calidad de los resultados en el resto de *datasets*, más complejos y realistas por la presencia de ruidos y artefactos en ellos.

Estos resultados son coherentes con el trabajo [SRDTB+12], punto de partida de este PFC. En el citado trabajo se realizan más experimentos, pero si se presta atención al realizado usando un *kernel* RBF para el clasificador como el utilizado aquí, se puede ver como algunos *datasets* bajan su porcentaje de acierto (especialmente el F), otros suben y, globalmente, mejora ligeramente el resultado. Las diferencias en los resultados, más pequeñas de lo esperado en el planteamiento del PFC, se pueden deber al uso de librerías diferentes (se ha desarrollado un sistema en C++ frente a un prototipo en Matlab) y a usar un algoritmo distinto en la etapa de *clustering*: *k-means* en este trabajo frente a EM en [SRDTB+12]. Ver Cuadro 6.3.

En cuanto al caso del *dataset* F, aunque no es favorable, no hay que perder de vista que en el estado del arte no existen experimentos con datos de estas características, extraídos de un entorno real con multitud de artefactos e interferencias.

6.5 Análisis de tiempo

Aunque el tiempo consumido en realizar el *training* y *testing* no es algo crítico, se ha realizado un análisis del tiempo por cada *dataset* (Tabla 6.4) y del tiempo que consume el entrenamiento (Tabla 6.5).

Para medir los tiempos se ha usado la librería *chrono* que pertenece al estándar C++11 y

Set	Sistema		Δ (%)
	ESI (%)	PFC (%)	
A	99.92	100	0.08
E	97.75	100	2.3
F	42.91	11.3	-73.66
G	62.69	93.33	48.1
H	89.49	96.74	8.1
I	100	98.09	-1.91
J	100	98.48	-1.52
K	71.98	93.89	30.4
Media	83.09	86.1	3.62

Cuadro 6.3: Comparativa con los datos de [SRDTB$^+$12]

que está definida en el archivo de cabecera *chrono*

Set	(s)
A	0.5007
E	0.5008
F	0.2097
G	15.4947
H	0.4709
I	1.4908
J	0.913
K	1.1861

Cuadro 6.4: Tiempo consumido durante el *testing* por cada *dataset*

Los tiempos corresponden a la ejecución del sistema sobre un procesador Intel® Core™ i5-2450M

Set	(s)
A, E	1.08

Cuadro 6.5: Tiempo consumido durante el *training*

Conclusiones y propuestas

7

Este capítulo presente las conclusiones más relevantes derivadas de la realización de este trabajo, así como las posibles líneas futuras de investigación que se abren tras el mismo.

7.1 Conclusiones

Este trabajo se proponía como objetivo general el de la construcción de un sistema de bajo coste para el diagnóstico automático de epilepsia, para lo cual se marcaron una serie de sub-objetivos más específicos tal y como describieron en el **Capítulo 4: Método de trabajo**.

Una vez desarrollado el sistema y analizado en el **Capítulo 6: Resultados** los porcentajes de precisión obtenidos por el clasificador, se puede concluir que estos resultados mejoran los de los trabajos previos hasta la fecha y revisados en el **Capítulo 3: Antecedentes**. Además, este trabajo también mejora ligeramente los resultados obtenidos por su versión prototipada implementada utilizando Matlab [SRDTB+12] donde se propone el uso del modelo BoW para la diagnosis de epilepsia.

No es pequeño el reto de intentar diagnosticar de forma automática una enfermedad, sin intervención humana, y quizás quede un largo camino por recorrer hasta que una empresa de estas características sea viable tecnológicamente. No era este, ni mucho menos, el objetivo buscado. También supone un gran reto trasladar estos resultados de un entorno controlado y académico a un entorno real. Sin embargo los resultados obtenidos invitan a continuar esta línea.

En cuanto al modelo BoW implementado, explicado con detalle a lo largo de este documento, aunque éste sea susceptible de mejoras lo cierto es que los resultados en la mayoría de los *datasets* están cerca del 100 % de aciertos, sin perder de vista que están construidos con datos preprocesados. Una herramienta completamente funcional y eficaz debería ser capaz de predecir en entornos menos controlados. Otra limitación importante que se plantea es la dificultad de probar en una situación real el sistema. Cuando se empezó a perfilar el problema, a mediados de 2013, se contaba con la posibilidad de acudir al HRUM y hacer pruebas junto a los profesionales que colaboraron en el planteamiento del PFC. Finalmente no fue posible por una limitación temporal, pero en cualquier caso, esto requeriría una fuerte colaboración entre diversas instituciones, profesionales, etc., y resolver problemas de orga-

nización incluso de índole muy distinta a los tratados aquí como cumplir protocolos clínicos u otra serie de medidas.

En cuanto al hardware EEG tampoco cabe ninguna duda de que la tecnología progresará y será posible contar con alternativas más baratas y de mayor calidad y, a ser posible, de código abierto. Aunque hemos tenido la posibilidad de disponer de un sistema hardware para la captura de la actividad cerebral a un coste razonable, la dependencia de este hardware a las librerías de código cerrado ha imposibilitado la migración de sistema software a arquitecturas distintas de la Intel.

7.2 Propuestas

Las líneas de trabajo que surgen para ir avanzando por el camino que se abre a partir del trabajo realizado aquí son múltiples y se pueden proponer como nuevos PFC e incluso tesis de máster o doctorales, que sin duda serían de interés para los estudiantes. Para terminar se exponen una serie de propuestas en este sentido que se han ido identificando a lo largo de los más de 12 meses de trabajo:

En cuanto al algoritmo:

1. Algunas de las etapas de las que consta el algoritmo se pueden modificar para mejorar la precisión del clasificador. En la etapa de *clustering* se puede sustituir el algoritmo *k-means* por el algoritmo EM. Como demuestra [SRDTB+12] mejora sustancialmente al formar «gaussianos» que pueden tener forma elíptica y por tanto, hacer un mejor agrupamiento en espacios complejos. Otra de las mejoras más evidentes se puede obtener del módulo de clasificación. SVM es una herramienta compleja que requiere de una profunda comprensión teórica. Configurar los distintos parámetros pudiera arrojar resultados diferentes.
2. Contar con más y mejores conjuntos de datos para entrenar el sistema mejoraría la calidad del modelo obtenido, a partir del cual se hacen las predicciones. Sin embargo no es algo trivial.

En cuanto al EEG:

1. El producto adquirido tiene algunas limitaciones obvias que no se presentan en equipos profesionales. Con el uso del mismo, varios sensores se han deteriorado perdiéndose la información de ese canal. A nivel de hardware poco se puede hacer, salvo valorar otros productos, pero se puede conocer a través de la API la calidad de la señal obtenida por cada sensor sin mucho esfuerzo, y en base a esto tomar una decisión: ignorarlo, avisar al usuario, no realizar ninguna predicción hasta conseguir una mayor calidad de señal, etc.
2. El software Emotiv adquirido junto al hardware EEG tiene algunas funciones que pueden resultar útiles, como la que detecta expresiones faciales. Esta aplicación puede

detectar algunos artefactos como guiños, sonrisas o movimientos oculares. Se podría aprovechar esta ventaja para obtener una señal limpia.

3. El hardware también dispone de giróscopos que informan de los movimientos realizados con la cabeza. También se podría aprovechar esta información cuando se registra la señal y evitar los segmentos que no contengan una señal limpia.

ANEXOS

Scripts

```
#code modified from http://glowingpython.blogspot.com.es/2011/08/how-to-plot-frequency-spectrum-with.html

from numpy import sin, cos, linspace, pi
from pylab import plot, show, title, xlabel, ylabel, subplot
from scipy import fft, arange

def plotSpectrum(y, Fs):

 n = len(y) # length of the signal
 k = arange(n)
 T = n/Fs
 frq = k/T # two sides frequency range
 frq = frq[range(n/2)] # one side frequency range
 Y = fft(y)/n # fft computing and normalization
 Y = Y[range(n/2)]

 plot(frq,abs(Y),'r') # plotting the spectrum
 xlabel('Freq (Hz)')
 ylabel('|Y(freq)|')

#main
Fs = 500.0;  # sampling rate
Ts = 1.0/Fs; # sampling interval
t = arange(0,1,Ts) # time vector
ff = 5;   # frequency of the signal
w= 2*pi
y = sin(w*ff*t) + sin(w*2*ff*t) + sin(w*5*ff*t)

subplot(2,1,1)
plot(t,y)
xlabel('Time')
ylabel('Amplitude')
subplot(2,1,2)
plotSpectrum(y,Fs)
show()
```

Listado A.1: Señal estacionaria con su espectro de frecuencias

```
#code modified from http://glowingpython.blogspot.com.es↩
   /2011/08/how-to-plot-frequency-spectrum-with.html

from numpy import sin, cos, linspace, pi
from pylab import plot, show, title, xlabel, ylabel, ↩
   subplot
from scipy import fft, arange

#main
Fs = 500.0;  # sampling rate
Ts = 1.0/Fs; # sampling interval
t = arange(0,1,Ts) # time vector

ff = 5;   # frequency of the signal
w= 2*pi
y = sin(w*ff*t)

subplot(2,1,1)
plot(t,y)
xlabel('Time')
ylabel('Amplitude')
show()
```

Listado A.2: Señal estacionaria

```
#code modified from http://glowingpython.blogspot.com.es↩
   /2011/08/how-to-plot-frequency-spectrum-with.html

from numpy import sin, cos, linspace, pi, concatenate
from pylab import plot, show, title, xlabel, ylabel, ↩
   subplot
from scipy import fft, arange, hstack
import matplotlib.pyplot as plt

Fs = 1000.0;  # sampling rate
Ts = 1.0/Fs; # sampling interval
t = arange(0,1,Ts) # time vector

ff = 10;   # frequency of the signal
w= 2*pi
y1 = sin(w*ff*t[:400])
y2 = sin(w*2*ff*t[400:800])
y3 = sin(w*5*ff*t[800:1000])
y = hstack((y1,y2,y3))

plot(t, y)
ylabel('Amplitud')
xlabel('Tiempo (s)')
show()
```

Listado A.3: Señal no-estacionaria

```
#code modified from http://glowingpython.blogspot.com.es/2011/08/how-to-plot-frequency-spectrum-with.html

from numpy import sin, cos, linspace, pi, concatenate
from pylab import plot, show, title, xlabel, ylabel, subplot
from scipy import fft, arange, hstack
import matplotlib.pyplot as plt
def plotSpectrum(y, Fs):
 """
 Plots a Single-Sided Amplitude Spectrum of y(t)
 """
 n = len(y) # length of the signal
 k = arange(n)
 T = n/Fs
 frq = k/T # two sides frequency range
 frq = frq[range(n/5)] # one side frequency range

 Y = fft(y)/n # fft computing and normalization
 Y = Y[range(n/5)]

 plot(frq,abs(Y),'r') # plotting the spectrum
 xlabel('Freq (Hz)')
 ylabel('|Y(freq)|')

 tickpos=arange(0,200,10)
 ticklabels=[]
 for point in tickpos:
         ticklabels.append(point)

 plt.xticks(tickpos, ticklabels)

#main
Fs = 1000.0;  # sampling rate
Ts = 1.0/Fs; # sampling interval
t = arange(0,1,Ts) # time vector

ff = 10;   # frequency of the signal
w= 2*pi
y1 = sin(w*ff*t[:400])
y2 = sin(w*2*ff*t[400:800])
y3 = sin(w*5*ff*t[800:1000])
y = hstack((y1,y2,y3))
subplot(2,1,1)
plot(t, y)
ylabel('Amplitud')
xlabel('Tiempo (s)')
subplot(2,1,2)
plotSpectrum(y,Fs)
show()
```

Listado A.4: Señal no-estacionaria con su espectro de frecuencias

Presupuesto de los equipos EEG

B

Equipo	Precio
actiCHAMP	18.000 €
TREA	12.303 €
Emotiv	750 $

Cuadro B.1: Precios de los distintos equipos EEG

Diagrama de flujo para comunicar con el EmoEngine

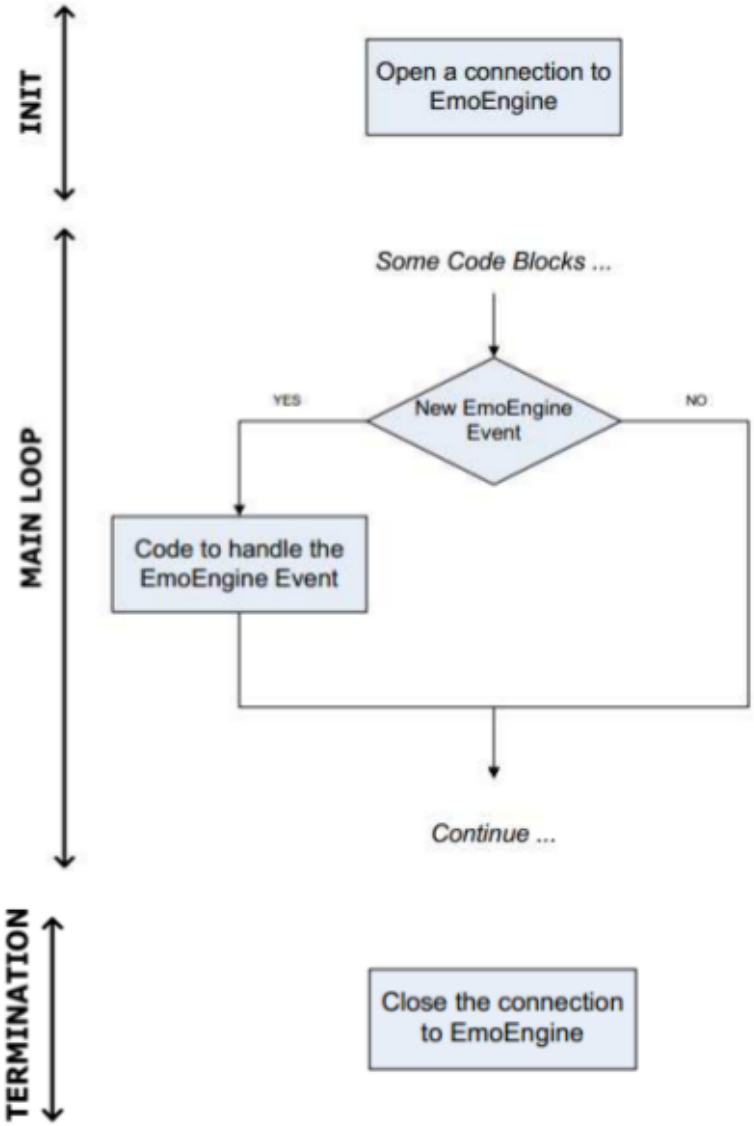

Figura C.1: Diagrama de flujo de alto nivel para las aplicaciones que incorporan el EmoEngine. Durante la inicialización, y antes de llamar a las funciones de la API Emotiv, la aplicación debe establecer una conexión al EmoEngine llamando a *EE_EngineConnect*

Comparativa «mini PCs»

Modelo	Procesador	Precio (€)
Intel NUC DC3217BY	Intel Core i3	153
Shuttle XH61V	Intel Core i7	173
Zotac ID41	Atom D525	182
Zotac ZBOX ID18	Intel 1007U	118
Intel NUC DN2820FYKH	Intel Celeron N2820	119

Cuadro D.1: Comparativa de varios modelos de mini PCs

Makefile

```
CC          = g++
DEBUG       = -g
INCLUDEXML = -I/usr/include/libxml2
IPYTHON     = -I /usr/include/python2.7
FLAGS       = -ansi -Wall -std=c++11 -pedantic -O3 $(DEBUG)
LFLAGS      = -m64
LIBS        = $(SUBLIBS)  -L/usr/lib/x86_64-linux-gnu -L./lib/ -ledk
EMBEDPY     = -lpython2.7 `python-config --cflags` `python-config --ldflags`

EXECTRAIN  = main-train
EXECPREDICT= main-predict
EXECHEAD   = main-headset
INC         =  include
BIN         =  bin
OBJ         = obj
DATA        = data

all: main-train main-predict main-headset

main-train:  $(OBJ)/svm.o $(OBJ)/Classifier.o $(OBJ)/SlideWindow.o $(OBJ)/Signal.o $(OBJ)/FeatureVector.o $(OBJ)/KMeansClustering.o $(OBJ)/BoWSystem.o $(OBJ)/BoWTraining.o
	$(CC) -o $(BIN)/$(EXECTRAIN)     $(FLAGS) $(IPYTHON) main-train.cpp $^  -lmlpack $(EMBEDPY)

main-predict: $(OBJ)/svm.o $(OBJ)/Classifier.o $(OBJ)/SlideWindow.o $(OBJ)/Signal.o $(OBJ)/FeatureVector.o $(OBJ)/KMeansClustering.o $(OBJ)/BoWSystem.o $(OBJ)/BoWPredictDataSet.o
	$(CC) -o $(BIN)/$(EXECPREDICT) $(FLAGS) $(IPYTHON) main-predict.cpp $^ -lmlpack $(EMBEDPY)

main-headset: $(OBJ)/svm.o $(OBJ)/Classifier.o $(OBJ)/SlideWindow.o $(OBJ)/Signal.o $(OBJ)/FeatureVector.o $(OBJ)/KMeansClustering.o $(OBJ)/BoWSystem.o $(OBJ)/BoWPredictHeadSet.o $(OBJ)/Engine.o
	$(CC) -o $(BIN)/$(EXECHEAD) $(FLAGS) $(IPYTHON) main-headset.cpp $^ -lmlpack $(EMBEDPY) $(LIBS)
```

```
29 $(OBJ)/Engine.o: Engine.cpp $(INC)/Engine.hpp $(INC)/↩
      IEngine.hpp
30         $(CC) $(LFLAGS) $(FLAGS) -c Engine.cpp -o obj/↩
              Engine.o

32 $(OBJ)/BoWPredictHeadSet.o: $(OBJ)/BoWSystem.o ↩
      BoWPredictHeadSet.cpp $(INC)/BoWPredictHeadSet.hpp
33         $(CC) $(FLAGS) $(IPYTHON) $(INCLUDEXML) -c ↩
              BoWPredictHeadSet.cpp -o $(@)

35 $(OBJ)/BoWPredictDataSet.o: $(OBJ)/BoWSystem.o ↩
      BoWPredictDataSet.cpp $(INC)/BoWPredictDataSet.hpp
36         $(CC) $(FLAGS) $(IPYTHON) $(INCLUDEXML) -c ↩
              BoWPredictDataSet.cpp -o $(@)

38 $(OBJ)/BoWTraining.o: $(OBJ)/BoWSystem.o BoWTraining.cpp $↩
      (INC)/BoWTraining.hpp
39         $(CC) $(FLAGS) $(IPYTHON) $(INCLUDEXML) -c ↩
              BoWTraining.cpp -o $(@)

41 $(OBJ)/BoWSystem.o: BoWSystem.cpp $(INC)/BoWSystem.hpp
42         $(CC) $(FLAGS) $(IPYTHON) -c $< -o $(@)

44 $(OBJ)/KMeansClustering.o: KMeansClustering.cpp $(INC)/↩
      KMeansClustering.hpp include/clustering.hpp
45         $(CC) $(FLAGS) $(INCLUDEXML) -c $< -o $(@) -↩
              larmadillo -lmlpack -lm

47 $(OBJ)/FeatureVector.o: FeatureVector.cpp $(INC)/↩
      FeatureVector.hpp
48         $(CC) $(FLAGS)  $(IPYTHON) -c $<  -o $(@) $(↩
              EMBEDPY)

50 $(OBJ)/Signal.o: Signal.cpp $(INC)/Signal.hpp
51         $(CC) $(FLAGS) -c $<  -o $(@)

53 $(OBJ)/SlideWindow.o: SlideWindow.cpp $(INC)/SlideWindow.h
54         $(CC) $(FLAGS) -c $<  -o $(@)

56 $(OBJ)/Classifier.o: Classifier.cpp $(INC)/Classifier.hpp ↩
      $(INC)/svm.h
57         $(CC) $(FLAGS) -c $<  -o $(@)
58         echo 'Classifier compiled'

60 $(OBJ)/svm.o : svm.cpp $(INC)/svm.h
61         $(CC) -Wall -Wconversion -O3 -fPIC $(DEBUG) -c svm↩
              .cpp -o $(@)

63 clean:
64         rm -f  $(BIN)/$(EXECTRAIN) $(BIN)/$(EXECPREDICT) $↩
              (DATA)/centroids.csv $(DATA)/labels.csv $(DATA)↩
              /svm_model $(OBJ)/*.o
```

Bibliografía

[AD90] Hauser AW y Hesdorffer DC. Epilepsy: frequency, causes and consequences. *New York, Demos Publications*, 1990.

[AG13] Eduardo Aguilar Gallego. *Proyecto Fin de Carrera: Plataforma para la interacción con dispositivos de Electroencefalografía*. 9 2013.

[ALR+01] R. G. Andrzejak, K. Lehnertz, C. Rieke, F. Mormann, P. Davi, y C. E. Elger. Indications of nonlinear deterministic and finite dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Phys, Rev. E*, 64, 2001.

[Ban05] World Bank. World Bank list of economies. *Epilepsia*, 2005.

[bit] Atlassian Bitbucket. `https://bitbucket.org/`.

[BLZ08] F. S. Bao, D. Y. Lie, y Y. Zhang. A New Approach to Automated Epileptic Diagnosis Using EEG and Probabilistic Neural Network. *20th IEEE International Conference on Tools with Artificial Intelligence*, 2:pp. 482–486, 2008.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[Bra04] World Health Organization Regional Office for Afrca Brazzaville. Global Campaign Against Epilepsy. Epilepsy in the WHO African Region: bridging the gap. 2004. (AFR/MNH/04.1).

[CA98] Fraley C. y Raftery A.E. How Many Clusters? Which Clustering Method? Answers Via Model- Based Cluster Analysis. *Department of Statistics University of Washington*, 1998.

[CCS+13] Ryan R. Curtin, James R. Cline, Neil P. Slagle, William B. March, P. Ram, Nishant A. Mehta, y Alexander G. Gray. MLPACK: A Scalable C++ Machine Learning Library. *Journal of Machine Learning Research*, 14:801–805, 2013.

[CL11] Chih-Chung Chang y Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CS14] Enrique J. Carmona Suárez. Tutorial sobre Máquinas de Vectores Soporte (SVM. Technical report, Dpt. de Inteligencia Artificial, ETS de Ingeniería Informática, Universidad Nacional de Educación a Distancia (UNED), 7 2014.

[Dau92] Ingrid Daubechies. Ten Lecturs on Wavelets. *Society for Industrial and Applied Mathematics Philadelphia, PA, USA*, 1992.

[Dur10] Bahadir Durak. A Classification Algorithm Using Mahlanobis Distance Clustering Of Data With Applications On Biomedical Data Sets. Technical report, The Graduate School Of Natural And Applied Sciences Of Middle East Technical University, 1 2010.

[EWW+08] J. Echauz, G. Worrell, S. Wong, O. Smart, A. Gardner, y B. Litt. Computation Applied to Clinical Epilepsy and Antiepileptic Devices. *Soltesz I, Staley K, editors*, página p. 530–558, 2008.

[FBFK11] T. Fathima, M. Bedeeuzzaman, O. Farooq, y Y. U. Khan. Wavelet Based Features for Epileptic Seizure Detection. *MES Journal of Technology and Management*, 2:pp. 108–112, 2011.

[Fou13] Free Software Foundation. GSL - GNU Scientific Library, 2013. `http://www.gnu.org/software/gsl/`.

[FvEBb+05] Robert S. Fisher, Walter van Emde Boas, Warren blume, Christian elger, Pierre Genton, Phillip Lee, y Jr. Jerome Engel. Epileptic Seizures and Epilepsy: Definitions Proposed by the International League Against Epilepsy (ILAE) and the International Bureau for Epilepsy (IBE). *Epilepsia*, 46, 2005.

[GHJV96] E. Gamma, R. Helm, R. Johnson, y J. Vlissides. *Dessign Patterns*. Addison-Wesley, 1996.

[GKVL06] A. B. Gardner, A. M. Krieger, G. Vachtsevanos, y B. Litt. One-Class Novelty Detection for Seizure Analysis from Intracranial EEG. *Journal of Machine Learning Research*, 7:pp. 1025–1044, 2006.

[Got82] J. Gotman. Automatic recognition of epileptic seizures in the EEG. *Electroencephalography and Clinical Neurophysiology*, 54:pp. 530–540, 1982.

[Got90] J. Gotman. Automatic seizure detection: improvements and evaluation. *Electroencephalography and Clinical Neurophysiology*, 76:pp. 317–324, 1990.

[GS94] David Garlan y Mary Shaw. An Introduction to Software Architecture. *World Scientific Publishing*, 1 1994.

[GVSC03] B. Gonzalez-Vellon, S. Sanei, y J. A. Chambers. Support vector machines for seizure detection. *Signal Processing and Information Technology*, páginas pp. 126–129, 2003.

[HK01] J. Han y M. Kamber. Data Mining: Concepts and Techniques. *Morgan Kaufmann Publishers*, 2001.

[Hof06] Martin Hofmann. Support Vector Machines — Kernels and the Kernel Trick. 6 2006.

[HR12] S. J. Husain y K. S. Rao. Epileptic Seizures Classification from EEG Signals using Neural Networks. *International Proceedings on Computer Science and Information Technology*, 37:pp. 269–273, 2012.

[Hus11] Rafat Hussain. Wavelet1d, 2011. `http://code.google.com/p/wavelet1d/` .

[Jan10] S. Janjarasjitt. Classification of the Epileptic EEGs Using the Wavelet-Based Scale Variance Feature. *International Journal of Applied Biomedical Engineering*, 3, 2010.

[JFM+12] E. Johnson, R. Fraser, J. Miller, N. Temkin, J. Barber, P. Ciechanowski, y N. Chaytor. A comparison of epilepsy self-management needs: Provider and patient perspectives. *Epilepsy and Behavior*, 25:150–155, 2012.

[Joa98] T. Joachims. Making large-Scale SVM Learning Practical. LS8-Report 24, Universität Dortmund, LS VIII-Report, 1998.

[Jor04] Michael I. Jordan. Statistical Learning Theory: The Kernel Trick. Technical report, 2004.

[KGDA05] M. K. Kıymık, I. Güler, A. Dizibüyük, y M. Akın. Comparison of STFT and wavelet transform methods in determining epileptic seizure activity in EEG signals for real-time application. *Computers in biology and medicine*, 35:pp. 603–616, 2005.

[Klo09] Wlodzimierz Klonowski. Everything you wanted to ask about EEG but were afraid to get the right answer. *Nonlinear Biomedical Physics*, 3(1):2, 2009. url: `http://www.nonlinearbiomedphys.com/content/3/1/2`.

[Kni07] Henrik Kniberg. *Scrum and XP from the trenches*. C4Media, edición 1ª, 2007.

[LCD+13] Karl LaFleur, Kaitlin Cassady, Alexander Doud, Kaleb Shades, Eitan Rogin, y Binh He. Quadcopter control in three-dimensioanl space using a noninvasive motor imagery-based brain-computer interface. *Journal of Neural Engineering*, 10, 2013.

[LCL+07] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, y B. Arnaldi. A review of classification algorithms for EEG-based brain–computer interfaces. *Journal of Neural Engineering*, 4:pp. 1741–2552, 2007.

[Mal89] Stephane G. Mallat. A theory for multiresolution signal decomposition: The wavelet rep- resentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:674–693, 1989.

[MdCF02] Julio Martínez Malo y Rosa María de Castro Fernández. Análisis de la teoría de ondículas orientada a las aplicaciones en ingeniería eléctrica: Fundamentos, 2002.

[MDD09] Nils Brede Moe, Torgeir Dungsøyr, y Tore Dybå. A teamwork model for understanding an agile team: A case study of a Scrum project. *Science Direct*, 2009.

[OAND10] I. Omerhodzic, S. Avdakovic, A. Nuhanovic, y K. Dizdarevic. Energy Distribution of EEG Signals: EEG Signal Wavelet-Neural Network Classifier. *International Journal of Biological and Life Sciences*, 6:pp. 210–215, 2010.

[oF13] U. o. Freiburg. Seizure Prediction Project,» University of Freiburg, 2013. 2013. https://epilepsy.uni- freiburg.de/freiburg-seizure-prediction-project/eeg-database. [Último acceso:25 01 2013].

[oFG13] University Hospital of Freiburg. Germany. Seizure Prediction Project. 2013. `freiburg.de/freiburg-seizure-prediction-project/eeg-database`.

[OHO11] U. Orhan, M. Hekim, y M. Ozer. EEG signals classification using the K-means clustering and a multilayer perceptron neural network model. *Expert Systems with Applications*, 38:pp. 13475–13481, 2011.

[Oli] José A. Olivas. Modelos y Aplicaciones de la IA. Algoritmos de Clustering y clasificación.

[Org05] World Health Organization. Atlas. Epilepsy care in the world. 2005.

[Org12] W. H. Organization. Epilepsy Fact sheet No 999. October 2012. [En línea]. Available: http://www.who.int/mediacentre/factsheets/fs999/en/index.html. [Último acceso: 22 01 2013.

[P97] Jallon P. Epilepsy in developing countries. *Epilepsia*, página 38:1143–1151, 1997.

[P02] Jallon P. Epilepsy and epileptic disorders, an epidemiological study. *Epileptic Disorders*, página 4:1–12, 2002.

[PAW13] Jussi Ronkainen Pekka Abrahamsson, Outi Salo y Juhani Warsta. Agile Software Development Methods: Review and analysis, 2013.

[PM08] L. M. Patnaik y O. K. Manyam. Epileptic EEG detection using neural networks and post-classification. *Computer methods and programs in biomedicine*, 91:pp. 100–109, 2008.

[Poo09] D.B. Poole. *Do It Yourself Agile*. 2009.

[RM] Lior Rokack y Oded Maimon. *Clustering Methods*, capítulo 15, páginas 322–352.

[San10] Conrad Sanderson. Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. 2010.

[SBS99] B. Schölkopf, C. Burges, y A. J. Smola. Advances in Kernel Methods: Support Vector Learning. *MIT Press*, 1999.

[SE08] A. Subasi y E. Erçelebi. Classification of EEG signals using neural network and logistic regression. *Computer Methods and Programs in Biomedicine*, 78:pp. 87–99, 2008.

[SEC+04] A. Shoeb, H. Edwards, J. Connolly, B. Bourgeois, S. T. Treves, y J. Guttag. Patient-specific seizure onset detection. *Epilepsy and Behavior*, páginas pp. 483–498, 2004.

[SFV07] O. Smart, H. Firpi, y G. Vachtsevanos. Genetic programming of conventional features to detect seizure precursors. *Engineering Applications of Artificial Intelligence*, 20:pp. 1070–1085, 2007.

[SJN07] John Stamatoyonnapoulos Shane J. Neph, Michael S. Kuehn. Wavelets Library. Department of Genome Sciences, University of Washington, 2007. `http://faculty.washington.edu/dbp/WMTSA/NEPH/wavelets.html`.

[Smi05] S.J.M. Smith. EEG in the diagnosis, classification, and management of patients with epilepsy. *Neurol Neurosurg Psychiatry*, 2005.

[SRDTB+12] María José Santofimia Romero, Xavier Del Toro, Jesús Barba, Julio Dondo, Francisca Romero, Patricia Navas, Ana Rubio, y Juan Carlos. López. A System for Epileptic Seizure Focus Detection Based on EEG Analysis. *UCAmI*, páginas 407–414, 2012.

[ST09] N. Sivasankari y K. Thanushkodi. Automated Epileptic Seizure Detection in EEG Signals Using FastICA and Neural Network. *Int. J. Advance. Soft Comput. Appl*, 2009.

[TTF07] A. T. Tzallas, M. G. Tsipouras, y I. Fotiadis. Automatic Seizure Detection Based on Time-Frequency Analysis and Artificial Neural Networks. *Computational Intelligence and Neuroscience*, 2007.

[WLSN12] J. Wang, P. Liu, M. Fenghua She, y A. Nahavandi. Bag-of-Words Representation for Biomedical Time Series Classification. 2012. CoRR, vol. abs/1212.2262.

[WSEG04] S. B. Wilson, M. L. Scheuer, R. G. Emerson, y A. J. Gabor. Seizure detection: evaluation of the Reveal algorithm. *Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology*, 115:pp. 2280–2291, 2004.

[yJS13] Ken Schwaber y Jeff Sutherland. *La Guía de Scrum™*. Scrum.org, edición 2, 7 2013.

[YZLW12] Q. Yuan, W. Zhou, Y. Liu, y J. Wang. Epileptic seizure detection with linear and nonlinear features. *Epilepsy and Behavior*, 24:pp. 415–421, 2012.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **arco-pfc** que se puede encontrar en:
`https://bitbucket.org/arco_group/arco-pfc`