Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Clasificación de enfermedades cardíacas a partir de los parámetros del modelo 3DFMMecg

---

GRADO EN ESTADÍSTICA

Autor:
Adolfo Fernández Santamónica

Tutoras:
Cristina Rueda Sabater
Yolanda Larriba González

Agradecimientos

A todas aquellas personas que habéis estado a mi lado: no puedo expresar con palabras lo agradecido que me siento por todo vuestro apoyo a lo largo de este proyecto, ni tampoco podía entregar este trabajo sin dejar por escrito mi gratitud hacia vosotros.

A mis padres José Luis y María José, por vuestro apoyo y cariño incondicional y por celebrar con orgullo todos mis logros.

A mis hermanos David y Fabio, por vuestro constante apoyo y admiración por mi trabajo, que aún no entendiéndolo en profundidad, siempre me habéis preguntado por él con ilusión. Ojalá os sirva de referencia para vuestro futuro académico.

A mi tutora Cristina, por haberme ofrecido esta oportunidad de trabajo única y por haber sido una inagotable fuente de energía y motivación en todo momento.

A mi tutora Yolanda, por haber estado siempre dispuesta a echarme una mano con todo lo que he necesitado y por haberte involucrado tanto en mi trabajo.

Resumen

El electrocardiograma (ECG) es probablemente el método de diagnóstico no invasivo más importante de la medicina. Numerosos estudios han tratado el problema de la clasificación de enfermedades cardíacas mediante el uso de técnicas de *deep learning* generalmente, llegando a obtener una alta precisión en muchos casos. Sin embargo, la interpretación clínica que ofrecen es bastante limitada y el coste computacional llega a ser desorbitado.

Se propone un enfoque interpretable a través de la descomposición de los latidos del ECG en sus cinco ondas primordiales (P, Q, R, S y T) mediante el novedoso modelo 3DFMM$_{ecg}$, formulado simultáneamente para las 12 derivaciones del ECG. A partir de estadísticos construidos sobre los parámetros de dicho modelo y de distintas covariables demográficas, se han entrenado distintos modelos de *machine learning* y se ha llevado a cabo un análisis sobre la base de datos PTB-XL.

Se han obtenido resultados muy competitivos, de hasta 0,933 de macro AUC con tan solo 228 variables, algunas de las cuales han demostrado tener una gran nivel discriminante entre las clases.

En este trabajo, se han desarrollado diversos modelos construidos en base a los parámetros del modelo 3DFMM$_{ecg}$ que permiten hacer un diagnóstico automático interpretable y eficaz en tiempo real, de forma que pueda ser útil como apoyo en la toma de decisiones de los profesionales sanitarios.

Palabras clave: ECG, modelo FMM, aprendizaje automático, clasificación, PTB-XL

Abstract

The electrocardiogram (ECG) is probably the most important noninvasive diagnostic method in medicine. Numerous studies have addressed the problem of classifying cardiac diseases by using deep laerning techniques in general, achieving high accuracy in many cases. However, the clinical interpretation they offer is quite limited and the computational cost becomes exorbitant.

An interpretable approach is proposed through the decomposition of ECG beats into their five primordial waves (P, Q, R, S and T) by means of the novel 3DFMM$_{ecg}$ model, formulated simultaneously for all 12 ECG leads. Based on statistics constructed on the parameters of this model and different demographic covariates, different machine learning models were trained and an analysis was performed on the PTB-XL database.

Very competitive results of up to 0.933 macro AUC have been obtained with only 228 variables, some of which have been shown to have a high discriminant level between classes.

In this work, several models built on the basis of the parameters of the 3DFMM$_{ecg}$ model have been developed to provide an interpretable and efficient automatic diagnosis in real time, so that it can be useful as a decision support for health professionals.

Keywords: ECG, FMM model, machine learning, classification, PTB-XL

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. El electrocardiograma

La señal del electrocardiograma (ECG) [1] es un registro de la actividad eléctrica del corazón. Es probablemente el método de diagnóstico no invasivo más importante de la medicina y se utiliza desde hace décadas para la detección de alteraciones y patologías cardíacas con un riesgo mínimo para el paciente. Para hacer una interpretación del ECG, conviene familiarizarse con la fisiología del corazón. El corazón es un órgano muscular que se divide en cuatro cámaras: dos aurículas (derecha e izquierda) y dos ventrículos (derecho e izquierdo). Las aurículas están situadas en la parte superior y los ventrículos en la parte inferior. Durante el ciclo cardíaco, la sangre fluye de las aurículas a los ventrículos y, luego, es bombeada hacia las arterias durante la contracción ventricular. El corazón genera una señal eléctrica que es producida por la despolarización y repolarización de las células del músculo cardíaco. La despolarización ocurre cuando los iones de sodio (Na+) ingresan a las células musculares cardíacas, lo que provoca una carga eléctrica positiva en las células. Esto desencadena la contracción del músculo cardíaco y produce el primer sonido del corazón. La repolarización ocurre cuando los iones de potasio (K+) salen de las células musculares cardíacas, lo que restaura la carga eléctrica negativa en las células y permite que el corazón se relaje y se prepare para el siguiente latido. Esto produce el segundo sonido del corazón.

Para registrar el comportamiento del campo eléctrico producido por las células cardíacas, se colocan una serie de electrodos sobre la piel del paciente, tanto en las extremidades como en localizaciones específicas del tórax. Al igual que para realizar una fotografía no se suele tomar una única instantánea, ya que se quiere ver el resultado desde distintos ángulos, con el ECG sucede lo mismo. Las 12 derivaciones del ECG son las direcciones en las que se registra la actividad eléctrica del corazón, y dan lugar a distintas series de tiempo con las diferencias de potenciales (voltajes) medidas entre dos puntos. Estas series temporales se etiquetan con el nombre de su respectiva derivación y, según el plano eléctrico en el que se registren, se pueden clasificar en dos grupos: las derivaciones de las extremidades del plano frontal y las derivaciones precordiales del plano horizontal (ver Figura 1.1).

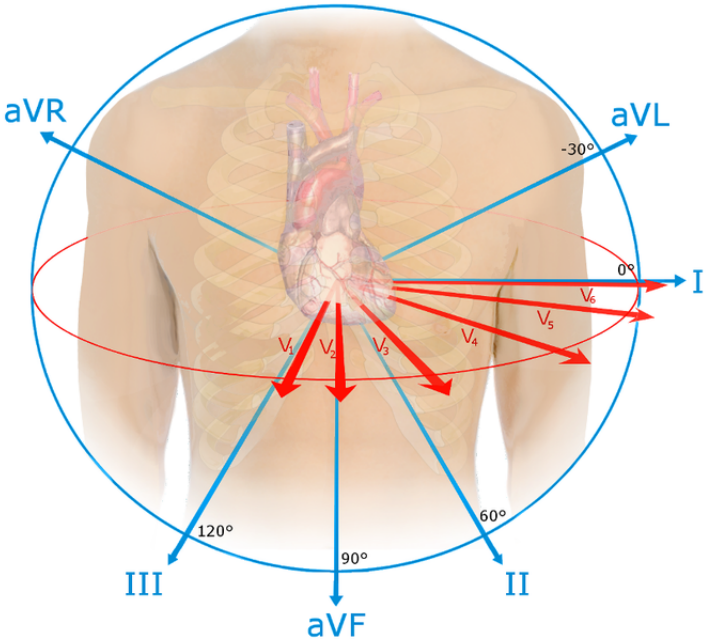


Figura 1.1: Orientación espacial de las derivaciones del ECG [2].

Las derivaciones del plano frontal registran las diferencias de potencial a través de los electrodos situados en las extremidades. Estas se pueden clasificar a su vez en otros dos grupos: derivaciones bipolares y derivaciones monopolares aumentadas. Las primeras son tres (I, II y III). Estas derivaciones forman lo que se conoce como el triángulo de Einthoven, ya que verifican una ecuación matemática conocida como la ley de Einthoven:

$$II = I + III \tag{1.1}$$

Por otro lado, las derivaciones monopolares aumentadas son también tres (aVR, aVL y AVF), y, matemáticamente, se pueden obtener mediante combinaciones lineales de las anteriores:

$$aVL = \frac{I - II}{2} \tag{1.2}$$

$$aVR = -\frac{I + II}{2} \tag{1.3}$$

$$aVF = \frac{II + III}{2} \tag{1.4}$$

Por último, las derivaciones del plano horizontal son monopolares, se nombran con una V mayúscula y el correspondiente número del 1 al 6 (V1, V2, V3, V4, V5 y V6).

El ECG posee información de todas estas derivaciones en latidos consecutivos a lo largo del tiempo. A la hora de representar el ECG, el formato más común es el que se describe en la Figura 1.2, el cual recoge información de las 12 derivaciones a lo largo de 10s (segundos). Está formado por una rejilla de colores rojizos o anaranjados en la que se colocan las derivaciones en color negro o azul para que destaquen. En el eje de abscisas se mide el tiempo y en el eje de ordenadas se mide el voltaje. Por norma general, cada cuadrado pequeño de la rejilla equivale 0,04s y 0,1mV (milivoltios).

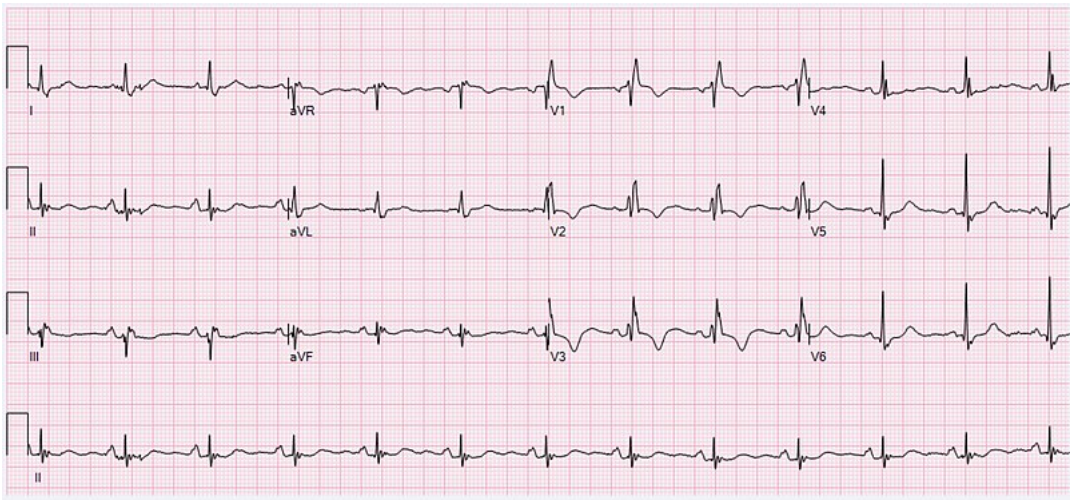


Figura 1.2: ECG de una mujer de 78 años con ritmo sinusal normal y bloqueo de rama derecha [3].

Si nos acercamos a visualizar un segmento en el que se recoja un único latido, por ejemplo en la derivación II, se puede observar que el latido del corazón está compuesto por diferentes ondas, las cuales representan los distintos eventos que se producen durante el ciclo cardíaco. En la Figura 1.3 se puede observar la morfología de un latido de un ECG normal, aunque las individualidades y patologías de cada paciente pueden provocar modificaciones sustanciales. La onda P es la primera del ciclo cardíaco y representa la despolarización auricular. A continuación, aparece el complejo QRS conformado por las ondas Q, R y S generalmente. Este complejo indica la despolarización de los ventrículos y la repolarización de las aurículas, que ocurren simultáneamente, aunque muchas veces la repolarización auricular queda oculta en el QRS y no llega a apreciarse en el ECG. Finalmente, la onda T cierra el ciclo a través de la repolarización ventricular.

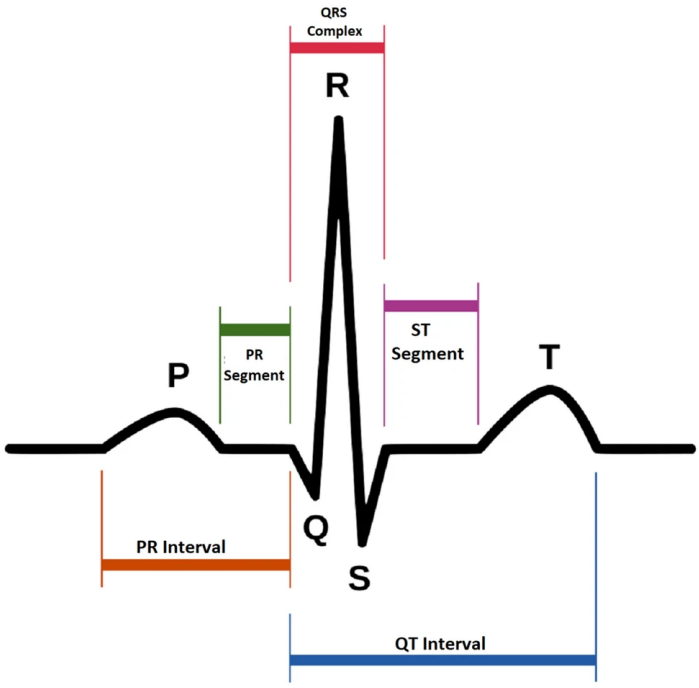


Figura 1.3: Morfología de un latido de un ECG normal típico [4].

También existen otra serie de elementos de interés clínico. Los segmentos PR y ST, miden la distancia entre los comienzos o finales de dos ondas. Los intervalos PR y QT, miden el tiempo

entre dos eventos de actividad eléctrica. El intervalo RR mide el tiempo entre las ondas R de dos latidos distintos.

Existen multitud de enfermedades cardíacas que pueden afectar la actividad eléctrica del corazón y, por tanto, producir cambios en los patrones del ECG. En este trabajo se ha llevado a cabo un estudio sobre la base de datos PTB-XL [5], la cual se describirá más en detalle en el Capítulo 3. Se han considerado las siguientes cinco clases de diagnósticos debido a su gran importancia clínica, y siguiendo el criterio de otros autores [6, 7]:

- CD (*conduction disturbance*): También conocido como bloqueo, muestra un problema con el sistema eléctrico del corazón, el cual regula su ritmo para asegurar un bombeo de sangre eficaz. En muchas ocasiones está ligado a la herencia genética del individuo.
- HYP (*hypertrophy*): La hipertrofia cardíaca es una enfermedad en la cual la pared del corazón es más gruesa de lo normal, dificultando así el bombeo de sangre. Una de las principales causas es una alta presión arterial. Se caracteriza por presentar unos picos prominentes en las ondas R del ECG.
- MI (*myocardial infarction*): El infarto de miocardio ocurre cuando una parte del músculo cardíaco no obtiene suficiente sangre. Normalmente ocurre por una obstrucción en los vasos sanguíneos. Es una enfermedad sumamente peligrosa y, por tanto, de vital interés.
- NORM (*normal*): El patrón eléctrico estándar que debería tener un ECG, con las cinco ondas principales y una frecuencia cardíaca de entre 60 y 100 latidos por minuto.
- STTC (*ST/T-wave changes*): Consiste en un trastorno en el segmento ST, lo cual indica una disfunción cardíaca. La causa puede ser variada, incluyendo la isquemia cardíaca o la inflamación del miocardio. Puede además ser un indicador de riesgo de otros eventos cardíacos como el infarto de miocardio.

En la Figura 1.4 se puede ver un ejemplo de señales de la derivación II de ECG representativos de cada una de estas clases.

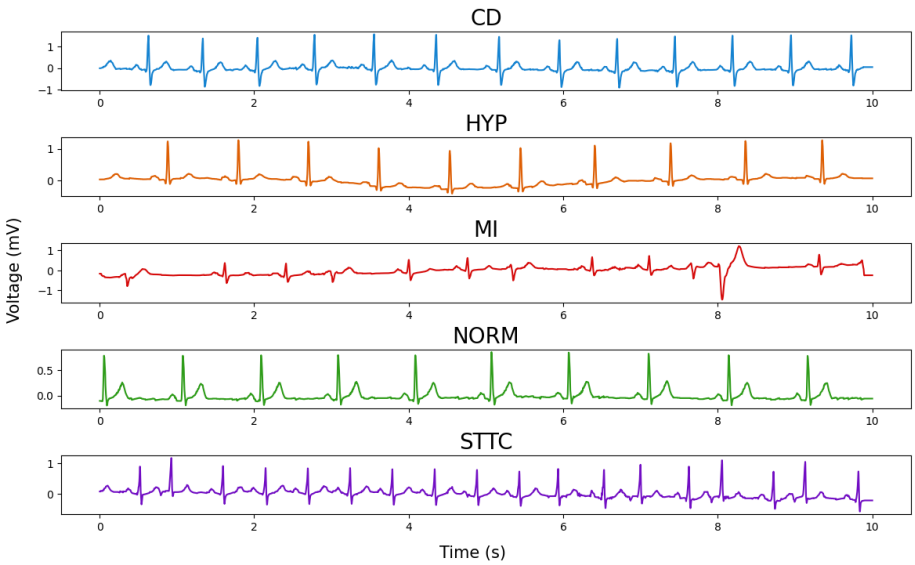


Figura 1.4: Derivación II de ECG representativos de las clases CD, HYP, MI, NORM y STTC.

1.2. Objetivos

La clasificación de ECG es un problema ampliamente estudiado en multitud de trabajos en la literatura, muchos de ellos, capaces de obtener resultados de clasificación muy precisos a través de complejos modelos de *deep learning* generalmente [8, 9, 10]. Estos modelos de aprendizaje profundo trabajan con la señal completa del ECG y cuentan con millones de parámetros cuya significación clínica es bastante limitada. Además, los cardiólogos aún se encuentran reticentes a este tipo de técnicas. Especialistas como el profesor Alan Fraser afirman que las redes neuronales son fundamentalmente "estúpidas" y que se las puede "engañar" fácilmente; con grandes cantidades de datos pueden encontrar patrones incluso aunque no existan, y mostrar correlaciones sin ningún tipo de causalidad [11].

Recientemente, el Grupo de Inferencia con Restricciones (GIR) [12, 13] de la Universidad de Valladolid ha desarrollado un modelo enormemente prometedor, formulado simultáneamente para las 12 derivaciones del ECG, el modelo 3DFMM$_{ecg}$ [14] (*3D Frequency Modulated Möbius*). Pese a que ya se ha comprobado su utilidad en el diagnóstico de algunas patologías cardíacas utilizando reglas de clasificación directamente sobre los parámetros del modelo, aún no se ha probado a combinarlo con el potencial de los algoritmos de *machine learning* (ML). La motivación de este trabajo nace de la necesidad de dirigir su desarrollo hacia un diagnóstico interpretable en tiempo real, teniendo por tanto los siguientes objetivos:

- Estudiar y familiarizarse con el modelo 3DFMM$_{ecg}$.
- Analizar el potencial del modelo 3DFMM$_{ecg}$ para la clasificación de enfermedades cardíacas.
- Entrenar y comparar diversos modelos de ML mediante diversas técnicas tales como la regresión logística, SVM, Random Forest, XGBoost y LightGBM.
- Conocer cuáles son las variables que más influyen en la clasificación de ECG.

Este proyecto está estrechamente relacionado con el Trabajo de Fin de Grado en Ingeniería Informática «ECGMiner: un software para la digitalización de electrocardiogramas» [15], en el cual se ha diseñado una herramienta de digitalización que permite obtener señales de ECG a partir de imágenes. Aunque los dos trabajos tienen objetivos claramente diferenciados, ambos tienen como denominador común el ECG y el modelo 3DFMM$_{ecg}$, por lo que su correspondiente explicación teórica será común en ambas memorias.

1.3. Estructura de la memoria

En cuanto a la estructura de este documento, se organiza de la siguiente manera:

Capítulo 1. Introducción. En este capítulo se describe el problema a tratar, los objetivos del trabajo y las asignaturas relacionadas.

Capítulo 2. Metodología. En este capítulo se explica el fundamento teórico del modelo FMM, de los algoritmos de aprendizaje empleados, de la metodología de evaluación de modelos y de la explicabilidad de modelos complejos.

Capítulo 3. Datos. En este capítulo se describen los datos utilizados así como el preprocesado que se ha realizado de los mismos.

Capítulo 4. Resultados. En este capítulo se explica el diseño experimental llevado a cabo y se exponen y discuten los resultados del ajuste de los distintos modelos.

Capítulo 5. Conclusiones. En este capítulo se exponen las conclusiones obtenidas tras la realización del trabajo y se proponen distintas líneas de trabajo futuro.

1.4. Asignaturas relacionadas

Las asignaturas del Grado en Estadística que han tenido más relevancia a la hora de realizar este trabajo son las siguientes:

- Análisis de Datos, Análisis Multivariante y Análisis de Datos Categóricos: introducen los fundamentos del problema de la clasificación y distintos modelos de predicción.

- Técnicas de Aprendizaje Automático y Minería de Datos: se presentan distintas técnicas de preprocesado de los datos y de evaluación de modelos.

- Modelos Estadísticos Avanzados: se estudian los modelos GLM, entre los que se encuentra la regresión logística.

- Métodos Estadísticos de Computación Intensiva: se introducen distintas técnicas de regularización y se explica su importancia para evitar el sobreajuste.

- Fundamentos de Programación, Paradigmas de Programación y Programación Orientada a Objetos: sientan las bases de la programación para producir un software de calidad.

Capítulo 2

Metodología

2.1. El enfoque FMM

El enfoque FMM (*Frequency Modulated Möbius*) es un procedimiento para el análisis de señales que compite con la clásica descomposición de Fourier o la descomposición en ondículas, combinando una formulación con muy buenas propiedades computacionales y estadísticas junto con la enorme ventaja que posee gracias a la interpretabilidad de sus parámetros y su robustez frente al ruido. En esta sección se presentará el modelo $3DFMM_{ecg}$ y se motivará con distintos argumentos matemáticos y biológicos.

2.1.1. Señales oscilatorias

Una señal oscilatoria [16] es una señal que se repite cíclicamente a intervalos regulares a lo largo del tiempo. Durante este periodo, la señal puede ser más simple y llegar a oscilar una única vez (ver Figura 2.1), o ser más complicada y tener más de una oscilación, como es el caso del ECG.

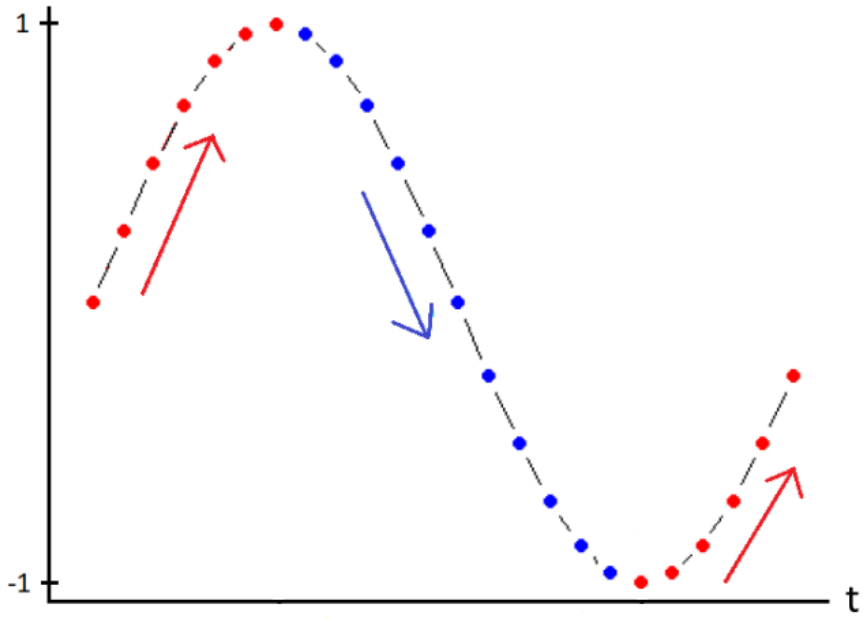


Figura 2.1: Esquema de una señal circular con una única oscilación. Adaptado de [16].

Las señales se pueden registrar en cualquier intervalo de tiempo, pero es habitual realizar una transformación del espacio euclídeo al círculo unidad, definiéndose así la señal como:

$$\mu(t) = \cos\left(\phi(t)\right) , \quad 0 < \phi(t) \leq 2\pi , \quad 0 < t \leq 2\pi \tag{2.1}$$

Un ejemplo de señal oscilatoria simple con una única oscilación es la señal $\cos\left(\phi(t)\right)$ tal que:

$$\phi(t) = \beta + 2\arctan\left(\omega \tan\left(\frac{t-\alpha}{2}\right)\right) ; \quad t \in (0, 2\pi] \tag{2.2}$$

A partir de ella, se define la señal compleja:

$$S(t) = \tau(c) = e^{i\phi(t)} \tag{2.3}$$

donde $\tau(c)$ se conoce como la transformación de Möbius, definida en el círculo unidad como:

$$\tau(c) = b\frac{c-a}{1-\bar{a}c} ; \quad a, b, c \in \mathbb{C} ; \quad |a| < 1 ; \quad |b| = 1 ; \quad |c| = 1 \tag{2.4}$$

De esta manera, una onda de Möbius es la parte real de una señal compleja definida por la transformación de Möbius:

$$W(t, \alpha, \beta, \omega) = \cos\left(\phi(t)\right) \tag{2.5}$$

Los parámetros que caracterizan la onda son el parámetro de localización α, el parámetro de anchura ω y el parámetro de dirección de onda β. A partir de este objeto matemático surge la onda FMM, la cual añade un cuarto parámetro de escala A: $AW(t, \alpha, \beta, \omega)$. En la Figura 2.2 se pueden observar distintos ejemplos ilustrativos de como cambian las ondas a medida que se varían los distintos parámetros de las mismas.

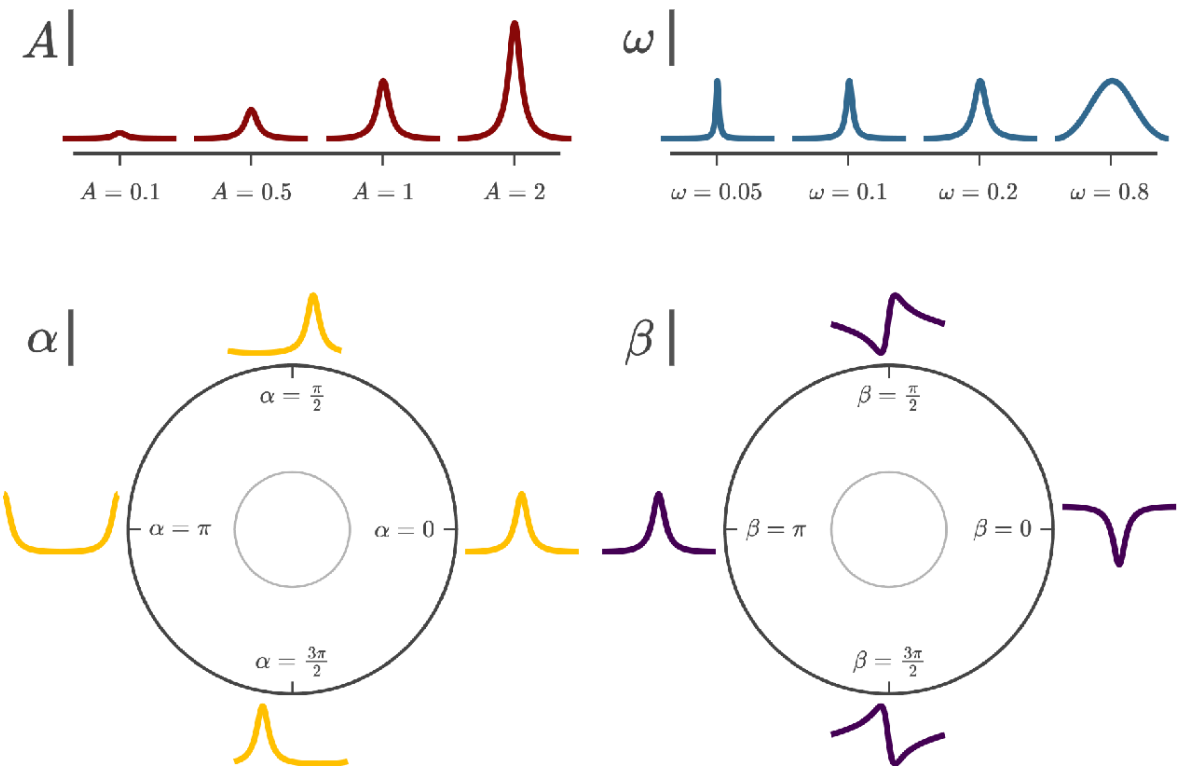


Figura 2.2: Ondas FMM de la forma $AW(t, \alpha, \beta, \omega)$ para varios valores de los parámetros $(A, \alpha, \beta, \omega)$. Salvo que se indique lo contrario $A = 1$, $\alpha = 0$, $\beta = \pi$ y $\omega = 0{,}2$ [17].

2.1.2. El modelo FMM$_{ecg}$

En la práctica, muchas de las señales oscilatorias asociadas a procesos biológicos, como son las señales del ECG, cuentan con más de una oscilación. De ello, surge la necesidad de formular modelos con varias componentes que tengan en cuenta la presencia de múltiples oscilaciones.

El modelo FMM$_{ecg}$ [18] es un modelo paramétrico multicomponente, formulado como la suma del conjunto de ondas de una derivación del ECG, a lo que se le añade además un término de ruido. Sea $X(t_i)$ la observación i-ésima del voltaje medido sobre una derivación $(t_1 < ... < t_n)$, se define el modelo:

$$X(t_i) = M + \sum_{J \in \{P,Q,R,S,T\}} A_J W(t_i, \alpha_J, \beta_J, \omega_J) + \epsilon(t_i) \tag{2.6}$$

donde

1. $M \in \mathbb{R},\ \beta_J \in (0, 2\pi],\ \omega_J \in [0, 1],\ A_J \in \mathbb{R}^+$
2. $\alpha_P \leq \alpha_Q \leq \alpha_R \leq \alpha_S \leq \alpha_T$
3. $(\epsilon(t_1), ..., \epsilon(t_n))' \sim N_n(0, \sigma I)$

El parámetro M es el *intercept* o término independiente del modelo, el cual describe el nivel de voltaje base de la señal. La restricción entre los parámetros α indica que las ondas se encuentran localizadas de acuerdo al orden biológico $(P \to Q \to R \to S \to T)$. Se asume que los errores $\epsilon(t_i)$ son independientes y están normalmente distribuidos con media 0 y desviación típica σ.

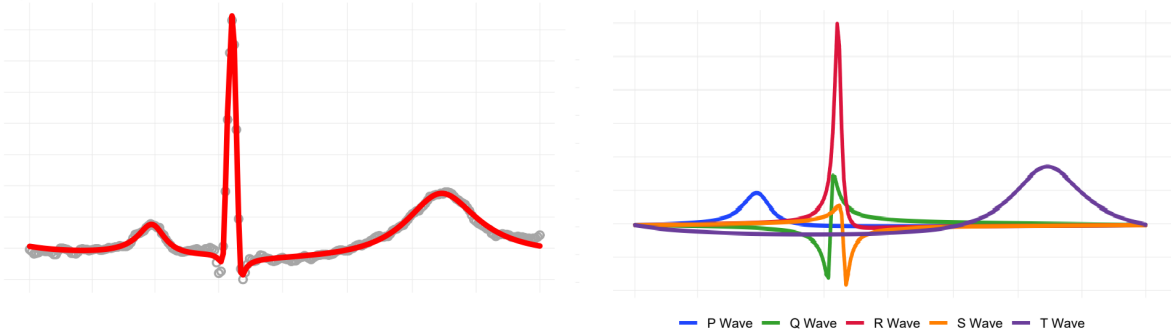


Figura 2.3: Ejemplo ilustrativo del ajuste del modelo FMM$_{ecg}$ sobre un latido típico de un ECG. Adaptado de [19].

2.1.3. El modelo 3DFMM$_{ecg}$

Los modelos cardíacos parten de la premisa general de que el campo eléctrico en el corazón es un proceso tridimensional representado por un vector $\vec{D}(t)$ que varía su magnitud y su dirección con el tiempo [20, 21]. Habitualmente $\vec{D}(t)$ describe tres bucles. Los bucles P y T son elípticos, mientras que el bucle QRS tiene una forma más irregular (ver Figura 2.4). Por otro lado, las señales del ECG son proyecciones de $\vec{D}(t)$ en 12 direcciones de dicho vector, las conocidas como derivaciones $(Lset \in \{I, II, III, aVR, aVL, aVF, V1, V2, V3, V4, V5, V6\})$. Adicionalmente, dichas señales son observadas con un cierto error.

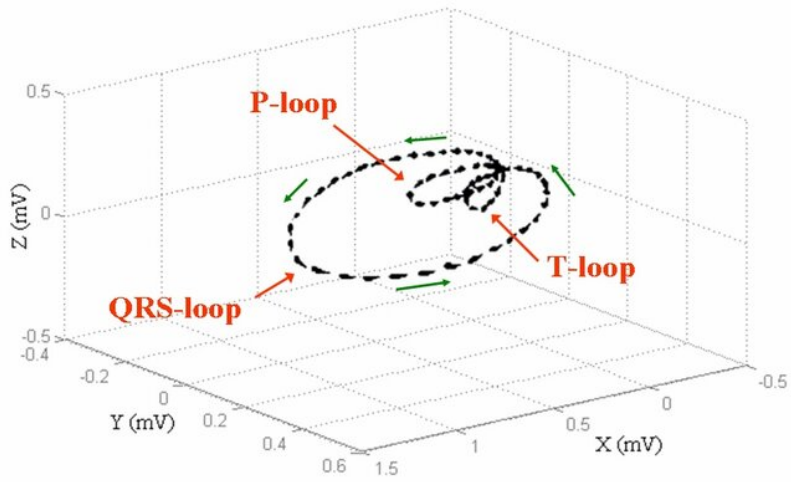


Figura 2.4: Trayectoria tridimensional que describe el vector $\vec{D}(t)$ para un ciclo cardíaco normal. Adaptado de [21].

La novedad del modelo 3DFMM$_{ecg}$ [14] parte de la asunción de que $\vec{D}(t)$ combina las señales eléctricas de exactamente cinco fuentes diferenciadas, correspondiéndose con las cinco señales primordiales del ECG:

$$\vec{D}(t) = \vec{d}_P(t) + \vec{d}_Q(t) + \vec{d}_R(t) + \vec{d}_S(t) + \vec{d}_T(t) \tag{2.7}$$

Por tanto, el vector $\vec{D}(t)$ se puede expresar como una suma de ondas, cada una de las cuales se corresponde con una onda FMM. Estas ondas se encuentran sincronizadas, lo que quiere decir que los parámetros α y ω son compartidos por todas las derivaciones. Sea $X^L(t_i)$ la observación i-ésima del voltaje medido sobre la derivación L, $(t_1 < ... < t_n)$, $(t_i \in (0, 2\pi])$, se define el modelo 3DFMM$_{ecg}$ como:

$$X^L(t_i) = M^L + \sum_{J \in \{P,Q,R,S,T\}} A_J^L W(t_i, \alpha_J, \beta_J^L, \omega_J) + \epsilon^L(t_i) \tag{2.8}$$

donde, para $L \in Lset$ y $J \in \{P, Q, R, S, T\}$:

1. $M^L \in \mathbb{R},\ \beta_J^L \in (0, 2\pi],\ \omega_J \in [0, 1],\ A_J^L \in \mathbb{R}^+$

2. $\alpha_P \leq \alpha_Q \leq \alpha_R \leq \alpha_S \leq \alpha_T$

3. $(\epsilon^L(t_1), ..., \epsilon^L(t_n))' \sim N_n(0, \sigma^L I)$

Las restricciones descritas anteriormente garantizan la identificabilidad de los parámetros. Para ello, se realiza una primera etapa de preprocesado [14], seguida del propio algoritmo de identificación con el objetivo de obtener el estimador máximo verosímil, los cuales se describirán a continuación.

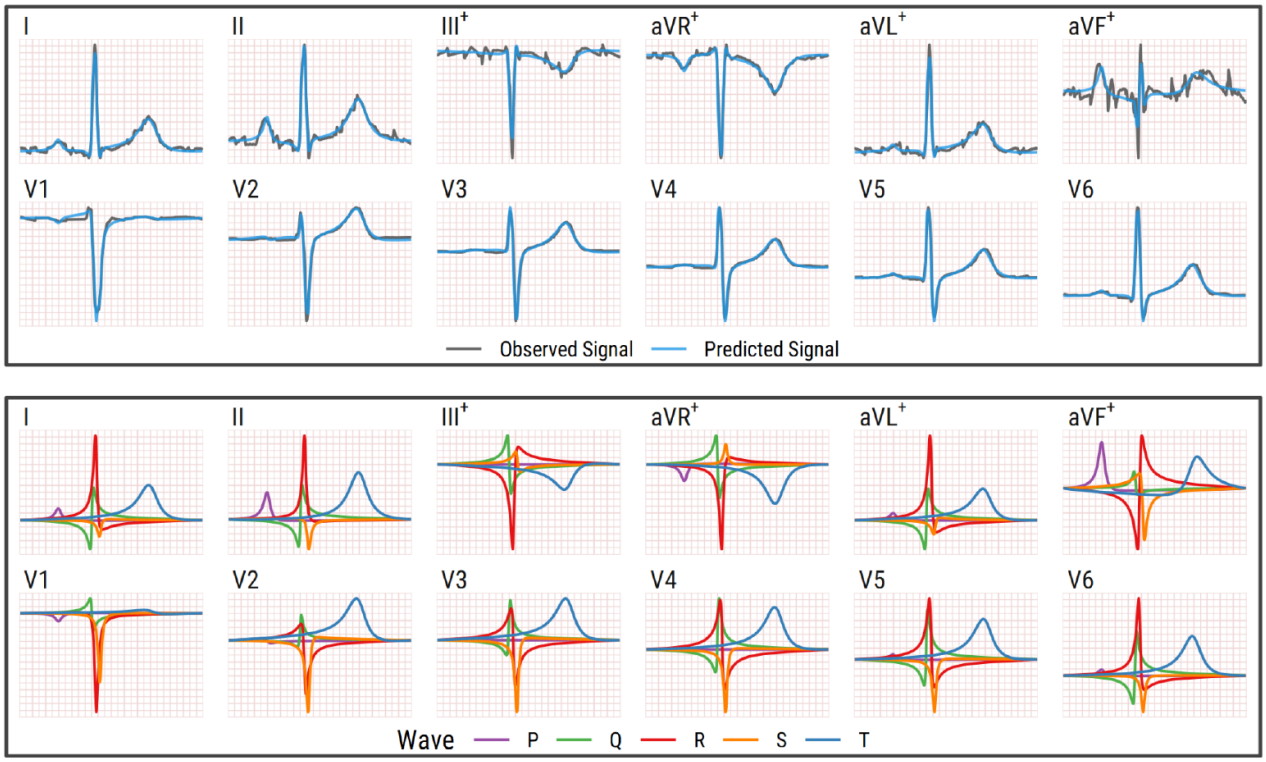


Figura 2.5: Ejemplo ilustrativo del ajuste del modelo 3DFMM$_{ecg}$ sobre las 12 derivaciones de un ECG. Adaptado de [14].

Preprocesado

Las señales del ECG suelen contener gran cantidad de ruido, desviaciones y artefactos que pueden afectar gravemente a la calidad de su análisis. Es por ello que se necesita una etapa previa de preprocesado para obtener fragmentos del ECG fiables y dividirlos latido a latido. Los distintos pasos de este proceso son:

1. **Eliminación de la línea base**

   Para cada uno de los fragmentos del ECG, se aplica un filtro de baja frecuencia basado en el ajuste de un modelo de regresión polinómica. Restando estas estimaciones de cada una de las señales se consigue corregir las posibles desviaciones.

2. **Detección mono-derivación de complejos QRS**

   Se realiza un análisis independiente mediante el algoritmo de Pan Tompkins [22] para cada una de las señales, y así detectar los complejos QRS de cada una de las derivaciones individualmente. Adicionalmente, se emplea la mediana de los intervalos RR (tiempo entre cada par de complejos QRS) para eliminar posibles falsos positivos, es decir, los complejos QRS demasiado largos o demasiado cortos.

3. **Detección multi-derivación de complejos QRS**

   Las detecciones obtenidas del paso anterior son combinadas para obtener un conjunto de anotaciones de referencia común. Las detecciones individuales se ordenan incrementalmente para construir grupos de al menos cuatro detecciones adyacentes de manera que los grupos

estén lo suficientemente lejos los unos de los otros. A continuación, se calcula la mediana de cada uno de los grupos, llamada t^{QRS}. Con esto se consigue reducir el número de falsos positivos y de complejos no detectados en alguna de las derivaciones. A partir de este punto, solo se trabaja con las derivaciones del conjunto $Lred = \{I, II, V1, V2, V3, V4, V5, V6\}$.

4. **Segmentación del ECG y revisión**

   Cada uno de los latidos segmentados se obtienen a través de $[t^{QRS}-40\,\%RR,\ t^{QRS}+60\,\%RR]$. Por otro lado, se lleva a cabo un último filtrado para eliminar los latidos insignificantes:

   - Los primeros y últimos latidos que sean notablemente inferiores que la mediana de los intervalos RR o con una proporción de observaciones constante en cualquiera de los extremos.
   - Los latidos para los cuales t^{QRS} se sitúa fuera de un 35 %-45 % de la longitud del latido.
   - Los latidos con una gran diferencia entre los puntos finales de la señal con respecto a la amplitud del dichos latidos.
   - Los latidos con problemas de conductividad cuya amplitud se exceda de los límites estándar o presenten anomalías de amplitud respecto a la señal completa.

   Se considera que el ECG podrá analizarse correctamente y no se descartará por tener valores inaceptables de ruido o distorsión si hay al menos:

   - Tres latidos en el conjunto de anotaciones de referencia.
   - Tres derivaciones con más de tres latidos.
   - Un 20 % de los latidos esperados de acuerdo con la frecuencia de muestreo.

   La salida del preprocesado está compuesta por las anotaciones t^{QRS} y por la segmentación de latidos del ECG.

Algoritmo de identificación y estimación

El problema de identificación de ondas se reduce a resolver el siguiente problema de optimización:

$$\arg\min_{\theta\in\Theta} \sum_{L\in Lset} \frac{1}{\sigma^L} \sum_{i=1}^{n} \{X^L(t_i) - [M^L + \sum_{J\in\{P,Q,R,S,T\}} A_J^L W(t_i, \alpha_J, \beta_J^L, \omega_J)]\}^2 \tag{2.9}$$

donde θ es el vector de parámetros del modelo y Θ es el espacio paramétrico. Para un ECG atípico o con mucho ruido, Θ puede ser más reducido para conseguir una correcta identificación de las ondas. Tanto σ^L como $L \in Lset$ son identificados durante el proceso de optimización. Ocasionalmente, se puede haber descartado alguna derivación en la etapa de preprocesado debido a artefactos ruidosos. Se requiere información sobre al menos una de las derivaciones I, II, V2 o V5. A continuación, se inicia un proceso iterativo MI con dos etapas diferenciadas: el paso M en el que se realiza la estimación de las ondas y el paso I en el que se asignan las ondas. El algoritmo finaliza cuando no se obtienen diferencias significativas en la función objetivo del problema de optimización. La Figura 2.6 muestra esquemáticamente mediante un diagrama de flujo el funcionamiento del algoritmo.

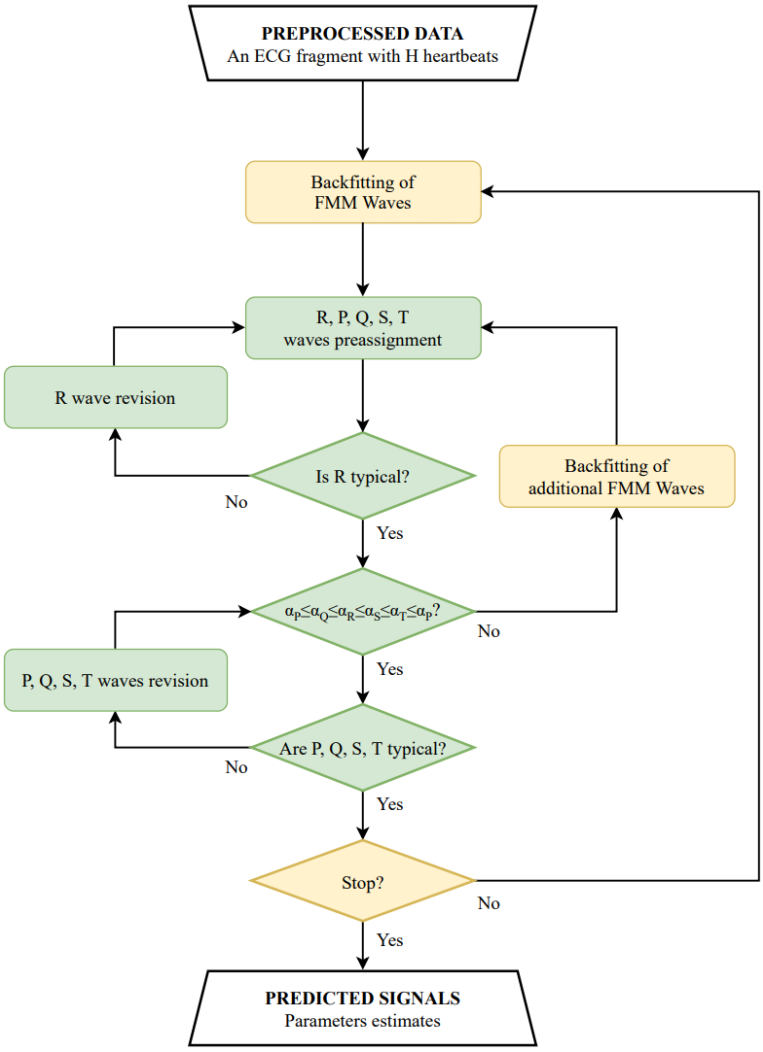


Figura 2.6: Diagrama de flujo del algoritmo de identificación del modelo 3DFMM$_{ecg}$. Los bloques amarillos se corresponden con el paso M y los verdes con el paso I. Adaptado de [14].

Paso M: El paso M obtiene al menos 5 ondas FMM mediante un algoritmo de *backfitting* ajustado simultáneamente sobre todas las derivaciones de $Lred$. Típicamente son 5 ondas, pero en presencia de ruido o morfologías patológicas, las ondas de interés pueden estar ocultas y se pueden necesitar más ondas. Los valores σ^L se inicializan a 1 y son actualizados en cada iteración del algoritmo, promediando las diferencias cuadráticas entre los valores esperados y los valores predichos por el modelo. Los estimadores finales de M^L, A_J^L y B_J^L se obtienen resolviendo un problema de regresión lineal múltiple.

Paso I: La onda R se asigna en primer lugar, siendo aquella de mayor variabilidad explicada entre las más cercanas al t^{QRS} y teniendo además una deflexión positiva en las derivaciones I o II y un pico negativo en la derivación V2. A continuación, se preasignan las ondas P, Q, S y T usando $\alpha_P \leq \alpha_Q \leq \alpha_R \leq \alpha_S \leq \alpha_T$. En la mayoría de los casos esta preasignación se corresponde con la asignación final, aunque pueden requerirse reasignaciones estableciendo umbrales sobre los parámetros principales del modelo. En concreto, los componentes ruidosos son detectados por poseer valores muy bajos o muy altos de ω.

Medidas de evaluación

Para estudiar el ajuste de este modelo, hay que confeccionar una métrica que tenga en cuenta todos los latidos de las 12 derivaciones del ECG. En el artículo original se propone una medida de calidad global $\bar{R} \in [0, 1]$. Esta medida se puede interpretar como la proporción de variabilidad de la señal que es explicada a partir de los valores predichos. Cuanto más próxima sea a 1, mejor será el modelo. Se formula como:

$$\bar{R} = \frac{1}{12} \sum_{L \in Lset} median(R^2_{L_b}) \tag{2.10}$$

donde $R^2_{L_b}$ es el coeficiente de determinación (R cuadrado) para un latido b de una derivación L con valor medio $\overline{X}^L_b$:

$$R^2_{L_b} = 1 - \frac{\sum_{i=1}^{n} (X^L_b(t_i) - \widehat{X^L_b}(t_i))^2}{\sum_{i=1}^{n} (X^L_b(t_i) - \overline{X}^L_b)^2} \tag{2.11}$$

Por otro lado, PRD es otra métrica ampliamente usada en la literatura relativa a la compresión de datos:

$$PRD_{L_b} = \sqrt{\frac{\sum_{i=1}^{n} (X^L_b(t_i) - \widehat{X^L_b}(t_i))^2}{\sum_{i=1}^{n} (X^L_b(t_i) - \overline{X}^L_b)^2}} = \sqrt{1 - R^2_{L_b}} \tag{2.12}$$

2.2. Algoritmos de aprendizaje supervisado

El aprendizaje supervisado es una rama del ML que consiste en el entrenamiento de modelos mediante conjuntos de datos etiquetados, con el objetivo de encontrar patrones en los mismos y obtener una alta capacidad de generalización a la hora de realizar predicciones sobre nuevos datos. Se parte de un conjunto de tuplas (x_i, y_i), donde $x_i = (1, x_{i1}, x_{i2}, ..., x_{ip})'$ es el vector de p atributos de la observación i-ésima e y_i su correspondiente etiqueta de clase. Se trata de construir un modelo que aproxime y a partir de x minimizando una función de pérdida o coste $\mathscr{L}$.

En este tipo de problemas surge un compromiso entre lo que se conoce como el sesgo (*bias*) y la varianza (*variance*) del modelo. El sesgo se refiere a la tendencia de un modelo a simplificar demasiado las relaciones entre los datos, lo que puede llevar a un subajuste (*underfitting*). Por el contrario, la varianza se refiere a la sensibilidad excesiva del modelo a las fluctuaciones de los datos de entrenamiento. Modelos excesivamente complejos pueden resultar en un sobreajuste (*overfitting*), lo que se traducirá en una mala capacidad de generalización. Existen distintos métodos de regularización que permiten restringir el proceso de estimación de los modelos para tratar de evitar posibles problemas de sobreajuste.

A continuación, se detallarán los distintos algoritmos que se han utilizado para abordar el problema de la clasificación de ECG. A lo largo de estas secciones se ha mantenido la notación estándar utilizada en el ámbito del ML, pudiendo ocasionalmente existir alguna coincidencia con la utilizada previamente en este trabajo.

2.2.1. Regresión logística

La regresión logística [23] fue creada en 1958 por David Cox y se usa principalmente en problemas de clasificación. Forma parte de los modelos lineales generalizados (*Generalized Linear Models* - GLM) y es una técnica popularmente empleada por su sencillez e interpretabilidad. Los modelos GLM tratan de modelar linealmente una función de la esperanza de una variable aleatoria. Esta función se conoce como función de enlace (*link function*). En el caso de la regresión logística, la función de enlace que se emplea es la función *logit* de la probabilidad de ocurrencia π de cierto suceso. La Figura 2.7 contiene un ejemplo muy sencillo de la aplicación de este modelo.

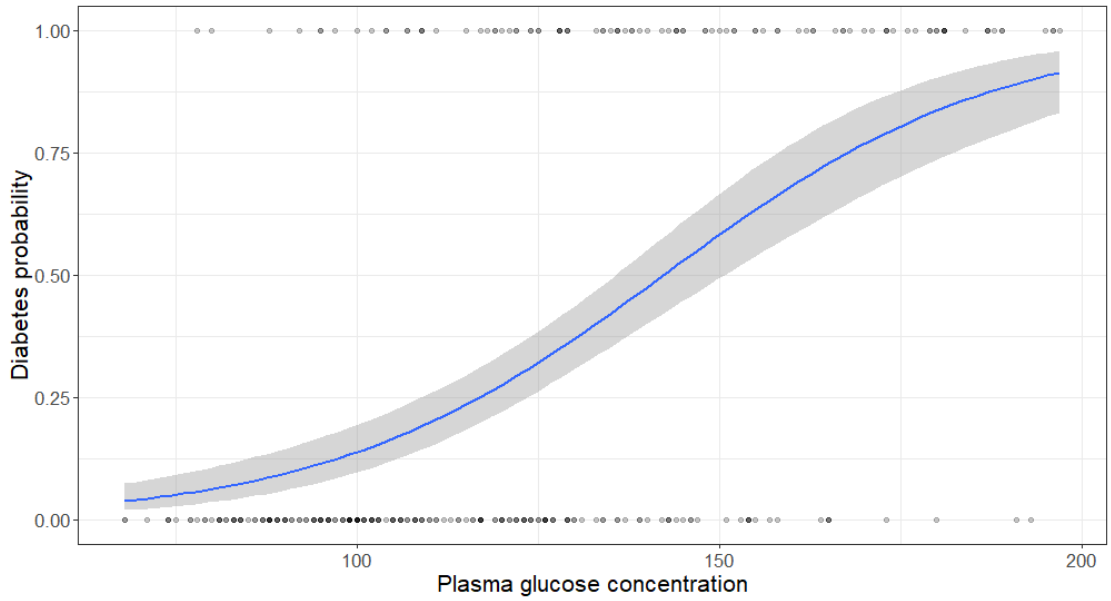


Figura 2.7: Modelo de regresión logística sobre la probabilidad de diabetes en función de la concentración de glucosa plasmática.

Sea $\theta = (\theta_0, \theta_1, \theta_2, ..., \theta_p)'$ el vector con los parámetros a estimar (siendo θ_0 el término independiente) y π_i la probabilidad de pertenencia a la clase positiva para la observación i, la expresión del modelo es:

$$log(\frac{\pi_i}{1-\pi_i}) = \theta' x_i + \theta_0 \tag{2.13}$$

Aplicando la función inversa a ambos lado de la ecuación, podemos llegar a una expresión explícita para la probabilidad π_i:

$$\pi_i = \frac{1}{1 + e^{-(\theta' x_i + \theta_0)}} \tag{2.14}$$

Para asignar la etiqueta de clase a cada instancia, basta con elegir un valor límite a partir del cual se toma la decisión de asignarlo a una u otra clase. Por defecto, se suele tomar el valor 0,5, de manera que simplemente se asigne a la clase con mayor probabilidad.

$$\hat{y}_i = \begin{cases} -1, & \text{si } \hat{\pi}_i < 0,5. \\ +1, & \text{si } \hat{\pi}_i \geq 0,5. \end{cases} \tag{2.15}$$

2.2.2. Support Vector Machine (SVM)

SVM [24] es una familia de algoritmos de clasificación y regresión desarrollados por Vladimir Vapnik y su equipo en los laboratorios de AT&T Bell en 1963. En el caso de la clasificación, están basados en construir un hiperplano de separación máxima entre las envolventes convexas de las clases (ver Figura 2.8). La envolvente convexa es el conjunto convexo más pequeño que contiene a un conjunto dado de puntos. Se toman una serie de puntos o vectores de referencia, conocidos como vectores de soporte (*support vectors*), los cuales permiten definir el hiperplano que discrimine las clases entre sí. Normalmente, estos vectores son las observaciones pertenecientes a la envolvente de su respectiva clase que se encuentran más próximas a la envolvente de otras clases. Son modelos bastante resistentes al sobreajuste y con almacenamiento muy eficiente, al solo constar únicamente de los vectores de soporte.

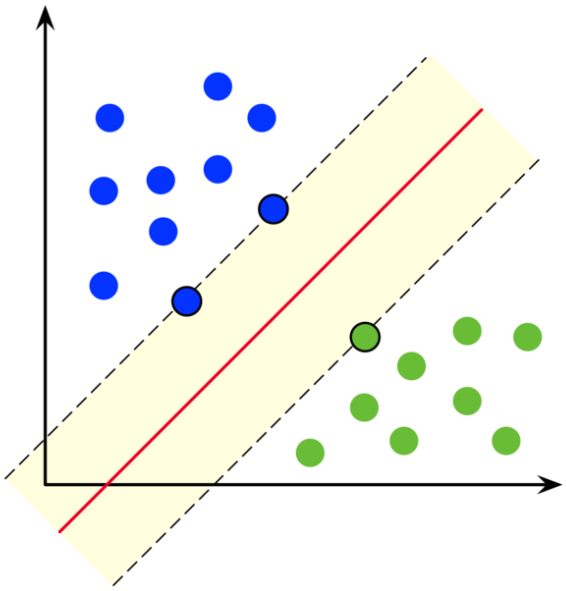


Figura 2.8: Hiperplano de margen máximo para un modelo SVM entrenado sobre dos clases. Las muestras en los márgenes se denominan vectores de soporte. Adaptado de [25].

Se define el hiperplano $\theta' x + \theta_0 = 0$ tal que $||\theta|| = 1$. La regla de clasificación es inducida a partir de dicho hiperplano como $sign(\theta' x + \theta_0)$. De esta forma, se trata de hallar el hiperplano que dé lugar al mayor margen posible entre los puntos de la clase positiva y los de la clase negativa, el cual se obtiene resolviendo el siguiente problema de optimización:

$$\begin{gathered} \arg\max_{\theta,\theta_0,||\theta||=1} \mathcal{M} \\ \text{sujeto a } y_i(\theta' x + \theta_0) \geq \mathcal{M};\ i = 1, ..., n \end{gathered} \tag{2.16}$$

En la mayoría de casos reales, las clases pueden solaparse en el espacio. Una forma de lidiar con este problema es maximizar $\mathcal{M}$, pero permitiendo a algunos puntos situarse en el "lado equivocado" del margen. Otro recurso que les confiere una gran flexibilidad es hacer uso de lo que se conoce como el truco del *kernel*, que consiste en proyectar puntos que no son linealmente separables a espacios de mayor dimensión para mejorar su separabilidad. La función *kernel* más elemental es la lineal, definida como el producto escalar entre dos observaciones i y j:

$$\mathcal{K}(x_i, x_j) = x_i \cdot x_j \tag{2.17}$$

Se puede generalizar fácilmente esta idea para un polinomio de grado d, al cual se le añade además un parámetro de regularización γ:

$$\mathcal{K}(x_i, x_j) = (\gamma(x_i \cdot x_j))^d \tag{2.18}$$

Finalmente, otro núcleo muy popular que también se ha usado en este proyecto es el núcleo de base radial (*Radial Based Function* - RBF), el cual emplea la norma euclídea.

$$\mathcal{K}(x_i, x_j) = e^{\gamma||x_i - x_j||^2} \tag{2.19}$$

Para todos los núcleos existe la posibilidad de añadir un término de regularización inversa $\mathcal{C}$ para controlar el sobreajuste. Escoger un bajo valor de $\mathcal{C}$ suavizará la superficie de decisión (mayor regularización), mientras que un alto valor de $\mathcal{C}$ proporcionará un mayor ajuste sobre el conjunto de entrenamiento (menor regularización).

En la Figura 2.9 se presenta un ejemplo de clasificación sencillo en el que se pueden observar las diferencias de aplicar cada uno de estos *kernel*.

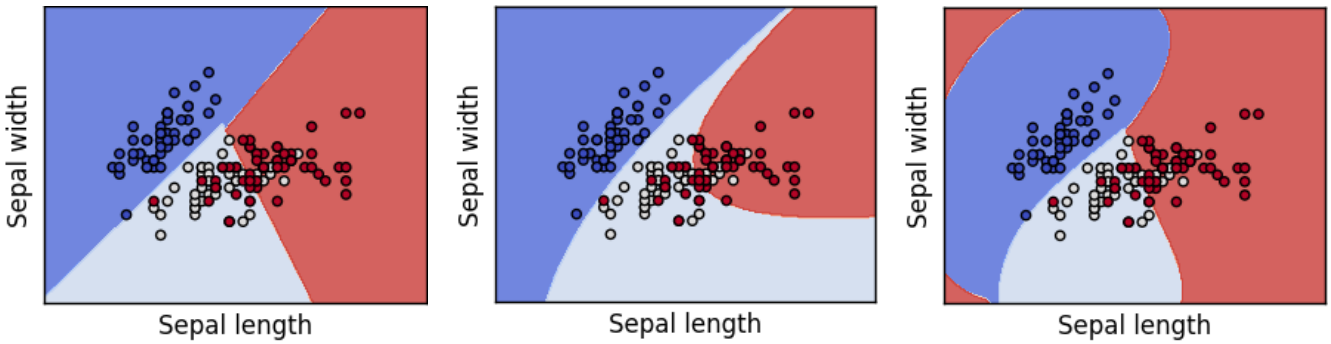


Figura 2.9: Representación gráfica del resultado de un modelo clasificador SVM sobre las tres clases de flores Iris setosa (azul), virginica (rojo) y versicolor (gris) a partir del tamaño y la anchura del sépalo. (a) Kernel lineal, (b) Kernel polinómico de grado 3, (c) Kernel RBF [26].

2.2.3. Random Forest

Random Forest [27] es una técnica aprendizaje supervisado de *ensemble*, es decir, que combina las predicciones de varios modelos. Concretamente, Random Forest está basado en combinar árboles de decisión mediante *bagging* (*bootstrap aggregating*). Tim Kam Ho propuso el fundamento teórico de estos modelos en 1995.

Los árboles de decisión son un algoritmo conocido por su sencillez e interpretabilidad. La principal limitación de los árboles de decisión es su alta variabilidad y, por tanto, su tendencia al sobreajuste. Se parece a un diagrama en forma de árbol, donde cada nodo interno representa una regla binaria sobre una variable y cada rama representa una posible respuesta o resultado (ver Figura 2.10). El punto de inicio se denomina "nodo raíz", el cual tendrá dos nodos hijos, ya sean terminales o no-terminales (hojas). Un nodo no terminal tendrá a su vez dos hijos, y así sucesivamente. Las hojas contendrán directamente la etiqueta de clase.

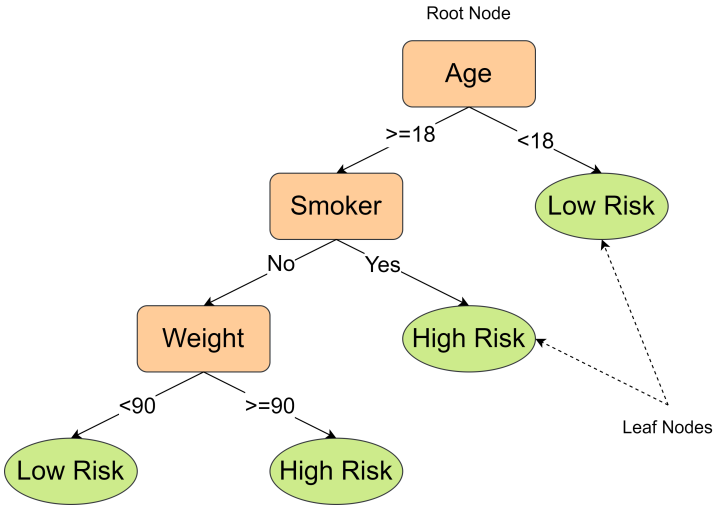


Figura 2.10: Árbol de decisión aplicado a la clasificación del riesgo cardiovascular. Los menores de edad tienen un bajo riesgo, mientras que los adultos fumadores o con más de 90Kg sufren un alto riesgo.

En cada nodo interno, se intenta escoger la mejor característica posible para dividir el conjunto de datos en dos grupos, de tal forma que se maximice la homogeneidad dentro de cada grupo y la heterogeneidad entre los grupos.

Bagging realiza remuestreo *bootstrap* (muestreo con reemplazamiento) sobre el conjunto original de datos, de manera que para cada una de las muestras *bootstrap* se entrena un modelo. Para una nueva observación, la predicción se obtendrá mediante un sistema de voto mayoritario. Una de las principales ventajas de este enfoque es la posibilidad de conocer la importancia relativa de cada una de las variables. Esta importancia relativa se calcula a partir de lo que se conoce como error *out-of-bag*, que consiste en calcular el error para cada uno de los clasificadores con las muestras que no formaron parte de su entrenamiento.

Random Forest utiliza una variante de *bagging* de árboles de decisión en la cual, aparte de hacer un muestreo de las observaciones, se utiliza un subconjunto aleatorio de variables para cada uno de los árboles. Esto permite obtener una mayor capacidad de generalización, reduciendo la dimensión de cada subárbol y obteniendo una gran diversidad al combinar todos ellos (ver Figura 2.11).

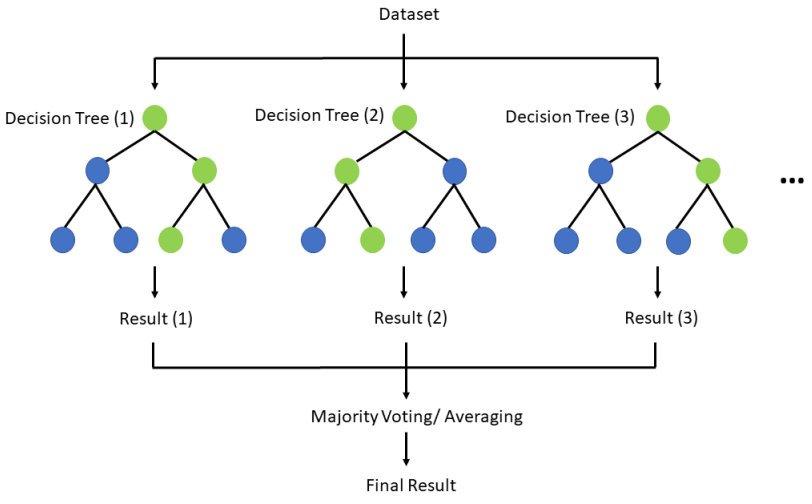


Figura 2.11: Estructura general de Random Forest [28].

2.2.4. Gradient Boosting

Gradient Boosting [29, 30, 31] es un algoritmo de aprendizaje automático de *ensemble* que se basa en la combinación de varios modelos más simples y débiles (*weak learners*) para crear un modelo final más robusto y preciso. Para entender cómo funciona Gradient Boosting, primero se deben entender dos conceptos clave: *boosting* y *gradient descent*.

El *boosting* es una técnica que se utiliza para mejorar la precisión de los modelos de aprendizaje automático mediante la combinación de múltiples modelos más simples. A diferencia de *bagging*, los modelos débiles no se entrenan en paralelo, sino en secuencia. La idea básica es entrenar un modelo inicial del que se calcularán los errores y luego agregar iterativamente modelos adicionales que se centren en los casos que el modelo anterior no pudo clasificar correctamente (ver Figura 2.12).

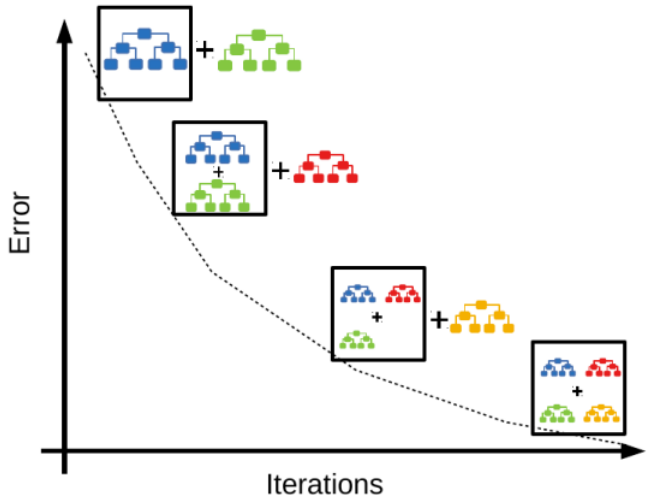


Figura 2.12: Estructura general de *boosting* [32].

Por otro lado, el descenso de gradiente (*gradient descent*) es un algoritmo utilizado para encontrar el mínimo global de una función. Consiste en avanzar iterativamente con un cierto tamaño de paso hacia la dirección que más haga disminuir el gradiente desde una posición dada. Un tamaño de paso excesivamente pequeño puede provocar que, en funciones que no sean estrictamente convexas, el algoritmo se quede atascado en un mínimo local, lo cual además ralentiza significativamente el tiempo de cómputo. Por otro lado, un tamaño de paso demasiado grande puede provocar que el algoritmo no converja.

El algoritmo de Gradient Boosting parte de una función inicial:

$$F_0(x) = \arg\min_{\rho} \sum_{i=1}^{n} \mathscr{L}(y_i, \rho) \tag{2.20}$$

A continuación, se va realizando un proceso iterativo ($m = 1, ..., M$) en el que se va actualizando dicha función. El primer paso de cada iteración es calcular los pseudo-residuales r_i, que se corresponden con el menos gradiente de la función de pérdida:

$$r_i = -\left[\frac{\partial \mathscr{L}(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}; \quad i = 1, ..., n \tag{2.21}$$

Estos residuales son los que deberán ir disminuyendo a medida que avancen las iteraciones. A continuación, se ajusta un modelo con parámetros ν y λ de la forma:

$$\nu_m = \arg\min_{\nu,\lambda} \sum_{i=1}^{n} [r_i - \lambda h(x_i; \nu)]^2 \tag{2.22}$$

Los modelos que se suelen emplear se denominan como "débiles", debido a que su capacidad predictiva es limitada y en muchos casos de precisión es bastante baja. Típicamente se emplean árboles de decisión no muy extensos, pero se podrían utilizar otros. La función h puede tener cualquier forma que pueda optimizarse convenientemente sobre ν. El siguiente paso será elegir el tamaño de paso del descenso de gradiente ρ_m, el cual se obtiene minimizando la siguiente función:

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{n} \mathscr{L}(y_i, F_{m-1}(x_i) + \rho h(x_i; \nu_m)) \tag{2.23}$$

Finalmente, se actualiza la estimación de $F_m(x)$ de acuerdo con la ecuación:

$$F_m(x) = F_{m-1}(x) + \rho_m h(x; \nu_m) \tag{2.24}$$

Existen implementaciones más modernas que consiguen explotar al máximo el potencial predictivo de este algoritmo. Entre las versiones más populares destacan XGBoost y LightGBM. Ambos cuentan con una gran cantidad de hiperparámetros configurables que permiten modificar el comportamiento del modelo para poder adaptarse mejor a cada problema.

Extreme Gradient Boosting (XGBoost)

XGBoost [33] es una implementación del algoritmo de Gradient Boosting introducido por Tianqi Chen en 2016 como un método escalable con un gran potencial de desempeño en multitud de áreas de conocimiento. Algunas de las características que introduce son:

- **Paralelización**: Permite aprovechar al máximo la capacidad de computación de la máquina, haciendo uso de procesamiento multinúcleo y de la tarjeta gráfica.
- **Regularización**: Dispone de multitud de hiperparámetros como el número de árboles, la profundidad máxima de cada árbol, proporción de variables usadas por cada árbol...
- **Trabajo nativo de datos ausentes**: La mayoría de algoritmos tradicionales de ML requieren una previa imputación de los valores ausentes o, en los mejores casos, realizan una imputación sencilla internamente. XGBoost aprende durante el entrenamiento la dirección por la que debe continuar en caso de que se encuentre con valores perdidos. Esto no solo facilita el trabajo de programación sino que, de hecho, suele ofrecer mejores resultados que cualquier método de imputación tradicional.

Light Gradient Boosting Machine (LightGBM)

LightGBM [34] surge de la mano de Microsoft en 2016 como competencia a XGBoost. Se diseñó con el objetivo principal de reducir al máximo el tiempo de cómputo, haciendo uso de una técnica basada en histogramas que agrupa valores continuos en un rango de valores discretos, lo que requiere también menos memoria. Las características principales de paralelización, regularización y tratamiento nativo de datos ausentes también están presentes. La principal diferencia estructural consiste en la manera en la que hace crecer los árboles. En contraposición al crecimiento por niveles

(horizontal) de XGBoost, LightGBM apuesta por un crecimiento por hojas (vertical), que consiste en centrarse en dividir únicamente aquellas hojas con mayor pérdida (ver Figura 2.13). Esto puede ser un arma de doble filo, ya que aunque se puede tunear los hiperparámetros de profundidad, es más propenso al sobreajuste, siendo XGBoost más robusto en este sentido.

Figura 2.13: Estrategias estructurales del crecimiento de los árboles en modelos de *boosting*. (a) Crecimiento por niveles, (b) crecimiento por hojas [35].

No hay un claro vencedor, ambos algoritmos darán mejor o peor resultado dependiendo del problema concreto, por lo que resulta interesante aprovechar el potencial de ambos.

2.3. Evaluación de modelos

Tan importante es entrenar un clasificador como ser capaz de evaluarlo correctamente para demostrar su rendimiento y su capacidad de generalización a la hora de predecir nuevas muestras. Habitualmente el conjunto de datos se divide en lo que se llama subconjunto de entrenamiento y subconjunto de test. El primero se utiliza para aprender de él y ajustar así los distintos parámetros del modelo, mientras que el test se utiliza para evaluar la capacidad de generalización, al ser datos que no se han visto hasta el momento. Además, muchos de los modelos actuales permiten una gran variedad de configuraciones, por lo que se suele recurrir a un tercer subconjunto de validación sobre el que se prueban distintas combinaciones para tratar de quedarse con la mejor de ellas antes de evaluar sobre el conjunto de test. A continuación, se detallarán algunos conceptos clave para comprender la metodología de evaluación de modelos.

2.3.1. Validación hold-out

El método de *hold-out* [36] o método de reserva consiste, como su propio nombre indica, en reservar un cierto número de observaciones del conjunto de datos del que se dispone. Este subconjunto queda apartado completamente del proceso de entrenamiento y se usa únicamente para estimar el error de generalización. El dilema que existe es la elección del tamaño de este subconjunto. Si este subconjunto de test es muy pequeño, es posible que no sea una muestra representativa de la población y que, por tanto, no se obtenga una buena estimación del error de generalización. Por el contrario, si se es conservador y se guarda un gran porcentaje para test, no quedarán apenas datos para entrenar un modelo que ofrezca buenos resultados.

Algunas de las formas clásicas de dividir el conjunto de datos es reservar 80 % para entrenamiento y un 20 % para test o, en caso de que se necesite un conjunto de validación, reducir a un 70 % los datos de entrenamiento y dejar un 15 % para validación y otro 15 % para test (ver Figura 2.14).

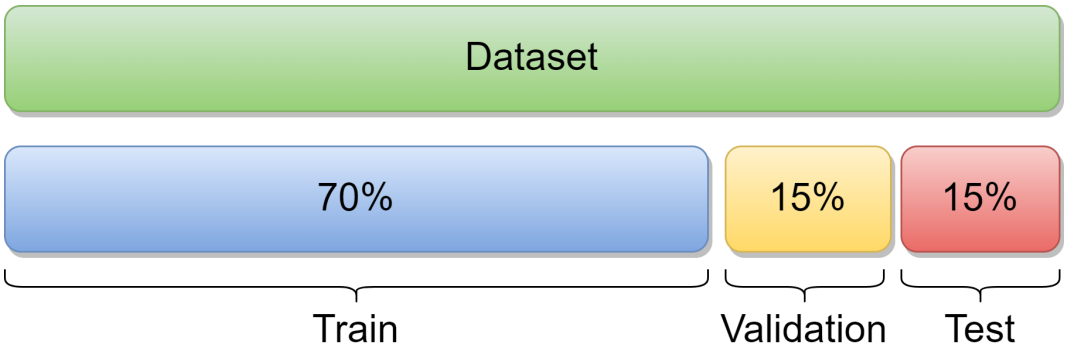


Figura 2.14: Estrategia clásica de reparto de los datos mediante *hold-out* de los subconjuntos de entrenamiento, validación y test.

En algunos contextos se habla de *hold-out* repetido; se repite el proceso de *hold-out* sobre distintos conjuntos aleatorios de entrenamiento y test y se promedian los errores. Esta técnica no es del todo recomendable, ya que es probable que si se repite muchas veces exista solapamiento entre los distintos conjuntos de test y que, por tanto, no sean experimentos del todo independientes.

2.3.2. Validación cruzada k-fold

La validación cruzada [36], o también comúnmente llamada *k-fold*, surge como alternativa al *hold-out* repetido, de manera que se obtengan varias estimaciones de error pero tratando de conservar la independencia entre los distintos experimentos. Se realiza un muestreo aleatorio para repartir los datos en k carpetas (*folds*) de igual tamaño. Se realizan por tanto k iteraciones, en cada de las cuales se calculará el error sobre la carpeta k-ésima, habiendo entrenado un modelo sobre el resto de los datos. Al final del proceso se habrán obtenido k estimaciones del error (ver Figura 2.15). Normalmente se suelen realizar $k = 10$ iteraciones, aunque esto depende muchas veces de la capacidad de cómputo disponible. Esta técnica, además de evitar por completo el solapamiento entre subconjuntos, garantiza que todos los datos se han utilizado tanto para entrenamiento como para test.

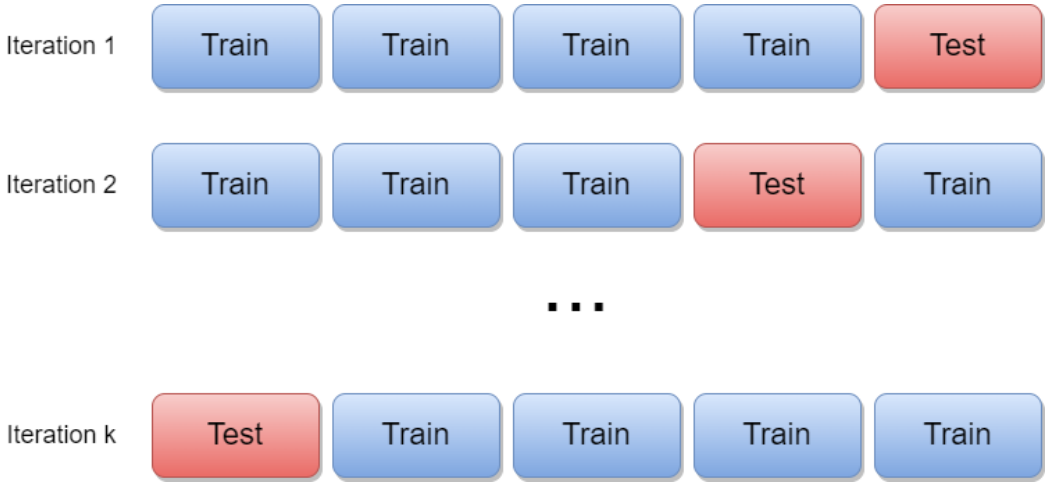


Figura 2.15: Diagrama del proceso de validación cruzada de k particiones.

2.3.3. Estratificación y agrupamiento

Para las estrategias de división de los datos mencionadas anteriormente, e incluso para cualquier otra, hay que tener cuidado de asegurarse de que la distribución del conjunto de entrenamiento es similar a la del conjunto de test. La estratificación ayuda a garantizar que los datos de entrenamiento y prueba sean representativos de todas las clases del conjunto de datos original. Por ejemplo, si hay un conjunto de datos con 70 % de instancias de la clase A y 30 % de la clase B, hay que asegurarse de que este reparto 70-30 se dé tanto en el subconjunto de entrenamiento como en el de test.

Además, hay que tener cuidado cuando aparecen grupos naturales de observaciones. Cuando se divide un *dataset* que contiene múltiples observaciones de un mismo individuo o fuente, se deben dividir de forma que todas las observaciones de un individuo determinado se incluyan en el mismo subconjunto. En el contexto de los ECG, se han realizado estudios acerca de su prometedor potencial biométrico, debido a la naturaleza tan individualizada de las señales de ECG [37]. El haber entrenado un clasificador con el ECG de un paciente y evaluarlo sobre otro ECG del mismo, puede provocar que el modelo sea capaz de clasificarlo correctamente por haber "memorizado" las particularidades del ECG de ese paciente concreto, no por ser capaz de detectar su patología realmente.

2.3.4. Métricas de evaluación

Partiendo de un problema de clasificación binaria, con una clase positiva y una clase negativa, aparecen los siguientes escenarios para una observación.

- TP (*True Positive*): Observación de clase positiva clasificada correctamente.
- FP (*False Positive*): Observación de clase positiva clasificada incorrectamente.
- TN (*True Negative*): Observación de clase negativa clasificada correctamente.
- FN (*False Negative*): Observación de clase negativa clasificada incorrectamente.

A partir de esto, se pueden construir diversas métricas para evaluar un clasificador. En este trabajo se utilizarán las siguientes:

- ACC (*Accuracy*): El complementario de la tasa de error, mide la proporción de aciertos respecto al total. No debe ser el criterio principal a tener en cuenta si las clases están desbalanceadas.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

- SE (*Sensitivty*): También llamada tasa de verdaderos positivos. Mide la proporción de positivos correctamente identificados.

$$SE = \frac{TP}{TP + FN}$$

- SP (*Specifity*): El complementario de la tasa de falsos positivos. Mide la proporción de negativos correctamente identificados.

$$SP = \frac{TN}{TN + FP}$$

- BACC (*Balanced accuracy*): Es el promedio entre la sensibilidad y la especificidad. Resulta interesante para comprobar si existe un equilibrio entre ambas medidas.

$$BACC = \frac{SE + SP}{2}$$

- AUC (*Area Under the ROC Curve*): Mide el área bajo la curva de la curva ROC (*Receiver Operating Characteristic curve*), la cual es una representación gráfica que resulta útil para observar la relación entre SE y SP según varía el umbral de clasificación entre clases. En el eje de abscisas se representa la tasa de falsos positivos y en el eje de ordenadas la tasa de verdaderos positivos (ver Figura 2.16). La referencia es la clasificación aleatoria, que se correspondería con una la línea diagonal que pasase por los puntos $(0, 0)$ y $(1, 1)$ y, por tanto, tenga un AUC de 0,5. El clasificador perfecto sería aquel alcanzase el punto $(0, 1)$, obteniéndose un AUC igual a 1.

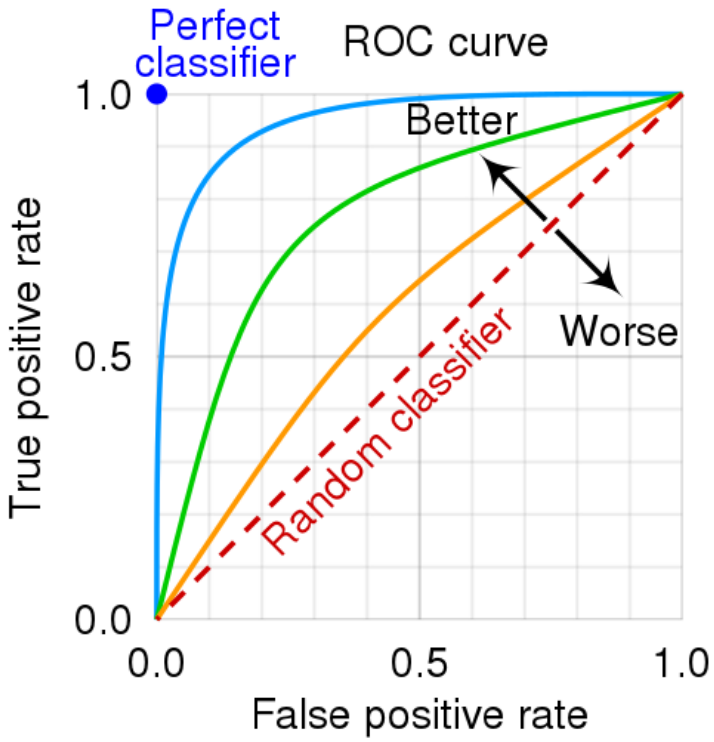


Figura 2.16: Curva ROC de la tasa de verdaderos positivos en función de la tasa de falsos positivos. La diagonal punteada (roja) muestra el rendimiento de un clasificador aleatorio. Se muestran otros tres clasificadores de ejemplo (azul, verde y naranja) [38].

La matriz de confusión es otra herramienta que también es útil para comprobar si se están confundiendo dos clases entre sí. Cada columna de la matriz representa las clases predichas, mientras que cada fila representa a las instancias en la clase real. Para el caso de los problemas multiclase se suele usar el enfoque *one-vs-rest*, que consiste en tomar una clase de referencia como positiva y agrupar todas las demás clases en una sola. De esta forma, se pueden computar las métricas tal y como se haría en un problema binario. Si se repite este proceso, se obtendrán las métricas deseadas para cada una de las clases. También se pueden tomar todas las combinaciones de parejas de clases (enfoque *one-vs-all*) aunque es menos habitual, ya que aparecen muchas combinaciones en problemas con un gran número de clases.

Por otro lado, existen varias alternativas para promediar estas métricas. Habitualmente se usa la media aritmética (*macro average*) si se le quiere dar la misma importancia a todas las clases, o una media ponderando con la distribución de cada clase (*micro average*) en caso de que se le quiera dar más peso a las clases más pobladas.

Ajuste de hiperparámetros

Los hiperparámetros son variables de configuración externa que se fijan antes del proceso de entrenamiento de un modelo. La diferencia con los parámetros, es que estos últimos sí que se van ajustando progresivamente durante el entrenamiento. El ajuste de hiperparámetros es un proceso crucial si quiere maximizar el rendimiento de un modelo. Los algoritmos más potentes ofrecen una gran cantidad de hiperparámetros para poder adaptarse a las necesidades de cada problema, pero cuentan con la desventaja de que el ajuste de los mismos es por prueba y error, por lo que suele ser un proceso bastante costoso.

Se define una rejilla de valores para cada uno de los hiperparámetros y se prueban distintas combinaciones de los mismos. Se elegirá aquella configuración que optimice una métrica determinada sobre el conjunto de validación. La aproximación clásica *grid search* consiste en probar todas las combinaciones posibles de la rejilla; es útil cuando no hay muchos hiperparámetros que ajustar. Sin embargo, muchos de los algoritmos actuales tienen un gran número de hiperparámetros, por lo que resulta demasiado costoso computacionalmente. En contraposición surge *randomized search*, el cual prueba solo un subconjunto aleatorio de la rejilla, por lo que resulta más interesante cuando existen muchos hiperparámetros posibles o la capacidad de cómputo es limitada.

2.4. Explicabilidad de modelos complejos

El aprendizaje automático se está convirtiendo en una herramienta cada vez más importante en el día a día, debido a que es capaz de encontrar patrones complejos que el ser humano puede pasar por alto o, incluso, desconocer por completo. Sin embargo, muchos de estos modelos se denominan como "caja negra", debido a que por su naturaleza o por su complejidad se hace prácticamente imposible una interpretación directa. En este trabajo, se han utilizado algunos métodos de explicabilidad que resultan ciertamente útiles para la interpretación de modelos complejos y para la explicación de las predicciones individuales.

2.4.1. LIME

LIME, *Local Interpretable Model-Agnostic Explanations* [39] es una técnica de explicación de modelos de aprendizaje automático que ayuda a entender cómo se toman las decisiones de un modelo. La interpretación se lleva a cabo a nivel local, es decir, para una observación en particular. LIME comprueba como varían las predicciones cuando se añade cierto ruido al conjunto de datos.

Se denota $x \in \mathbb{R}^p$ como la representación original de la instancia que se quiere explicar y $\tilde{x} \in \{0,1\}^q$ como un vector binario de representación simplificada interpretable con q variables. Se define una explicación como un modelo $g \in G$, siendo G el conjunto de potenciales modelos interpretables. Por otro lado, se define una función de complejidad $\Omega(g)$. Por ejemplo, para un árbol puede ser su profundidad, y para una regresión logística puede ser el número total de parámetros estimados.

Sea $f : \mathbb{R}^p \to \mathbb{R}$ el modelo a explicar, $f(x)$ es la probabilidad de que la observación x pertenezca a una cierta clase. Por otra parte, $\psi_x(w)$ es una medida de proximidad entre una instancia w y x. Finalmente, sea $l(f, g, \psi_x)$ una medida de qué poco fiel es g a la hora de aproximar f en la localidad definida por ψ_x, la explicación LIME se obtiene de la siguiente ecuación:

$$\xi(x) = \arg\min_{g \in G} l(f, g, \psi_x) + \Omega(g) \tag{2.25}$$

En la Figura 2.17 se puede observar un sencillo ejemplo de la aplicación de LIME. La función de decisión compleja f está representada por el fondo azul/rosa, la cual es muy distinta a un modelo lineal. La cruz roja en negrita es la instancia que se está explicando. LIME muestrea las observaciones, obtiene predicciones utilizando f y las pondera por la proximidad a la instancia que se está explicando. En este caso se ha representado el tamaño de cada instancia de manera inversamente proporcional a la distancia respecto a la que se quiere explicar. La línea discontinua es la explicación aprendida, la cual es fiel localmente, pero no globalmente.

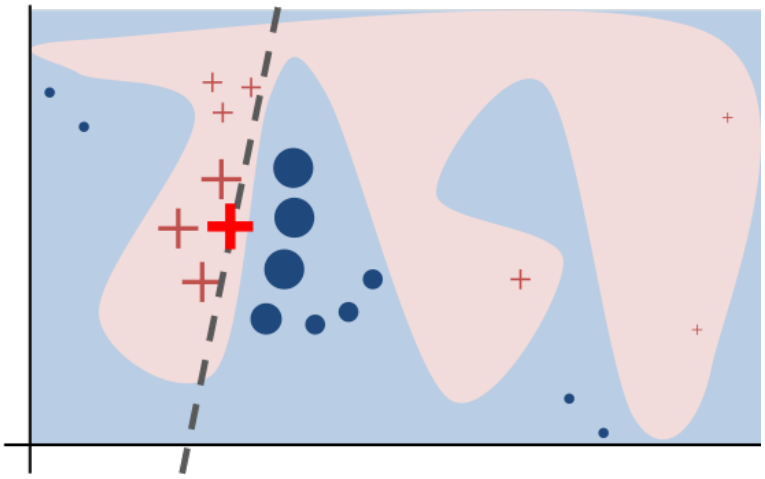


Figura 2.17: Ejemplo ilustrativo para presentar la intuición detrás de LIME [39].

2.4.2. Shapley value

Los valores Shapley [40] son una medida matemática utilizada en la teoría de juegos para determinar la contribución de cada jugador a una ganancia colectiva. En un problema de aprendizaje supervisado el valor de cada atributo se puede ver como un "jugador" y la propia predicción como la ganancia.

Suponer un problema de regresión de precios de coches. Se cuenta con un modelo que para un coche deportivo, con buena autonomía, de color negro y de segunda mano se tiene una predicción de 25,000€. Se sabe que la predicción media de los coches del mercado es de 15,000€. Para un modelo paramétrico sencillo como es una regresión lineal, basta con observar los coeficientes, sin embargo, para modelos más complejos esto no es tan inmediato. El valor de Shapley da una medida de la contribución marginal de cada variable a la predicción. Volviendo al ejemplo anterior, el ser un coche deportivo ha podido contribuir con 30,000€, el tener buena autonomía con 5,000€, el ser de color negro 0€ y el ser de segunda mano con -25,000€. Si se suman todas estas contribuciones se obtiene un total de 10,000€, que es la diferencia entre el precio predicho y el coste promedio de todos los coches.

Partiendo de un modelo de regresión lineal con p atributos y un término independiente θ_0,

$$f(x) = \theta_0 + \theta_1 x_1 + ... + \theta_p x_p \tag{2.26}$$

la contribución δ_i se define como:

$$\delta_i(f) = \theta_i x_i - E(\theta_i x_i) = \theta_i x_i - \theta_i E(x_i) \tag{2.27}$$

Para modelos que no estiman directamente un coeficiente para cada atributo, se calcula la contribución ponderada y sumada sobre todas las combinaciones posibles de valores de las características:

$$\delta_i(\upsilon) = \sum_{S \subseteq \{1,...,p\} \setminus \{i\}} \frac{|S|!(p - |S| - 1)!}{p!} (\upsilon(S \cup \{i\}) - \upsilon(S)) \tag{2.28}$$

donde S es el subconjunto de variables usadas en el modelo, x es el vector de atributos de la instancia x y $\upsilon(S)$ la predicción para los valores de las características que pertenecen al subconjunto.

2.4.3. SHAP

SHAP (*SHapley Additive exPlanations*) [41] es un método que combina las técnicas LIME y los valores Shapley para asignar la contribución de cada variable de entrada en la predicción del modelo. La explicación SHAP para un modelo g se define como:

$$g(\tilde{z}) = \delta_0 + \sum_{i=1}^{q} \delta_i \tilde{z}_i \tag{2.29}$$

donde $\tilde{z} \in \{0,1\}^q$, siendo q el número de variables simplificadas. Tres de sus propiedades más destacadas son:

- **Precisión local**: El modelo explicativo $g(\tilde{x})$ se corresponde con el modelo original $f(x)$ cuando $x = h_x(\tilde{x})$.

$$f(x) = g(\tilde{x}) = \delta_0 + \sum_{i=1}^{p} \delta_i \tilde{x}_i \tag{2.30}$$

- **Ausencia**: Las variables ausentes de la entrada original cuando $\tilde{x}_i = 0$ no tienen ningún impacto.

$$\tilde{x}_i = 0 \implies \delta_i = 0 \tag{2.31}$$

- **Consistencia**: Sea $f_x(\tilde{z}) = f(h_x(\tilde{z}))$ y $\tilde{z} \setminus i$ indica fijar $\tilde{z}_i = 0$. Para dos modelos f y $\tilde{f}$, si

$$\tilde{f}_x(\tilde{z}) - \tilde{f}_x(\tilde{z} \setminus i) \geq f_x(\tilde{z}) - f_x(\tilde{z} \setminus i) \tag{2.32}$$

para cada entrada $\tilde{z} \in \{0,1\}^q$, entonces $\delta_i(\tilde{f}, x) \geq \delta_i(f, x)$. Esto quiere decir que si un modelo cambia de modo que la contribución de una variable simplificada aumenta o se mantiene igual independientemente de los demás variables, la contribución de dicha variable no debería disminuir.

De esta forma, solo existe un posible modelo explicativo que cumpla la definición de la Ecuación (2.29) y satisfaga las propiedades descritas en las Ecuaciones (2.30), (2.31) y (2.32):

$$\delta_i(f, x) = \sum_{\tilde{z} \subseteq \tilde{x}} \frac{|\tilde{z}|!(q - |\tilde{z}| - 1)!}{q!} [f_x(\tilde{z}) - f_x(\tilde{z} \backslash i)] \tag{2.33}$$

donde $|\tilde{z}|$ es el número de entradas en $\tilde{z}$ que no son 0 y $\tilde{z} \subseteq \tilde{x}$ representa todos los vectores $\tilde{z}$ donde las entradas distintas de 0 son un subconjunto de las entradas distintas de 0 en $\tilde{x}$.

Algunos de los métodos más usados para estimar los valores de SHAP son:

- **KernelSHAP**: Se puede utilizar para aproximar cualquier función haciendo uso de la esperanza marginal, sin embargo, es bastante pesado computacionalmente.

- **LinearSHAP**: Usado para modelos lineales. Es capaz de trabajar con posibles colinearidades en las variables de entrada mediante muestreo, distribuyendo las contribuciones entre todas las variables correladas. Es además un método computacionalmente eficiente que puede aplicarse a conjuntos de datos de gran tamaño.

- **TreeSHAP**: Este método es específico para modelos basados en árboles de decisión. Utiliza la esperanza condicional y se beneficia de la propiedad aditiva de SHAP, que establece que el valor SHAP de un *ensemble* de árboles es la suma de los SHAP de todos sus árboles. Es una aproximación de los verdaderos valores SHAP, ya que estima la esperanza condicional durante el proceso de entrenamiento del modelo.

Capítulo 3

Datos

3.1. Descripción

La base de datos utilizada ha sido PTB-XL [5], disponible en Physionet [42, 43]. PTB-XL es una base de datos abierta que comprende 21,837 ECG de 12 derivaciones de 10 segundos de duración correspondientes a 18,885 pacientes. Ha sido estudiada ampliamente en la literatura, habiendo incluso formado parte de las fuentes de datos del PhysioNet Challenge 2020 [44]. Está balanceada con respecto al sexo (52 % de hombres y 48 % de mujeres) y cubre un amplío espectro de edades, desde los 0 hasta los 95 años (mediana de 62 y rango intercuartílico de 22). Existen una compleja jerarquía de hasta 71 etiquetas distintas para los ECG, de acuerdo con el estándar SCP-ECG [45], que se dividen en tres grupos: forma, ritmo y diagnóstico. A su vez, las etiquetas de diagnóstico se organizan jerárquicamente en un sistema de superclases y subclases, de tal forma que varios diagnósticos pueden pertenecer a una misma subclase, la cual pertenece a su vez a una superclase (CD, HYP, MI, NORM o STTC). En particular, cada ECG puede tener varias etiquetas debido a que, en muchos casos, varias anomalías conviven en un mismo paciente. Además, cada una de las etiquetas está asignada con una cierta verosimilitud (entre 0 y 100), dependiendo del nivel de certeza que tuvo el cardiólogo a la hora de realizar el diagnóstico. Se utilizó el fichero proporcionado directamente por Physionet, el cual contiene la información demográfica y las etiquetas de cada electrocardiograma. Además, se usaron los resultados del ajuste del modelo 3DFMM$_{ecg}$ sobre la base de datos PTB-XL descritos en [14], constituidos por las anotaciones de las ondas R y los parámetros ajustados para cada latido de cada ECG.

3.2. Preprocesado

3.2.1. Filtrado de las observaciones

Se realizó un proceso de filtrado de los ECG siguiendo los criterios descritos en [6, 7, 14]:

1. Se eliminaron todas aquellos ECG que no tuvieran alguna etiqueta de diagnóstico, es decir, que solo tuvieran etiquetas de ritmo o de forma. Por tanto, de los 21,837 ECG originales, se descartaron 407 ECG.

2. Se eliminaron todos aquellos ECG que no tuvieran al menos una etiqueta con 100 % de verosimilitud para no introducir ruido por diagnósticos de dudosa fiabilidad, obteniendo un total de 17,259 ECG.

3. Se eliminaron 154 ECG por ser altamente ruidosos o por contener artefactos generados por un marcapasos. Finalmente, se obtuvo un *dataset* con un total de 17,105 ECG. La distribución de superclases queda reflejada en la Figura 3.1. Se observa que hay una gran cantidad de observaciones que pertenecen a más de una superclase. Son casos que pueden ocurrir con relativa frecuencia en el mundo real, y no resulta recomendable eliminarlos del estudio ya que introduciría un sesgo muy grande en los clasificadores.

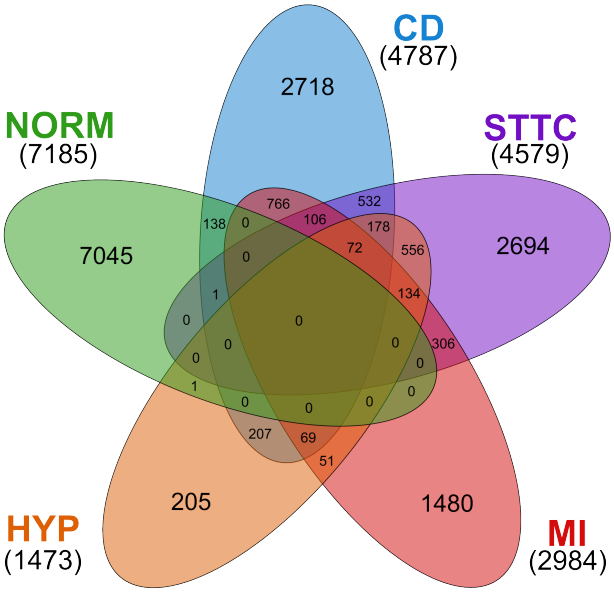


Figura 3.1: Diagrama de Venn de la distribución de superclases de PTB-XL tras el proceso de filtrado.

3.2.2. Etiquetado de las observaciones

Debido a la presencia de múltiples etiquetas por cada clase, hay que establecer un criterio para enfocar el problema desde un punto de vista de aprendizaje supervisado. Existen algunos enfoques conocidos como *multilabel* que permiten asignar múltiples etiquetas a una misma observación, sin embargo, se decidió no utilizar dichos enfoques por varios motivos. En primer lugar, para evitar posibles ambigüedades y potenciales conflictos a la hora de asignar múltiples etiquetas a una misma observación. En segundo lugar, porque la mayoría de implementaciones populares de los algoritmos de ML no soportan de manera nativa el enfoque *multilabel*, además de que el esfuerzo computacional requerido para un correcto entrenamiento aumenta enormemente y quedaría fuera de los límites de este trabajo. En tercer lugar, por simplicidad de la evaluación de los modelos, ya que las métricas de evaluación de modelos son más difícilmente interpretables y requieren de medidas más complejas que contemplen errores parciales dentro de cada predicción. Por estos motivos, se decidió emplear un enfoque multiclase, permitiendo una única etiqueta por observación. Para ello, se utilizó la prioridad descrita en el consenso de expertos chinos de 2017 [46, 47], basada en el estándar SCP-ECG [45]. Estos establecen cuatro niveles de riesgo para cada diagnóstico (0 - *No Risk*, 1 - *Low Risk*, 2 - *Medium Risk* y 3 - *High Risk*), por lo que se etiquetó cada ECG con la superclase correspondiente al diagnóstico con mayor riesgo. En la Figura 3.2 se observa la distribución de superclases del *dataset* final.

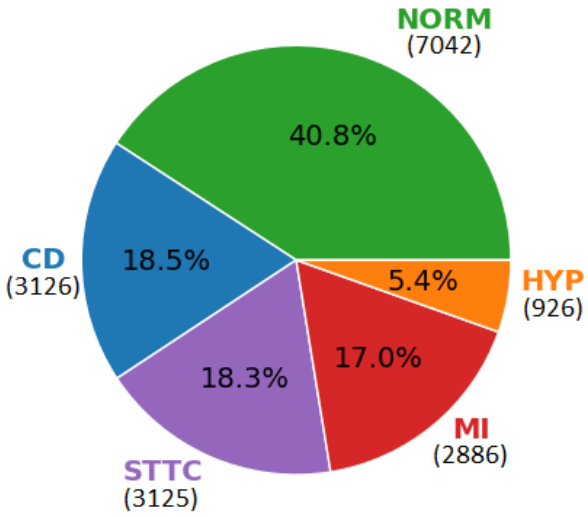


Figura 3.2: Diagrama de sectores con la distribución de superclases del *dataset* final con 17,105 ECG.

3.2.3. Extracción de características

A partir de la información de PTB-XL y de los resultados del ajuste del modelo 3DFMM$_{ecg}$ se extrajeron un total de 228 características, las cuales están divididas en los siguientes tres grupos:

Características demográficas

Indican las características propias del paciente al que se la he realizado la prueba del ECG.

- *age*: Edad del paciente.
- *sex*: Sexo del paciente (hombre 0, mujer 1).
- *height*: Altura del paciente (cm).
- *weight*: Peso del paciente (Kg).

Se extrajeron por tanto $1 + 1 + 1 + 1 = 4$ de estas características.

Características de ritmo

Hacen referencia a la secuencia y la disposición de los latidos del corazón. De las medidas detalladas a continuación, se calcularon dos estadísticos sobre el conjunto de latidos de cada ECG: la media y la desviación típica.

- RR: Duración de los intervalos RR entre cada par de latidos del ECG (ms).
- HR: Frecuencia cardíaca del ECG entre cada par de latidos del ECG (latidos/minuto). Calculada a partir de los intervalos RR como $60000/RR$.

Se extrajeron por tanto $2 \times (1 + 1) = 4$ de estas características.

Características generadas con los parámetros del modelo 3DFMM$_{ecg}$

Se obtuvieron a partir de los resultados del ajuste del modelo 3DFMM$_{ecg}$ sobre las 5 ondas P, Q, R, S y T del conjunto de 8 derivaciones $Lred = \{I, II, V1, V2, V3, V4, V5, V6\}$. Se calcularon dos estadísticos (media y desviación típica) sobre el conjunto de latidos de cada ECG. Además, se tuvo en cuenta que los parámetros α_J y β_J^L son circulares y, por tanto, se consideraron estadísticos específicos: media y desviación típica circular [48, 49].

- M^L: Término independiente de la derivación L.
- A_J^L: Parámetro de amplitud de la onda J de la derivación L.
- α_J: Parámetro de localización de la onda J.
- β_J^L: Parámetro de dirección del pico de la onda J de la derivación L.
- ω_J: Parámetro de anchura de la onda J.
- R_L^2: Medida de la bondad de ajuste sobre la derivación L.
- dRP, dRQ, dRS, y dRT: Distancias coseno entre ondas del ECG. Se definen como $1 - \cos(\alpha_R - \alpha_{J'})$, siendo α_R el parámetro de localización de la onda R y $\alpha_{J'}$ el parámetro de localización de la onda J' ($J' \in \{P, Q, S, T\}$).

Se extrajeron por tanto $2 \times (8 + 40 + 5 + 40 + 5 + 8 + 4) = 220$ de estas características.

3.2.4. Estandarización

Los algoritmos basados en árboles son invariantes frente a la escala, debido a que simplemente dividen los datos en diferentes regiones de acuerdo a sus características. Sin embargo, otros modelos sí que requieren que todas las variables estén en la misma escala para poder tratarlas a todas por igual. De otra forma, características medidas en escalas menores pueden ser despreciadas por el algoritmo de clasificación. Para los modelos de regresión logística y SVM se realizó una estandarización previa de cada variable, restando su media y dividiéndola por su desviación típica.

3.2.5. Imputación de valores ausentes

Como se comentó en el Capítulo 2, los algoritmos XGBoost y LightGBM incorporan su propio tratamiento nativo de valores ausentes, aunque esto no es lo habitual. La mayoría de algoritmos de aprendizaje requieren una matriz de diseño completa. Para el resto de algoritmos, se optó por imputar los valores perdidos de cada una de las variables con su mediana. Se eligió la mediana frente a la media por su robustez ante valores extremos.

Capítulo 4

Resultados

4.1. Diseño experimental

En primer lugar, se realizó una partición de los datos en dos subconjuntos, reservando un 80 % como conjunto de entrenamiento+validación y un 20 % de test. Sobre el primero, se empleó 10-fold para ajustar los hiperparámetros correspondientes mediante *randomized search* de los algoritmos de regresión logística, SVM, Random Forest, XGBoost y LightGBM. Para cada algoritmo, se escogió el modelo con la configuración de hiperparámetros que maximizase el promedio de macro AUC sobre los 10 *folds*. A continuación, se reajustaron cada uno de los correspondientes modelos con la totalidad de los datos de entrenamiento y validación. Finalmente, se evaluaron sobre el conjunto de test, de lo que se obtuvieron las curvas ROC para cada superclase (CD, HYP, MI, NORM, STTC), la matriz de confusión, las distintas métricas de clasificación con promedio macro *one-vs-rest*. Cabe destacar que tanto para la primera partición como para la propia validación cruzada se estratificó y se mantuvieron los ECG de un mismo paciente dentro del mismo conjunto.

Las distintas pruebas se ejecutaron en un ordenador equipado con una CPU AMD Ryzen 5 3600 @ 3,60 GHz (compuesta por 6 núcleos y un total de 12 hilos) bajo el sistema operativo Microsoft Windows 10 (versión 22H2 - compilación OS 19045.2486) utilizando Python 3.9.10.

4.2. Ajuste de modelos

A continuación, se detallarán los resultados obtenidos del mejor modelo ajustado con cada uno de los algoritmos estudiados.

4.2.1. Regresión logística

Hiperparámetros

Se obtuvo un AUC promedio de 0,898 con la siguiente configuración de hiperparámetros:

- penalty = "l1" (tipo de regularización)
- solver = "saga" (algoritmo de optimización)
- C = 0,1 (valor inverso del nivel de regularización)

En las curvas ROC de la Figura 4.1 se puede ver que los ECG normales y con hipertrofia tienen un AUC de en torno a 0,95 y los STTC de 0,909. Los CD y MI ofrecen unos resultados algo más bajos, con un AUC de 0,87 aproximadamente. En la Figura 4.2 se observa que se obtiene un AUC de 0,91 aproximadamente sobre el conjunto de test y un tiempo de entrenamiento de 1m 45s. En la matriz de confusión destacan 156 ECG con CD que están siendo clasificados erróneamente como normales.

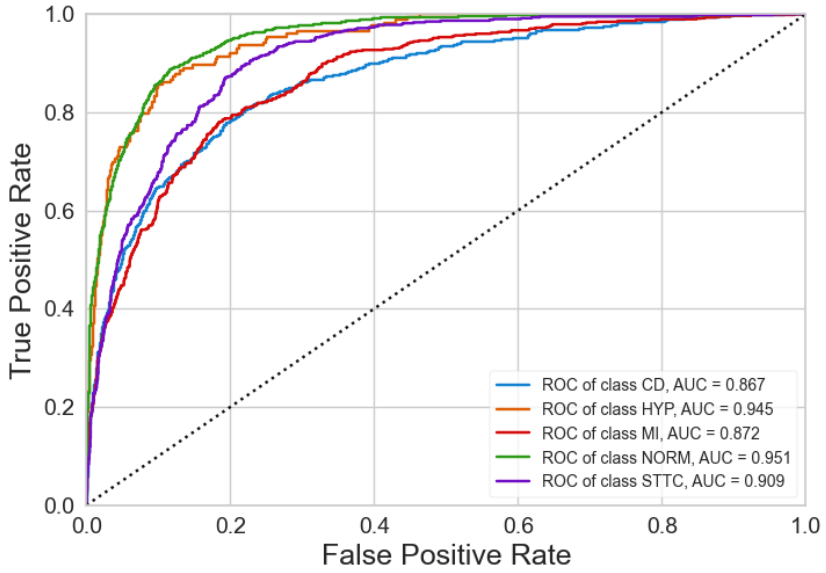
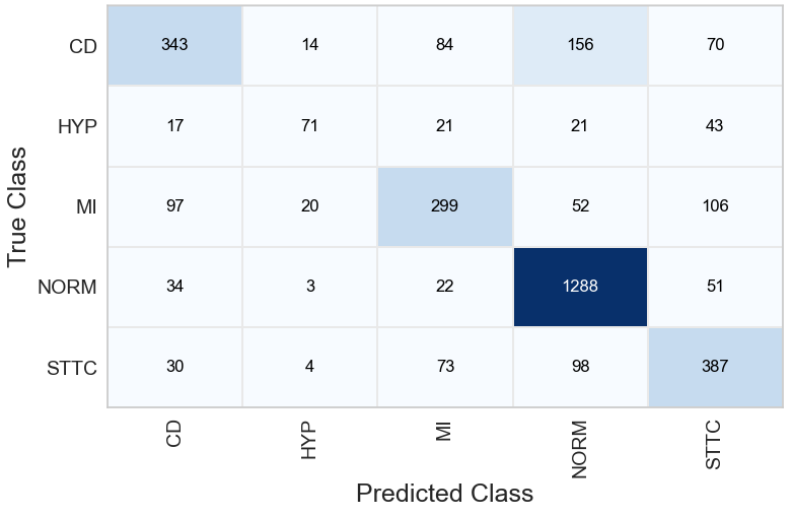


Figura 4.1: Curvas ROC del modelo de regresión logística para cada superclase.



- AUC = 0,909
- BACC = 0,761
- SE = 0,604
- SP = 0,919
- ACC = 0,702
- Training time = 1m 45s

Figura 4.2: Matriz de confusión y métricas sobre el conjunto de test del modelo de regresión logística.

4.2.2. SVM

Hiperparámetros

Se obtuvo un AUC promedio de 0,911 con la siguiente configuración de hiperparámetros:

- kernel = "rbf" (función kernel)

- gamma = 0,001 (parámetro de la función kernel RBF)
- C = 10 (valor inverso del nivel de regularización)

En las curvas ROC de la Figura 4.3 se observa que se obtienen unos resultados algo mejores que con el modelo de regresión logística. El AUC sobre la hipertrofia consigue superar el 0,95 y el infarto cardíaco el 0,9. El resto de clases también mejoran ligeramente. El promedio de los AUC es de 0,923 aproximadamente y el tiempo de entrenamiento ronda el minuto. En la Figura 4.4 vuelve a destacar el caso de los CD clasificados incorrectamente como normales, 145 observaciones concretamente.

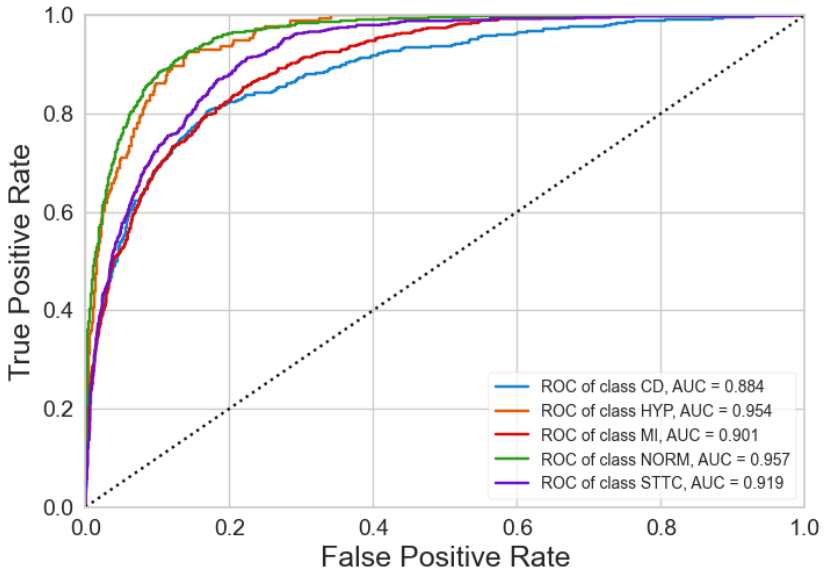
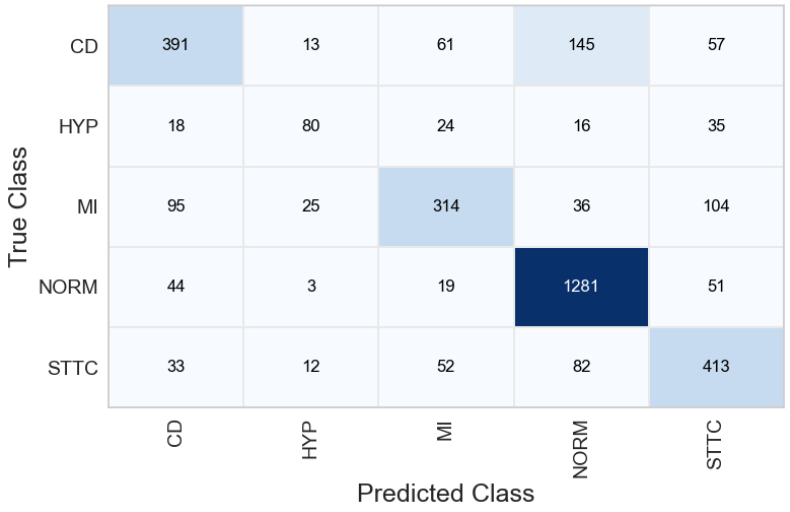


Figura 4.3: Curvas ROC del modelo SVM para cada superclase.



- AUC = 0,923
- BACC = 0,784
- SE = 0,642
- SP = 0,926
- ACC = 0,728
- Training time = 1m 3s

Figura 4.4: Matriz de confusión y métricas sobre el conjunto de test del modelo SVM.

4.2.3. Random Forest

Hiperparámetros

Se obtuvo un AUC promedio de 0,907 con la siguiente configuración de hiperparámetros:

- n_estimators = 350 (número de arboles de decisión)
- max_depth = None (máxima longitud de cada árbol)
- max_leaf_nodes = None (máximo número de hojas de cada árbol)
- max_features = "sqrt" (número máximo de atributos usados en cada árbol)
- min_samples_leaf = 2 (mínimo numero de observaciones en cada nodo hoja)

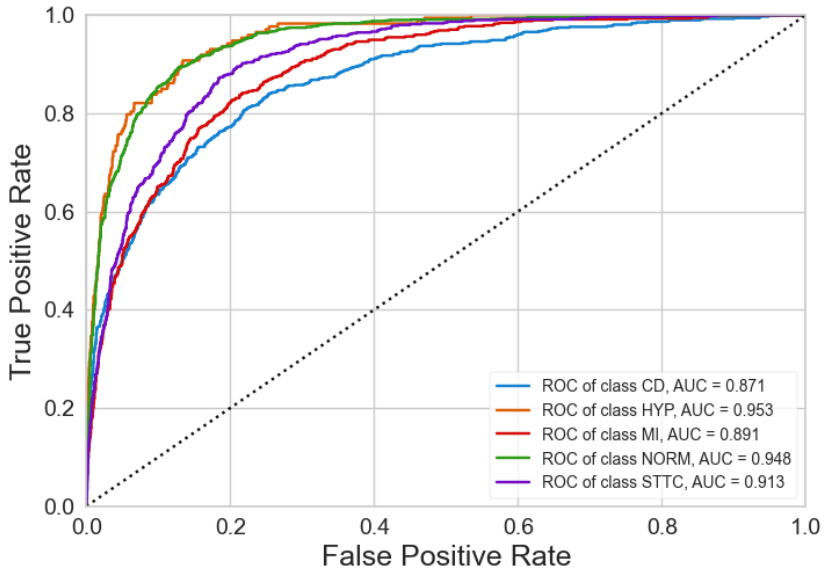
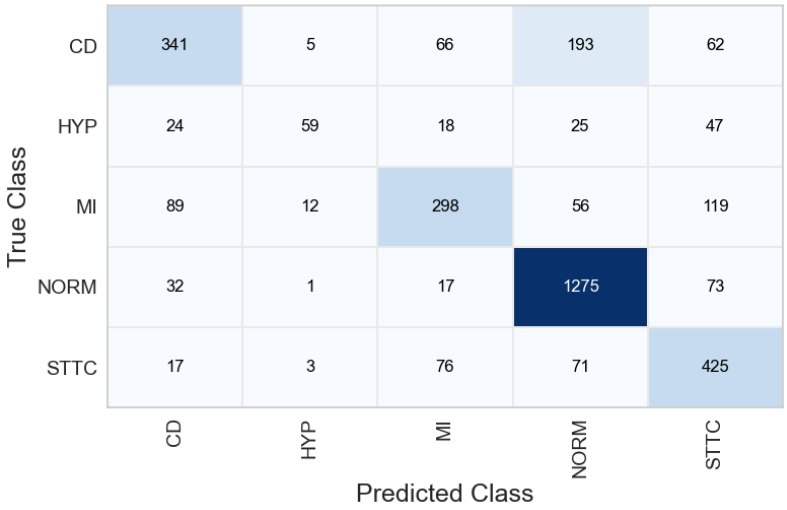


Figura 4.5: Curvas ROC del modelo Random Forest para cada superclase.



- AUC = 0,915
- BACC = 0,759
- SE = 0,600
- SP = 0,919
- ACC = 0,704
- Training time = 14s

Figura 4.6: Matriz de confusión y métricas sobre el conjunto de test del modelo Random Forest.

A la vista de los AUC de cada superclase (ver Figura 4.5), se observa que el modelo Random Forest tiene un desempeño intermedio entre la regresión logística y SVM. El AUC promedio es de 0,915 y, aunque es ligeramente menor que el de SVM, el tiempo de entrenamiento es significativamente menor (14s).

4.2.4. XGBoost

Hiperparámetros

Se obtuvo un AUC promedio de 0,919 con la siguiente configuración de hiperparámetros:

- n_estimators = 125 (número de arboles de decisión)
- learning_rate = 0,2 (tasa de aprendizaje de cada árbol)
- colsample_bytree = 0,9 (proporción de variables usadas en cada árbol)
- max_depth = 10 (máxima longitud de cada árbol)
- min_child_weight = 1 (suma mínima de pesos de instancia necesaria en una hoja)
- reg_alpha = 0 (término de regularización L1)
- reg_lambda = 1 (término de regularización L2)

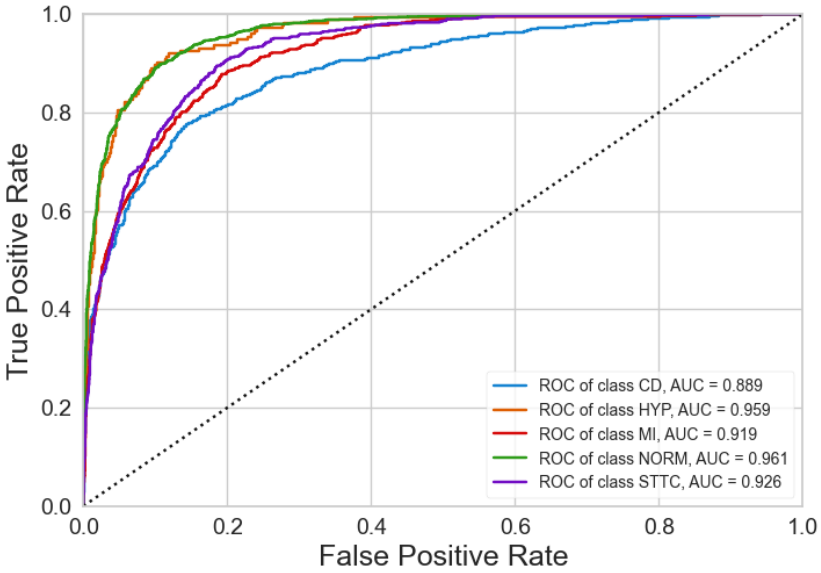
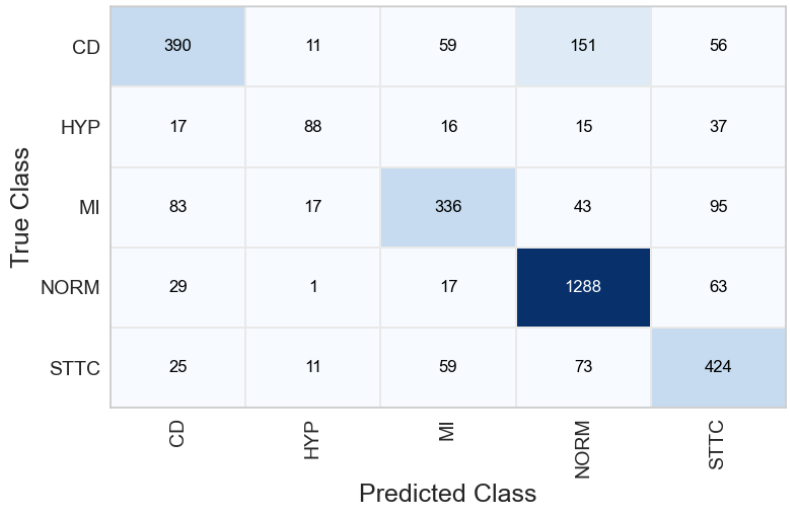


Figura 4.7: Curvas ROC del modelo XGBoost para cada superclase.

El modelo XGBoost consigue un mejor desempeño que los anteriores modelos. Destaca principalmente el aumento de la capacidad de clasificación sobre el infarto de miocardio, llegando a aproximarse más a la curva ROC de STTC que a la de CD (ver Figura 4.7), a diferencia de lo que ocurría con el resto de modelos. Consigue además superar los 0,93 de macro AUC (ver Figura 4.8), aunque es el segundo modelo más costoso computacionalmente en cuanto al tiempo entrenamiento, tardando algo más de un minuto y medio.



- AUC = 0,931
- BACC = 0,796
- SE = 0,663
- SP = 0,930
- ACC = 0,742
- Training time = 1m 42s

Figura 4.8: Matriz de confusión y métricas sobre el conjunto de test del modelo XGBoost.

4.2.5. LightGBM

Hiperparámetros

Se obtuvo un AUC promedio de 0,923 con la siguiente configuración de hiperparámetros:

- n_estimators = 250 (número de arboles de decisión)
- learning_rate = 0,05 (tasa de aprendizaje de cada árbol)
- colsample_bytree = 0,8 (proporción de variables usadas en cada árbol)
- max_depth = 30 (máxima longitud de cada árbol)
- min_child_samples = 32 (mínimo número de observaciones en cada hoja)
- reg_alpha = 2 (término de regularización L1)
- reg_lambda = 2 (término de regularización L2)

Se observa que el modelo LightGBM obtiene muy buenos resultados de clasificación, superando al resto de algoritmos y con un desempeño muy similar a XGBoost. Salvo la curva ROC del infarto de miocardio, el resto tienen mayor área que las respectivas curvas de XGBoost. El AUC promedio es de 0,933 y su tiempo de entrenamiento es de tan solo 12,3s (ver Figura 4.10).

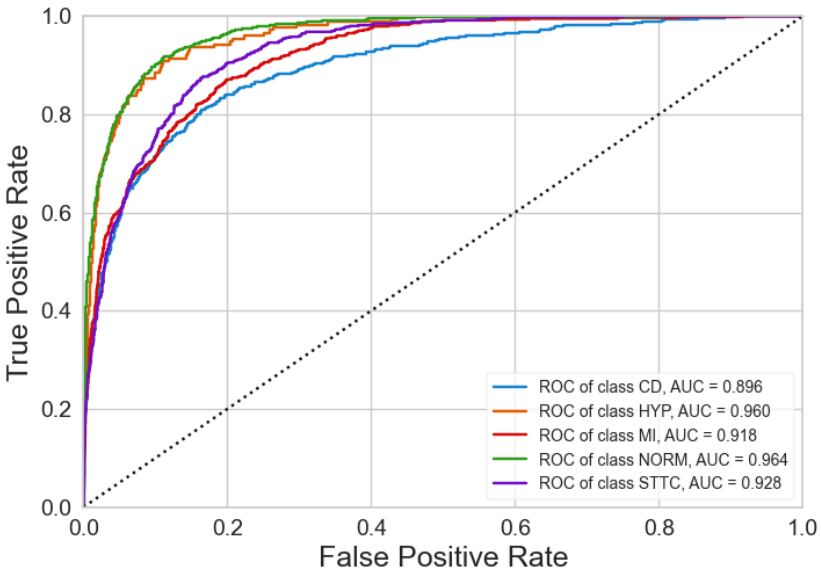


Figura 4.9: Curvas ROC del modelo LightGBM para cada superclase.

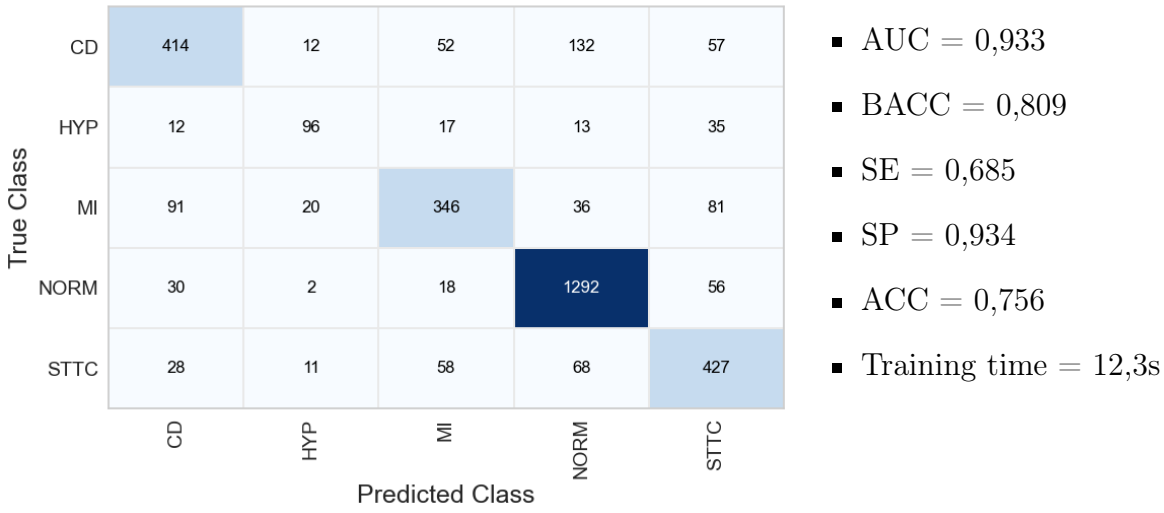


Figura 4.10: Matriz de confusión y métricas sobre el conjunto de test del modelo LightGBM.

4.3. Selección e interpretación del modelo final

Tras haber analizado los resultados de todos los modelos, se puede concluir que todos han tenido un desempeño satisfactorio, con macro AUC de más de 0,9. Aunque el vencedor indiscutible es el modelo LightGBM, con un macro AUC de 0,933, una macro sensibilidad de 0,685 y una macro especificidad de 0,934. Es sin duda el que obtiene métricas más altas junto con XGBoost, sin embargo, el tiempo de entrenamiento de LightGBM es de casi siete veces menos, por lo que se ha decidido escoger este último como modelo final.

Se han construido varios gráficos resumen para poder hacer una interpretación más detallada del modelo final. En dichos gráficos se puede observar la distribución de los valores SHAP para las 10 variables más importantes en la clasificación de cada una de las superclases. Las variables se

encuentran ordenadas en orden descendente según su importancia. La posición en el eje de abscisas hace referencia al valor SHAP. Para una variable concreta, las observaciones que se sitúen a la derecha (respecto al 0) tendrán mayor probabilidad de ser clasificadas como la clase en cuestión a igualdad del resto de variables, y viceversa. Por otro lado, el color hace referencia al valor que toma cada variable; los colores más rosados indican valores más altos de la variable y los azulados valores más bajos. A continuación, se detallarán las conclusiones obtenidas para cada una de las superclases.

Superclase CD

La distancia coseno media entre las ondas R y S resulta ser la variable más importante para la detección de bloqueos, teniendo estos valores más altos de la misma. La media del nivel de voltaje de la derivación II (M_II_mean) toma valores más bajos en los individuos de esta clase. También se observa que estos tipos de ECG tienen valores más altos de amplitud, concretamente de las variables A_S_II_mean, A_S_V1_mean, A_T_II_mean. También, valores altos en las variables de anchura ω_S_mean y ω_R_mean inducen a una clasificación de esta superclase.

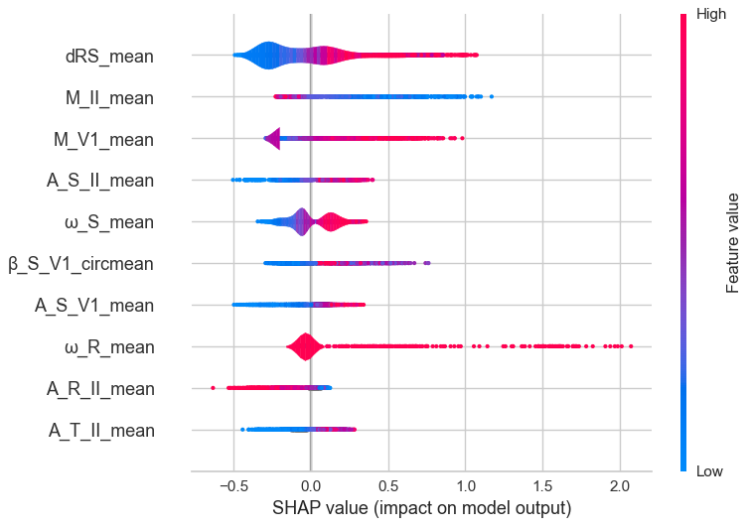


Figura 4.11: Top 10 variables más importantes del modelo final de la clasificación de CD mediante SHAP.

Superclase HYP

Los ECG con hipertrofia tienen valores más altos de amplitud de onda principalmente de la onda R (A_R_V5_mean, A_R_V1_mean, A_R_V6_mean y A_R_V2_mean), tal y como se esperaba. Por otra parte, se aprecia que el clasificador es más propenso a asignar a los individuos de mayor edad con hipertrofia. Llama también la atención la variable dRT_mean, la cual toma valores muy altos en presencia de la hipertrofia.

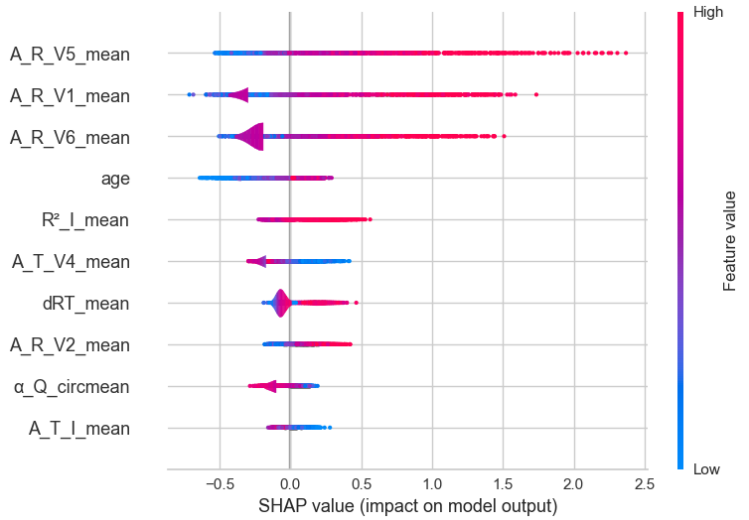


Figura 4.12: Top 10 variables más importantes del modelo final de la clasificación de HYP mediante SHAP.

Superclase MI

La clasificación sobre MI está fuertemente motivada por valores bajos del nivel de voltaje medio de la derivación II (M_II_mean). Los individuos de menor edad tienen, por el contrario, menor probabilidad de ser clasificados con infarto de miocardio. También se aprecian amplitudes bajas de la onda Q, concretamente de la derivación V2 y V3 (A_Q_V2_mean y A_Q_V3_mean). Por otro lado, amplitudes altas en la onda P de la primera derivación monopolar (A_P_I_mean) sí que favorecen la clasificación como infarto. Se puede observar que las mujeres obtienen valores de SHAP más bajos, por lo que existe una menor tendencia a clasificarlas con infarto a igualdad del resto de variables. Aunque aparecen otras variables como el peso (weight) y la variable de dirección de onda β_Q_II_mean, no aparece una división clara de los puntos de corte a la vista del gráfico, por lo que no se puede hacer una interpretación directa.

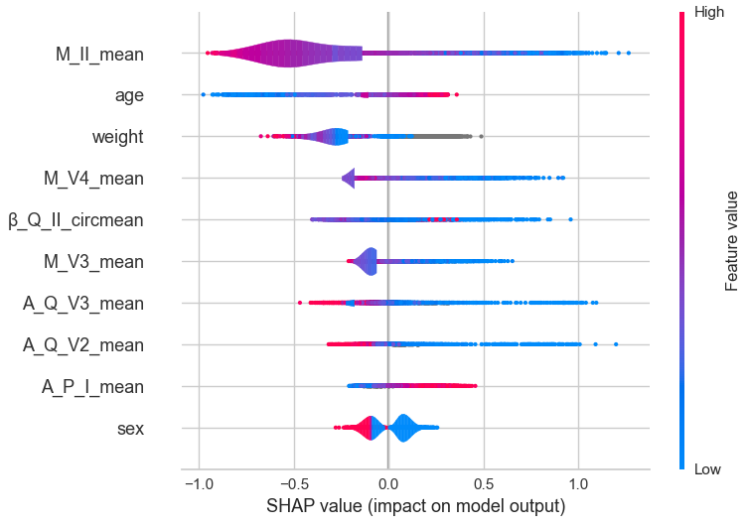


Figura 4.13: Top 10 variables más importantes del modelo final de la clasificación de MI mediante SHAP.

Superclase NORM

Los ECG clasificados como normales tienden a tener valores más altos del nivel de voltaje medio de la derivación II (M_II_mean), al contrario de lo que ocurría con la hipertrofia y el infarto de miocardio. También aparece el voltaje medio de la derivación V1 (M_V1_mean), la cual sí que toma valores más bajos en los ECG normales. Se concluye además que tener valores extremadamente bajos de amplitud de la onda T en derivaciones tales como la II, la V5 o la I (A_T_II_mean, A_T_V5_mean y A_T_V1_mean) favorece una clasificación como no normal. De la variables de amplitud de la onda T y de las medias circulares β (β_T_I_circmean y β_T_V6_circmean) se puede decir que toman valores más altos en los ECG normales. Los pacientes más jóvenes también suelen ser clasificados como normales, lo cual resulta lógico. Finalmente, también es bastante reseñable la gran cantidad de individuos con valores bajos de ω_R_mean clasificados como normales, al contrario que ocurría con CD.

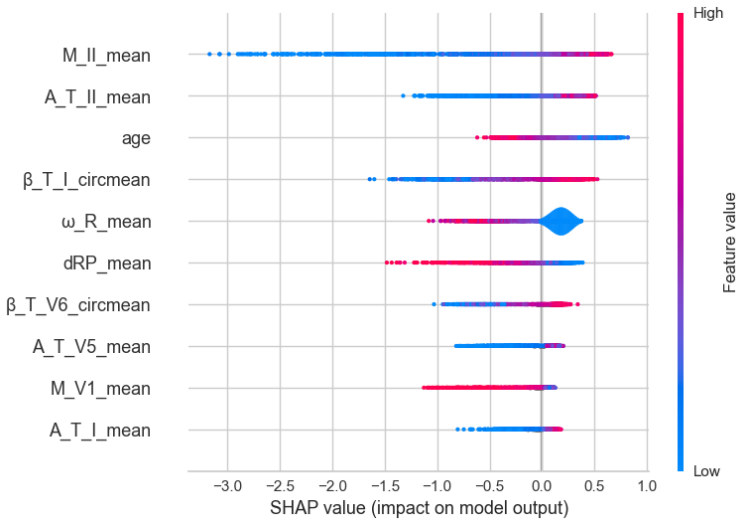


Figura 4.14: Top 10 variables más importantes del modelo final de la clasificación de NORM mediante SHAP.

Superclase STTC

Los ECG clasificados con STTC tienen valores más bajos de las amplitudes de la onda T en la gran mayoría de derivaciones: A_T_V5_mean, A_T_V6_mean, A_T_I_mean, etc. Por el contrario, los individuos con este tipo de patología tienen valores promedios más altos de amplitud de la onda R, concretamente de las derivaciones V5 y V6 (A_R_V5_mean y A_R_V6_mean). El nivel de voltaje medio no tiene tanta relevancia como la tiene en la clasificación de otras superclases ya que aparece en penúltima posición, pero se puede apreciar que toma valores más altos en presencia de STTC. Finalmente, se observa que las observaciones con valores extremadamente altos del promedio de la distancia entre las ondas R y T (dRT_mean) son clasificadas también con este tipo de enfermedad.

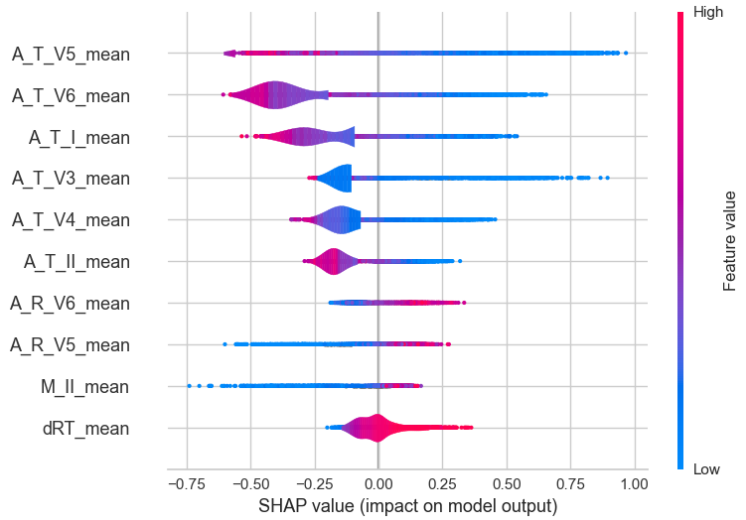


Figura 4.15: Top 10 variables más importantes del modelo final de la clasificación de STTC mediante SHAP.

4.4. Comparación con otros trabajos

En la literatura reciente, existen otras investigaciones que también se han centrado en la clasificación de la superclase de la base de datos PTB-XL. Como se comentó anteriormente en el Capítulo 1, la mayoría hacen uso de arquitecturas de aprendizaje profundo que toman como entrada las señales completas del ECG, por lo que la comparación en términos interpretativos es limitada. En el artículo de 2021 [6] se presenta una arquitectura de red convolucional con 58.269 parámetros. Obtienen un AUC de 0,910, siendo incluso algo inferior a los 0,933 que se obtuvieron con el modelo LightGBM propuesto en este trabajo. Otro artículo más reciente de 2022 [7] consigue mejorar los resultados de clasificación, combinando estas mismas redes convoluciones profundas con modelos de *machine learning* conectados a la última capa. Concretamente, conectando un SVM con *kernel* lineal, estiman un AUC medio de 0,938, apenas una centésima superior al del modelo LightGBM de este trabajo. Sin embargo, hay que tener en cuenta que ninguno de los artículos menciona que se haya respetado la estructura de grupos de los ECG, ni que se haya estratificado, ni tampoco explican claramente el criterio que han tomado a la hora de etiquetar los ECG. Con todo esto se puede deducir que quizás puedan estar realizando unas estimaciones del error bastante optimistas. Además, hay que tener en cuenta las otras dos claras ventajas añadidas de este trabajo: la simplicidad (228 variables y 12s de entrenamiento) y la interpretación clínica obtenida gracias a los parámetros del modelo 3DFMM_{ecg}.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

La combinación del modelo 3DFMM$_{ecg}$ con algoritmos de ML ha sido completamente satisfactoria, habiendo alcanzado más de 0,9 de macro AUC sobre el conjunto de test. La principal ventaja del enfoque propuesto reside en su gran capacidad interpretativa. Tan solo se han usado 228 variables predictoras y se han medido tiempos de entrenamiento que en ningún caso superan los 2 minutos, en contraste con la gran abundancia de arquitecturas de redes neuronales profundas con decenas de miles de parámetros propuestas en la literatura científica.

En cuanto a los distintos modelos ajustados, todos han logrado obtener resultados de un orden similar, aunque destacan claramente las técnicas de *boosting* por encima del resto. Concretamente, con LightGBM se han obtenido los mejores resultados (0,933 de macro AUC) e incluso el menor tiempo de entrenamiento, de 12s aproximadamente. Las superclases mejor clasificadas han sido NORM e HYP, con un AUC de en torno a 0,96. En siguiente lugar, los ECG con MI o STTC se clasifican con un AUC de 0,92. La superclase peor clasificada es CD con un AUC ligeramente inferior al resto, de casi 0,90. En cualquier caso, los valores obtenidos son comparables con los de la literatura reciente o incluso mejores en muchos casos.

Se han encontrado estadísticos sobre los parámetros del modelo 3DFMM$_{ecg}$ con un gran potencial discriminante entre las clases. En [50] se utilizaron reglas de clasificación sencillas directamente sobre los parámetros del modelo 3DFMM$_{ecg}$ de algunas enfermedades concretas de bloqueo de rama (subconjunto de CD) de la base de datos PTB-XL. En dicho artículo ya se observó la importancia de las variables ω_R y ω_S. En este trabajo, también se ha constatado la importancia de otras variables como la distancia media entre las ondas R y S, y los parámetros A_S^{II} y A_S^{V1}. En cuanto a la hipertrofia cardíaca, los resultados obtenidos en este trabajo reflejan el conocimiento previo sobre la patología, siendo las amplitudes de la onda R en diferentes derivaciones los parámetros más relevantes. En [51] se hace uso del modelo FMM$_{ecg}$ (precursor del modelo 3DFMM$_{ecg}$) para la clasificación del infarto de miocardio; concluyen que las variables de amplitud de la onda T son las más influyentes. En este trabajo, se ha comprobado además la importancia de los parámetros A_Q^{V2} y A_Q^{V3}. Para STTC, las variables más importantes han sido las amplitudes de las ondas T de las derivaciones V5 y V6 principalmente, lo cual es coherente con la propia morfología de la enfermedad, al manifestarse alteraciones en el segmento ST. También es reseñable la importancia de la amplitud de la onda R en las derivaciones V5 y V6, del nivel de voltaje medio de la derivación II y de la distancia media entre las ondas R y T.

Finalmente, cabe destacar que durante la etapa de experimentación de este trabajo, en la que se consideraron otras patologías más allá de las superclases, se constató que valores altos del parámetro A_P^{II} identifican la patología "RAO/RAE" (*Right Atrial Overload/Right Atrial Enlargement*), la cual es un tipo de hipertrofia. Específicamente, con la regla *naive*: $A_P^{II} > 100$, se obtiene un AUC de 0,941, una sensibilidad de 0,973 y una especificad de 0,909. Dicha regla tiene además una gran interpretación electrofisiológica y es útil en la práctica clínica, debido a que los cardiólogos elaboran su diagnóstico según las anomalías que observen en la onda P.

5.2. Trabajo futuro

A la vista de los resultados, surgen multitud de líneas de trabajo futuro. Por un lado, la definición de nuevas características a partir de los parámetros del modelo 3DFMM$_{ecg}$ o el uso de otras técnicas de ML tales como Naive Bayes, KNN o análisis discriminante, entre otras variantes, pueden llevar a afinar los resultados de clasificación. Por otro lado, queda completamente abierto el problema *multilabel*, el cual puede resolverse usando diferentes estrategias.

En definitiva, a pesar de que el ECG ha sido utilizado durante más de un siglo, todavía siguen surgiendo nuevas metodologías y enfoques para abordar el problema del diagnóstico médico automático, el cual no es en absoluto trivial. Aún queda mucho para conseguir un procedimiento de clasificación fiable que se incorpore en el día a día de la práctica clínica, mediante el cual se puedan detectar enfermedades en etapas más tempranas y se elaboren tratamientos más efectivos. Este trabajo es una contribución para conseguir alcanzar dicho logro.

Bibliografía

[1] Antoni Bayés de Luna. *Manual de Electrocardiografía Básica*. 13º ed. CADUCEO MULTIMEDIA, S. L., 2014. Cap. 1.

[2] Npatchett. *Spatial orientation of EKG leads*. 27 de mar. de 2015. URL: https://commons.wikimedia.org/wiki/File:EKG_leads.png (Último acceso 18-03-2023).

[3] Ewingdo. *ECG NSR with RBBB 74 bpm*. 16 de nov. de 2020. URL: https://commons.wikimedia.org/wiki/File:ECG_NSR_with_RBBB_74_bpm.jpg (Último acceso 18-03-2023).

[4] Saira Aziz, Sajid Ahmed y Mohamed-Slim Alouini. «ECG-based machine-learning algorithms for heartbeat classification». En: *Scientific Reports* 11.1 (sep. de 2021). DOI: 10.1038/s41598-021-97118-5. URL: https://doi.org/10.1038/s41598-021-97118-5.

[5] P. Wagner et al. «PTB-XL, a large publicly available electrocardiography dataset». En: *Scientific Data* 7 (2020). DOI: 10.1038/s41597-020-0495-6. URL: https://doi.org/10.1038/s41597-020-0495-6.

[6] Sandra Śmigiel, Krzysztof Pałczyński y Damian Ledziński. «ECG Signal Classification Using Deep Learning Techniques Based on the PTB-XL Dataset». En: *Entropy* 23.9 (2021). ISSN: 1099-4300. DOI: 10.3390/e23091121. URL: https://www.mdpi.com/1099-4300/23/9/1121.

[7] Krzysztof Pałczyński et al. «Study of the Few-Shot Learning for ECG Classification Based on the PTB-XL Dataset». En: *Sensors* 22.3 (2022). ISSN: 1424-8220. DOI: 10.3390/s22030904. URL: https://www.mdpi.com/1424-8220/22/3/904.

[8] Nils Strodthoff et al. «Deep Learning for ECG Analysis: Benchmarks and Insights from PTB-XL». En: *CoRR* abs/2004.13701 (2020). arXiv: 2004.13701. URL: https://arxiv.org/abs/2004.13701.

[9] Zicong Li y Henggui Zhang. «Automatic Detection for Multi-Labeled Cardiac Arrhythmia Based on Frame Blocking Preprocessing and Residual Networks». En: *Frontiers in Cardiovascular Medicine* 8 (mar. de 2021). DOI: 10.3389/fcvm.2021.616585. URL: https://doi.org/10.3389/fcvm.2021.616585.

[10] Raymond Ao y George He. «Image based deep learning in 12-lead ECG diagnosis». En: *Frontiers in Artificial Intelligence* 5 (ene. de 2023). DOI: 10.3389/frai.2022.1087370. URL: https://doi.org/10.3389/frai.2022.1087370.

[11] Mark Nicholls. *AI in cardiology: a marriage made in heaven – or hell?* 2021. URL: https://healthcare-in-europe.com/en/news/ai-in-cardiology-a-marriage-made-in-heaven-or-hell.html (Último acceso 27-04-2023).

[12] Cristina Rueda Sabater. *Grupo de Investigación de Inferencia Estadística con Restricciones*. URL: http://www.eio.uva.es/gir-ier/ (Último acceso 16-02-2023).

[13] Cristina Rueda Sabater. *The FMM Project*. URL: http://www.eio.uva.es/the-fmm-project/ (Último acceso 16-02-2023).

[14] Cristina Rueda et al. «A unique cardiac electrocardiographic 3D model. Toward interpretable AI diagnosis». En: *iScience* 25.12 (2022), pág. 105617. ISSN: 2589-0042. DOI: https://doi.org/10.1016/j.isci.2022.105617. URL: https://www.sciencedirect.com/science/article/pii/S2589004222018892.

[15] Adolfo Fernández Santamónica, Rocío Carratalá Sáez y Cristina Rueda Sabater. «ECGMiner: un software para la digitalización de electrocardiogramas». Trabajo de Fin de Grado. Departamento de Informática de la Universidad de Valladolid, 2023.

[16] Yolanda Larriba et al. «Order restricted inference in chronobiology». En: *Statistics in Medicine* 39 (3 nov. de 2019), págs. 265-278. DOI: 10.1002/sim.8397. URL: https://onlinelibrary.wiley.com/doi/10.1002/sim.8397.

[17] Alejandro Rodríguez-Collado y Cristina Rueda. «Functional Clustering of Neuronal Signals with FMM Mixture Models». En: 2022.

[18] Cristina Rueda, Yolanda Larriba y Adrian Lamela. «The hidden waves in the ECG uncovered revealing a sound automated interpretation method». En: *Scientific Reports* 11.1 (feb. de 2021), pág. 3724. ISSN: 2045-2322. DOI: 10.1038/s41598-021-82520-w. URL: https://doi.org/10.1038/s41598-021-82520-w.

[19] Cristina Rueda et al. «The FMM Approach to Analyze Biomedical Signals: Theory, Software, Applications and Future». En: *Mathematics* 9.10 (2021). ISSN: 2227-7390. DOI: 10.3390/math9101145. URL: https://www.mdpi.com/2227-7390/9/10/1145.

[20] J H Holt Jr et al. «A study of the human heart as a multiple dipole electrical source. I. Normal adult male subjects». en. En: *Circulation* 40.5 (nov. de 1969), págs. 687-696.

[21] Mario Versaci, Giovanni Angiulli y Fabio La Foresta. «A Modified Heart Dipole Model for the Generation of Pathological ECG Signals». En: *Computation* 8.4 (2020). ISSN: 2079-3197. DOI: 10.3390/computation8040092. URL: https://www.mdpi.com/2079-3197/8/4/92.

[22] Jiapu Pan y Willis J. Tompkins. «A Real-Time QRS Detection Algorithm». En: *IEEE Transactions on Biomedical Engineering* BME-32.3 (1985), págs. 230-236. DOI: 10.1109/TBME.1985.325532.

[23] Maria Teresa González Arteaga. *Apuntes de la asignatura "Modelos Estadísticos Avanzados"*. Departamento de Estadística e Investigación Operativa, Facultad de Ciencias. Universidad de Valladolid. 2023.

[24] Trevor Hastie, Robert Tibshirani y Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2º ed. Springer, 2009. Cap. 12. URL: https://hastie.su.domains/Papers/ESLII.pdf.

[25] Larhmam. *SVM margin*. 2018. URL: https://commons.wikimedia.org/wiki/File:SVM_margin.png (Último acceso 25-04-2023).

[26] scikit-learn. *Plot different SVM classifiers in the iris dataset*. 2016. URL: https://scikit-learn.org/0.18/auto_examples/svm/plot_iris.html (Último acceso 25-04-2023).

[27] Trevor Hastie, Robert Tibshirani y Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2º ed. Springer, 2009. Cap. 15. URL: https://hastie.su.domains/Papers/ESLII.pdf.

[28] TseKiChun. *Random forest explain*. 2021. URL: https://commons.wikimedia.org/wiki/File:Random_forest_explain.png (Último acceso 25-04-2023).

[29] L. Breiman. «Arcing the edge». En: 1997.

[30] Jerome H. Friedman. «Greedy function approximation: A gradient boosting machine.» En: *The Annals of Statistics* 29.5 (2001), págs. 1189-1232. DOI: 10.1214/aos/1013203451. URL: https://doi.org/10.1214/aos/1013203451.

[31] Llew Mason et al. «Boosting Algorithms as Gradient Descent». En: *Advances in Neural Information Processing Systems*. Ed. por S. Solla, T. Leen y K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/96a93ba89a5b5c6c226e49b88973f46e-Paper.pdf.

[32] Aratrika Pal. *Gradient Boosting Trees for Classification: A Beginner's Guide*. 2020. URL: https://medium.com/swlh/gradient-boosting-trees-for-classification-a-beginners-guide-596b594a14ea (Último acceso 29-04-2023).

[33] Tianqi Chen y Carlos Guestrin. «XGBoost: A Scalable Tree Boosting System». En: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, págs. 785-794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[34] Guolin Ke et al. «Lightgbm: A highly efficient gradient boosting decision tree». En: *Advances in neural information processing systems* 30 (2017), págs. 3146-3154.

[35] Microsoft Corporation. *Leaf-wise (Best-first) Tree Growth*. 2023. URL: https://lightgbm.readthedocs.io/en/latest/Features.html (Último acceso 26-04-2023).

[36] Carlos Alonso González. *Apuntes de la asignatura "Minería de Datos"*. Departamento de Informática, Facultad de Ciencias. Universidad de Valladolid. 2023.

[37] Mohit Ingale et al. «ECG Biometric Authentication: A Comparative Analysis». En: *IEEE Access* 8 (2020), págs. 117853-117866.

[38] Martin Thoma. *Roc curve*. 2018. URL: https://commons.wikimedia.org/wiki/File:Roc_curve.svg?uselang=ca (Último acceso 25-04-2023).

[39] *"Why Should I Trust You?" Explaining the Predictions of Any Classifier*. 2016, págs. 1135-1144. DOI: 10.1145/2939672.2939778.

[40] Alexandre Heuillet, Fabien Couthouis y Natalia Díaz Rodríguez. «Collective eXplainable AI: Explaining Cooperative Strategies and Agent Contribution in Multiagent Reinforcement Learning with Shapley Values». En: *CoRR* abs/2110.01307 (2021). URL: https://arxiv.org/abs/2110.01307.

[41] Scott M Lundberg y Su-In Lee. «A Unified Approach to Interpreting Model Predictions». En: *Advances in Neural Information Processing Systems*. Ed. por I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[42] A. L. Goldberger et al. «PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals». En: *Circulation* 101.23 (2000), págs. 215-220. DOI: 10.1161/01.CIR.101.23.e215. URL: https://www.ahajournals.org/doi/10.1161/01.cir.101.23.e215.

[43] P. Wagner et al. *PTB-XL, a large publicly available electrocardiography dataset (version 1.0.1)*. 2020. URL: https://doi.org/10.13026/x4td-x982.

[44] Erick A Perez Alday et al. «Classification of 12-lead ECGs: the PhysioNet/Computing in Cardiology Challenge 2020». En: *Physiological Measurement* 41.12 (dic. de 2020), pág. 124003. DOI: 10.1088/1361-6579/abc960. URL: https://dx.doi.org/10.1088/1361-6579/abc960.

[45] *ISO Central Secretary. Health informatics – Standard communication protocol – Part 91064: Computer-assisted electrocardiography. Standard ISO 11073-91064:2009, International Organization for Standardization, Geneva, CH (2009).*

[46] *Chinese Electrocardiographic Society, CVEWG 2017 consensus of Chinese experts on ECG critical value (in chinese). J Clin Electrocardiol. 2017;026(006):401–02.*

[47] Guodong Wei et al. «Estimating critical values from electrocardiogram using a deep ordinal convolutional neural network». En: *BMC Medical Informatics and Decision Making* 22.1 (nov. de 2022), pág. 295. ISSN: 1472-6947. DOI: 10.1186/s12911-022-02035-w. URL: https://doi.org/10.1186/s12911-022-02035-w.

[48] Arthur Pewsey, Markus Neuhäuser y Graeme D. Ruxton. *Circular Statistic in R.* 1º ed. Oxford, 2013. Cap. 3.

[49] N.I. Fisher. *Statistical analysis of circular data.* 1º ed. Cambridge University Press, 1993. Cap. 2.

[50] Cristina Rueda et al. «Compelling new electrocardiographic markers for automatic diagnosis». En: *Computer Methods and Programs in Biomedicine* 221 (2022), pág. 106807. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2022.106807. URL: https://www.sciencedirect.com/science/article/pii/S0169260722001894.

[51] Yingyu Yang et al. «Explainable Electrocardiogram Analysis with Wave Decomposition: Application to Myocardial Infarction Detection». En: *Statistical Atlases and Computational Models of the Heart. Regular and CMRxMotion Challenge Papers*. Ed. por Oscar Camara et al. Cham: Springer Nature Switzerland, 2022, págs. 221-232. ISBN: 978-3-031-23443-9.

Apéndice A

Anexos

A.1. Dependencias

En este anexo se indica la lista con las dependencias necesarias para la correcta ejecución de los experimentos.

```
asttokens==2.2.1
backcall==0.2.0
black==23.3.0
click==8.1.3
cloudpickle==2.2.1
colorama==0.4.6
comm==0.1.3
contourpy==1.0.7
cycler==0.11.0
debugpy==1.6.7
decorator==5.1.1
executing==1.2.0
fonttools==4.39.3
importlib-metadata==6.3.0
importlib-resources==5.12.0
ipykernel==6.22.0
ipython==8.12.0
jedi==0.18.2
joblib==1.2.0
jupyter_client==8.1.0
jupyter_core==5.3.0
kiwisolver==1.4.4
lazypredict==0.2.12
lightgbm==3.3.5
llvmlite==0.39.1
matplotlib==3.7.1
matplotlib-inline==0.1.6
mypy-extensions==1.0.0
nest-asyncio==1.5.6
numba==0.56.4
numpy==1.23.5
packaging==23.0
pandas==2.0.0
parso==0.8.3
pathspec==0.11.1
pickleshare==0.7.5
Pillow==9.5.0
```

```
platformdirs==3.2.0
prompt-toolkit==3.0.38
psutil==5.9.4
pure-eval==0.2.2
Pygments==2.14.0
pyparsing==3.0.9
python-dateutil==2.8.2
pytorch-tabnet==4.0
pytz==2023.3
pywin32==306
pyzmq==25.0.2
scikit-learn==1.2.2
scipy==1.10.1
seaborn==0.12.2
shap==0.41.0
six==1.16.0
slicer==0.0.7
stack-data==0.6.2
threadpoolctl==3.1.0
tomli==2.0.1
torch==1.13.1
tornado==6.2
tqdm==4.65.0
traitlets==5.9.0
typing_extensions==4.5.0
tzdata==2023.3
wcwidth==0.2.6
xgboost==1.7.5
yellowbrick==1.5
zipp==3.15.0
```

A.2. Preprocesamiento

En este anexo se incluye el Jupyter Notebook (convertido a un script de Python) que se ha usado para la etapa de preprocesado.

```python
# %% [markdown]
# ## Imports

# %%
import re

import pandas as pd
from scipy.stats import circmean, circstd

# %% [markdown]
# ## Dataset loading

# %% [markdown]
# ### PTB-XL database
# It contains each patient's demographic information and diagnosis codes.

# %%
ptbxl = pd.read_csv("./data/ptbxl_database.csv")
ptbxl["patient_id"] = ptbxl["patient_id"].astype(int)
print(ptbxl.shape)
ptbxl.head()
```

```
22
23
24 # %% [markdown]
25 # ### 3DFMMecg parameters
26 # It contains the parameters of the models fitted on each of the ECG beats.
27
28 # %%
29 LEADS = ["I", "II", "III", "V1", "V2", "V3", "V4", "V5", "V6"]
30 WAVES = ["P", "Q", "R", "S", "T"]
31
32 fmm = pd.read_csv("./data/ptbxl_3dfmm.csv", sep=";")
33 fmm = fmm.rename(columns={"EcgId": "ecg_id"})
34 fmm = fmm.rename(
35     columns=lambda x: re.sub("Alpha", "α_", x)
36     if any(w in x for w in WAVES)
37     else x
38 )
39 fmm = fmm.rename(
40     columns=lambda x: re.sub("Beta", "β_", x)
41     if any(w in x for w in WAVES)
42     else x
43 )
44 fmm = fmm.rename(
45     columns=lambda x: re.sub("Omega", "ω_", x)
46     if any(w in x for w in WAVES)
47     else x
48 )
49 fmm = fmm.rename(
50     columns=lambda x: re.sub("^A", "A_", x)
51     if any(l in x for l in LEADS)
52     else x
53 )
54 fmm = fmm.rename(columns=lambda x: re.sub("R2", "$R^2$", x))
55
56 m = [x for x in fmm.columns if re.findall("M_", x)]
57 alpha = [x for x in fmm.columns if re.findall("α", x)]
58 beta = [x for x in fmm.columns if re.findall("β", x)]
59 omega = [x for x in fmm.columns if re.findall("ω", x)]
60 amplitude = [x for x in fmm.columns if re.findall("A_", x)]
61 r2 = [x for x in fmm.columns if re.findall("$R^2$", x)]
62 dist = [x for x in fmm.columns if re.findall("^d\w\w", x)]
63 fmm = fmm.loc[
64     :, ["ecg_id", "Age", "Sex"] + alpha + beta + omega + amplitude + r2 + dist + m
65 ]
66
67 print(fmm.shape)
68 fmm.head()
69
70
71 # %% [markdown]
72 # ### 3DFMMecg annotations
73
74 # %% [markdown]
75 # It contains the R-wave annotations for each of the ECG beats.
76
77 # %%
78 beat_annot_df = pd.read_csv("./data/ptbxl_beats.txt", sep="\t")
79 print(beat_annot_df.shape)
```

```
beat_annot_df.head()


# %% [markdown]
# ## Labelling

# %% [markdown]
# 1. Non-diagnostic labels are removed
#
# 2. Eliminate labels that do not have 100% likelihood.
#
# 3. The superclass is chosen according to the diagnosis classified as having a higher risk.

# %%
def get_labels(x:str) -> str:
    """
    Given a dictionary of SCP-ECG codes with their respective likelihoods, the superclass label is returned according to the max diagnostic risk criterion.

    Args:
        x (str): String with dictionary format with the SCP-ECG codes.

    Returns:
        str: Superclass label.
    """

    DIAGNOSTICS = (
        # Risk 0
        ["NORM"]
        # Risk 1
        + [
            "SEHYP",
            "RVH",
            "LPFB",
            "RAO/RAE",
            "ILBBB",
            "ISCLA",
            "LAO/LAE",
            "1AVB",
            "ISC_",
            "CRBBB",
            "NST_",
            "IRBBB",
            "LAFB",
            "NDT",
        ]
        # Risk 2
        + [
            "INJIN",
            "INJLA",
            "INJIL",
            "2AVB",
            "ISCAN",
            "DIG",
            "EL",
            "ISCIL",
            "ISCIN",
```

```
            "ISCAS",
            "INJAS",
            "IVCD",
            "ISCAL",
            "CLBBB",
            "LVH",
        ]
        # Risk 3
        + [
            "ANEUR",
            "3AVB",
            "PMI",
            "IPMI",
            "INJAL",
            "IPLMI",
            "LMI",
            "WPW",
            "LNGQT",
            "ALMI",
            "AMI",
            "ILMI",
            "ASMI",
            "IMI",
        ]
    )

    SUB2SUPER = {
        "NORM": "NORM",
        "LAFB/LPFB": "CD",
        "IRBBB": "CD",
        "ILBBB": "CD",
        "CLBBB": "CD",
        "CRBBB": "CD",
        "_AVB": "CD",
        "IVCD": "CD",
        "WPW": "CD",
        "LVH": "HYP",
        "RVH": "HYP",
        "LAO/LAE": "HYP",
        "RAO/RAE": "HYP",
        "SEHYP": "HYP",
        "AMI": "MI",
        "IMI": "MI",
        "LMI": "MI",
        "PMI": "MI",
        "ISCA": "STTC",
        "ISCI": "STTC",
        "ISC_": "STTC",
        "NST_": "STTC",
        "STTC": "STTC",
    }

    DIAG2SUB = {
        "LAFB": "LAFB/LPFB",
        "IRBBB": "IRBBB",
        "1AVB": "_AVB",
        "IVCD": "IVCD",
        "CRBBB": "CRBBB",
```

```
        "CLBBB": "CLBBB",
        "LPFB": "LAFB/LPFB",
        "WPW": "WPW",
        "ILBBB": "ILBBB",
        "3AVB": "_AVB",
        "2AVB": "_AVB",
        "LVH": "LVH",
        "LAO/LAE": "LAO/LAE",
        "RVH": "RVH",
        "RAO/RAE": "RAO/RAE",
        "SEHYP": "SEHYP",
        "IMI": "IMI",
        "ASMI": "AMI",
        "ILMI": "IMI",
        "AMI": "AMI",
        "ALMI": "AMI",
        "INJAS": "AMI",
        "LMI": "LMI",
        "INJAL": "AMI",
        "IPLMI": "IMI",
        "IPMI": "IMI",
        "INJIN": "IMI",
        "PMI": "PMI",
        "INJLA": "AMI",
        "INJIL": "IMI",
        "NORM": "NORM",
        "NDT": "STTC",
        "NST_": "NST_",
        "DIG": "STTC",
        "LNGQT": "STTC",
        "ISC_": "ISC_",
        "ISCAL": "ISCA",
        "ISCIN": "ISCI",
        "ISCIL": "ISCI",
        "ISCAS": "ISCA",
        "ISCLA": "ISCA",
        "ANEUR": "STTC",
        "EL": "STTC",
        "ISCAN": "ISCA",
    }

    # Get diagnostic labels
    codes = {k: v for k, v in eval(x).items() if k in DIAG2SUB.keys()}
    # Discard low likelihoods
    codes = [k for k,v in codes.items() if v >= 100]
    label = None
    if len(codes):
        # Get diagnostic with highest risk
        code = max(codes, key=lambda x: DIAGNOSTICS.index(x))
        # Get superclass of the diagnostic
        label = SUB2SUPER[DIAG2SUB[code]]
    return label


superdiag = ptbxl["scp_codes"].copy()
superdiag = superdiag.apply(get_labels)
print(superdiag.value_counts())
non_empty_indexes = [i for i, x in enumerate(superdiag) if x is not None]
```

```
superdiag = superdiag.iloc[non_empty_indexes]
ecg_ids = ptbxl["ecg_id"].iloc[non_empty_indexes]
superdiag_df = pd.DataFrame({"superclass": superdiag})
superdiag_df.insert(loc=0, column="ecg_id", value=ecg_ids)
print(superdiag_df.shape)


# %% [markdown]
# ## FMM model parameter statistics
# Mean and standard deviation are calculated for each parameter. Note that for alpha and beta the respective circular statistics are calculated.

# %%
vars_dict = {x: ["mean", "std"] for x in amplitude + omega + r2 + dist + m}
circvars_dict = {x: [circmean, circstd] for x in alpha + beta}
fmm_features = fmm.groupby("ecg_id").agg(vars_dict | circvars_dict)
fmm_features.columns = fmm_features.columns.to_flat_index()
fmm_features.columns = ["_".join(col) for col in fmm_features.columns.values]
fmm_features = fmm_features.reset_index()
print(fmm_features.shape)
fmm_features.head()


# %% [markdown]
# ## Rhythm variables
# The mean and standard deviation are calculated for the R–R intervals and the heart rates derived from them.

# %%
RR2BPM = lambda x: 60000 / x
FREQ = 500
SEC2MS = 1000

rhythm_features = pd.DataFrame({"ecg_id": beat_annot_df["ecg_id"].unique()})
beat_annot_df["RR"] = (
    beat_annot_df.groupby("ecg_id")["annoRef"].diff() / FREQ * SEC2MS
)
beat_annot_df["HR"] = RR2BPM(beat_annot_df["RR"])
beat_annot_df_group = beat_annot_df.groupby("ecg_id")
# RR Features
rhythm_features["RR_mean"] = beat_annot_df_group["RR"].mean().values
rhythm_features["RR_std"] = beat_annot_df_group["RR"].std().values
# Hr features
rhythm_features["HR_mean"] = beat_annot_df_group["HR"].mean().values
rhythm_features["HR_std"] = beat_annot_df_group["HR"].std().values
print(rhythm_features.shape)
rhythm_features.head()


# %% [markdown]
# ## Feature storing
# All datasets are merged into a single dataset and a resulting file is stored.

# %%
ptbxl_superclass_df = ptbxl[
    ["ecg_id", "patient_id", "age", "sex", "height", "weight"]
].merge(fmm_features, left_on="ecg_id", right_on="ecg_id")
ptbxl_superclass_df = ptbxl_superclass_df.merge(
```

```
    rhythm_features, left_on="ecg_id", right_on="ecg_id"
)
ptbxl_superclass_df = ptbxl_superclass_df.merge(
    superdiag_df, left_on="ecg_id", right_on="ecg_id"
)

ptbxl_superclass_df.to_csv("./data/ptbxl_superclass.csv", index=False)
print(ptbxl_superclass_df.shape)
ptbxl_superclass_df.head()
```

A.3. Modelado

En esta sección se incluye el Jupyter Notebook convertido a un script de Python que se ha usado para la etapa de modelado.

```
# %% [markdown]
# ## Imports

# %%
from typing import Dict, Iterable, Tuple

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import shap
from lightgbm import LGBMClassifier
from scipy import stats
from sklearn.base import BaseEstimator
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, confusion_matrix, make_scorer,
                             roc_auc_score)
from sklearn.model_selection import RandomizedSearchCV, StratifiedGroupKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from xgboost import XGBClassifier
from yellowbrick.classifier import ROCAUC, ConfusionMatrix
from yellowbrick.contrib.wrapper import wrap
from yellowbrick.style import set_palette

# %% [markdown]
# ## Dataset loading

# %%
ID = "ecg_id"
LABEL = "superclass"
GROUP = "patient_id"
ptbxl_superclass_df = pd.read_csv("./data/ptbxl_superclass.csv")
print(ptbxl_superclass_df.shape)
ptbxl_superclass_df.head()


# %% [markdown]
# ### Predictors
```

```
# Predictor variables, ecg id, superclass label and group number (patient id) are excluded.

# %%
X = ptbxl_superclass_df.loc[
    :,
    ~ptbxl_superclass_df.columns.isin([ID, LABEL, GROUP]),
]
print(X.shape)
X.head()

# %% [markdown]
# ### Labels
# Labels with the ECGs' superclasses.

# %%
y = ptbxl_superclass_df[LABEL]
print(y.shape)
y.head()

# %% [markdown]
# ### Groups
# ECG group number (patient ID).

# %%
groups = ptbxl_superclass_df[GROUP].astype(int)
print(groups.shape)
groups.head()

# %% [markdown]
# ## Train-Test split
# 80% are reserved for training and 20% for test. It is done in a stratified way and by groups.

# %%
def train_test_idx(
    X: pd.DataFrame,
    y: pd.Series,
    groups: pd.Series,
    k: int = 10,
    test_k: int = 2,
    random_state=3533516674,
) -> Tuple[Iterable[int], Iterable[int]]:
    """
    Train-Test split stratified and by groups.

    Args:
        X (pd.DataFrame): DataFrame with predictors.
        y (pd.Series): Series with labels.
        groups (pd.Series): Series with group ids.
        k (int, optional): Number of folds to divide the dataset into.
            Defaults to 10.
        test_k (int, optional): Number of folds for test. Defaults to 2.
        random_state (int, optional): Random seed. Defaults to 3533516674.

    Returns:
        Tuple[Iterable[int], Iterable[int]]: Tuple with the list of train
            indexes and the list of test indexes.
```

```
    """
    X = X.reset_index(drop=True)
    y = y.reset_index(drop=True)
    groups = groups.reset_index(drop=True)
    skf = StratifiedGroupKFold(
        n_splits=k, shuffle=True, random_state=random_state
    )
    test_index = []
    for i, (_, test_fold_idx) in enumerate(skf.split(X=X, y=y, groups=groups)):
        if i < test_k:
            test_index.extend(test_fold_idx)
    train_index = list(set(X.index) - set(test_index))
    return train_index, test_index


# %%
train_index, test_index = train_test_idx(X, y, groups)
X_train = X.iloc[train_index, :]
X_test = X.iloc[test_index, :]
y_train = y.iloc[train_index]
y_test = y.iloc[test_index]
groups_train = groups.iloc[train_index]
groups_test = groups.iloc[test_index]
print(f"{X_train.shape=}")
print(f"{X_test.shape=}")


# %% [markdown]
# ## Models

# %%
def hyperparameter_tuning(
    estimator: BaseEstimator,
    hyperparams: Dict[str, list],
    X: pd.DataFrame,
    y: pd.Series,
    groups: pd.Series,
    k: int = 10,
    n_iter: int = 1,
    random_state: int = 1255869813,
    n_jobs: int = 10,
) -> RandomizedSearchCV:
    """
    Performs hyperparameter tuning by randomized search. The combination of
    hyperparameters chosen will be the one that maximizes the average macro AUC
    estimated by K-Fold cross-validation.

    Args:
        estimator (BaseEstimator): Estimator to tune.
        hyperparams (Dict[str, list]): Hyperparameter dictionary.
        X (pd.DataFrame): DataFrame with predictors.
        y (pd.Series): Series with labels.
        groups (pd.Series): Series with group ids.
        k (int, optional): Number of folds. Defaults to 10.
        n_iter (int, optional): Number of randomized search iterations. Defaults to
    1.
        random_state (int, optional): Random state. Defaults to 1255869813.
        n_jobs (int, optional): Number of threads. Defaults to 10.
```

```
155
156     Returns:
157         RandomizedSearchCV: Results of the randomized search.
158     """
159     skf = StratifiedGroupKFold(
160         n_splits=k, shuffle=True, random_state=random_state
161     )
162     cv = skf.split(X=X, y=y, groups=groups)
163
164     scorer = make_scorer(
165         roc_auc_score, average="macro", multi_class="ovr", needs_proba=True
166     )
167     search = RandomizedSearchCV(
168         estimator=estimator,
169         param_distributions=hyperparams,
170         cv=cv,
171         scoring=scorer,
172         refit=True,
173         n_iter=n_iter,
174         n_jobs=n_jobs,
175         verbose=False,
176         random_state=random_state,
177     )
178     search.fit(X, y)
179     score_mean = search.cv_results_["mean_test_score"][search.best_index_]
180     score_std = search.cv_results_["std_test_score"][search.best_index_]
181     QUANTILE = stats.t(df=k - 1).ppf(0.975)
182     score_error = QUANTILE * score_std / np.sqrt(k)
183     print(f"Best params = {search.best_params_}")
184     print(f"AUC = {score_mean:.3f} +- {score_error:.3f}")
185     return search
186
187
188 # %%
189 def sensitivity(cm: Iterable[Iterable[int]]) -> float:
190     """
191     Compute the macro sensitivity.
192
193     Args:
194         cm (Iterable[Iterable[int]]): Confusion matrix.
195
196     Returns:
197         float: Macro sensitivity.
198     """
199     n_clases = cm.shape[0]
200     s = np.empty(n_clases)
201     for c in range(n_clases):
202         cdf = pd.DataFrame(cm)
203         tp = cdf.iloc[c, c]
204         fn = cdf.iloc[c, ~cdf.columns.isin([c])].values.sum()
205         s[c] = tp / (tp + fn)
206     return s.mean()
207
208
209 def specificity(cm: Iterable[Iterable[int]]) -> float:
210     """
211     Compute the macro specificity.
212
```

```
    Args:
        cm (Iterable[Iterable[int]]): Confusion matrix.

    Returns:
        float: Macro specificity.
    """
    n_clases = cm.shape[0]
    s = np.empty(n_clases)
    for c in range(n_clases):
        cdf = pd.DataFrame(cm)
        tn = cdf.iloc[
            ~cdf.index.isin([c]), ~cdf.columns.isin([c])
        ].values.sum()
        fp = cdf.iloc[~cdf.index.isin([c]), c].values.sum()
        s[c] = tn / (tn + fp)
    return s.mean()


def compute_metrics(
    estimator: BaseEstimator, X_test: pd.DataFrame, y_test: pd.Series
) -> dict:
    """
    Compute macro AUC, balanced accuracy, sensitivity, specificity and accuracy.

    Args:
        estimator (BaseEstimator): Fitted estimator on which to compute the metrics.
        X_test (pd.DataFrame): DataFrame with predictors.
        y_test (pd.Series): Series with labels.

    Returns:
        dict: Dictionary with metric values.
    """
    y_pred = estimator.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    acc = accuracy_score(y_test, y_pred)
    se = sensitivity(cm)
    sp = specificity(cm)
    bacc = (se + sp) / 2
    proba = estimator.predict_proba(X_test)
    auc = roc_auc_score(y_test, proba, average="macro", multi_class="ovr")
    print(f"AUC = {auc:.3f}")
    print(f"BACC = {bacc:.3f}")
    print(f"SE = {se:.3f}")
    print(f"SP = {sp:.3f}")
    print(f"ACC = {acc:.3f}")
    metrics = {"auc": auc, "bacc": bacc, "se": se, "sp": sp, "acc": acc}
    return metrics


# %%
def plot_cm_ROC(
    estimator: BaseEstimator,
    X_train: pd.DataFrame,
    y_train: pd.Series,
    X_test: pd.DataFrame,
    y_test: pd.Series,
) -> None:
```

```
    """
    Plot the ROC curves for each class.

    Args:
        estimator (BaseEstimator): Fitted estimator on which to compute the ROC curves.
        X_train (pd.DataFrame): Training DataFrame with predictors.
        y_train (pd.Series): Training Series with labels.
        X_test (pd.DataFrame): Test DataFrame with predictors.
        y_test (pd.Series): Test Series with labels.
    """
    PALETTE = {
        "CD": "#1583d1",
        "HYP": "#de5f04",
        "MI": "#d60b0b",
        "NORM": "#2e9c19",
        "STTC": "#720ec4",
    }

    CLASSES = list(PALETTE.keys())
    set_palette(list(PALETTE.values()))
    # Visualizer of ROC curves
    visualizer = ROCAUC(
        wrap(estimator),
        classes=CLASSES,
        micro=False,
        macro=False,
        title=" ",
        fmt="%.4f",
    )
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    plt.xlabel("", fontsize=18)
    plt.ylabel("", fontsize=18)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    visualizer.show()
    # Visualizer of confussion matrix
    visualizer = ConfusionMatrix(
        wrap(estimator), classes=CLASSES, cmap="Blues", title=" "
    )
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    plt.xlabel("", fontsize=18)
    plt.ylabel("", fontsize=18)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    visualizer.show()


def plot_SHAP(
    estimator: BaseEstimator,
    X: pd.DataFrame,
    method: str = "kernel",
    preprocess: bool = True,
    max_display: int = 10,
) -> None:
    """
```

```
    Plot the summary of the most important features based on SHAP method.

    Args:
        estimator (BaseEstimator): Fitted estimator on which to compute the SHAP values.
        X (pd.DataFrame): DataFrame on which to calculate the SHAP values.
        method (str, optional): Type of SHAP method, could be linear or tree.
            Defaults to "kernel".
        preprocess (bool, optional): If true preprocess the DataFrame based on estimator's
            transform steps. Defaults to True.
        max_display (int, optional): Maximum number of variables to display. Defaults to 10.
    """
    PALETTE = {
        "CD": "#1583d1",
        "HYP": "#de5f04",
        "MI": "#d60b0b",
        "NORM": "#2e9c19",
        "STTC": "#720ec4",
    }

    X_t = X.copy()
    if preprocess:
        for step in estimator[:-1]:
            X_t.iloc[:, :] = step.transform(X_t)
        estimator = estimator[-1]
    explainer = None

    if method == "linear":
        explainer = shap.LinearExplainer(estimator, X_t)
        shap_values = explainer.shap_values(X_t)
    elif method == "tree":
        explainer = shap.TreeExplainer(estimator)
        shap_values = explainer.shap_values(X_t, check_additivity=False)
    else:
        explainer = shap.KernelExplainer(estimator.predict_proba, X_t)
        shap_values = explainer.shap_values(X_t, check_additivity=False)
    shap.summary_plot(
        shap_values,
        X_t.values,
        class_names=list(PALETTE.keys()),
        class_inds="original",
        color=lambda i: list(PALETTE.values())[i],
        feature_names=X_t.columns,
        max_display=max_display,
        show=False,
    )
    plt.xlabel("mean(|SHAP value|)")
    plt.show()


# %% [markdown]
# ### Logistic Regression

# %%
# Tuning
estimator = Pipeline(
```

```
    steps=[
        ("scaler", StandardScaler()),
        ("imputer", SimpleImputer(strategy="median")),
        (
            "model",
            LogisticRegression(
                max_iter=10000,
                random_state=3218926,
            ),
        ),
    ]
)
hyperparams = {
    "model__penalty": ["l1"],
    "model__solver": ["saga"],
    "model__C": [0.001, 0.01, 0.1, 1, 2, 5, 10, 50, 100, 1000],
}
search = hyperparameter_tuning(
    estimator, hyperparams, X_train, y_train, groups_train, n_iter=10
)
lr_model = search.best_estimator_


# %%
# Test
compute_metrics(lr_model, X_test, y_test)
plot_cm_ROC(lr_model, X_train, y_train, X_test, y_test)

# %%
# SHAP
plot_SHAP(lr_model, X_test, method="linear")

# %% [markdown]
# ### SVM

# %%
# Tuning
estimator = Pipeline(
    steps=[
        ("scaler", StandardScaler()),
        ("imputer", SimpleImputer(strategy="median")),
        ("model", SVC(probability=True)),
    ]
)

hyperparams = {
    "model__kernel": ["linear", "poly", "rbf"],
    "model__C": [0.001, 0.01, 0.1, 1, 2, 5, 10, 50],
    "model__gamma": [0.001, 0.01, 0.1, 1, 2, 5, 10, 50, "scale", "auto"],
}
search = hyperparameter_tuning(
    estimator, hyperparams, X_train, y_train, groups_train, n_iter=100
)
svm_model = search.best_estimator_


# %%
# Test
```

```
compute_metrics(svm_model, X_test, y_test)
plot_cm_ROC(svm_model, X_train, y_train, X_test, y_test)

# %%
# SHAP
plot_SHAP(svm_model, X_test)


# %% [markdown]
# ### Random Forest

# %%
# Tuning
estimator = Pipeline(
    steps=[
        ("scaler", StandardScaler()),
        ("imputer", SimpleImputer(strategy="median")),
        ("model", RandomForestClassifier(random_state=98741651)),
    ]
)
hyperparams = {
    "model__n_estimators": [100, 150, 200, 250, 300, 350, 400],
    "model__max_depth": [3, 5, 7, 11, 15, None],
    "model__max_leaf_nodes": [1, 3, 5, 7, 11, 15, None],
    "model__max_features": [0.6, 0.8, 0.9, "sqrt", "log2", None],
    "model__min_samples_leaf": [1, 2, 5, 7, 11, 15],
}
search = hyperparameter_tuning(
    estimator, hyperparams, X_train, y_train, groups_train, n_iter=100
)
rf_model = search.best_estimator_

# %%
# Test
compute_metrics(rf_model, X_test, y_test)
plot_cm_ROC(rf_model, X_train, y_train, X_test, y_test)

# %%
# SHAP
plot_SHAP(rf_model, X_test, method="tree")

# %% [markdown]
# ### XGBoost

# %%
# Tuning
# XGBoost needs target encoded
encoder = LabelEncoder()
y_train_encoded = encoder.fit_transform(y_train)
y_test_encoded = encoder.transform(y_test)
estimator = Pipeline(
    steps=[
        ("model", XGBClassifier(random_state=231986132)),
    ]
)

hyperparams = {
    "model__n_estimators": [100, 125, 150, 175, 200, 225, 250],
```

```
    "model__learning_rate": [0.001, 0.01, 0.05, 0.075, 0.1, 0.15, 0.2],
    "model__colsample_bytree": [0.7, 0.8, 0.9, 1.0],
    "model__max_depth": [-1, 5, 10, 15, 20, 25, 30, None],
    "model__min_child_weight": [1, 5, 7, 11, 15],
    "model__reg_alpha": [0, 0.01, 1, 2, 5, 10, 50],
    "model__reg_lambda": [0, 0.01, 1, 2, 5, 10, 50],
}
search = hyperparameter_tuning(
    estimator, hyperparams, X_train, y_train_encoded, groups_train, n_iter=100
)
xgb_model = search.best_estimator_


# %%
# Test
compute_metrics(xgb_model, X_test, y_test_encoded)
plot_cm_ROC(xgb_model, X_train, y_train_encoded, X_test, y_test_encoded)

# %%
# SHAP
plot_SHAP(xgb_model["model"], X_test, preprocess=False, method="tree")

# %% [markdown]
# ### LightGBM

# %%
# Tuning
estimator = Pipeline(
    steps=[
        ("model", LGBMClassifier(n_estimators=250, random_state=79891)),
    ]
)

hyperparams = {
    "model__n_estimators": [100, 125, 150, 175, 200, 225, 250],
    "model__learning_rate": [0.001, 0.01, 0.05, 0.075, 0.1, 0.15, 0.2],
    "model__colsample_bytree": [0.7, 0.8, 0.9, 1.0],
    "model__max_depth": [-1, 5, 10, 15, 20, 25, 30, None],
    "model__min_child_samples": [20, 22, 24, 26, 28, 30, 32, 34],
    "model__reg_alpha": [0, 0.01, 1, 2, 5, 10, 50],
    "model__reg_lambda": [0, 0.01, 1, 2, 5, 10, 50],
}

search = hyperparameter_tuning(
    estimator, hyperparams, X_train, y_train, groups_train, n_iter=100
)
lgbm_model = search.best_estimator_


# %%
# Test
compute_metrics(lgbm_model, X_test, y_test)
plot_cm_ROC(lgbm_model, X_train, y_train, X_test, y_test)

# %%
# SHAP
plot_SHAP(lgbm_model["model"], X_test, preprocess=False, method="tree")

```

```
# %% [markdown]
# ### Final model

# %%
best_model = lgbm_model["model"]
explainer = shap.TreeExplainer(best_model)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[0], X_test.values, feature_names = X_test.columns,
    plot_type="violin", max_display=10)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns,
    plot_type="violin", max_display=10)
shap.summary_plot(shap_values[2], X_test.values, feature_names = X_test.columns,
    plot_type="violin", max_display=10)
shap.summary_plot(shap_values[3], X_test.values, feature_names = X_test.columns,
    plot_type="violin", max_display=10)
shap.summary_plot(shap_values[4], X_test.values, feature_names = X_test.columns,
    plot_type="violin", max_display=10)
```