UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

SmartBeds

Aplicación de técnicas de minería de datos para detección de crisis epilépticas

Presentado por Alicia Olivares Gil
en Universidad de Burgos — 18 de junio de 2019
Tutores: Álvar Arnaiz González y José Francisco Díez Pastor

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Álvar Arnaiz González, profesor del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Alicia Olivares Gil, con DNI 71299943N, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «SmartBeds – Aplicación de técnicas de minería de datos para la detección de crisis epilépticas».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 18 de junio de 2019

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. Álvar Arnaiz González	D. José Francisco Díez Pastor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

Introducción

La epilepsia es un trastorno neurológico provocado por la alteración de la actividad normal de una región cerebral, que desencadena crisis caracterizadas por convulsiones musculares reiteradas y, en ocasiones, pérdida de la consciencia. Se trata de una de las enfermedades neurológicas más habituales, y aunque las crisis epilépticos pueden experimentarse de forma aislada o durante periodos de tiempo limitados, una gran cantidad de la población las sufre de forma crónica. Según la Federación Española de Epilepsia [2] alrededor de 700 000 personas en España padecen o han padecido epilepsia a lo largo de su vida, y más de 200 000 la padecen de forma activa.

Para una persona con epilepsia crónica, la detección inmediata de una crisis es vital para permitir la aplicación de primeros auxilios que ayuden a evitar consecuencias permanentes. Actualmente, en la bibliografía se habla de varias técnicas para la detección automática de crisis [14, 24], la mayoría basadas en Electroencefalogramas (EEG) o en el uso de dispositivos portátiles (*wearables*) como pulseras inteligentes basadas en la monitorización de las constantes vitales del paciente.

La detección mediante EEG se usa principalmente para diagnósticos médicos, ya que los dispositivos que se necesitan son demasiado costosos o aparatosos como para ser usados en el día a día del paciente. Por otro lado, las pulseras inteligentes resultan más convenientes para este cometido, ya que son más baratas y cómodas de utilizar. Sin embargo, ambas técnicas requieren del uso consciente y continuado de los dispositivos de detección, lo que puede suponer un inconveniente para pacientes dependientes o con necesidades especiales. Por esta razón se propone el uso de colchones inteligentes para la detección automática de crisis nocturnas, mediante sensores de presión y biométricos incorporados en el interior del propio colchón.

Sea cual fuera el dispositivo utilizado, la captación de los datos (actividad eléctrica del cerebro, constantes vitales, presiones, etc.) no basta para detectar una crisis. Es necesario un procesado adecuado para determinar si estos datos corresponden o no con una crisis epiléptica. Para ello, las técnicas de minería de datos permiten generar modelos de clasificación capaces de realizar esta tarea. Para este trabajo de fin de grado, el principal objetivo será encontrar un modelo de clasificación efectivo mediante la aplicación de este tipo de técnicas sobre los datos disponibles.

1.1. Estructura de la memoria

Esta memoria incluye los siguientes apartados:

- **Objetivos del proyecto:** se definen los objetivos generales, técnicos y personales que se persiguen con la realización de este trabajo.
- **Conceptos teóricos:** TODO
- **Técnicas y herramientas:** TODO
- **Aspectos relevantes del desarrollo del proyecto:** TODO
- **Trabajo relacionados:** TODO
- **Conclusiones y líneas de trabajo futuras:** TODO

1.2. Materiales adjuntos

- **Anexos:**
  - Plan de Proyecto Software
  - Especificación de Requisitos
  - Especificación de diseño
  - Documentación técnica de programación
  - Documentación de usuario
- **Cuaderno de investigación:** recoge las explicaciones de todas las técnicas probadas, los resultados y las comparativas de todos los experimentos realizados con el fin de encontrar el mejor modelo de clasificación para el problema.

- **Experimentos:** Conjunto de notebooks de jupyter que contienen todos los experimentos realizados.
- **App de Android:** Archivo .apk para la distribución e instalación de la aplicación para Android desarrollada con el fin de mostrar la aplicabilidad del modelo de clasificación.

A la memoria y a todos los demás materiales adjuntos se puede acceder a través del repositorio del proyecto en GitHub: https://github.com/aog0036/TFG-SmartBeds.

Objetivos del proyecto

En este apartado se exponen los objetivos perseguidos con la realización de este trabajo:

2.1. Objetivos generales

- Investigar sobre técnicas del estado del arte aplicadas a problemas similares.
- Aplicar técnicas de minería de datos siguiendo los pasos del Descubrimiento de Conocimiento en Bases de Datos (*KDD*) [10].
- Explorar, aplicar y comparar distintas formas de preprocesado de los datos (filtrado, normalización, transformación, etc.).
- Usar técnicas de proyección de datos a dos dimensiones para comprobar si los instancias de «crisis» son fácilmente separables de las instancias de «no crisis».
- Probar modelos de clasificación para conjuntos de datos con preprocesados basados en estadísticas simples.
- Probar modelos de clasificación para conjuntos de datos con preprocesados basados en características de series temporales.
- Probar modelos de clasificación mediante *ensembles* para conjuntos de datos desequilibrados.
- Probar detección de anomalías mediante un modelo *One-Class*.

- Comparar el rendimiento de los modelos obtenidos.
- Comparar distintas métricas usadas para evaluar el rendimiento los modelos obtenidos.
- Generar un modelo de clasificación capaz de detectar crisis epilépticas a partir de los datos disponibles.
- Desarrollar una app de Android para mostrar la aplicabilidad del modelo de clasificación generado.

2.2. Objetivos técnicos

- Aprender LaTeX, en concreto su uso en herramientas de escritorio y online (tales como Overleaf).
- Realizar y exponer los resultados de los experimentos en notebooks de jupyter empleando código Python.
- Usar bibliotecas de minería de datos en Python, en concreto sklearn [12], para la proyección de los datos y para la obtención y evaluación de los modelos.
- Generar un conjunto de transformadores compatibles con sklearn para poder aplicar las técnicas de preprocesado de forma directa.
- Utilizar bibliotecas de procesamiento y visualización de datos tales como Pandas y Matplotlib.
- Usar la biblioteca tsfresh [1] para extracción de características en series temporales.
- Usar la herramienta Weka [7] para probar modelos de clasificación mediante ensembles en conjuntos de datos desequilibrados.
- Desarrollar una app Android con soporte para API 23 y superiores.
- Usar peticiones HTTP desde la app para la comunicación con la API del servidor remoto.
- Usar Socket.IO desde la app para la obtención de datos en tiempo real del servidor remoto.
- Usar la plataforma GitHub junto con la extensión ZenHub para la gestión del proyecto.

- Aplicar la metodología Scrum adaptada a un proyecto con fines educativos.
- Realizar test TODO.

2.3. Objetivos personales

- Iniciarme en el campo de la investigación.
- Explorar técnicas y herramientas aplicadas a la minería de datos.
- Aprender a generar documentación con LaTeX.
- Iniciarme en el desarrollo de aplicaciones Android.

Conceptos teóricos

Este proyecto se centra principalmente en investigación, por lo que tiene una importante carga teórica. En el apéndice del cuaderno de investigación se explica el uso de todas las técnicas que hemos empleado en los experimentos, especificando implementaciones concretas, parámetros utilizados y resultados obtenidos. Por otra parte, en este apartado se van a exponer los conceptos teóricos en los que se basan esas técnicas sin entrar en los detalles de implementación.

Dado que el desarrollo de la investigación se ha basado en el proceso *KDD* (explicado más detenidamente en el apartado de Técnicas y Herramientas), este apartado se estructurará en función de los pasos que incluye este proceso:

3.1. Limpieza de datos

Los datos de los que disponemos tienen los siguientes atributos:

- **MAC_NGMATT** y **UUID_BSN**: identificador del colchón asociado alos tubos de presión e identificador del sensor de biométricas respectivamente.
- **DateTime**: día y hora en la que fue tomada cada instancia de dato.
- **P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12**: presiones, en mBar, captadas por los 12 tubos de presión integrados en el colchón.
- **HR, RR, SV, HRV, B2B**: valores de las constantes vitales que corresponden con la frecuencia cardíaca (pulsaciones/minuto), la frecuencia respiratoria (respiraciones/minuto), el volumen sistólico (mili-

litros), la variabilidad del ritmo cardíaco (milisegundos) y el tiempo entre pulsaciones (milisegundos) respectivamente.

- **SS**: potencia de la señal relativa a la matriz de tubos de presión. Un valor superior a 400 se considera aceptable.
- **STATUS**: estado de medición del sensor de constantes vitales. 0=*low signal*, 1=*ok signal*, 2=*high signal*, 3= *close to overload*, 4= *close to max HR*.

Por otra parte se nos proporcionaron unos rangos aproximados de horas en los que el paciente tuvo un ataque epiléptico, lo que nos permite etiquetar cada instancia de datos como «crisis» o «no crisis».

Eliminación de instancias por baja señal

Dos de estos atributos, SS y STATUS, hacen referencia a la calidad de la instancia de los datos, uno haciendo referencia a la matriz de tubos de presión y otro al sensor de constantes vitales, y dado que la fase de limpieza de datos consiste en eliminar el ruido y las instancias incorrectas, es aquí donde eliminaremos todas las instancias cuyos datos no consideremos fiables en función de los valores de estos dos atributos.

Eliminación manual del ruido

Una opción para eliminar el ruido de los datos es hacerlo de forma manual, inspeccionando los datos disponibles y modificando o eliminando aquellos que sean inconsistentes.

Eliminación del ruido mediante filtrado

Otra forma de eliminar el ruido de una señal consiste en usar filtros. Un filtro es un sistema que se aplica a una señal (en nuestro caso la señal de presiones en el tiempo) y la transforma para conseguir un objetivo concreto, en este caso el suavizado de la señal para eliminar el ruido. Se han probado dos tipos de filtro distintos:

- **Filtro de Butterworth** [27, 22]: Diseñado para filtrado de señales eléctricas, trata de producir la respuesta más plana que sea posible hasta la frecuencia de corte (Wn), dicho de otra forma, trata de mantener intactas las frecuencias por debajo de la frecuencia de corte

mientras disminuye las frecuencias superiores con razón proporcional a N, siendo N el orden de filtrado:

$$|H| = \frac{1}{\sqrt{1 + (W/Wn)^{2N}}}$$

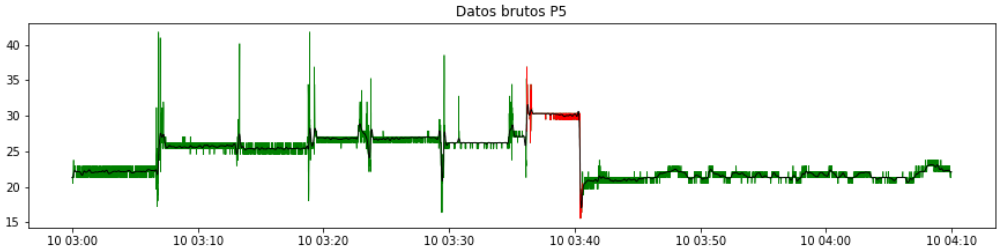

Figura 3.1: Ejemplo de señal de presión (P5) filtrada mediante un filtro de Butterworth con N=3 y Wn=0.05. En verde la señal original y en negro la señal filtrada.

- **Filtro de Savitzky–Golay** [26, 23]: Se basa en el cálculo de una regresión polinomial local para la que se debe especificar un tamaño de ventana (*window_lenght*) y el orden del polinomio utilizado en la regresión (*polyorder*).

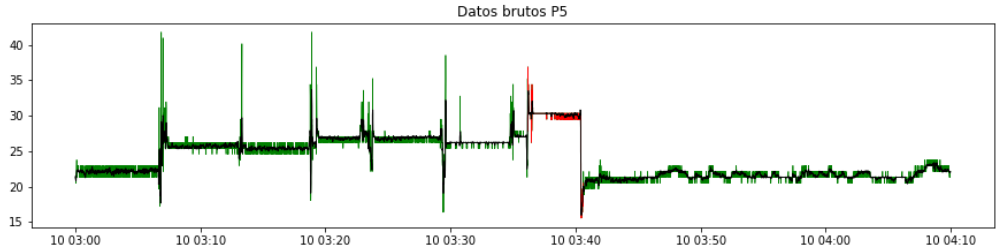

Figura 3.2: Ejemplo de señal de presión (P5) filtrada mediante un filtro de Svatzky-Golay con *window_length*=15 y *polyorder*=2. En verde la señal original y en negro la señal filtrada

Eliminación de atributos inconsistentes

Al trabajar con datos reales, cabe la posibilidad de que algunos de los atributos contengan datos inconsistente, ya sea por que se produjeron errores

(humanos o sistemáticos) durante la captación de los mismos o porque se corrompieron o perdieron durante su almacenamiento o su envío. Estos datos deberán ser eliminados para no interferir en la fase de minería de datos. Como se explicará en el apartado 4.5 de Aspectos Relevantes, en esta fase nos vimos obligados a eliminar los datos relativos a las constantes vitales.

3.2. Integración de datos

Los datos disponibles corresponden con las instancias recogidas a lo largo de varios meses de una sola cama (un solo paciente). Estos datos nos han sido proporcionados en varios archivos de extensión .csv, por lo que la fase de integración ha consistido en recoger los datos en varios subconjuntos, uno por cada noche, considerando que una noche corresponde con el periodo de tiempo en el que el paciente estuvo acostado en la cama de forma ininterrumpida.

3.3. Selección de datos

La selección de los datos consiste en desechar aquellos atributos o instancias que no resultan útiles para la extracción de conocimiento. A estas alturas del proceso seguimos teniendo datos relativos a la identificación del colchón, a la hora en la que se tomó la instancia de dato y a las presiones.

Eliminación de identificadores

Al contar con los datos de un único paciente, los atributos de identificación de la cama MAC_NGMATT y UUID_BSN no proporcionan ningún tipo de información útil para la extracción de conocimiento, por lo que ni siquiera son tenidos en cuenta en la fase de integración.

Eliminación de atributos de baja variabilidad

Este proceso consiste en detectar aquellos atributos que varían menos de un umbral determinado durante el tiempo, y por lo tanto no proporcionarán información relevante. Como se explicará en el apartado 4.5 de Aspectos relevantes, este proceso nos llevó a desechar la mitad de los datos relativos a los tubos de presión.

En este punto ya hemos seleccionado los atributos que serán considerados como **Datos en Bruto**: DateTime, P1, P2, P3, P4, P5 y P6.

Selección de atributos para el entrenamiento

Tras la fase de transformación de datos, la cual se abordará a continuación, se contará con una gran cantidad atributos calculados a partir de los datos en bruto. En la mayoría de los casos, resulta más eficiente entrenar el modelo de clasificación solo con un subconjunto de ellos que proporcione la mayor imformación, y por ello es necesario plantear estrategias para seleccionar el mejor subconjunto. En este proyecto se han planteado dos estrategias distintas:

- Escoger los atributos mejor situados en un ranking elaborado en función del rendimiento del clasificador en base a una métrica de evaluación concreta (estos conceptos se explican en fases posteriores).
- Emplear un algoritmo genético para encontrar el mejor subconjunto de atributos.

Algoritmo genético

Un algoritmo genético [8] es un método iterativo de búsqueda y optimización inspirado en la evolución biológica y el mecanismo de selección natural. En el contexto de un algoritmo genético una posible solución del problema se denomina «individuo», y en cada iteración el algoritmo trabajará con un conjunto de individuos llamado «población» o «generación». Los individuos tienen dos tipos de representación:

- **El genotipo:** es la representación del individuo que se empleará en las operaciones de cruce y mutación.
- **El fenotipo:** es la representación del individuo que se empleará en la operación de evaluación.

En cada iteración (o generación) se llevarán a cabo 4 operaciones:

1. **Selección** de los mejores individuos de la generación en base a una función de evaluación, que en nuestro caso se basará en el rendimiento de un clasificador entrenado con los atributos indicados por el individuo.
2. **Cruce** de algunos de los mejores individuos generando otros nuevos mediante la combinación de sus genotipos.
3. **Mutación** de algunos de los mejores individuos generando otros nuevos mediante la modificación de una pequeña parte de sus genotipos.

4. **Reemplazo** de los individuos de la anterior generación por los nuevos individuos generados.

Cada uno de estos pasos puede emplear distintos mecanismos o implementaciones en función del genotipo de los individuos, el tipo de problema y otras consideraciones a tener en cuenta.

3.4. Transformación de datos

Técnicas de reducción de la dimensionalidad

El primer tipo de transformación de datos que se utilizó fueron las técnicas de reducción de dimensionalidad [6]. Estas técnicas consisten en proyectar datos en un plano, reduciendo su dimensión (número de atributos de cada instancia, en nuestro caso seis) a dos. Este tipo de enfoques se basan en la hipótesis de que la dimensionalidad de los conjuntos de datos es artificialmente alta, y que se puede mantener la misma información con una representación de los mismos con una dimensionalidad menor.

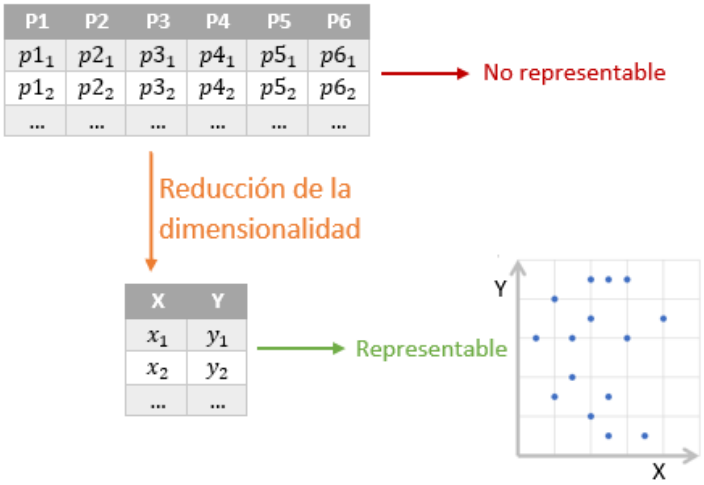

Figura 3.3: Abstracción de la reducción de la dimensionalidad de los datos.

Con esto, el objetivo que se persiguió fue comprobar si las instancias de «crisis» eran fácilmente separables de las instancias de «no crisis» en una representación bidimensional de los datos. Existen varios tipos de técnicas, cada una basada en unos criterios concretos para realizar la proyección, y se

pueden clasificar principalmente en transformaciones lineales y transformaciones no lineales.

Transformaciones lineales

Este tipo de transformaciones supone que los datos observados son combinación lineal de una cierta base, y en ellas se engloba el **Análisis de Componentes Principales** (*PCA*) [18].

PCA es una de las técnicas más utilizadas y consiste en escoger el nuevo sistema de coordenadas (o base) de forma que se maximice la varianza entre las instancias. Cabe destacar que para el nuevo sistema de coordenadas se puede escoger cualquier dimensionalidad menor o igual a la del conjunto original de datos. Sin embargo, a mayor dimensionalidad, mayor coste de computación, y no es posible representar en papel proyecciones de más de 2 dimensiones, por lo que solo se ha probado esta última.

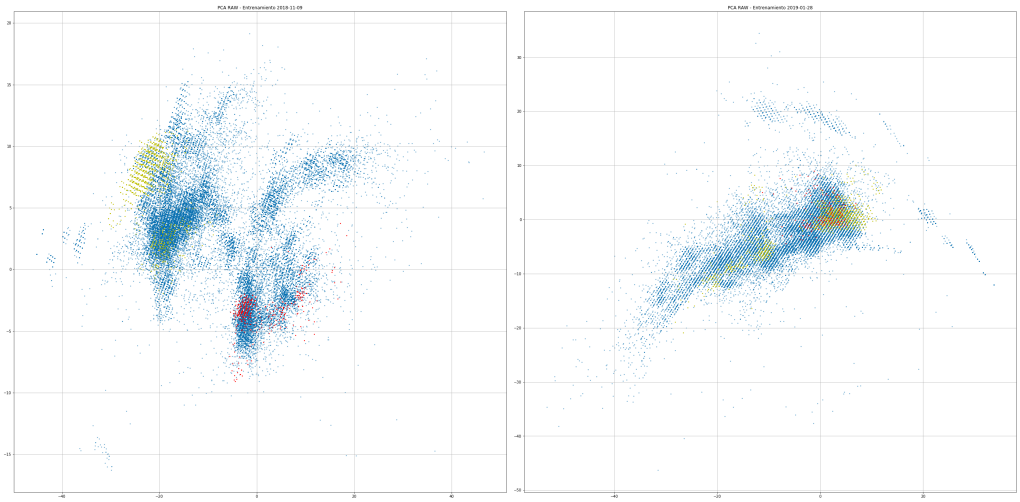

Figura 3.4: *PCA* aplicado a la noche de la crisis 1 (izquierda) y a la noche de la crisis 2 (derecha).

En la figura 3.4 los puntos rojos corresponden con instancias de «crisis» y los azules con instancias de «no crisis». Como se puede ver no existe separación entre ellas. Los puntos amarillos corresponden con instancias de la otra crisis, y se puede ver que tampoco comparten espacio.

Transformaciones no lineales

Las transformaciones no lineales [16], no asumen que los datos sean una combinación lineal de una base. Existen varias técnicas de este tipo y en nuestro caso se han probado con éxito las siguientes:

- ***Isomap* [19]:** Trata de reducir la dimensionalidad manteniendo las distancias geodésicas entre todas las instancias.

- ***Locally Linear Embedding* (*LLE*) [20]:** Reduce la dimensionalidad preservando las distancias dentro de los «vecindarios» locales. Se puede considerar como un conjunto de *PCA*s locales que se comparan globalmente para encontrar la mejor reducción no lineal.

- ***Multi-Dimensional Scaling* (*MDS*):** [21] Es una técnica bastante utilizada en marketing y ciencias sociales que se basa en la similitud o disimilitud de los datos. Busca reducir la dimensionalidad tratando las distancias como valores proporcionales a la disimilitud de las instancias (lo que se parecen entre ellas).

Como se aprecia en la figura 3.5 con algunas de estas técnicas se consigue un grado algo mayor de separación, pero las instancias siguen sin ser fácilmente separables. En este caso, y como se puede observar, hemos aplicado las técnicas no solo a los datos en bruto sino también a los datos estadísticos, lo que nos lleva al siguiente tipo de transformación.

Cálculo de estadísticas móviles

Las técnicas de minería de datos se pueden aplicar directamente a los datos en bruto, pero en ocasiones ofrecen un mejor rendimiento aplicadas a datos estadísticos calculados a partir de estos. Para calcular estadísticas móviles se debe definir el tamaño de la ventana (N), de forma que para cada ventana se calculará un valor estadístico a partir de los datos contenidos en ella. Se han usado tres estadísticas simples:

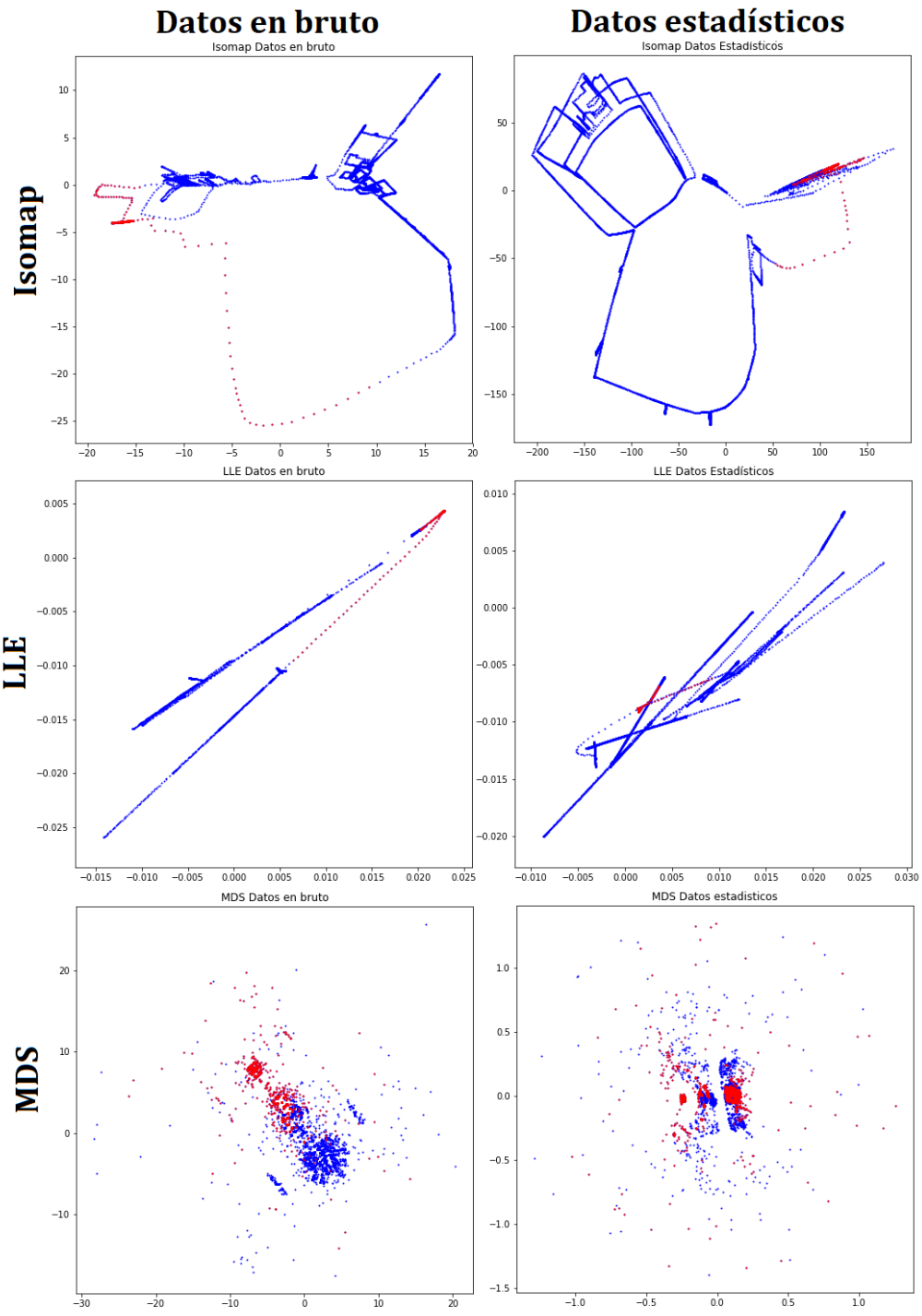

Figura 3.5: Comparativa de las técnicas de transformación no lineal.

- **Media móvil:** Es el valor promedio de los valores contenidos en la ventana.

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- **Desviación típica móvil:** Es una medida de la dispersión de los valores contenidos en la ventana.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

- **Rango móvil:** Es la diferencia entre el valor máximo y el valor mínimo de los contenidos en la ventana.

Cálculo de características de series temporales

Además de las descritas en el apartado anterior, se pueden calcular multitud de estadísticas más complejas, como es el caso de las características de series temporales.

Una serie temporal [28] no es más que una secuencia de instancias medidas en determinado momento de tiempo y ordenadas cronológicamente, por lo que nuestro conjunto de datos corresponde con la definición de serie temporal. Las series temporales se suelen usar para estudiar la relación causal entre diversas variables que cambian en el tiempo y pueden influirse entre sí. Para este cometido existen una serie de técnicas específicas basadas en cálculos estadísticos más complejos.

Para este proyecto se ha empleado la biblioteca tsfresh [1] que extrae algunas de esas características a partir de una serie temporal, y hemos tratado de usarlas para generar el modelo de clasificación.

3.5. Minería de datos

En esta fase se ha tratado de generar un modelo capaz de encontrar los patrones que definen una crisis epiléptica. Para ello el modelo debe ser capaz de aprender a partir de los datos, y en función de como lo haga se puede hablar de dos tipos básicos de aprendizaje: supervisado y no supervisado.

- Un modelo de **aprendizaje supervisado** se entrena con un conjunto de datos etiquetado, es decir, que cada instancia de dato de entrenamiento tiene asociado un valor categórico que ofrece información sobre a qué clase pertenece (para un problema de clasificación) o un

valor nominal (para un problema de regresión). Este es un problema de clasificación y las dos clases posibles son «crisis» o «no crisis». Tras el entrenamiento, el modelo deberá ser capaz de etiquetar nuevas instancias de dato distintas a las que se han usado en el entrenamiento como instancias de «crisis» o «no crisis» basándose en lo aprendido. En definitiva, el ciclo de vida de un clasificador supervisado tiene dos fases:

1. **Entrenamiento:** El modelo aprende a partir de un conjunto de datos etiquetado.
2. **Predicción:** El modelo predice la clase de un dato no etiquetado, es decir, cuya clase real se desconoce.

- En el **aprendizaje no supervisado** el modelo no tiene conocimiento sobre a qué clase pertenece cada instancia. Con este tipo de aprendizaje se pueden, por ejemplo, agrupar los datos en subconjuntos en función de sus características. Las técnicas de reducción de la dimensionalidad podrían ser consideradas técnicas de aprendizaje no supervisado, ya que elaboran la proyección sin conocer la clase de cada instancia, pero podrían darnos información sobre la existencia de dos clases si en las proyecciones bidimensionales existiera una clara separación entre dos grupos (*clusters*) de instancias.

Por otra parte, dentro de los modelos de clasificación podemos distinguir entre clasificadores simples o *ensembles*.

- Los **clasificadores simples** emplean un solo modelo que será el encargado de predecir la clase de las nuevas instancias.
- Los ***ensembles*** [9] son conjuntos de clasificadores simples cuyas predicciones se combinan para obtener una predicción final. La idea es construir varios modelos que no necesariamente predigan la misma clase para la misma instancia, y será la combinación de estas predicciones, por ejemplo, mediante votación, la que determinará la clase que predice el *ensemble*. Para que este sistema de clasificación tenga sentido los modelos que forman el *ensemble* deben ser algo distintos entre sí, de lo contrario no ofrecerían ninguna ventaja, y esa diferencia puede radicar en varios mecanismos, por ejemplo:
  - El proceso de construcción del modelo no es determinista, por lo que al entrenar varias instancias de ese modelo no se generan clasificadores exactamente iguales.

- El *ensemble* se compone de modelos obtenidos con distintos métodos, llamados métodos heterogéneos.
- Cada modelo se entrena con un conjunto de datos diferente.
- Cada modelo se entrena con un subconjunto de atributos diferente para el conjunto de datos.

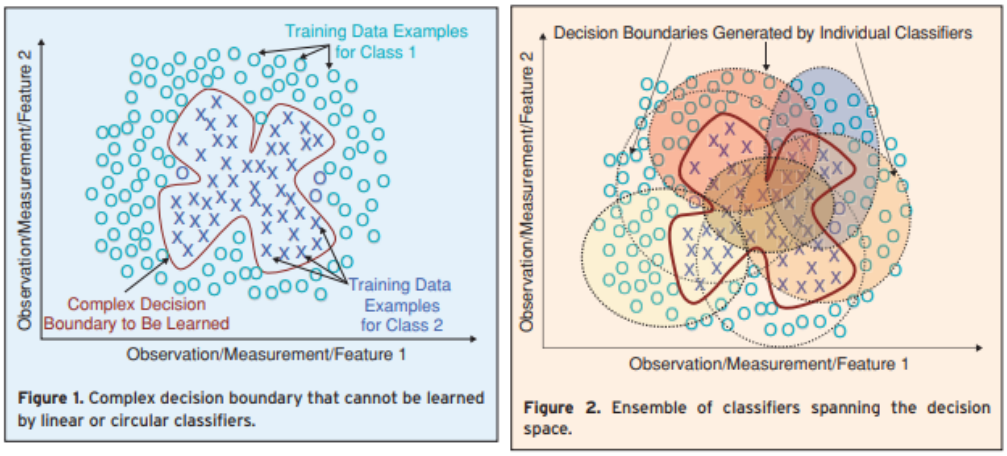

Figura 3.6: Uso de *ensembles* para problemas de clasificación complejos [13].

Teniendo en cuenta estas distinciones a continuación se exponen los modelos de clasificación que se han probado en este proyecto.

Random Forest

El clasificador *Random Forest* es un *ensemble* que emplea árboles de decisión, y se trata de uno de los más sencillos y utilizados. Para construir cada árbol emplea la técnica de *Bagging*, que consiste en entrenar cada modelo con un conjunto de datos del mismo tamaño que el original pero generado mediante muestreo uniforme con reemplazo (los datos pueden aparecer varias veces en el mismo conjunto). Mediante este mecanismo se reduce la varianza y se evita el sobreajuste.

Además, durante la construcción de los árboles que componen el *ensemble*, en cada nodo de decisión se tendrá en cuenta únicamente un subconjunto aleatorio de atributos del conjunto de datos.

Dado que todos los modelos del *ensemble* son independientes los unos de los otros, tienen la misma importancia, por lo que su voto tendrá el mismo peso a la hora de calcular la predicción final.

One-Class

La detección de anomalías One-Class consiste en entrenar el modelo con instancias de una sola clase de forma que a la hora evaluar nuevas instancias, devolverá información sobre si cada una pertenece o no a la clase basándose, de alguna forma, en su similitud con las instancias usadas en el entrenamiento.

Por ejemplo, si entrenamos el clasificador solo con instancias etiquetadas como «no crisis» y utilizamos datos de ambas clases para la predicción, esta puede devolver dos valores:

- La nueva instancia corresponde con la clase, por lo que se predice «no crisis».
- La nueva instancia no corresponde con la clase, por lo que se predice «crisis».

Durante el entrenamiento no se indica explícitamente a qué clase pertenece cada instancia, pero dado que todas las instancias de entrenamiento pertenecen a la misma, sí se le está proporcionando cierta información sobre la clase a la que pertenecen, y por lo tanto puede ser considerado como un algoritmo de aprendizaje supervisado.

AdaBoost.M1

Se trata de un *ensemble* para clasificación multi-clase que emplea aprendizaje supervisado. Emplea el método de *Boosting*, que a diferencia de *Bagging* es iterativo, y genera cada modelo del *ensemble* influenciado por el rendimiento del anterior. En este caso se busca que cada modelo dé más importancia a las instancias que son mal clasificadas por el modelo anterior. A diferencia de *Bagging* este mecanismo puede producir sobreajuste, sobre todo si el conjunto de datos tiene demasiado ruido.

Debido a la forma en la que se construyen los modelos simples que lo componen, a la hora de calcular la predicción final para una nueva instancia el voto de cada modelo tendrá un peso distinto en función de su precisión.

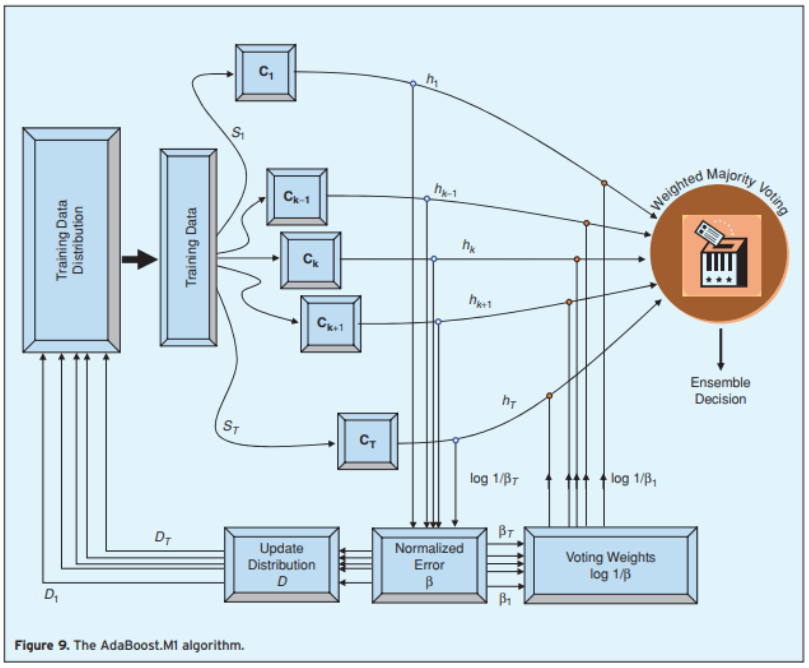

Figura 3.7: Funcionamiento del algoritmo AdaBoost.M1 [13].

Rotation Forest

Rotation Forest [15] es también un *ensemble* para clasificación que emplea aprendizaje supervisado mediante árboles de decisión. No se basa en un algoritmo iterativo como *Boosting*, por lo que los clasificadores simples pueden ser generados de forma paralela. Se trata de un método algo más complejo que los anteriores, basado en la aplicación del ya mencionado *PCA* a subconjuntos, preferiblemente disjuntos, de atributos. Además, al igual que en *Bagging*, cada árbol se entrena con conjuntos de datos construidos mediante muestreo con reemplazo a partir del conjunto original.

3.6. Evaluación de patrones

En el contexto de este trabajo, la utilidad de los patrones encontrados corresponderá con la precisión del clasificador en sus predicciones. Dado que al clasificar instancias nuevas no conocemos a priori la clase a la que pertenecen, no podemos saber si la clasificación ha sido correcta, por lo que para calcular la precisión de un clasificador se debe usar una partición de entrenamiento y test del conjunto original de datos disponibles:

- La **partición de entrenamiento** contiene la mayoría de los datos (por ejemplo, el 75 %), los cuales serán usados para entrenar el clasificador.
- El resto corresponden con la **partición de test**, que serán usados para comparar su clase real (su etiqueta) con la clase predicha por el clasificador entrenado.

La precisión corresponde con el porcentaje de acierto del clasificador para la partición de entrenamiento, pero en ocasiones, conviene usar métricas más complejas para medir el rendimiento de un clasificador. En este trabajo se ha trabajado con dos métricas distintas: el área bajo la curva ROC y la precisión media, pero antes de explicarlos es necesario presentar algunos conceptos básicos.

En un problema de clasificación binaria como este, si consideramos una instancia de «crisis» como un positivo (P) y una instancia de «no crisis» como un negativo (N) podemos definir los siguientes valores:

- Verdaderos positivos (VP): número de instancias de «crisis» que son clasificadas como «crisis».
- Falsos positivos (FP): número de instancias de «no crisis» que son clasificadas como «crisis».
- Verdaderos negativos (VN): número de instancias de «no crisis» que son clasificadas como «no crisis».
- Falsos negativos (FN): número de instancias de «crisis» que son clasificadas como «no crisis».

Estos valores se suelen visualizar en forma de matriz de confusión, la cual tiene la estructura de la figura 3.8.

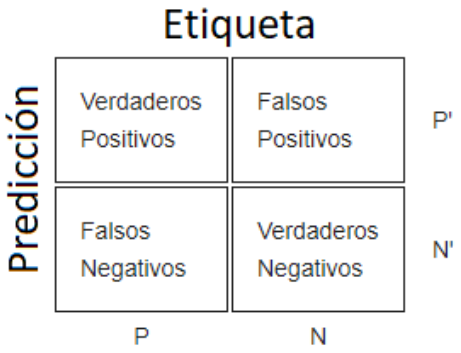

Figura 3.8: Disposición de la matriz de confusión4 [29].

Área bajo la curva ROC

A partir de los valores anteriores se pueden calcular los siguientes:

- **Sensibilidad o Razón de Verdaderos Positivos (VPR):**

$$VPR = \frac{VP}{P} = \frac{VP}{(VP+FN)}$$

- **Razón de Falsos positivos (FPR):**

$$VPR = \frac{VP}{P} = \frac{VP}{(VP+FN)}$$

Algunos clasificadores binarios, como los que se usan en este trabajo, además de predecir la etiqueta de la clase a la que creen que pertenece la instancia, son capaces de devolver la probabilidad entre 0 y 1 de que la instancia pertenezca a esa clase. Usando estas probabilidades podríamos clasificar las instancias en función de un umbral, por ejemplo, una instancia sería considerada «crisis» si la probabilidad de que pertenezca a esa clase es mayor o igual a 0.8 y «no crisis» si es menor.

El área bajo la curva ROC (*Receiver Operating Characteristic*) representa la relación entre la VPR y la FPR según se varía ese umbral. Se representará la VPR en el eje y, y FPR en el eje x. Cada punto corresponderá con los valores VPR y FPR resultantes de escoger cada uno de los posibles umbrales de clasificación, y la unión de esos puntos será lo que llamamos la curva ROC del clasificador.

Nos interesa que el VPR (*sensibilidad*) sea lo mayor posible y que el FPR ($1 - especificidad$) sea lo menor posible, por lo que el clasificador óptimo corresponderá con aquel que presenta un área bajo la curva ROC (AUC) igual a 1, mientras que el peor clasificador será aquel con un AUC igual a 0.5, ya que ofrecería una predicción equivalente a lanzar una moneda al aire.

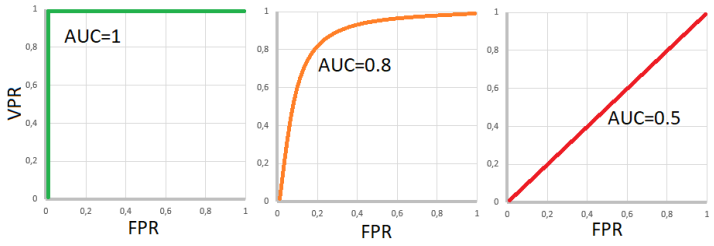

Figura 3.9: ROC. De izquierda a derecha: mejor caso, caso intermedio y peor caso.

Precisión media

Se definen las siguientes variables:

$$Precision = \frac{VP}{VP+FP}$$

$$Recall = \frac{VP}{VP+FN}$$

La precisión media (AV) [17] se define como lo siguiente:

$$AV = \sum_{i=1}^{n}(Recall_i - Recall_{i-1})Precission_i$$

donde $Precission_i$ y $Recall_i$ son los valores de *Precision* y *Recall* calculados para el umbral de clasificación i-ésimo, y n el número de posibles umbrales. Cuanto mayor sea este valor, mejor se considera el clasificador.

3.7. Presentación del conocimiento

En este caso una posible presentación útil del conocimiento adquirido sería la representación de la probabilidad de pertenencia a la clase «crisis» de cada instancia ordenadas cronológicamente, es decir, cómo varía la probabilidad

de crisis epiléptica del paciente con el tiempo en función del modelo. Esta será una de las gráficas que serán visibles en la aplicación de Android desarrollada.

Figura 3.10: Ejemplo de representación de la probabilidad de crisis en función del tiempo.

Técnicas y herramientas

En este apartado se presentan las técnicas metodológicas y las herramientas que se han usado en las distintas fases del desarrollo del proyecto.

4.1. Técnicas metodológicas

Scrum

Como se explica más detenidamente en el apartado de Planificación temporal del Apéndice A, para la planificación y desarrollo del proyecto se ha utilizado la metodología ágil *Scrum*, manteniendo el enfoque incremental de los *sprints* pero adaptándola al contexto de un trabajo con fines educativos.

KDD

En la fase de investigación se ha seguido el proceso de Descubrimiento de Conocimiento en Bases de Datos o *KDD* [10], dedicado a encontrar un modelo válido y útil en la medida en la que sirva para describir los patrones subyacentes de los datos. El término *KDD* suele ser empleado a menudo como sinónimo de minería de datos, pero esta corresponde en realidad con uno solo de los pasos del proceso. Se suele hablar de las siguientes fases englobados en el proceso de *KDD*:

- **Limpieza de datos:** Consiste en eliminar los datos inconsistentes o con ruido.
- **Integración de datos:** Se integran los datos de todas las fuentes disponibles en un formato uniforme y adecuado para los pasos posteriores.

- **Selección de datos:** Se trata de seleccionar solo aquellos datos relevantes para la tarea de análisis.
- **Transformación de datos:** Se realizan transformaciones y cálculos a partir de los datos en bruto con el fin de aplicar las técnicas de minería a las formas más apropiadas de los mismos.
- **Minería de datos:** Aplicación de técnicas para encontrar patrones subyacentes.
- **Evaluación de patrones:** Se estudia hasta qué punto los patrones encontrados son interesantes.
- **Presentación del conocimiento:** Consiste en ofrecer una representación comprensible y útil de los patrones y del conocimiento extraído.

Estos pasos no se aplican necesariamente de forma secuencial, ya que en muchos casos conviene volver a pasos anteriores tras la evaluación de los resultados del paso actual.

4.2. Herramientas en la fase de investigación

Anaconda

Anaconda es un administrador de paquetes y de entornos considerado un estándar para el desarrollo de minería de datos en lenguajes como Python y R. Al instalar Anaconda se tienen automáticamente disponibles más de 200 paquetes, además de ofrecer la posibilidad de añadir nuevos de forma sencilla.

Licencia: *New BSD License*

Jupyter Notebook

Los experimentos se han desarrollado en código Python distribuído en múltiples jupyter notebooks, ya que ofrecen un formato de estructuración y documentación del código muy adecuado para la investigación. *Jupyter Notebook* se encuentra disponible al instalar Anaconda.

Licencia: *Modified BSD License*

Scikit-Learn

Es la principal biblioteca empleada en la fase de investigación. Incluye modelos de clasificación, predicción y clustering de todo tipo y herramientas para entrenarlos, explotarlos y evaluarlos de forma sencilla. Está especialmente diseñada para operar con las bibliotecas *NumPy* y *SciPy*, y es compatible con *Pandas*.

Licencia: *New BSD License*

Weka

En algunas partes de la investigación se han usado modelos de Weka, otra plataforma para aprendizaje automático y minería de datos escrita en Java. Fue utilizada sobre todo para el entrenamiento de *ensembles* aplicados a conjuntos de datos desequilibrados.

Licencia: *GNU General Public License*

tsfresh

Es una biblioteca de Python dedicada al cálculo de grandes cantidades de características de series temporales. Permite calcular 64 tipos distintos de características y dispone de herramientas para filtrarlas en función de su relevancia. Es compatible con *Pandas*.

Licencia: *MIT License*

DEAP

Es el framework de python para computación evolutiva más utilizado. Se ha empleado como una de las alternativas para intentar encontrar la mejor selección de características de series temporales mediante un algoritmo genético, esperando que al aplicar este conjunto en la fase de minería se obtuviesen los mejores resultados posibles.

Licencia: *GNU Lesser General Public License v3.0*

tmux

Se trata un multiplexador de terminales que permite abrir varias sesiones simultáneamente y dejarlas corriendo en segundo plano. Esta herramienta ha resultado especialmente útil para la ejecución de los experimentos más

costosos en cuanto a tiempo de ejecución. Debido a que las ejecuciones de los experimentos se han llevado a cabo sobre un equipo de cómputo del grupo de investigación de los tutores mediante una conexión ssh que a menudo se cerraba en medio de un trabajo, ha sido necesario abrir una sesión de tmux en el equipo para cada una de estas ejecuciones de forma que, aunque se perdiera la conexión, el proceso siguiera corriendo en segundo plano y pudiéramos acceder a los resultados reactivando la sesión cuando el trabajo finalizase.

Licencia: *BSD License*

Otras bibliotecas relevantes

- *NumPy* y *Pandas* para la gestión, modificación y presentación de los datos.
- *Matplotlib* para la presentación gráfica de los resultados.
- *pickle* para la serialización de los datos y los resultados de los experimentos.

4.3. Herramientas en la fase de diseño de la aplicación

StarUML

Software de edición de diagramas UML utilizado en la fase de modelado.

Licencia: Propietaria aunque dispone de una demo gratuíta.

Material Design

Es una guía de estilo desarrollada por Google e integrada a partir de la versión *Lollipop* (5.0) de Android. Esta guía de estilos trata los elementos de la interfaz como elementos matariales, con unas dimensiones y una posición definida dentro del espacio (no solo en el plano, también en una tercera dimensión representada mediante el atributo *elevation*), propone una serie de dimensiones idóneas para cada tipo de elemento (textos, botones, tarjetas, etc.), y define la forma de generar el esquema de colores de la interfaz.

Pencil

Software de prototipado de interfaces gráficas. Permite incluir paquetes para incorporar elementos propios de *Material Design* a los prototipos, por lo que proporciona una visión más próxima al aspecto final de las interfaces de la aplicación que otros tipos de prototipado.

Licencia: *GNU Public License version 2*

4.4. Herramientas en la fase de desarrollo de la aplicación

Android Studio

Es el entorno de desarrollo integrado oficial de Android, disponible para Windows, GNU/Linux y MacOS. Android Studio incluye, entre otras muchas cosas, un editor de código, un editor gráfico de *layouts*, emuladores para todas las versiones de Android existentes, y soporte para construcción automática con Gradle.

Licencia: *Apache License 2.0*

Gradle

Herramienta para la automatización de la construcción del software para proyectos Java. Es la herramienta soportada de forma oficial por Android.

Licencia: *Apache License 2.0*

Android Support Library

Es la biblioteca que gestiona la compatibilidad de funciones de versiones avanzadas con su equivalente en versiones anteriores de Android. Además, incluye *layouts*, elementos y utilidades que no están disponibles en el framework oficial. Aunque la biblioteca recomendada actualmente para este cometido es *AndroidX*, la cual incluye las mismas funcionalidades y algunas más, se ha escogido *Android Support Library* por su simplicidad y por su documentación clara y completa, lo que resulta de mucha utilidad cuando se desarrolla una aplicación Android por primera vez.

Licencia: *Apache License 2.0*

MPAndroidChart

Es una biblioteca para generación de gráficos en aplicaciones Android. En este caso, y aunque no está pensada para ello, se ha utilizado para visualizar gráficas dinámicas cuyos datos se modifican en tiempo real. Para implementar esta funcionalidad se tuvo en cuenta también la biblioteca SciChart, al ser la biblioteca de referencia para generación de gráficos en tiempo real de Android, pero se descartó al tratarse de un software de pago.

Licencia: *Apache License 2.0*

Socket.IO-client Library

Es una biblioteca para gestión de comunicación mediante sockets para Java. El uso de esta biblioteca viene impuesto por la implementación de la API del servidor remoto, que gestiona el envío de datos en tiempo real de esta forma.

Licencia: *MIT License*

4.5. Otras herramientas generales

- **Git** como sistema de control de versiones distribuido.
- **GitHub** como plataforma para el *hosting* del repositorio del proyecto.
- **ZenHub** como extensión de GitHub para la gestión del proyecto basada en la metodología *Scrum*.
- **GitKraken** como cliente de Git mediante interfaz gráfica. (*GNU Public License*).
- **Overleaf** como editor colaborativo de LaTeX online para la generación del cuaderno de investigación conjunto.
- **TeXstudio** como editor de LaTeX para la generación de la memoria y los anexos. (*GNU General Public License v2*).
- **FileZilla** como aplicación para transferencia de ficheros. Soporta los protocolos FTP, SFTP y FTPS. (*GNU General Public License v2*).

Aspectos relevantes del desarrollo del proyecto

En este apartado se presentarán los aspectos relevantes del desarrollo del proyecto, exponiendo los problemas y las principales decisiones tomadas en cada una de las fases. Se distinguirá, de nuevo, entre la fase de investigación y la fase de desarrollo de la aplicación de Android.

5.1. Fase de invstigación

A continuación se expondrán las distintas situaciones que obligaron a tomar decisiones sobre cómo proseguir con la investigación y se explicará la razón por la que se tomó cada decisión concreta. También se hablará de detalles de implementación que fueron necesarios para abordar algunos de esos problemas.

Cabe destacar que la fase de investigación se hizo de forma conjunta con mi compañero de proyecto José Luis Garrido Labrador, y que las decisiones se tomaron entre los dos y con la ayuda de nuestros tutores.

Exploración del estado del arte

Como en cualquier proyecto de investigación, el paso inicial consistió en realizar una exploración del estado del arte sobre métodos y técnicas empleados para la detección automática de ataque epilépticos o de otros problemas similares, como la detección de caídas. En esta exploración se encontró que aunque la detección automática de ataques epilépticos es un campo muy explorado, la mayoría de las técnicas que se encuentran se basan

en el uso de Electroencefalogramas (EEG) y pulseras inteligentes basadas en la monitorización de las constantes vitales. Por otro lado, no se encontraron artículos relativos a técnicas o metodologías de aprendizaje a partir de los datos de sensores de presión instalados en colchones o similares.

Por otro lado, dado que el conjunto de datos disponibles es claramente desequilibrado (se tienen muchas más instancias de «no crisis» que de «crisis») también se realizó una búsqueda de técnicas aplicadas a este tipo de problemas, lo que nos llevó a los modelos de clasificación expuestos en el apartado 2.3 de Conceptos teóricos.

Etiquetado de los datos

Uno de los principales problemas que se tuvo que afrontar fue el del etiquetado de los datos, ya que, aunque los proveedores de los datos indicaron también un rango de horas en los que supuestamente tuvo lugar una crisis epiléptica, estos rangos eran muy aproximados y demasiado amplios para lo que dura una crisis epiléptica, según la bibliografía, sin que se produzcan consecuencias fatales o irreversibles.

Esto nos obligó a realizar una inspección visual de los datos, y ajustar los tiempos de la crisis teniendo en cuenta:

- Los rangos de horas proporcionados por los proveedores de los datos.
- Información relativa al aspecto de las señales en entornos próximos al inicio y final de esos rangos.
- Información relativa a las proyecciones obtenidas mediante las técnicas de reducción de la dimensionalidad, de forma que, si existía algún tipo de separación, se aplicaron las etiquetas de «crisis» a solamente aquellas instancias dentro de los rangos que mostraban una separación más clara con las instancias de «no crisis».

En la figura 5.11 se muestra la aplicación de la técnica de transformación MDS para el etiquetado de la primera crisis. Las instancias se han dividido en colores en función de su instante de tiempo respecto al rango proporcionado por los proveedores de los datos. Azul: antes de la crisis, Negro: después de la crisis, Verde: primeros 5 minutos de la crisis, Rojo: minutos centrales de la crisis, Naranja: últimos 5 minutos de la crisis. Como se puede observar, las instancias rojas presentan una separación mucho mayor con el resto, y además coinciden con aquellas cuyas señales de presión mostraban una

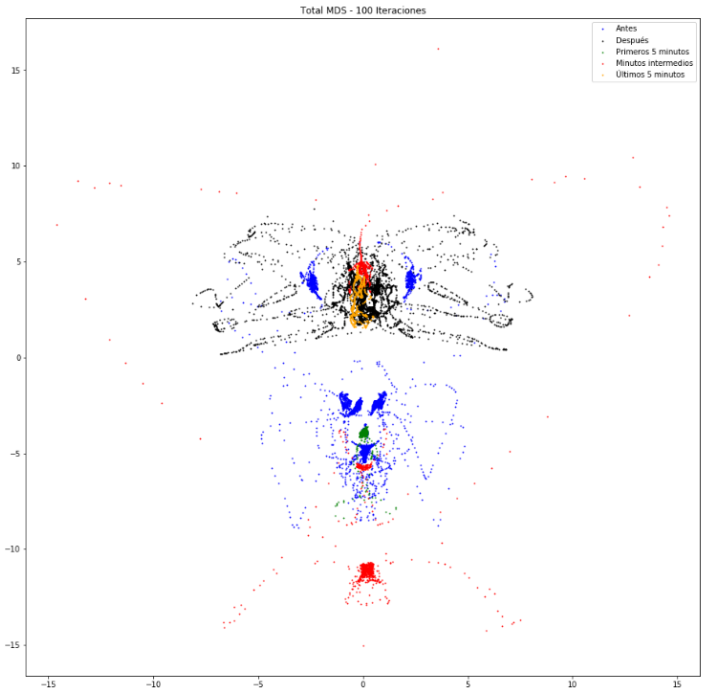

Figura 5.11: Aplicación de *MDS* para el etiquetado de la primera crisis.

diferencia más significativa dentro de su entorno, por lo que esas fueron las instancias etiquetadas como «crisis».

Se debe tener en cuenta que, aunque se realizaron estos ajustes, el etiquetado de los datos no puede ser considerado otra cosa que aproximado, y se asume que por esta razón los datos tendrán cierto grado de ruido. Este motivo junto con la escasez de datos de «crisis» en el conjunto de datos disponible, representan los mayores problemas a los que me he tenido que enfrentar en el proyecto.

Limpieza de datos

Del total de datos, hubo que desechar muchas de las instancias por distintas razones. En primer lugar se eliminaron aquellas instancias cuya señal era de mala calidad. Concretamente eliminamos todas aquellas instancias con un SS menor de 400 (como recomendaban los proveedores de los datos), pero no tuvimos en cuenta el valor de STATUS ya que, como se explicará más adelante, los valores de las constantes vitales no se usaron para la búsqueda del modelo de clasificación.

Por otra parte, mediante una inspección inicial de los datos se detectó que algunos datos de los tubos de presión en ocasiones eran negativos (lo cual no debería ocurrir), y aparecían valores bajos de presión en momentos en los que la cama debería estar vacía. Por esta razón, se decidió considerar todo valor de presión menor a 5 como ruido convirtiéndolo automáticamente a 0. De esta forma nos deshicimos también de los valores negativos. Para eliminar el ruido del resto de los datos se aplicó un filtro de Butterworth.

En esta inspección también se advirtió que los atributos referentes a las constantes vitales eran nulos de forma intermitente y en una gran cantidad de las instancias. Como supimos más adelante, esto se debió al mal funcionamiento del sensor durante las primeras etapas de recogida de los datos. Aunque algunas técnicas de minería de datos soportan la presencia de atributos desconocidos (*missing*) en algunas de las instancias, la cantidad de instancias con constantes vitales nulas o incorrectas era tan grande que se tomó la decisión de no tener en cuenta estos datos para el resto de experimentos. Cabe destacar que esto supone un gran inconveniente, ya que la mayoría de técnicas de detección automática encontrados en el estado del arte (quitando las centradas en EEG) se basan en datos biométricos.

Selección de atributos

A continuación se detectaron los datos con baja variabilidad para su eliminación. Tras llevar a cabo este proceso se eliminaron los atributos P7, P8, P9, P10, P11 y P12. Esto tiene sentido ya que esos campos corresponden con la matriz de tubos de presión de una de las mitades del modelo de colchón de matrimonio, y el colchón con el que se trabaja en este proyecto es individual. Por esta razón eliminar estos datos no supone ninguna pérdida de información, y además, reduce a la mitad la cantidad de datos con la que deben tratar los modelos, lo que repercute positivamente en los tiempos.

Transformadores

Una vez seleccionado el abanico de operaciones que íbamos a tener en cuenta para la búsqueda de un preprocesado óptimo, se codificaron estas operaciones como transformadores compatibles con sklearn para poder aplicarlas de forma sencilla y sistemática. Para ello se generaron una serie de clases que heredaban la clase `sklearn.base.TransformerMixin`. En total se codificaron 9 transformadores.

Elección del tamaño de ventana

Como se ha comentado en el apartado 2.3 de Conceptos teóricos, para calcular estadísticas móviles del conjunto de datos debemos escoger un tamaño de ventana. Para determinar el valor óptimo para este problema, se realizó un barrido de tamaños aplicados a la media y la desviación móviles, datos que se usaron para entrenar y testear un clasificador *Random Forest*. El proceso que se siguió se explica también con detenimiento y detalle en el cuaderno de investigación.

Para calcular las estadísticas móviles durante la exploración se usó nuestro transformador `StatisticsTransformer` que utiliza internamente las funciones `df.rolling(window_length).mean()` o `.std()` según el caso, donde `df` es el `pandas.DataFrame` que contiene los datos originales. Esta función devuelve un nuevo `pandas.DataFrame` con las estadísticas móviles calculadas para cada una de las ventanas (una fila por cada desplazamiento de la ventana). De esta forma, nos evita ir desplazando la ventana manualmente mediante un bucle.

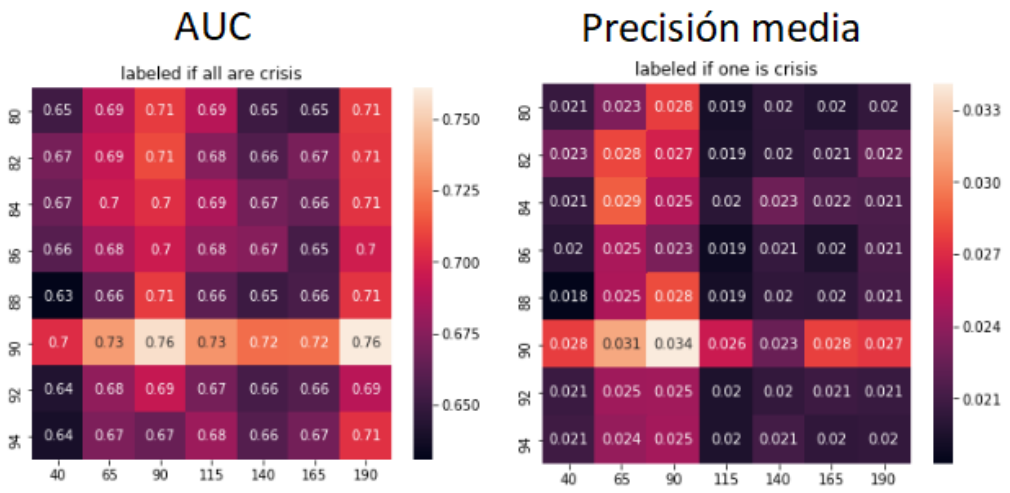

Figura 5.12: Mapa de calor de los resultados finales de la exploración para cada uno de las métricas de evaluación. En el eje y el tamaño de la ventana para la desviación típica, en el eje x el tamaño para la media.

Para cada una de las métricas de evaluación del clasificador (AUC y Precisión media) se hicieron una serie de barridos de parámetros, centrándose cada vez más en los rangos en los que se lograba un mejor rendimiento,

y llegando finalmente a los resultados que se muestran en la figura 5.12. Como se puede observar, los mejores valores se lograron para un tamaño de ventana de 90 tanto para la desviación típica como para la media, y también para ambas métricas de evaluación. Por esta razón se usó este valor en los experimentos y también como tamaño de ventana para la extracción de características de series temporales.

Extracción de características de series temporales para una ventana de datos

La extracción de características de series temporales aplicada a un cierto tamaño de ventana no es tan sencilla y merece un comentario sobre cómo se ha resuelto. La función encargada de la extracción de características pertenece la biblioteca `tsfresh` y se llama `extract_features`. Esta función recibe un `pandas.DataFrame` como parámetro y devuelve una fila de características por cada grupo de filas con el mismo «id» (debemos añadir un atributo «id» a nuestro conjunto de datos en bruto). Puedes indicarle mediante un diccionario pasado como parámetro el conjunto de características concreto que se desea calcular, o por el contrario, hacer que calcule todo su repertorio de características. Esta misma biblioteca ofrece una función llamada `utilities.dataframe_functions.roll_time_series()` que, a efectos prácticos, permite la extracción de características aplicada a una ventana, pero lo hace con un coste de memoria tan grande que no resulta práctica para conjuntos de datos como el nuestro. Básicamente, crea un `pandas.DataFrame` con un «id» distinto para cada ventana. Pongamos un ejemplo para comprender su funcionamiento: si tenemos un conjunto de datos con 10 filas y queremos aplicar una ventana de tamaño 3, esta función devuelve:

- 3 filas correspondientes a las filas 1, 2, y 3 del conjunto original y con un «id»=1.
- 3 filas correspondientes a las filas 2, 3, y 4 del conjunto original y con un «id»=2.
- así sucesivamente.

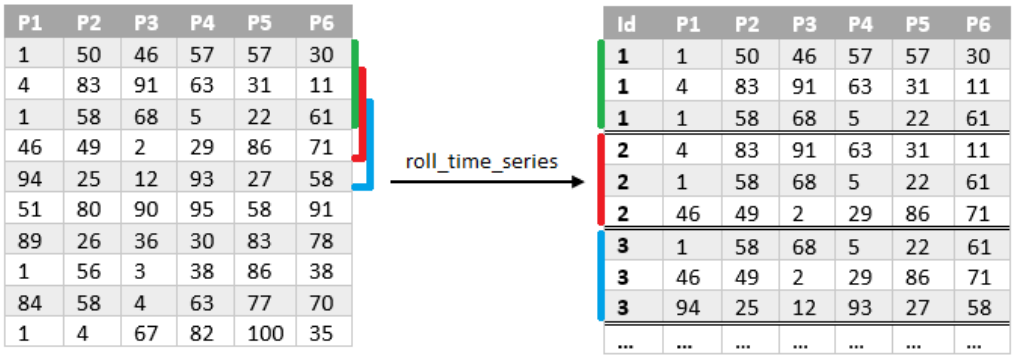

Figura 5.13: Ejemplo del funcionamiento de la función `roll_time_series`

Por cómo funciona en relación al atributo «id», si aplicamos este nuevo conjunto de datos a la función `extract_features` obtenemos lo que buscamos, una fila de características por cada ventana, pero el coste computacional y de memoria que supone la generación de este nuevo conjunto, que tiene una enorme cantidad de datos repetidos, es tan alto que no resulta viable. Por ello, en lugar de usar esta función, se tomó la decisión de realizar el desplazamiento de la ventana de forma manual mediante un bucle.

Filtrado de características de series temporales

Al ejecutar la extracción de características de las datos de las dos noches en las que tuvo lugar una crisis epiléptica, obtenemos 4 764 características en total, 794 por cada tubo de presión. Este número de atributos es excesivo para aplicarlo directamente a un clasificador, por lo que planteamos varias estrategias para quedarnos con aquellos que aporten mayor información. Estas estrategias se encuentran recogidas y explicadas en el cuaderno de investigación, por lo que aquí se expone la que, por presentar un mejor rendimiento, se empleó como entrada a métodos de selección posteriores.

Este filtrado consistió en usar la función `select_features` de la biblioteca tsfresh, que realiza una selección supervisada de las características (recibe el el valor de las etiquetas de cada instancia). Al aplicar esta función el número de características se redujo a 1 731. Además, asumimos que una característica solo sería relevante si resulta relevante para todos los tubos de presión, por lo que eliminamos aquellas que, tras el filtrado, no permanecieran para todos los tubos, quedándonos únicamente con 744 (124 para cada tubo de presión).

Selección final de características mediante algoritmo genético

Una vez filtradas las características más relevantes, es decir, que aportan más información, el último paso consiste en seleccionar un subconjunto más pequeño de ellas que sea suficiente para lograr un buen rendimiento del clasificador. De nuevo, se plantearon varias estrategias cuyo desarrollo se puede consultar en el cuaderno de investigación, y aquí se expondrá la que obtuvo un mejor rendimiento.

Esta estrategia es la que emplea un algoritmo genético implementado mediante la biblioteca DEAP. Una de las decisiones que hay que tomar a la hora de diseñar un algoritmo genético es qué tipo de genotipo se va a utilizar. En este caso se decidió que el genotipo sería un array unidimensional en el que cada gen (cada posición del array) contiene un número entero entre 0 y 123, que hace referencia a cada una de las 124 características resultantes del filtrado. El tamaño de este array corresponderá con el tamaño máximo de características que pueden ser seleccionadas, y se tomó la decisión de restringir este valor a únicamente 10 características, para que fuera cual fuera el tipo de clasificador escogido finalmente, la clasificación fuese rápida. Esto es importante ya que, en teoría, debe aplicarse a un sistema de recepción de datos en tiempo real.

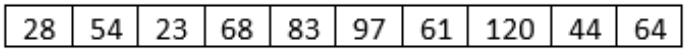

Figura 5.14: Genotipo del mejor individuo encontrado usando el AUC como métrica de evaluación.

Se aclara que 10 es el tamaño **máximo** de características porque, debido a los mecanismos de mutación y cruce utilizados, una característica puede aparecer más de una vez en un mismo individuo, en cuyo caso solo se tendría en cuenta una vez. Esta decisión se tomó porque de los mecanismos de cruce y mutación que ofrece DEAP, ninguno está preparado para trabajar con pseudopermutaciones (permutaciones en las que el tamaño del array no corresponde con el número de valores que puede contener). Aunque en DEAP es posible crear mecanismos de cruce y mutación personalizados que resolverían este problema, el beneficio que se obtiene con ello es mínimo o nulo. Esto se debe a que al tratar de maximizar el rendimiento, prevalecerán aquellos individuos que aporten mayor información al clasificar, es decir, aquellos con un mayor número de características.

Esto nos lleva a la decisión de cómo implementar la función de adaptación. Como ya hemos comentado, tenemos características relativas a dos noches en las que tuvo lugar una crisis epiléptica, y queremos realizar una evaluación lo más dura posible para que estemos seguros de que el clasificador vaya a predecir bien nuevas instancias. Por esta razón decidimos que la evaluación de los individuos se realizará mediante validación cruzada entre dos días (este tipo de evaluación se usó a menudo en otras fases de la investigación):

1. Se entrena un clasificador *Random Forest* con las características indicadas por el genotipo del individuo, pero solo con los datos pertenecientes a la noche de la primera crisis.

2. Se calcula el rendimiento del clasificador de acuerdo a la métrica pertinente (se usó tanto el AUC como la Precisión media) tomando los datos pertenecientes a la noche de la segunda crisis como partición de test.

3. Se realiza el proceso inverso.

4. La función de adaptación devolverá la media de los dos rendimientos obtenidos, el cual se tratará de maximizar.

Las últimas decisiones que se deben tomar en relación a la configuración del algoritmo son el tamaño de la población y el número de generaciones que se ejecutan. En ambos casos se escoge un valor de 50 teniendo en cuenta lo costosa que es la evaluación de cada individuo, y se confirma la decisión al ver que tras la ejecución se observa que no se producen mejoras en las últimas generaciones.

One-Class y *ensembles* para desequilibrados

Las características seleccionadas mediante el algoritmo genético no se probaron unicamente en un clasificador *Random Forest*, también se probaron en otros de los clasificadores comentados en el apartado 2.3 de Conceptos teóricos. Sin embargo, fue mi compañero José Luis Garrido Labrador el que se centró más en esta parte de la investigación, por lo que los aspectos relevantes relativos a estos clasificadores, así como a la detección de anomalías One-Class, se recogerán en su memoria.

Ejecución remota de experimentos con *tmux*

Como ya se ha explicado al hablar de tmux en el apartado 3.7 de Técnicas y herramientas, nos hemos topado con un problema al tratar de ejecutar

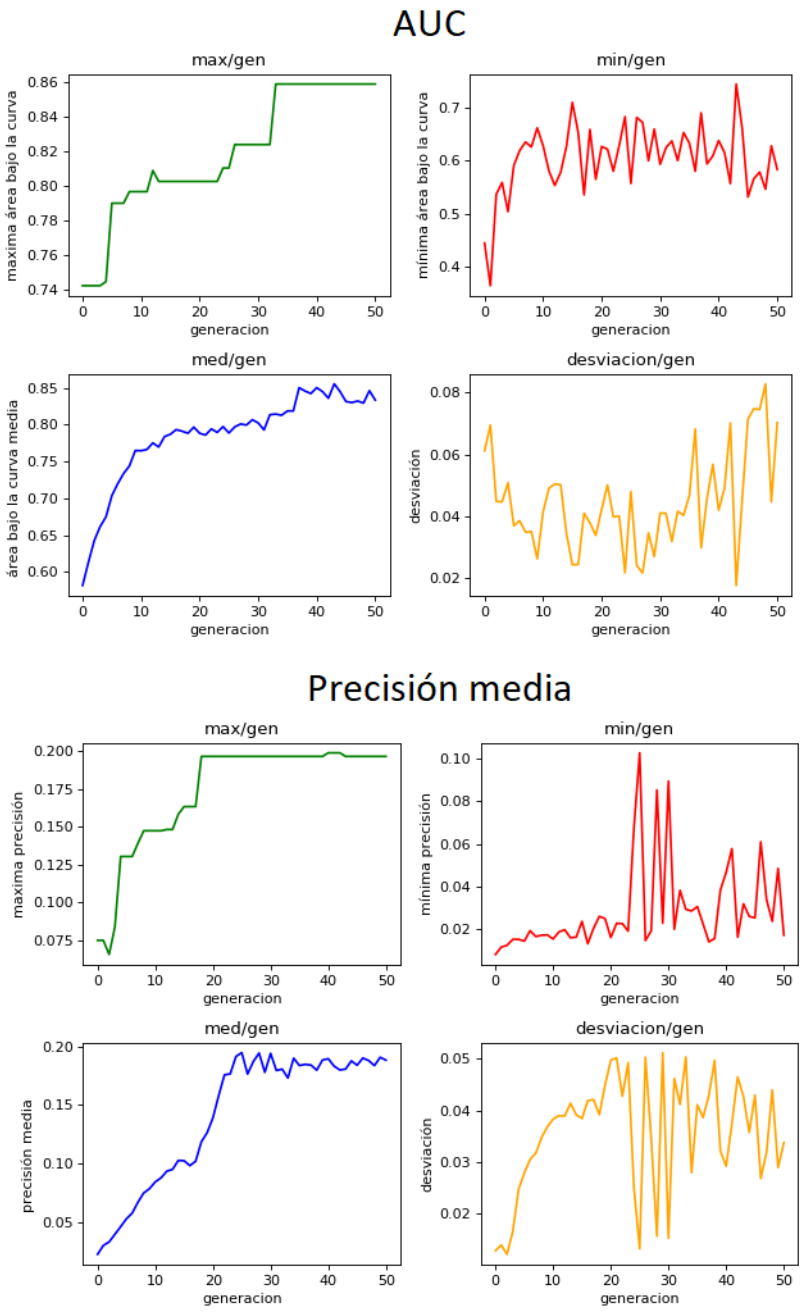

Figura 5.15: Evolución del algoritmo genético para cada una de las métricas de evaluación consideradas.

experimentos computacionalmente costosos en el equipo de cómputo del grupo de investigación de los tutores.

Normalmente, para lanzar los experimentos en el equipo, nos hemos conectado mediante ssh, y hemos lanzado el proceso de jupyter notebook sin interfaz gráfica mediante el comando `jupyter notebook --no-browser`. Este comando devuelve una URL que podemos copiar en nuestro navegador local para acceder a la interfaz gráfica del proceso y ejecutar los experimentos. Para ejecutar notebooks sin necesidad de acceder a la interfaz gráfica se ha usado el comando `jupyter nbconvert`.

Sin embargo, mediante este proceso, para experimentos muy costosos la conexión ssh se cerraba en mitad del trabajo y se perdían los resultados. Por ello, hemos usado el multiplexador de terminales tmux, que permite iniciar varias sesiones y ejecutarlas en segundo plano. De esta forma aunque se pierda la conexión ssh, el proceso seguirá corriendo en el equipo, y podremos acceder a los resultados una vez termine.

5.2. Fase de desarrollo de la aplicación

A continuación se expondrán los problemas y las decisiones que se tomaron referentes al desarrollo de la aplicación de Android. A diferencia de la fase de investigación, la fase de desarrollo se llevó a cabo de forma individual, pero debido a que la lógica de negocio se basó en la comunicación con la API desarrollada por mi compañero, algunas de las decisiones estuvieron condicionadas por las suyas.

Formación en Android

El desarrollo de una aplicación en Android requiere de unos conocimientos previos de los que no disponía antes de comenzar este proyecto, por lo que el primer paso consistió en buscar recursos que me permitieran iniciarme en el tema de la forma más eficiente posible. Estos recursos fueron principalmente los siguientes:

- El curso online gratuíto *Android Development for Beginners* de Google (Udacity) [25], que se estructura en los siguientes temas:

  - *User input*
  - *Multiple App Screens*
  - *Networking*
  - *Data Storage*
- La página web de *Android Developers* [3], que contiene la documentación oficial de las herramientas del SDK de Android y la API.
- La página web de *Stack Overflow* [11], donde se encontró solución a muchas de las pequeñas cuestiones que surgieron a lo largo del desarrollo.

El curso me permitió adquirir las bases necesarias para comenzar con el desarrollo de la aplicación, y en los demás recursos encontré solución a cuestiones de implementación más concretas.

Inicio del desarrollo

Para empezar, una de las primeras decisiones que se tomó fue para qué versiones de Android se iba a desarrollar la aplicación. Tras consultar el porcentaje de usuarios de cada versión de Android en el momento [5], se tomó la decisión de dar soporte a la versión 6.0 *Marshmallow* (API 23) y superiores, cubriendo un 74,8 % de los dispositivos Android del mercado. Aunque la versión 5.1 contaba con un porcentaje relativamente importante de usuarios, del 11,5 %, la aplicación resultante no iba a ser puesta en producción de inmediato, por lo que se consideró que las versiones soportadas eran suficientes.

Resumen de la arquitectura de la aplicación

La arquitectura puede verse desde dos puntos de vista distintos. Si consideramos el sistema como una entidad completa, incluyendo la aplicación cliente y la API del servidor, podemos hablar, como hace mi compañero en su memoria, de arquitectura Modelo-Vista-Controlador (MVC), dentro de la cual mi aplicación sería considerada como una Vista, ya que la lógica de negocio está enteramente contenida en la API.

Por otro lado, si hablamos de aplicaciones que trabajan directamente con los datos proporcionados por una API encontramos menciones a lo que comúnmente se denomina «arquetectura de microservicios». Según esta

arquitectura, cada funcionalidad está contenida en un proceso del servidor, de forma que cada parte de la lógica de negocio se encuentra encapsulada y los procesos son independientes entre sí. El cliente se limita a hacer peticiones a los microservicios de la API del servidor para acceder a la lógica de negocio.

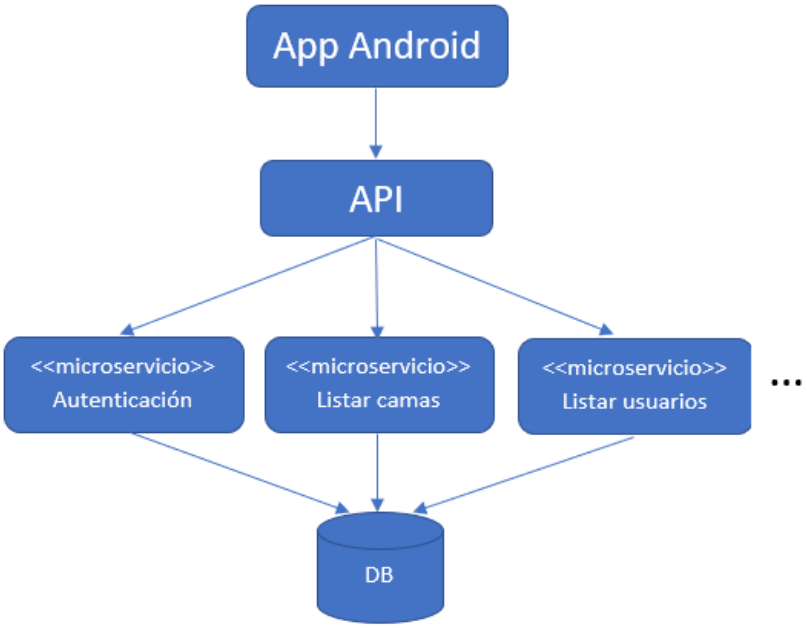

Figura 5.16: Resumen de la arquitectura de microservicios.

Comunicación con la API

Como se expone detenidamente en el apéndice de Especificación de requisitos, la aplicación ofrecerá, a grandes rasgos, tres funcionalidades principales:

1. **Gestión de usuarios:** permite gestionar qué usuarios y con qué credenciales pueden acceder al sistema. Esta funcionalidad solo estará disponible para el usuario con rol de admnistrador.

2. **Gestión de camas:** permite gestionar las camas instaladas y qué usuarios tienen acceso a sus datos. También estará disponible solo para el usuario administrador.

3. **Visualización de camas:** permite visualizar en tiempo real los datos captados por las camas a las que el usuario tiene acceso. El usuario

administrador tendrá acceso a todas las camas instaladas, el resto solo a las especificadas por el administrador.

Todas estas funcionalidades se basan en los servicios disponibles en la API de mi compañero. Las especificaciones de la API se exponen en el Manual del programador contenido en los anexos de su trabajo, donde se explica que se distingue entre dos tipos de comunicación:

1. La mayor parte de los servicios se basan en peticiones POST a rutas específicas en función del servicio. Este tipo de comunicación abarca servicios de autentificación, consulta y modificación de datos de gestión (tanto de usuarios como de camas).

2. De forma particular, el sistema que proporciona los datos captados por las camas en tiempo real se maneja a través de mensajes entre *WebSockets* usando la librería Socket.IO.

Peticiones POST

Dado que el funcionamiento de la aplicación depende casi enteramente de la comunicación con la API, debemos estar seguros de que existe conexión antes de realizar cada petición POST. Además, tal y como se implementa la API, un usuario solo puede mantener la autentificación en un dispositivo al mismo tiempo, de forma que, si el usuario «alicia» entra al sistema desde un dispositivo, y posteriormente lo hace desde otro, solo mantendrá la autentificación desde este último y perderá las credenciales desde el primero. Para gestionar estos dos problemas se define el siguiente esquema general de comunicación para cada petición POST:

1. Se comprueba que existe conexión a internet.
   a) Si existe se continúa.
   b) Si no existe se espera a recuperar la conexión o se pregunta al usuario si desea volver a la pantalla de inicio.
2. Se comprueba que el usuario sigue teniendo credenciales en el sistema.
   a) Si es así se continúa.
   b) Si no, se muestra un mensaje comunicando que se ha abierto sesión desde un dispositivo distinto y se redirige a la pantalla de inicio.
3. Se realiza la petición POST
   a) Si la respuesta es satisfactoria se refleja el resultado en la interfaz o se indica que la operación se ha llevado a cabo con éxito.
   b) Si la respuesta no es satisfactoria pero no se debe a un error del servidor, se muestra un mensaje de error al usuario.
   c) Si la respuesta no es satisfactoria debido a un error del servidor se muestra un mensaje de error y se redirige a la pantalla de inicio.

Además, se tomó la decisión de que cada petición a la API, tanto POST como mediante WebSockets para la recepción de mensajes en tiempo real, se gestione por un hilo independiente. De esta forma se pueden recibir y mostrar datos de varias camas al mismo tiempo de forma sencilla y transparente.

Comunicación en tiempo real con *Socket.IO*

Al emplear esta biblioteca en la API, también debe usarse en el cliente para que la comunicación en tiempo real sea efectiva.

A rasgos generales la comunicación se realiza en dos pasos:

1. Se conecta un *Socket* a la ruta genérica del servidor y se emite un evento «give_me_data».
2. Se conecta otro *Socket* a la ruta formada por por la concatenación de la ruta genérica y el namespace que identifica la cama que se desea escuchar. A través de este *Socket* se recibirán los eventos «package» con los datos en tiempo real.

Autenticación

La autenticación se resuelve de una forma muy sencilla desde el servidor. La primera petición POST que se debe realizar es la de autenticación, que recibe un nombre de usuario y una contraseña. Si el usuario tiene permiso para acceder al sistema, recibe un *token* que se guarda como variable de sesión. Todas las peticiones POST sucesivas recibirán como parámetro este *token*.

Actualización de la interfaz en tiempo real

Para la funcionalidad de visualización de camas, la aplicación va a recibir continuamente los datos captados por la cama en tiempo real, y queremos que la interfaz se actualice cada vez que llega un dato.

Dado que es un hilo el que gestiona la recepción de datos en tiempo real, para que la interfaz de usuario se actualice cada vez que se recibe un dato necesitamos implementar un mecanismo que permita que el hilo se lo notifique. La solución más sencilla que puede venir a la mente es el uso de un patrón observador.

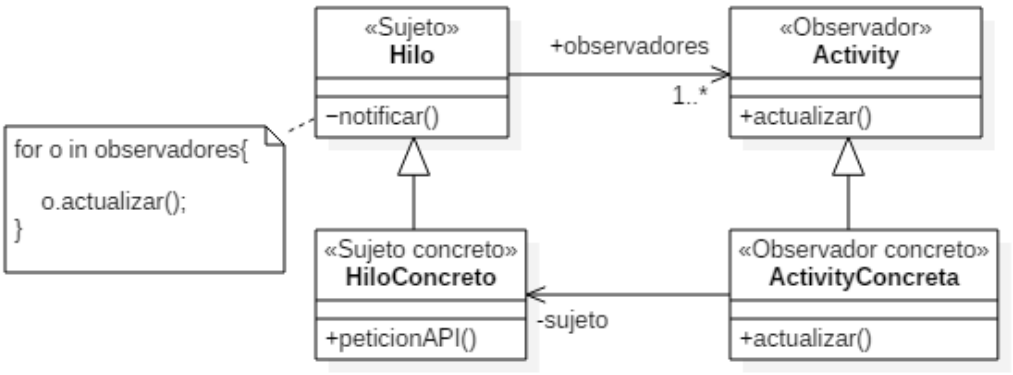

Figura 5.17: Posible aplicación del patrón observador.

Según se muestra en la figura 5.17, la interfaz (Activity) tendría una referencia al hilo que realiza la petición, el que a su vez tendría una referencia a la interfaz. Desde su referencia a la interfaz, el hilo podría ejecutar el método `actualizar()`.

Esta es una solución muy simple y efectiva para muchas plataformas, pero no funciona en el caso de Android, ya que la actualización de los elementos

de la interfaz (*Layouts*) solo se puede realizar desde el hilo principal. Al llamar al método `actualizar()` desde el hilo, saltaría una excepción si dentro de este método se manipula algún elemento de la interfaz.

Android ofrece varias alternativas para abordar este problema, y en mi caso he usado la que me ha parecido más sencilla: **el uso de *Handlers*** [4]. La idea es la misma, pero en este caso el *Activity* tiene una referencia a un objeto *Handler* que contiene el método `handleMessage`, el cual será el encargado de actualizar la interfaz. Cuando el *Activity* lanza el hilo, le pasa la referencia al objeto *Handler* como argumento. Cuando el hilo reciba un nuevo dato, se lo notificará ejecutando el método `sendMessage` del *Handler*. Esta llamada activará el método `handleMessage`, que a diferencia del método `actualizar` del patrón observador, sí se ejecuta en el hilo principal.

5.3. Aspectos relevantes generales

Este proyecto se me asignó gracias a la concesión de la **beca de colaboración de estudiantes en departamentos universitarios**, en este caso el departamento de Ingeniería Civil. Esta beca tiene como objetivo la implicación del estudiante en un proyecto de investigación, ofreciendo la posibilidad de ampliar sus conocimiento y concretar sus intereses para la posible incorporación a futuras tareas docentes o investigadoras.

La realización de este proyecto me ha permitido experimentar a grandes rasgos en qué consiste la labor de un investigador, y la experiencia, definitivamente, me ha llevado a considerar muy positivamente esta salida laboral.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[2] Federación Española de Epilepsia. Qué es la epilepsia. http://www.fedeepilepsia.org/epilepsia/que-es-la-epilepsia/, 2013. [Internet; consultado 11-Junio-2019].

[3] Android Developers. Android developers. https://developer.android.com/. [Internet; descargado 17-junio-2019].

[4] Android Developers. Handler. https://developer.android.com/reference/android/os/Handler. [Internet; descargado 17-junio-2019].

[5] Android Developers. Paneles de control. https://developer.android.com/about/dashboards/, 2018. [Internet; descargado 17-junio-2019].

[6] I.K. Fodor. A survey of dimension reduction techniques. *Meat Sci.*, 9:10–20, 01 2002.

[7] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[8] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[9] Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

[10] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 03 1996.

[11] Stack Overflow. Stack overflow. https://stackoverflow.com/. [Internet; descargado 17-junio-2019].

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, Third 2006.

[14] Sriram Ramgopal, Sigride Thome-Souza, Michele Jackson, Navah Ester Kadish, Iván Sánchez Fernández, Jacquelyn Klehm, William Bosl, Claus Reinsberger, Steven Schachter, and Tobias Loddenkemper. Seizure detection, seizure prediction, and closed-loop warning systems in epilepsy. *Epilepsy & behavior*, 37:291–307, 2014.

[15] Juan José Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2006.

[16] Scikit-Learn. Comparison of manifold learning methods. https://scikit-learn.org/stable/auto_examples/manifold/plot_compare_methods.html#sphx-glr-auto-examples-manifold-plot-compare-methods-py, 2019. [Internet; descargado 14-junio-2019].

[17] Scikit-Learn. Precision-recall. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html, 2019. [Internet; descargado 16-junio-2019].

[18] Scikit-Learn. sklearn.decomposition.pca. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA, 2019. [Internet; descargado 14-junio-2019].

[19] Scikit-Learn. sklearn.manifold.isomap. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html#sklearn.manifold.Isomap, 2019. [Internet; descargado 17-junio-2019].

[20] Scikit-Learn. sklearn.manifold.locallylinearembedding. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html#sklearn.manifold.LocallyLinearEmbeddingl, 2019. [Internet; descargado 17-junio-2019].

[21] Scikit-Learn. sklearn.manifold.mds. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html#sklearn.manifold.MDS, 2019. [Internet; descargado 17-junio-2019].

[22] SciPy. scipy.signal.butter. https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.butter.html, 2014. [Internet; descargado 17-junio-2019].

[23] SciPy. scipy.signal.savgol_filter. https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.savgol_filter.html#scipy.signal.savgol_filter, 2014. [Internet; descargado 17-junio-2019].

[24] Alexandros T Tzallas, Markos G Tsipouras, Dimitrios G Tsalikakis, Evaggelos C Karvounis, Loukas Astrakas, Spiros Konitsiotis, and Margaret Tzaphlidou. Automated epileptic seizure detection methods: a review study. In *Epilepsy-histological, electroencephalographic and psychological aspects*. IntechOpen, 2012.

[25] Udacity. Android development for beginners by google. https://eu.udacity.com/course/android-development-for-beginners--ud837, 2018. [Internet; descargado 17-junio-2019].

[26] Wikipedia. Filtro de savitzky–golay — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Filtro_de_Savitzky%E2%80%93Golay&oldid=85308976, 2015. [Internet; descargado 14-junio-2019].

[27] Wikipedia. Filtro de butterworth — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Filtro_de_Butterworth&oldid=90230742, 2016. [Internet; descargado 14-junio-2019].

[28] Wikipedia. Serie temporal — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Serie_temporal&oldid=109386775, 2018. [Internet; descargado 14-junio-2019].

[29] Wikipedia. Curva roc — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Curva_ROC&oldid=114286952, 2019. [Internet; descargado 17-junio-2019].