

Resumen

Este proyecto es una solución informática basada en una arquitectura *cloud* y la inteligencia artificial (IA), con la que se puede, mediante algoritmos de Machine Learning tales como random forest, predecir el rango de días que quedan para que un paciente reciba el alta médica a partir de sus datos médicos. La aplicación también hace uso de la computación *serverless*. Esta solución también permitirá predecir y clasificar el estado del paciente, con lo que el personal sanitario podrá tomar las decisiones oportunas para que este tenga la mejor recuperación posible.

Para que todo esto sea posible, se lleva a cabo una investigación sobre las distintas tecnologías que fundamentan el proyecto, haciendo especial hincapié en aquellos relacionados con la creación, evaluación e interpretación de los modelos de Machine Learning. El análisis y desarrollo de la arquitectura del sistema es una parte fundamental del trabajo, comprendiendo esta la descripción del sistema, modelado del mismo y su implementación con los servicios que Amazon Web Services ofrece. También se desarrollan dos aplicaciones con sendas interfaces gráficas que sirven al personal sanitario para hacer uso de los distintos recursos del sistema.

Finalmente, se llevan a cabo pruebas con usuarios, con especial interés en que estas sean realizadas por el propio personal sanitario.

Abstract

This project is a computing solution based on cloud architecture and Artificial Intelligence (AI), which can, using Machine Learning algorithms such as random forest, predict the range of days remaining until a patient is discharged from hospital based on their medical data. The application also makes use of serverless computing. This solution will also predict and classify the patient's condition, allowing healthcare staff to make the right decisions to ensure the patient has the best possible recovery.

To make all this possible, research is being carried out on the different technologies that underpin the project, with special emphasis on those related to the creation, evaluation and interpretation of Machine Learning models. The analysis and development of the system architecture is a fundamental part of the work, comprising the description of the system, its modelling and its implementation with the services offered by Amazon Web Services. Two applications are also developed, each with its own graphical interface, which are used by the healthcare personnel to make use of the different resources of the system.

Finally, tests are carried out with users, with special interest in the fact that these tests are carried out by the healthcare personnel themselves.

Descriptores

SARS-CoV-2, inteligencia artificial, arquitectura *cloud*, random forest, *serverless*.

Índice

1.	<i>Introducción</i>	1
2.	<i>Fundamentos teóricos</i>	3
2.1	Regresión lineal	3
2.1.1	Regresión lineal simple	3
2.1.2	Coeficiente de las ecuaciones normales	4
2.1.3	Optimización mediante el descenso del gradiente	4
2.1.4	Análisis de los residuos	5
2.2	Otros algoritmos de Machine Learning	5
2.2.1	Random forest	6
2.2.2	AdaBoost	9
2.3	Métricas para la evaluación de modelos	11
2.3.1	La división entrenamiento/prueba/validación	11
2.3.2	Métricas de clasificación	12
2.3.3	Métricas de regresión	15
2.3.4	La curva AUC-ROC	16
2.3.5	Sesgo contra varianza	16
2.3.6	Otras buenas prácticas	18
2.4	Interpretación del modelo	19
2.4.1	Funcionamiento de los valores SHAP	19
2.4.2	Cálculo de distribuciones marginales	19
2.5	Cómo manejar los datos que faltan	22
2.5.1	Imputación frente a eliminación de los datos	22
2.5.2	Borrado	22
2.5.3	Métodos específicos de series temporales	23
2.5.4	Media, mediana y moda	23
2.5.5	Regresión lineal	24
2.5.6	Imputación múltiple	24
2.5.7	KNN (K-Nearest Neighbors)	25
2.5.8	Cuál emplear	25
3.	<i>Especificación de requisitos</i>	27
3.1	Descripción del sistema	27
3.2	Modelado	27
3.2.1	Casos de uso	27
3.2.2	Base de datos	28
3.2.3	Diagramas de actividades	29
3.2.4	Diagramas de secuencias	31
3.3	Modelo de arquitectura	32
3.3.1	Un modelo de escritura, N modelos de lectura	33
3.3.2	Escalabilidad	33
3.3.3	Requisitos del negocio	33
3.3.4	Concurrencia	33
3.3.5	No hay semántica de «lee tus propios escritos»	33
3.3.6	Comandos/Escrituras: efectos secundarios	34
3.3.7	Flujo	34
3.3.8	Ventajas del modelo de arquitectura CQRS	35

3.3.9	Desventajas del modelo de arquitectura CQRS.....	35
3.4	Implementación de la arquitectura del sistema en AWS.....	35
3.4.1	Qué es AWS (Amazon Web Services)	35
3.4.2	Funcionamiento de AWS	35
3.4.3	Servicios de AWS empleados	35
3.4.4	Ventajas de AWS	37
3.4.5	Desventajas de AWS	37
3.4.6	Implementación del patrón CQRS en AWS	37
3.4.7	Despliegue de la arquitectura del entorno local a AWS	39
3.5	Metodologías ágiles	39
3.5.1	Temas	39
3.5.2	Product Backlog.....	40
3.5.3	Sprints Backlog.....	41
3.5.4	En Progreso.....	41
3.5.5	Revisando.....	41
3.5.6	Sprints finalizados.....	41
3.5.7	Procedimiento.....	42
4.	Desarrollo	43
4.1	Implementación del modelo de Machine Learning	43
4.1.1	Preparación para el modelado.....	43
4.1.2	Cómo hacer frente al desequilibrio de las clases	47
4.1.3	Modelado de Machine Learning.....	47
4.1.4	Interpretabilidad del modelo	50
4.1.5	Implementación de la pipeline.....	51
4.2	Implementación del sistema en AWS	52
4.2.1	Implementación del API Gateway	52
4.2.2	Implementación de las funciones Lambda.....	53
4.2.3	Implementación del SNS.....	56
4.2.4	Implementación de la cola SQS.....	56
4.2.5	Implementación de las base de datos DynamoDB	57
4.3	Desarrollo de las aplicaciones Electron	58
5.	Resultados obtenidos y conclusiones.....	65
5.1	Resultados.....	65
5.2	Conclusiones	65
5.2.1	Conclusiones generales	65
5.2.2	Los riesgos éticos que se esconden tras los algoritmos.....	66
6.	Futuras líneas de investigación.....	69
6.1	Entorno de clúster de Spark	69
6.2	Trabajo con series temporales	69
6.3	Inclusión de un conjunto de validación	70
6.4	Investigación sobre técnicas empleadas para reducir el sobreajuste en modelos con pocos datos.....	70
6.5	Inclusión de nuevos casos de uso	70
6.6	Sustitución de DynamoDB por Aurora para la base de datos de estados de los pacientes.....	71

Índice de Figuras

Figura 2-1 Representación gráfica del reparto de datos para entrenar cada árbol individual	7
Figura 2-2 Representación gráfica del proceso de entrenamiento del algoritmo random forest	8
Figura 2-3 Representación gráfica de casos de uso del algoritmo random forest	9
Figura 2-4 Gráfico que muestra la relación de at con la tasa de error. Fuente: McCormick, C. (2013)	11
Figura 2-5 Matriz de confusión. Fuente: Jordan, J. (2017)	12
Figura 2-6 Matriz de confusión sin normalizar. Fuente: Jordan, J. (2017)	13
Figura 2-7 Estados de los elementos predichos. Fuente: Jordan, J. (2017)	14
Figura 2-8 Infraajuste. Fuente: Scikit-learn (s.f.)	17
Figura 2-9 Sobreajuste. Fuente: Scikit-learn (s.f.)	17
Figura 2-10 Comparación de los tipos de generalización de un modelo. Fuente: Scikit-learn (s.f.)	18
Figura 2-11 Inspección visual de los datos. Fuente: Scikit-learn (s.f.)	18
Figura 2-12 Gráfico de las características de un modelo de ejemplo	19
Figura 2-13 Primer paso en el proceso del cálculo de los valores SHAP	20
Figura 2-14 Segundo paso en el proceso del cálculo de los valores SHAP	20
Figura 2-15 Tercer paso en el proceso del cálculo de los valores SHAP	21
Figura 2-16 Ejemplo del método de interpolación lineal. Fuente: variable «tsAirgap» de la librería «imputeTS» con los datos interpolados en rojo	23
Figura 2-17 Esquema del funcionamiento de la imputación múltiple	24
Figura 3-1 Casos de uso del subsistema dashboard	28
Figura 3-2 Casos de uso del subsistema alta de pacientes	28
Figura 3-3 Base de datos de Health Twin	29
Figura 3-4 Diagrama de actividades para añadir un nuevo paciente al sistema	29
Figura 3-5 Diagrama de actividades para añadir un nuevo paciente manualmente	30
Figura 3-6 Diagrama de actividades para añadir varios pacientes por medio de un fichero CSV	30
Figura 3-7 Diagrama de secuencias de análisis para añadir un paciente manualmente	31
Figura 3-8 Diagrama de secuencias de análisis para añadir varios pacientes por medio de un fichero CSV	31
Figura 3-9 Diagrama de secuencias de análisis para obtener el listado del estado de los pacientes	32
Figura 3-10 Representación del patrón CQRS en Health Twin	32
Figura 3-11 Diagrama que describe el sistema que implementa un patrón CQRS en AWS en Health Twin	38
Figura 3-12 Tarjeta de Trello que contiene los temas del desarrollo de Health Twin	40
Figura 3-13 Tarjeta de Trello que contiene los distintos ítems a completar en el desarrollo de Health Twin	40
Figura 3-14 Información de uno de los ítems a completar en el desarrollo de Health Twin	41
Figura 4-1 Código que describe la imputación múltiple para solucionar el problema de los valores NA	43
Figura 4-2 Código que describe cómo obtener la fecha de alta médica de los pacientes	44
Figura 4-3 Código que describe cómo obtener los días de estancia en UCI de los pacientes	44
Figura 4-4 Código que describe la transformación del atributo edad en etiquetas	44

Figura 4-5 Histograma de la distribución de la variable que representa los días hasta el alta	45
Figura 4-6 Código que describe la creación de los rangos de días de estancia en UCI de los pacientes	46
Figura 4-7 Histograma de la distribución de los rangos de días de alta	46
Figura 4-8 Código que describe cómo solucionar el desequilibrio de las clases	47
Figura 4-9 Gráfico que representa el porcentaje de varianza explicada frente al número de componentes	48
Figura 4-10 Curvas ROC para random forest con sus respectivos valores AUC para las distintas clases	49
Figura 4-11 Curvas ROC para AdaBoost con sus respectivos valores AUC para las distintas clases	49
Figura 4-12 Representación gráfica del impacto medio de las distintas variables en el resultado del modelo	50
Figura 4-13 Código de la función que describe el funcionamiento de un transformer personalizado	51
Figura 4-14 Código que describe la creación de la pipeline	52
Figura 4-15 Ejemplo que describe el formato de la documentación de la API de Health Twin creada	52
Figura 4-16 Anotaciones YAML que describen la configuración de las funciones Lambda ...	53
Figura 4-17 Código de la función que crea el evento con la información del paciente	53
Figura 4-18 Código de la función que emite el evento a SNS con la información del paciente	54
Figura 4-19 Código que describe cómo SNS recibe información y la convierte en un evento	54
Figura 4-20 Código de la función que guarda el evento del paciente en la base de datos de eventos	55
Figura 4-21 Código de la función que guarda el estado del paciente en la base de datos de estados	55
Figura 4-22 Anotaciones YAML que describen cómo otorgar permisos de invocación a una función Lambda	56
Figura 4-23 Código de la función que permite consultar el estado de los pacientes	56
Figura 4-24 Código de la función que permite consultar el estado de un paciente en concreto	56
Figura 4-25 Anotaciones YAML que describen la creación del servicio SNS	56
Figura 4-26 Anotaciones YAML que describen cómo crear una cola SQS y suscribirla al servicio SNS	57
Figura 4-27 Anotaciones YAML que describen cómo otorgar permisos a la conexión entre SNS y SQS	57
Figura 4-28 Anotaciones YAML que describen cómo configurar las base de datos DynamoDB de eventos	58
Figura 4-29 Anotaciones YAML que describen cómo configurar las base de datos DynamoDB de estados	58
Figura 4-30 Pantalla de inicio de sesión de la aplicación Health Twin Admission	59
Figura 4-31 Pantalla de selección del método de introducción de los datos de los pacientes	60
Figura 4-32 Pantalla de la primera parte del formulario de introducción de datos del paciente	60
Figura 4-33 Pantalla de la segunda parte del formulario de introducción de datos del paciente	61

Figura 4-34 Pantalla de introducción de datos de los pacientes a través de un fichero CSV	61
Figura 4-35 Pantalla de inicio de sesión de la aplicación Health Twin	62
Figura 4-36 Pantalla en la que se visualiza el listado de todos los pacientes	62
Figura 4-37 Pantalla en la que se visualiza la información relativa a un paciente en concreto	63
Figura 4-38 Código de la petición POST que permite introducir nuevos pacientes en las bases de datos	63
Figura 4-39 Código de la petición GET que permite recuperar los datos del estado de los pacientes	64
Figura 4-40 Código de la petición GET que permite recuperar los datos del estado de un paciente en concreto	64

1. Introducción

Si hay un tema que ha eclipsado por completo el panorama nacional e internacional durante los años 2020 y 2021 es sin duda la pandemia provocada por el virus SARS-COV-2, más comúnmente conocido como COVID-19 o coronavirus.

Es sabido por todo el mundo que el personal sanitario ha llevado a cabo una labor titánica a la hora de tratar a pacientes durante todos estos meses. Muchas noches sin dormir, estrés acumulado y sobre todo muchísimo trabajo. Es por ello que muchas veces sus estimaciones realizadas sobre el devenir de los pacientes no son acertadas debido a la falta de tiempo y descanso. Debido a esto, los pacientes no pueden recibir el trato que el personal sanitario querría darles debido a la ingente cantidad de casos de coronavirus activos en todo el mundo.

Es por ello que se decide poner en marcha el proyecto Health Twin, una aplicación que ayuda al personal sanitario en su día a día lidiando con el COVID-19, permitiendo liberarles de una gran parte de su trabajo actual, con lo que pueden dedicar tiempo a lo que realmente quieren y es importante: los pacientes. Un trato más personalizado y cercano es posible gracias a esta solución.

Health Twin es una solución informática basada en una arquitectura *cloud* y *serverless* y la inteligencia artificial (IA), con lo que se logra, mediante algoritmos de Machine Learning como random forest, predecir rango de días que quedan para que un paciente reciba el alta médica a partir de sus datos médicos.

Esta solución también permite predecir el estado del paciente, con lo que el personal sanitario puede tomar las decisiones oportunas para que este tenga la mejor recuperación posible.

En definitiva, con este proyecto se digitaliza la gestión hospitalaria, en este caso, relativa a la COVID-19.

Toda esta información le será mostrada al personal sanitario mediante una interfaz gráfica a través de la aplicación. Con ella pueden:

- Consultar el listado con los pacientes, su estado y los días hasta su alta médica predichos gracias a nuestro modelo de regresión lineal y modelos de Machine Learning.
- Acceder a la información de un paciente en concreto, donde se mostrará toda la información relativa a su hospitalización de forma mucho más detallada.

Además, existe una segunda aplicación con la que el personal sanitario puede guardar el estado de los pacientes cuando entran en UCI, con lo que Health Twin realiza la predicción del rango de días que el paciente necesita hasta su alta médica y guarda toda esta información en la base de datos para que pueda ser consultada posteriormente.

Como se puede observar, el sistema no se limita a recibir información, sino que se lleva a cabo una comunicación proactiva entre los dos extremos del canal de comunicación.

Health Twin tiene una enorme escalabilidad y posibilidad de adaptarse a muchísimos entornos, sin embargo, en un comienzo se centrará en los pacientes en UCI por SARS-CoV-2.

Los datos para entrenar los modelos de inteligencia artificial provienen de bases de datos médicas *open source*. Estos datos son (en un principio) relativos a los pacientes en UCI por SARS-CoV-2.

Esta aplicación únicamente recibe los datos médicos de los pacientes, es decir, que no se tiene acceso a los datos personales del paciente, pues llegará exclusivamente un identificador del paciente que solo será conocido por aquellas personas trabajando con él. En resumen, la trazabilidad del paciente es imposible.

2. Fundamentos teóricos

En este capítulo se llevará a cabo un análisis en profundidad sobre todos los fundamentos en los que se basa la aplicación, haciendo especial hincapié en aquellos relacionados con la creación del modelo de Machine Learning, pues es el principal signo distintivo de la aplicación.

2.1 Regresión lineal

La regresión lineal se utiliza para encontrar la relación lineal entre el objetivo y uno o más predictores. Hay dos tipos de regresión lineal: simple y múltiple. En este trabajo no se contempla la regresión lineal múltiple, debido a que no es empleada.

2.1.1 Regresión lineal simple

La regresión lineal simple es útil para encontrar la relación entre dos variables continuas. Una es el predictor o la variable independiente y la otra es la respuesta o la variable dependiente. Busca una relación estadística, pero no una relación determinista. Se dice que la relación entre dos variables es determinista si una variable puede ser expresada con precisión por la otra. Por ejemplo, utilizando la temperatura en grados Celsius es posible predecir con exactitud los grados Fahrenheit. La relación estadística no es precisa para determinar la relación entre dos variables. Por ejemplo, la relación entre la altura y el peso.

La idea central es obtener una línea que se ajuste mejor a los datos. La línea que mejor se ajusta es aquella cuyo error de predicción total (todos los puntos de datos) es lo más pequeño posible. El error es la distancia entre el punto y la línea de regresión.

Ejemplo en tiempo real

Se tiene un conjunto de datos que contiene información sobre la relación entre «número de horas estudiadas» y «notas obtenidas». Se ha observado a muchos estudiantes y se han registrado sus horas de estudio y su nota. Estos serán los datos de entrenamiento. El objetivo es diseñar un modelo que pueda predecir las notas si se da el número de horas estudiadas. Utilizando los datos de entrenamiento, se obtiene una recta de regresión que da un error mínimo. Esta ecuación lineal se utiliza entonces para cualquier dato nuevo. Es decir, si se da como entrada el número de horas estudiadas por un alumno, el modelo debe predecir su nota con un error mínimo.

$$Y(pred) = b_0 + b_1 * x$$

Los valores b_0 y b_1 deben elegirse de forma que minimicen el error. Si se toma la suma del error al cuadrado como métrica para evaluar el modelo (Montgomery, D.C., y Peck, E.A., 2012), el objetivo es obtener la línea que mejor reduzca el error.

$$Error = \sum_{i=1}^n (salida_real - salida_predicha)^2$$

Si no se eleva el error al cuadrado, los puntos positivos y negativos se anulan entre sí.

Para el modelo con un predictor (Montgomery, D.C., y Peck, E.A., 2012):

$$b_0 = \bar{y} - b_1 \bar{x}$$

$$b_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Si $b_1 > 0$, entonces x (predictor) e y (objetivo) tienen una relación positiva. Es decir, el aumento de x hará que aumente y .

Si $b_1 < 0$, entonces x (predictor) e y (objetivo) tienen una relación negativa. Es decir, el aumento de x hará que disminuya y .

Si el modelo no incluye $x = 0$, la predicción carecerá de sentido con solo b_0 . Por ejemplo, se tiene un conjunto de datos que relaciona la altura (x) y el peso (y). Si se toma $x = 0$ (es decir, la altura como 0), la ecuación solo tendrá el valor b_0 , que carece por completo de sentido, debido a que en tiempo real la altura y el peso nunca pueden ser cero. Esto se debe a la consideración de los valores del modelo más allá de su alcance.

Si el modelo incluye el valor 0, entonces b_0 será la media de todos los valores predichos cuando $x = 0$. Sin embargo, poner a cero todas las variables predictoras es a menudo imposible.

El valor de b_0 garantiza que los residuos tengan una media cero. Si no existe el término b_0 , la regresión se ve obligada a pasar por el origen. En ese caso, tanto el coeficiente de regresión como la predicción están sesgados.

2.1.2 Coeficiente de las ecuaciones normales

A parte de la ecuación anterior, también se puede calcular el coeficiente del modelo a partir de la ecuación normal (Montgomery, D.C., y Peck, E.A., 2012).

$$\theta = (X^T X)^{-1} X^T Y$$

Theta contiene el coeficiente de todos los predictores, incluido el término constante b_0 . La ecuación normal realiza el cálculo tomando la inversa de la matriz de entrada. La complejidad del cálculo aumenta a medida que aumenta el número de características. Se vuelve muy lento cuando el número de características aumenta.

2.1.3 Optimización mediante el descenso del gradiente

La complejidad de la ecuación normal hace que sea difícil de utilizar, y es aquí donde entra en escena el método de descenso de gradiente. La derivada parcial de la función de coste con respecto al parámetro puede dar el valor óptimo del coeficiente.

2.1.4 Análisis de los residuos

La aleatoriedad y la imprevisibilidad son los dos componentes principales de un modelo de regresión.

$$\text{Predicción} = \text{Determinista} + \text{Estadístico}$$

La parte determinista está cubierta por la variable predictora del modelo. La parte estocástica revela el hecho de que el valor esperado y el observado son imprevisibles. Siempre habrá alguna información que no se cubra. Esta información puede obtenerse a partir de la información residual.

El concepto de residuo puede ser explicado a través de un ejemplo. Considérese que se tiene un conjunto de datos que predice las ventas de zumo cuando se da una temperatura en un lugar. El valor predicho por la ecuación de regresión siempre tiene alguna diferencia con el valor real. Las ventas no coinciden exactamente con el verdadero valor de salida. Esta diferencia se denomina residuo (Montgomery, D.C., y Peck, E.A., 2012).

El gráfico de residuos ayuda a analizar el modelo utilizando los valores de los residuos. Se traza entre los valores predichos y el residuo. Sus valores están estandarizados. La distancia del punto a 0 especifica lo mala que es la predicción para ese valor. Si el valor es positivo, entonces la predicción es baja. Si el valor es negativo, entonces la predicción es alta. El valor 0 indica una predicción perfecta. La detección de un patrón residual puede mejorar el modelo.

Como se observa en el capítulo del libro de Montgomery, D.C., y Peck, E.A. (2012) dedicado a la regresión lineal, el patrón no aleatorio del gráfico de residuos indica que el modelo está:

- Falto de una variable que tiene una contribución significativa al objetivo del modelo.
- Falto de capturar la no linealidad (utilizando el término polinómico).
- Sin interacción entre los términos del modelo.

En este mismo capítulo también se pueden observar las características de un residuo:

- Los residuos no presentan ningún patrón.
- Los residuos adyacentes no deben ser iguales, debido a que indican que hay alguna información perdida por el sistema.

2.2 Otros algoritmos de Machine Learning

Pese a que la regresión lineal sea el algoritmo que pueda parecer tener más en común con la solución buscada, nunca es suficiente con probar un único algoritmo.

Además, durante la realización del modelo de Machine Learning se cambió el planteamiento del problema, debido a que, ¿por qué darle al médico los días exactos de estancia? De esta forma se pierde información que podría ser relevante. En este caso es más conveniente darle un margen de días, es decir, ¿qué probabilidad tiene un paciente de salir de la UCI en un rango de 3-5 días?

De esta forma el problema que antes era lineal, ha sido transformado en dicotómico (obtiene el alta de la UCI en 3-5 días = Sí/No).

Intentar emplear una regresión logística para estimar el modelo más sencillo es una aproximación que es necesaria hacer para poder llegar a un entendimiento real del problema y sus posibles soluciones. Sin embargo, no es un algoritmo eficiente para encontrar la solución a este problema.

A continuación se explican brevemente los algoritmos random forest y AdaBoost, debido a que son los que muestran un mejor desempeño para resolver el problema planteado.

2.2.1 Random forest

En el mundo del Machine Learning, los modelos random forest son un tipo de modelos no paramétricos que pueden utilizarse tanto para la regresión como para la clasificación. Son uno de los métodos de conjunto más populares que pertenecen a la categoría específica de métodos de ensamblaje (Cutler, A., Cutler, D.R., y Stevens, J.R., 2012).

Los métodos ensamblaje implican el uso de muchos aprendizajes para mejorar el rendimiento de cualquiera de ellos individualmente. Estos métodos pueden describirse como técnicas que utilizan un grupo de aprendizaje débiles (aquellos que, por término medio, obtienen solo resultados ligeramente mejores que un modelo aleatorio) de forma conjunta, con el fin de crear uno más fuerte y agregado.

En este caso, los algoritmos random forest son un conjunto de muchos árboles de decisión individuales.

Uno de los principales inconvenientes de los árboles de decisión es que son muy propensos a sobreajustarse, es decir, funcionan bien con los datos de entrenamiento, pero no son tan flexibles para hacer predicciones sobre muestras no vistas. Aunque hay soluciones para esto, como la poda de los árboles, esto reduce su poder de predicción. En general, son modelos con un sesgo medio y una varianza alta, pero son sencillos y fáciles de interpretar (Cutler, A., Cutler, D.R., y Stevens, J.R., 2012).

Los modelos de random forest combinan la simplicidad de los árboles de decisión con la flexibilidad y la potencia de un modelo de conjunto. En un bosque de árboles, se deja a un lado la alta varianza de un árbol específico y se vuelve de menor interés cada elemento individual, por lo que es posible hacer crecer árboles que tienen más poder predictivo que uno podado.

Aunque los modelos de random forest no ofrecen tanta capacidad de interpretación como un solo árbol, su rendimiento es mucho mejor, y no es necesario preocuparse tanto por afinar perfectamente los parámetros del bosque como se hace con los árboles individuales.

Construcción de un random forest

En primer lugar, como Cutler, A., Cutler, D.R., y Stevens, J.R. (2012) indican en su libro, es necesario crear un conjunto de datos *bootstrapped* (método de remuestreo) para cada árbol. Cuando se construye un árbol de decisión individual, se emplea un conjunto de datos de entrenamiento y todas las observaciones. Esto significa que, si no se es cuidadoso, el árbol puede ajustarse muy bien a estos datos de entrenamiento, y generalizar mal a nuevas observaciones no vistas. Para resolver este problema, se evita que el árbol crezca mucho, normalmente a costa de reducir su rendimiento.

Para construir un random forest es necesario entrenar N árboles de decisión. Los árboles no son entrenados utilizando siempre los mismos datos, ni se utiliza todo el conjunto de datos.

Aquí es donde entra en juego la primera característica aleatoria. Para entrenar cada árbol individual, se selecciona una muestra aleatoria de todo el conjunto de datos, como se muestra en la siguiente figura.

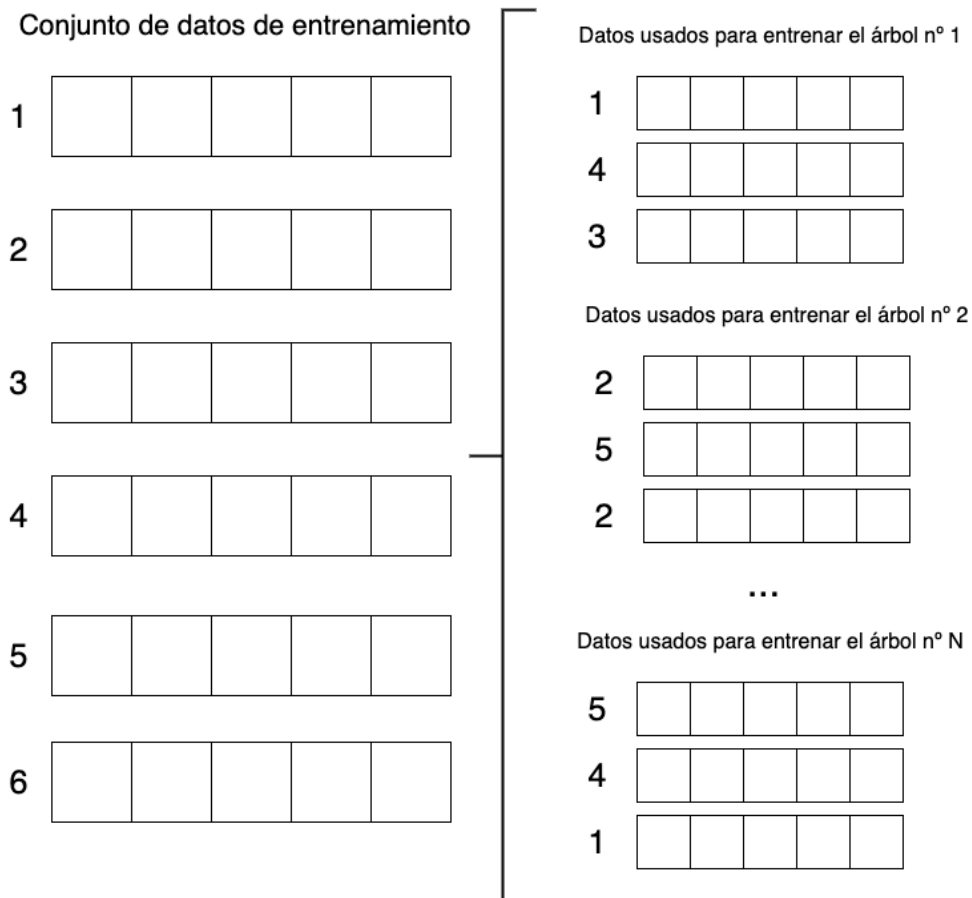


Figura 2-1 Representación gráfica del reparto de datos para entrenar cada árbol individual

De esta figura se pueden deducir varias cosas. En primer lugar, el tamaño de los datos utilizados para entrenar cada árbol individual no tiene por qué ser el de todo el conjunto de datos. Además, un punto de datos puede estar presente más de una vez en los datos utilizados para entrenar un solo árbol (como se puede observar en el árbol número 2).

Esto se denomina muestreo con reemplazo o *bootstrapping*: cada punto de datos se elige al azar de todo el conjunto de datos, y un punto de datos puede elegirse más de una vez (Cutler, A., Cutler, D.R., y Stevens, J.R., 2012).

Al utilizar diferentes muestras de datos para entrenar cada árbol individual se consigue reducir uno de los principales problemas que tienen, el sobreajuste. Si se entrena un bosque con muchos árboles y cada uno de ellos ha sido entrenado con datos diferentes, es posible solucionar este problema. Todos son muy aficionados a sus datos de entrenamiento, pero el bosque no es aficionado a ningún punto de datos específico. Esto permite hacer crecer árboles individuales más grandes, debido a que ya no es tan preocupante que un árbol individual se sobreajuste.

Si se emplea una porción muy pequeña del conjunto de datos para entrenar cada árbol individual, se aumenta la aleatoriedad del bosque (reduciendo el sobreajuste), pero normalmente a costa de un menor rendimiento.

En la práctica, la mayoría de las implementaciones de random forest, como la de Scikit-Learn (s.f.), eligen por defecto la muestra de los datos de entrenamiento utilizados para cada árbol

para que sea del mismo tamaño que el conjunto de datos original (sin embargo, no es el mismo conjunto de datos, porque se están eligiendo muestras aleatorias).

El hecho de entrenar un bosque de árboles utilizando estos conjuntos de datos aleatorios, y añadir un poco más de aleatoriedad con la selección de características, generalmente proporciona un buen compromiso de sesgo-varianza (Cutler, A., Cutler, D.R., y Stevens, J.R., 2012).

Para construir un árbol de decisión individual, en cada nodo se evalúa una determinada métrica y se elige la característica o variable de los datos que se encuentra en el nodo que minimiza esta métrica.

Este método funciona bien cuando se entrena un solo árbol, pero ahora se necesita un bosque entero de ellos. Por lo tanto, para lograr este objetivo, los modelos de conjunto son necesarios. Al igual que random forest, los modelos de conjunto funcionan mejor si los modelos individuales (árboles individuales en este caso) no están correlacionados. En random forest esto se consigue seleccionando aleatoriamente ciertas características para evaluarlas en cada nodo (Cutler, A., Cutler, D.R., y Stevens, J.R., 2012).

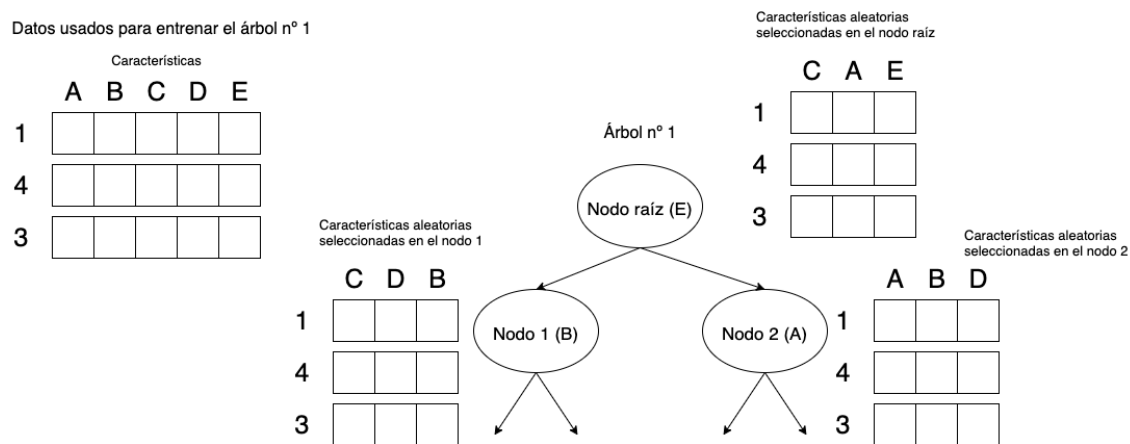


Figura 2-2 Representación gráfica del proceso de entrenamiento del algoritmo random forest

Como se puede ver en la imagen anterior, en cada nodo se evalúa únicamente un subconjunto de todas las características iniciales. En el nodo raíz se encuentran C, A y E (y E gana). En el nodo 1 se consideran C, D y B (y gana B). Por último, en el nodo 2 están presentes A, B y D (y gana A). Se continua haciendo esto hasta construir todo el árbol.

Gracias a este procedimiento, se evita incluir características que tienen un poder predictivo muy alto en cada árbol, mientras que se crean muchos árboles no relacionados (Cutler, A., Cutler, D.R., y Stevens, J.R., 2012). Este es el segundo barrido de la aleatoriedad. No solo se emplean datos aleatorios, sino también características aleatorias al construir cada árbol. Cuanto mayor sea la diversidad de árboles, mejor, pues se reduce la varianza y se obtiene un modelo de mejor rendimiento.

A continuación, se repite este proceso para los N árboles, seleccionando aleatoriamente en cada nodo de cada uno de los árboles qué variables entran en concurso para ser elegidas como la característica a dividir.

Hacer predicciones con random forest

Hacer predicciones con random forest es muy fácil. Simplemente es necesario tomar cada uno de los árboles individuales, pasar la observación para la que se desea hacer una predicción a través de ellos, obtener una predicción de cada árbol (sumando hasta N predicciones) y luego obtener una predicción global, agregada.

Hacer un *bootstrap* de los datos y luego utilizar un agregado para hacer una predicción se denomina *bagging* (Cutler, A., Cutler, D.R., y Stevens, J.R., 2012), y la forma en que se hace esta predicción depende del tipo de problema.

Para los problemas de regresión, la decisión agregada es la media de las decisiones de cada árbol de decisión. Para los problemas de clasificación, la predicción final es la predicción más frecuente realizada por el bosque.

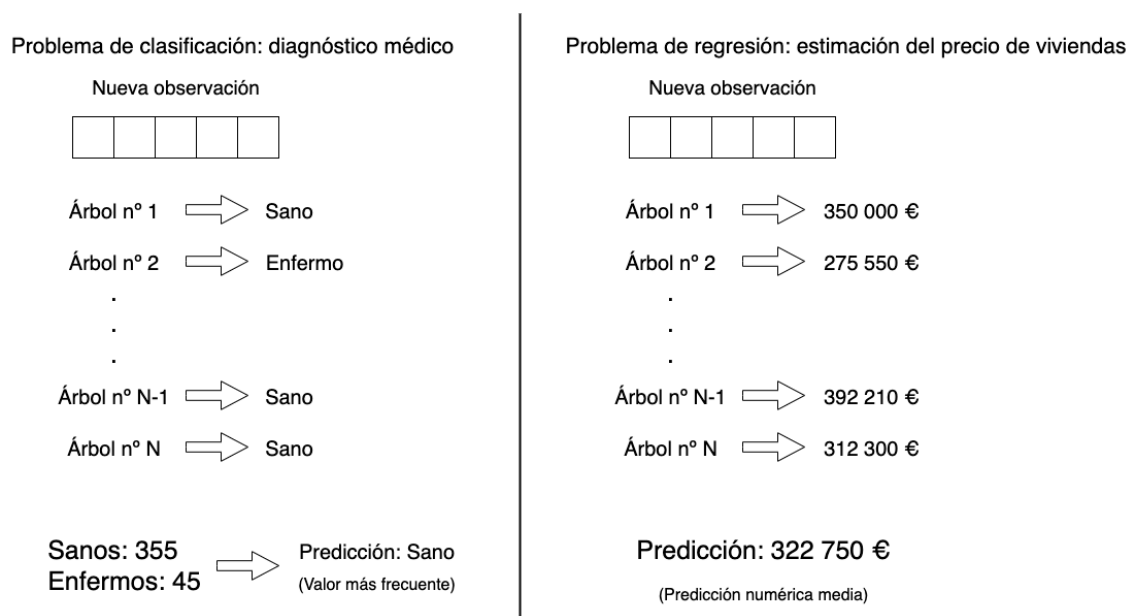


Figura 2-3 Representación gráfica de casos de uso del algoritmo random forest

La imagen anterior ilustra este procedimiento tan sencillo. Para el problema de clasificación se quiere predecir si un determinado paciente está enfermo o sano. Para ello, se recibe su historial médico y otra información por cada árbol del random forest, y se obtienen N predicciones (400 en este caso). En el ejemplo, 355 de los árboles indican que el paciente está sano y 45 dicen que está enfermo, por lo que el bosque decide que el paciente está sano.

Para el problema de regresión se requiere predecir el precio de una determinada casa. El algoritmo recibe las características de esta nueva casa por los N árboles, obteniendo una predicción numérica de cada uno de ellos. A continuación, se calcula la media de estas predicciones y se obtiene el valor final de 322 750 €.

2.2.2 AdaBoost

AdaBoost, al igual que el clasificador random forest, es otro clasificador de conjunto (los clasificadores de conjunto están formados por múltiples algoritmos clasificadores y su resultado es el resultado combinado de los resultados de esos algoritmos clasificadores).

El clasificador AdaBoost combina un algoritmo de clasificación débil para formar un clasificador fuerte. Un solo algoritmo puede clasificar mal los objetos. Pero si se combinan múltiples clasificadores con la selección del conjunto de entrenamiento en cada iteración y asignando la cantidad correcta de peso en la votación final, es posible tener una buena puntuación de precisión para el clasificador global.

En resumen, AdaBoost reentrena el algoritmo de forma iterativa eligiendo el conjunto de entrenamiento basado en la precisión del entrenamiento anterior. El peso de cada clasificador entrenado en cualquier iteración depende de la precisión alcanzada (Schapire, R.E., 2013).

Cada clasificador débil se entrena utilizando un subconjunto aleatorio del conjunto de entrenamiento. Pero el subconjunto aleatorio no es realmente 100 % aleatorio.

Después de entrenar un clasificador en cualquier nivel, AdaBoost asigna un peso a cada elemento de entrenamiento. A los elementos mal clasificados se les asigna un peso mayor para que aparezcan en el subconjunto de entrenamiento del siguiente clasificador con mayor probabilidad (Schapire, R.E., 2013).

Una vez entrenado cada clasificador, se le asigna un peso en función de su precisión. Al clasificador más preciso se le asigna un mayor peso para que tenga más impacto en el resultado final (Schapire, R.E., 2013).

Un clasificador con una precisión del 50 % recibe un peso de cero, y un clasificador con una precisión inferior al 50 % recibe un peso negativo.

Como Schapire, R.E. (2013) indica en su libro, la fórmula matemática es la siguiente:

$$H(x) = \text{sign}\left(\sum_{t=1}^T a_t h_t(x)\right)$$

$h_t(x)$ es la salida del clasificador débil t para la entrada x

a_t es el peso asignado al clasificador.

a_t se calcula de la siguiente manera: $a_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t} \rightarrow$ el peso del clasificador es directo y se basa en la tasa de error ϵ .

Inicialmente, todos los ejemplos de entrenamiento de entrada tienen el mismo peso.

A continuación, se puede observar un gráfico que muestra la relación de a_t con la tasa de error.

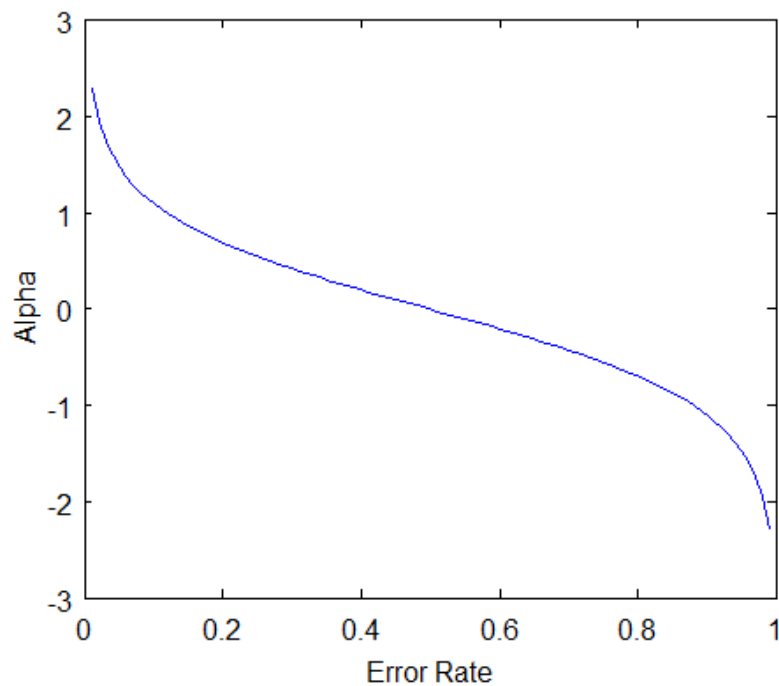


Figura 2-4 Gráfico que muestra la relación de α_t con la tasa de error. Fuente: McCormick, C. (2013)

Posteriormente, se lleva a cabo una actualización del peso de los ejemplos de entrenamiento.

Una vez entrenado el clasificador débil, se actualiza el peso de cada ejemplo de entrenamiento con la siguiente fórmula que Schapire, R.E. (2013) indica:

$$D_{t+1}(i) = \frac{D_t(i)^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

D_t es el peso en el nivel anterior.

Los pesos son normalizados dividiendo cada uno de ellos por la suma de todos los pesos, Z_t . Por ejemplo, si todos los pesos calculados suman 15.7, entonces se divide cada uno de los pesos por 15.7 para que sumen 1.0.

y_i es la coordenada y del ejemplo de entrenamiento (x_i, y_i) para simplificar.

AdaBoost, al igual que el clasificador random forest, ofrece resultados más precisos, debido a que depende de muchos clasificadores débiles para la decisión final.

2.3 Métricas para la evaluación de modelos

2.3.1 La división entrenamiento/prueba/validación

Lo más importante que se puede hacer para evaluar correctamente un modelo es no entrenar el modelo en todo el conjunto de datos. Una división típica de entrenamiento/prueba sería utilizar el 70 % de los datos para el entrenamiento y el 30 % de los datos para la prueba.

Es importante utilizar datos nuevos al evaluar el modelo para evitar la probabilidad de sobreajuste del conjunto de entrenamiento. Sin embargo, a veces es útil evaluar el modelo mientras se está construyendo para encontrar los mejores parámetros, pero no se puede utilizar el conjunto de pruebas para esta evaluación o se acabarán seleccionando los parámetros que

mejor funcionan en los datos de prueba, pero quizá no los parámetros que mejor generalizan. Para evaluar el modelo sin dejar de construirlo y ajustarlo, se crea un tercer subconjunto de datos conocido como conjunto de validación. Una división típica de entrenamiento/prueba/validación sería utilizar el 60 % de los datos para el entrenamiento, el 20 % de los datos para la validación y el 20 % de los datos para la prueba.

También hay que tener en cuenta que es muy importante barajar los datos antes de hacer estas divisiones para que cada una de ellas tenga una representación exacta del conjunto de datos.

2.3.2 Métricas de clasificación

Al realizar predicciones de clasificación, pueden producirse cuatro tipos de resultados.

- Los verdaderos positivos se producen cuando se predice que una observación pertenece a una clase y realmente pertenece a esa clase.
- Los verdaderos negativos se producen cuando se predice que una observación no pertenece a una clase y en realidad no pertenece a esa clase.
- Los falsos positivos se producen cuando se predice que una observación pertenece a una clase cuando en realidad no es así.
- Los falsos negativos se producen cuando se predice que una observación no pertenece a una clase cuando en realidad sí lo hace.

Estos cuatro resultados se suelen representar en una matriz de confusión. La siguiente matriz de confusión es un ejemplo para el caso de la clasificación binaria. Esta matriz se genera después de hacer predicciones sobre los datos de prueba y de identificar cada predicción como uno de los cuatro resultados posibles descritos anteriormente.

		Prediction	
		0	1
True Label	0	48 true negatives	8 false positives
	1	4 false negatives	37 true positives

Figura 2-5 Matriz de confusión. Fuente: Jordan, J. (2017)

También se puede ampliar esta matriz de confusión para trazar predicciones de clasificación multiclase. A continuación se muestra un ejemplo de matriz de confusión para clasificar las observaciones del conjunto de datos de la flor de Iris.

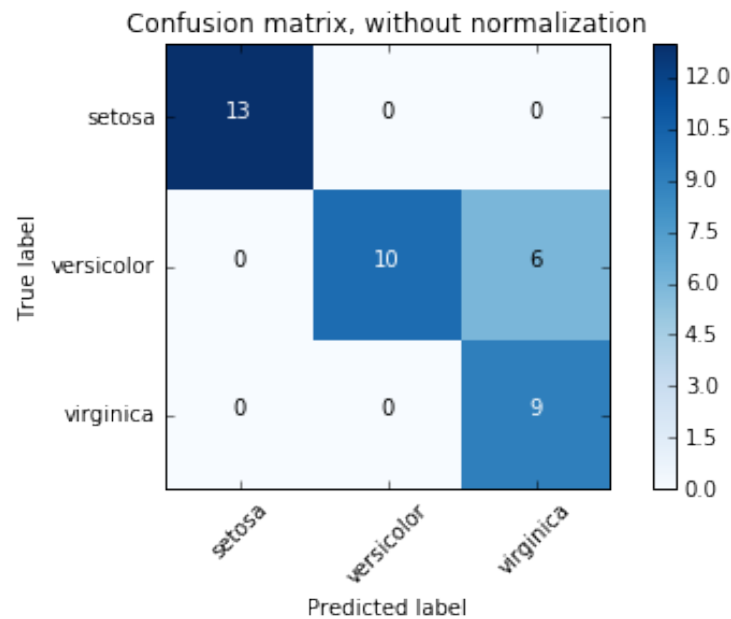


Figura 2-6 Matriz de confusión sin normalizar. Fuente: Jordan, J. (2017)

Las tres métricas principales utilizadas para evaluar un modelo de clasificación son la *accuracy*, la *precision* y el *recall* (Hossin, M., y Sulaiman, M.N., 2015).

La ***accuracy*** se define como el porcentaje de predicciones correctas para los datos de prueba. Se puede calcular fácilmente dividiendo el número de predicciones correctas por el número de predicciones totales.

$$accuracy = \frac{\text{predicciones correctas}}{\text{todas las predicciones}}$$

La ***precision*** se define como la fracción de ejemplos relevantes (verdaderos positivos) entre todos los ejemplos que se predice que pertenecen a una determinada clase.

$$precision = \frac{\text{positivos verdaderos}}{\text{positivos verdaderos} + \text{positivos falsos}}$$

El ***recall*** se define como la fracción de ejemplos que se predice que pertenecen a una clase con respecto a todos los ejemplos que realmente pertenecen a la clase.

$$recall = \frac{\text{positivos verdaderos}}{\text{positivos verdaderos} + \text{negativos falsos}}$$

El siguiente gráfico hace un gran trabajo al visualizar la diferencia entre precisión y *recall*.

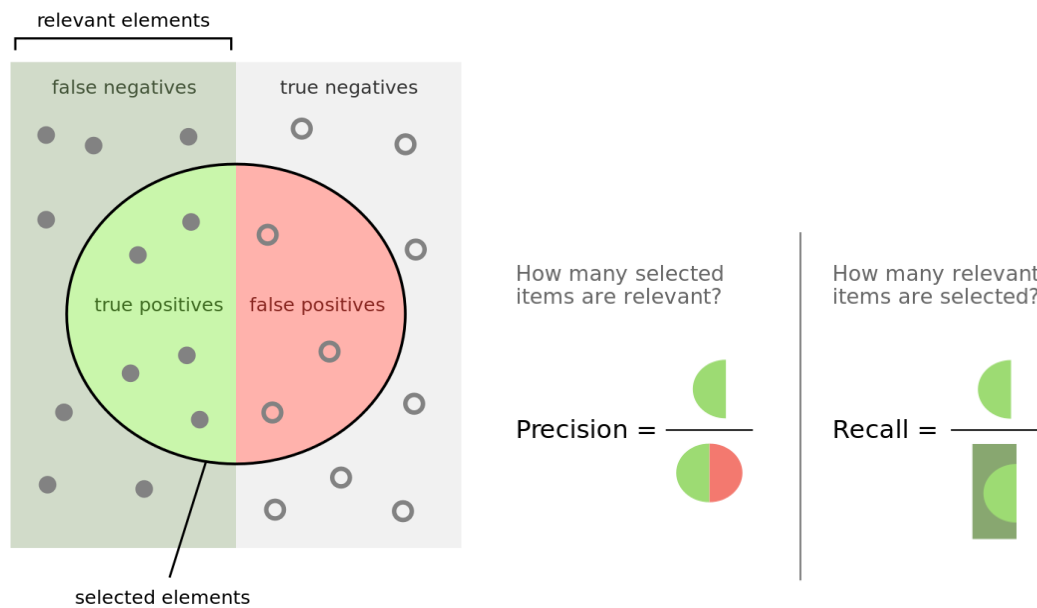


Figura 2-7 Estados de los elementos predichos. Fuente: Jordan, J. (2017)

La precisión y el *recall* son útiles en los casos en que las clases no están distribuidas uniformemente. El ejemplo más común es el desarrollo de un algoritmo de clasificación que predice si alguien tiene o no una enfermedad. Si solo un pequeño porcentaje de la población (por ejemplo, el 1 %) tiene esta enfermedad, sería posible construir un clasificador que siempre predijera que la persona no tiene la enfermedad, con lo que se habría construido un modelo con un 99 % de precisión y un 0 % de utilidad.

Sin embargo, si se mide el *recall* de este predictor inútil, quedaría claro que hay algo que no funciona en el modelo. En este ejemplo, el *recall* garantiza que no se está pasando por alto a las personas que tienen la enfermedad, mientras que la precisión garantiza que no se están clasificando erróneamente a demasiadas personas como si tuvieran la enfermedad cuando no es así. Evidentemente, no se querría un modelo que predijera incorrectamente que una persona tiene cáncer (la persona acabaría en un proceso de tratamiento doloroso y costoso por una enfermedad que no tiene), pero tampoco se querría predecir incorrectamente que una persona no tiene cáncer cuando en realidad sí lo tiene. Por ello, es importante evaluar tanto la precisión como el *recall* de un modelo.

En última instancia, es bueno tener un número para evaluar un modelo de aprendizaje automático, al igual que se obtiene una única calificación en un examen en la escuela. Por lo tanto, tiene sentido combinar las métricas de precisión y recuperación; el enfoque común para combinar estas métricas se conoce como puntuación *f* (Hossin, M., y Sulaiman, M.N., 2015).

$$F_{\beta} = (1 + \beta^2) \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

El parámetro β permite controlar el compromiso de importancia entre la precisión y la recuperación. $\beta < 1$ se centra más en la precisión, mientras que $\beta > 1$ se centra más en el *recall*.

Otro ejemplo sería el uso del aprendizaje automático para predecir el suicidio. En este caso, sería de mayor interés centrarnos mucho más en el *recall* del modelo que en su precisión. Sería

mucho menos perjudicial realizar una intervención con alguien que no estuviera considerando el suicidio que pasar por alto a alguien que estuviera considerando el suicidio. Sin embargo, la precisión sigue siendo importante porque no se quiere que haya demasiados casos en los que el modelo prediga falsos positivos.

Es importante tener en cuenta que la precisión por sí sola no es demasiado informativa respecto a la eficacia de un modelo.

2.3.3 Métricas de regresión

Las métricas de evaluación de los modelos de regresión son bastante diferentes de las métricas anteriores que han sido discutidas para los modelos de clasificación porque ahora se está prediciendo en un rango continuo en lugar de un número discreto de clases. Si el modelo de regresión predice que el precio de una casa es de 400 000 dólares y se vende por 405 000 dólares, es una predicción bastante buena. Sin embargo, en los ejemplos de clasificación solo es importante saber si una predicción era correcta o incorrecta, no se podía decir que una predicción era «bastante buena». Por lo tanto, se tiene un conjunto diferente de métricas de evaluación para los modelos de regresión.

La **varianza explicada** compara la varianza dentro de los resultados esperados, y la compara con la varianza en el error de nuestro modelo. Esta métrica representa esencialmente la cantidad de variación en el conjunto de datos original que el modelo es capaz de explicar (Hossin, M., y Sulaiman, M.N., 2015).

$$EV(y_{true}, y_{pred}) = 1 - \frac{Var(y_{true} - y_{pred})}{y_{true}}$$

El **error cuadrático medio** se define simplemente como la media de las diferencias al cuadrado entre la salida predicha y la salida real. El error cuadrático se utiliza habitualmente porque no tiene en cuenta si la predicción fue demasiado alta o demasiado baja, sino que simplemente informa de que la predicción fue incorrecta (Hossin, M., y Sulaiman, M.N., 2015).

$$MSE(y_{true}, y_{pred}) = \frac{1}{n_{samples}} \sum (y_{true} - y_{pred})^2$$

El **coeficiente R²** representa la proporción de la varianza del resultado que el modelo es capaz de predecir basándose en sus características (Hossin, M., y Sulaiman, M.N., 2015).

$$R^2(y_{true}, y_{pred}) = 1 - \frac{\sum (y_{true} - y_{pred})^2}{\sum (y_{true} - \bar{y})^2}$$

$$\bar{y} = \frac{1}{n_{samples}} \sum y_{true}$$

Este valor va de 0 a 1. El valor «1» indica que el predictor explica perfectamente toda la variación en Y. El valor «0» indica que el predictor «x» no explica ninguna variación en «y».

2.3.4 La curva AUC-ROC

Esta métrica está diferenciada del resto debido a que es la que se emplea en este trabajo como métrica principal para analizar la calidad del modelo de Machine Learning, debido a que, a menudo, se prefiere el AUC a la precisión (*accuracy*) para la clasificación por varias razones.

AUC significa Área Bajo la Curva, bajo la curva ROC. ROC son las siglas de Receiver Operating Characteristic. El objetivo implícito del AUC es hacer frente a situaciones en las que se tiene una distribución de muestras muy sesgada y no se quiere hacer un ajuste excesivo a una sola clase.

Un buen ejemplo es la detección de spam. Por lo general, los conjuntos de datos de spam están fuertemente sesgados hacia el spam, o el no-spam. Si un conjunto de datos está compuesto en un 90 % por spam, se puede obtener una precisión bastante buena diciendo simplemente que todos los correos electrónicos son spam, lo cual es obviamente algo que indica un clasificador no ideal.

El AUC mide cómo se compensan la tasa de verdaderos positivos (*recall*) y la tasa de falsos positivos. Lo más importante es que el AUC no es una función del umbral, sino que es una evaluación del clasificador a medida que el umbral varía en todos los valores posibles. Es, en cierto sentido, una métrica amplia que comprueba la calidad del valor interno que el clasificador genera y que luego compara con un umbral. No comprueba la calidad de una elección concreta de umbral (Narkhede, S., 2018).

El AUC tiene una interpretación diferente, y es que también es la probabilidad de que un ejemplo positivo elegido al azar se sitúe por encima de un ejemplo negativo elegido al azar, según el valor interno del clasificador para los ejemplos.

Esta métrica es muy completa, aunque aplicable en menos situaciones que otras métricas como el *accuracy*. No es estrictamente mejor que el *accuracy*; es diferente.

Además, la curva AUC-ROC es la métrica más empleada en los problemas de Machine Learning relacionados con la medicina. Esto se debe no solo a que los médicos pueden entenderla fácilmente, sino a que es posible ajustar el modelo según las necesidades del problema, es decir, priorizar la precisión o priorizar el *recall*.

En el problema que se aborda en este Trabajo de Fin de Grado, lo que resulta más interesante es dar prioridad al *recall*, pues no se quiere dejar de predecir el rango de días de alta para los pacientes.

2.3.5 Sesgo contra varianza

El objetivo último de cualquier modelo de aprendizaje automático es aprender de los ejemplos y generalizar cierto grado de conocimiento en relación con la tarea para la que está siendo entrenando. Algunos modelos de aprendizaje automático proporcionan el marco para la generalización sugiriendo la estructura subyacente de ese conocimiento. Por ejemplo, un modelo de regresión lineal impone un marco para aprender relaciones lineales entre la información que se le proporciona. Sin embargo, a veces se crea un modelo con demasiada estructura preestablecida, con lo que se limita la capacidad del modelo para aprender de los ejemplos, como el caso en el que se entrena un modelo lineal en un conjunto de datos exponencial. En este caso, el modelo está sesgado por la estructura y las relaciones preimpuestas.

Los modelos con un alto sesgo prestan poca atención a los datos presentados; esto también se conoce como infraajuste.

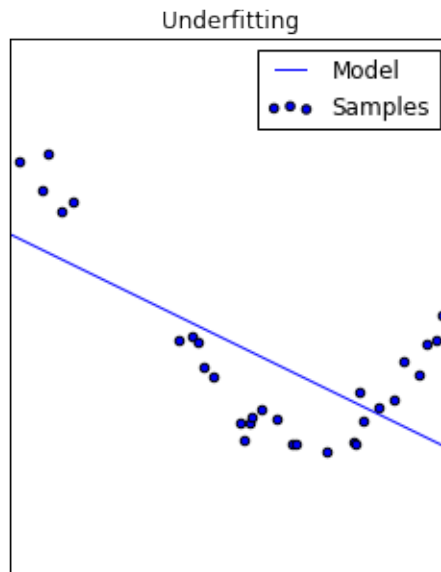


Figura 2-8 Infraajuste. Fuente: Scikit-learn (s.f.)

También es posible sesgar un modelo al intentar enseñarle a realizar una tarea sin presentar toda la información necesaria. Si se sabe que las restricciones del modelo no están sesgando su rendimiento y, sin embargo, es posible observar signos de infraajuste, es probable que no se estén utilizando suficientes características para entrenar el modelo.

En el otro extremo, a veces cuando se entrena un modelo, este aprende demasiado de los datos de entrenamiento. Es decir, el modelo capta el ruido de los datos además de la señal. Esto puede provocar fluctuaciones salvajes en el modelo que no representan la verdadera tendencia; en este caso, se dice que el modelo tiene una alta varianza. En este caso, el modelo no generaliza bien porque presta demasiada atención a los datos de entrenamiento sin tener en cuenta la generalización a nuevos datos. En otras palabras, se ha sobreajustado el modelo a los datos de entrenamiento.

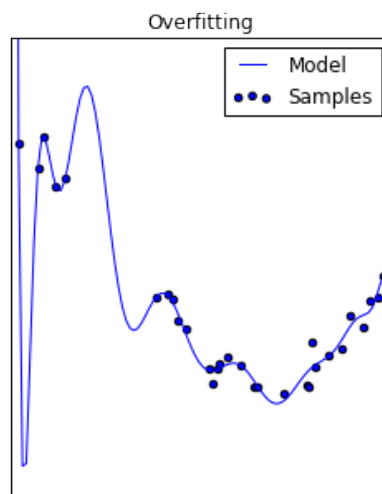


Figura 2-9 Sobreajuste. Fuente: Scikit-learn (s.f.)

En resumen, un modelo con un sesgo elevado no puede aprender la verdadera tendencia y no se ajusta a los datos. Un modelo con alta varianza aprende demasiado de los datos de entrenamiento y se ajusta demasiado a los datos. El mejor modelo se sitúa en algún punto intermedio de los dos extremos.

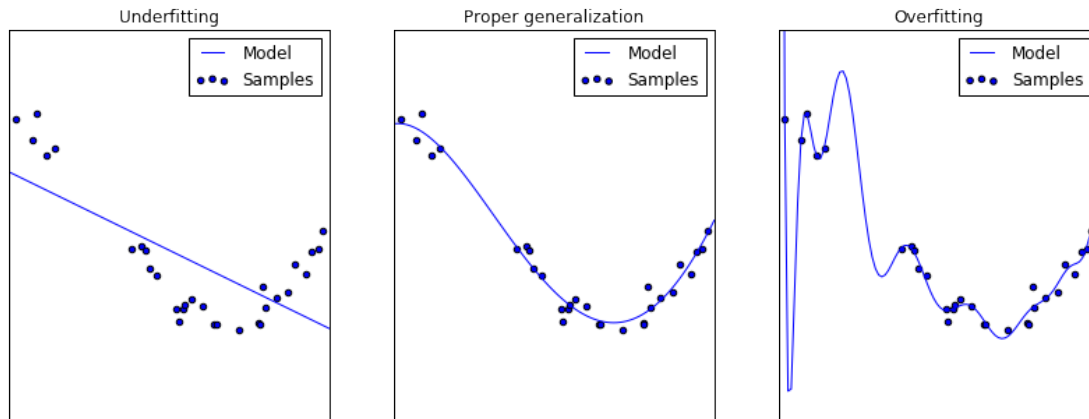


Figura 2-10 Comparación de los tipos de generalización de un modelo. Fuente: Scikit-learn (s.f.)

2.3.6 Otras buenas prácticas

En ocasiones, una de las cosas que se pueden hacer a la hora de evaluar modelos de clasificación es reducir el conjunto de datos a dos dimensiones y luego trazar las observaciones y el límite de decisión. A veces es útil inspeccionar visualmente los datos y el modelo al evaluar su rendimiento.

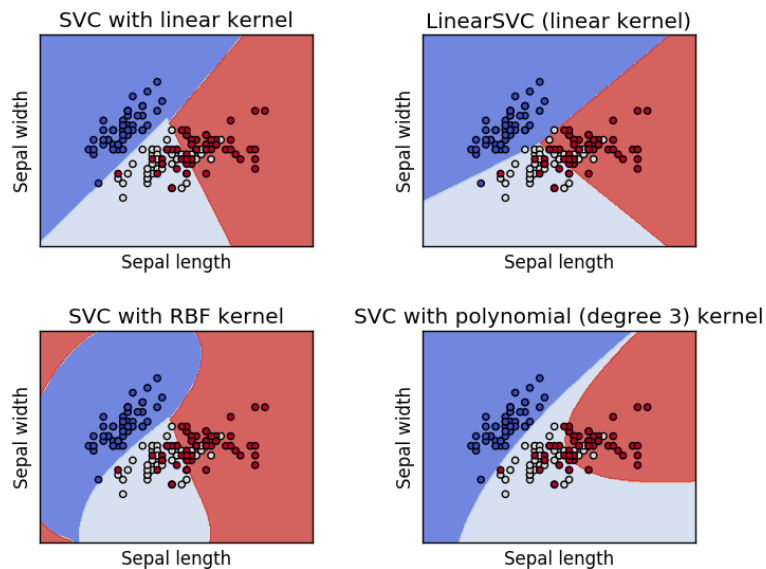


Figura 2-11 Inspección visual de los datos. Fuente: Scikit-learn (s.f.)

2.4 Interpretación del modelo

Se hace uso de los valores SHAP para la interpretación del modelo. SHAP (que significa SHapley Additive exPlanations) es un método para explicar las predicciones individuales de un modelo de Machine Learning.

2.4.1 Funcionamiento de los valores SHAP

SHAP se basa en los valores de Shapley, un concepto de la teoría de juegos desarrollado por el economista Lloyd Shapley. El método ayuda a explicar un modelo permitiendo ver cuánto contribuye cada característica a la predicción del modelo. Cada característica del modelo representará un «jugador», mientras que el «juego» será la predicción del modelo. Por lo tanto, se tratará de ver cuánto contribuye cada jugador al juego.

El proceso para hacer esto implica calcular la predicción del modelo con la característica y sin cada característica. Al obtener la diferencia entre estas dos predicciones, es posible ver cuánto contribuye esa característica a la predicción del modelo. Esta es la contribución marginal de la característica. Esto se hace para cada subconjunto de características y se toma la media de estas contribuciones para obtener el valor Shapley de la característica (Lundberg, S., y Lee, S. I., 2017).

2.4.2 Cálculo de distribuciones marginales

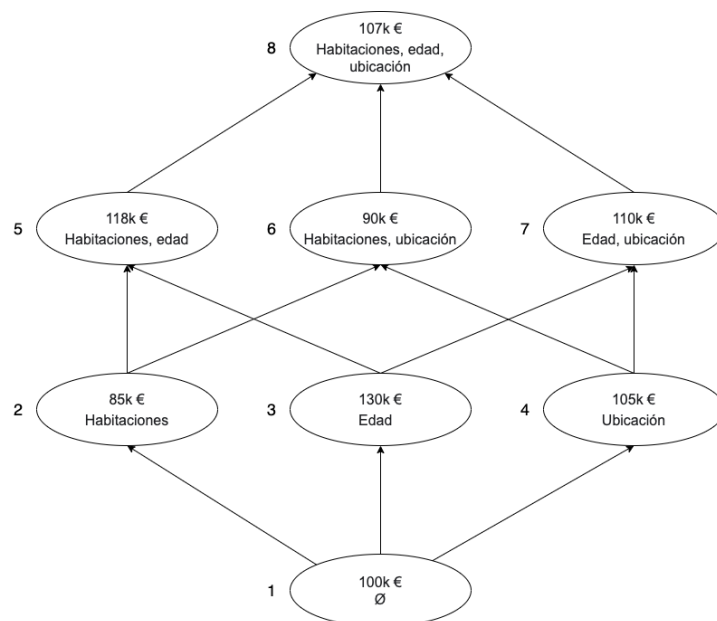


Figura 2-12 Gráfico de las características de un modelo de ejemplo

Para entender el concepto de los valores SHAP, se hará uso de un ejemplo. Se tiene un modelo que predice el precio de una casa. La imagen de arriba lo muestra en forma de gráfico. Existen tres características: Habitaciones, Edad y Ubicación. En total, se tienen 8 subconjuntos diferentes de características. Cada nodo del gráfico representará un modelo distinto, por lo que también se tienen 8 modelos diferentes. Cada modelo es entrenado en su subconjunto correspondiente y se predice la misma fila de datos.

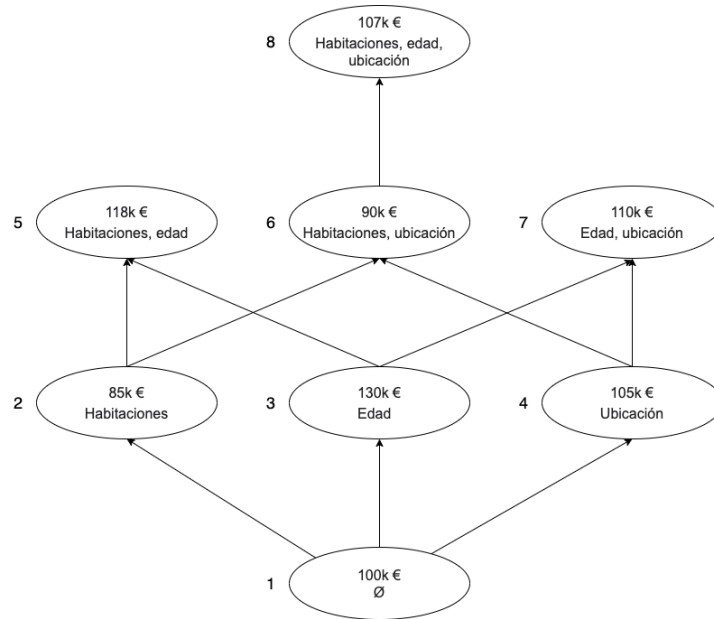


Figura 2-13 Primer paso en el proceso del cálculo de los valores SHAP

Cada nodo del gráfico está conectado a otro nodo mediante una arista dirigida. El nodo 1 no tiene características, lo que significa que solo predice el valor medio visto en los datos de entrenamiento (100k €). Siguiendo la arista de color azul que va al nodo 2, se observa que el modelo con una única característica Habitaciones predice un valor inferior a 85k €. Esto significa que la contribución marginal de Habitaciones al modelo que tiene Habitaciones como única característica es de -15k € (85k € - 100k €). Es necesario realizar este cálculo para cada modelo en el que se añada la característica Habitaciones.

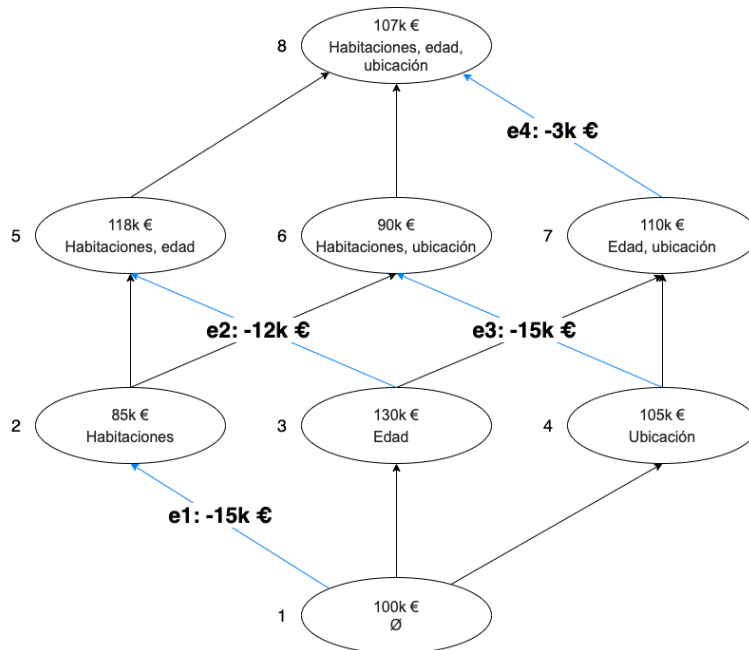


Figura 2-14 Segundo paso en el proceso del cálculo de los valores SHAP

La imagen de arriba destaca cada arista en la que se añade la característica Habitaciones y también muestra la contribución marginal de esa característica en cada modelo. Lo siguiente que hay que hacer es sacar la media de estas contribuciones marginales. El único problema es cómo ponderar cada una de ellas en la media. Se podría pensar en ponderar cada una de ellas por igual, pero no es el caso. Los modelos con menos características implicarían que las contribuciones marginales de cada característica serían mayores. Por lo tanto, los modelos con el mismo número de características deben tener el mismo peso.

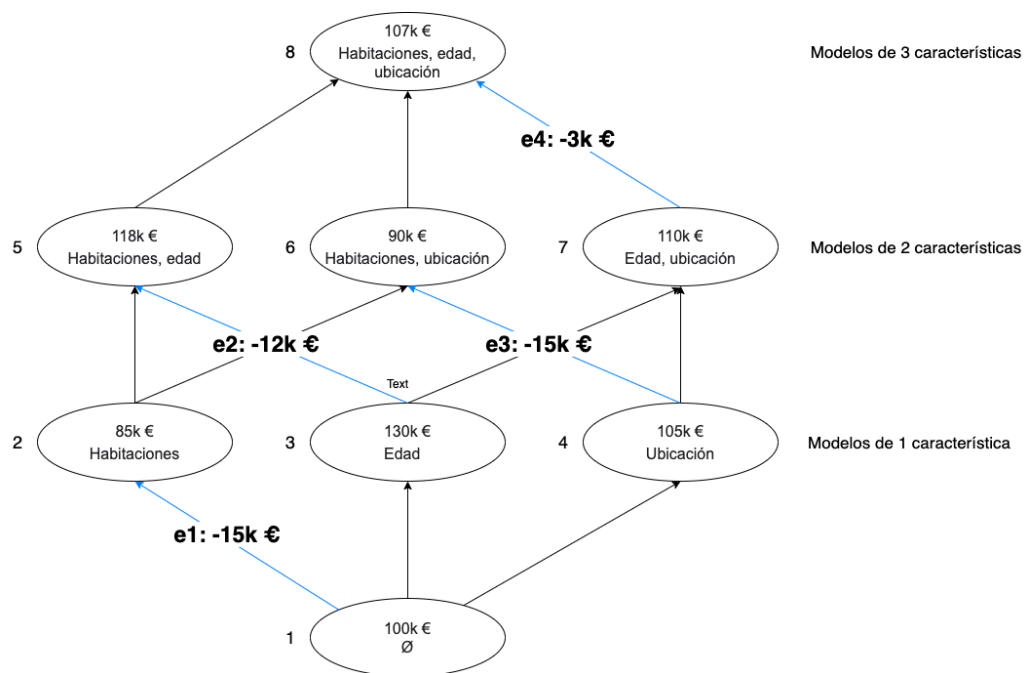


Figura 2-15 Tercer paso en el proceso del cálculo de los valores SHAP

Es posible agrupar el gráfico en filas como se ve arriba. Cada fila tiene modelos con un número diferente de características. Al promediar las contribuciones marginales, se busca que cada fila tenga el mismo peso (Lundberg, S., y Lee, S. I., 2017). Al tener tres filas, cada fila tiene un peso de 1/3. Para la fila con 2 modelos de características, existen dos modelos con la característica Habitaciones, por lo que cada uno de esos modelos debería tener un peso de 1/6. El desglose para el peso de cada «tipo» de modelo es el siguiente:

- Modelos de 1 característica: 1/3.
- Modelos de 2 características: 1/6.
- Modelos de 3 características: 1/3.

Y el cálculo final sería algo así:

$$\frac{1}{3}(-15k) + \frac{1}{6}(-12k) + \frac{1}{6}(-15k) + \frac{1}{3}(-3k)$$

El valor SHAP para la característica Habitaciones es, por tanto, de -10.5k €. Es posible repetir este proceso para cada característica de nuestro modelo para encontrar los valores de todas ellas. Lo bueno de este método en particular es que se puede observar cómo las características

afectan a las predicciones individuales en lugar de solo un efecto promedio sobre todos los ejemplos en el conjunto de datos.

2.5 Cómo manejar los datos que faltan

Uno de los problemas más comunes en la limpieza de datos y el análisis exploratorio es el manejo de los valores perdidos. En primer lugar, hay que entender que no hay una buena manera de tratar los datos que faltan. Hay diferentes soluciones para la imputación de datos dependiendo del tipo de problema: análisis de series temporales, Machine Learning, Regresión, etc. Por lo tanto, es difícil proporcionar una solución general.

A continuación se presentan las principales técnicas empleadas a la hora de manejar los datos faltantes en los conjuntos de datos. Las siguientes explicaciones se fundamentan en el *paper* de Soley-Bori, M. (2013), debido a que es un referente en cuanto al manejo de los valores faltantes.

2.5.1 Imputación frente a eliminación de los datos

Antes de pasar a los métodos de imputación de datos, es necesario entender la razón por la que faltan datos:

1. Falta al azar (MAR): Falta al azar significa que la propensión a que un punto de datos falte no está relacionada con los datos que faltan, pero sí con algunos de los datos observados.
2. Falta completamente al azar (MCAR): El hecho de que falte un determinado valor no tiene nada que ver con su valor hipotético y con los valores de otras variables.
3. Falta no aleatoria (MNAR): Dos posibles razones son que el valor omitido depende del valor hipotético (por ejemplo, las personas con salarios altos generalmente no quieren revelar sus ingresos en las encuestas) o el valor omitido depende del valor de alguna otra variable (por ejemplo, ¡supongamos que las mujeres generalmente no quieren revelar su edad! En este caso, el valor que falta en la variable de edad se ve afectado por la variable de género).

En los dos primeros casos, es seguro eliminar los datos con valores perdidos en función de su ocurrencia, mientras que en el tercer caso, eliminar las observaciones con valores perdidos puede producir un sesgo en el modelo. Por lo tanto, hay que tener mucho cuidado antes de eliminar las observaciones. Es importante tener en cuenta que la imputación no da necesariamente mejores resultados.

2.5.2 Borrado

Lista

La eliminación por lista (análisis de casos completos) elimina todos los datos de una observación que tiene uno o más valores perdidos. En particular, si los datos que faltan se limitan a un pequeño número de observaciones, se puede optar por eliminar esos casos del análisis. Sin embargo, en la mayoría de los casos, suele ser desventajoso utilizar la eliminación de la lista. Esto se debe a que los supuestos de MCAR son típicamente raros de soportar. Como resultado, los métodos de eliminación de la lista producen parámetros y estimaciones sesgados.

Por pares

La eliminación por pares analiza todos los casos en los que están presentes las variables de interés y, por tanto, maximiza todos los datos disponibles por una base de análisis. Un punto fuerte de esta técnica es que aumenta la potencia del análisis, pero tiene muchas desventajas.

Supone que los datos que faltan son MCAR. Si elimina los datos por pares, acabará con un número diferente de observaciones que contribuyen a diferentes partes de su modelo, lo que puede dificultar la interpretación.

Descartar variables

En líneas generales, siempre es mejor conservar los datos que descartarlos. A veces se pueden descartar variables si los datos faltan en más del 60 % de las observaciones, pero solo si esa variable es insignificante. Dicho esto, la imputación es siempre una opción preferible a la eliminación de variables.

2.5.3 Métodos específicos de series temporales

Última Observación Traslada (LOCF) y Siguiete Observación Traslada (NOCB)

Se trata de un enfoque estadístico habitual para el análisis de datos longitudinales de medidas repetidas en los que pueden faltar algunas observaciones de seguimiento. Los datos longitudinales hacen un seguimiento de la misma muestra en diferentes momentos. Estos dos métodos pueden introducir un sesgo en el análisis y no funcionan bien cuando los datos tienen una tendencia visible

Interpolación lineal

Este método funciona bien para una serie temporal con cierta tendencia, pero no es adecuado para los datos estacionales

Ajuste estacional más interpolación lineal

Este método funciona bien para datos con tendencia y estacionalidad

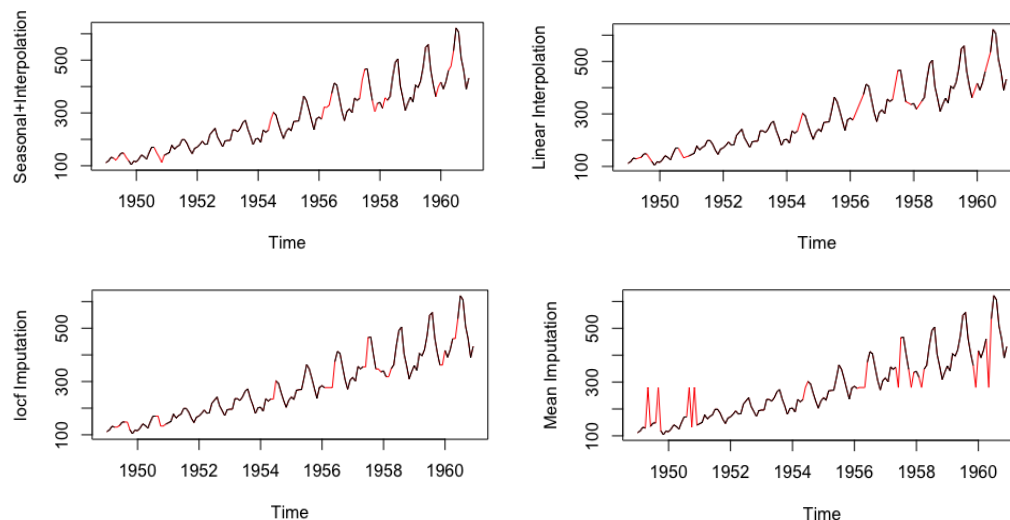


Figura 2-16 Ejemplo del método de interpolación lineal. Fuente: variable «tsAirgap» de la librería «imputeTS» con los datos interpolados en rojo

2.5.4 Media, mediana y moda

El cálculo de la media general, la mediana o la moda es un método de imputación muy básico, es la única función probada que no aprovecha las características de las series temporales ni la relación entre las variables. Es muy rápido, pero tiene claras desventajas. Una de las desventajas es que la imputación de la media reduce la varianza del conjunto de datos.

2.5.5 Regresión lineal

Para empezar, se identifican varios predictores de la variable con valores perdidos mediante una matriz de correlación. Se seleccionan los mejores predictores y se utilizan como variables independientes en una ecuación de regresión. La variable con datos ausentes se utiliza como variable dependiente. Los casos con datos completos para las variables predictoras se utilizan para generar la ecuación de regresión; la ecuación se utiliza entonces para predecir los valores que faltan en los casos incompletos. En un proceso iterativo, se insertan los valores de la variable que falta y luego se utilizan todos los casos para predecir la variable dependiente. Estos pasos se repiten hasta que hay poca diferencia entre los valores predichos de un paso al siguiente, es decir, convergen.

En teoría, proporciona buenas estimaciones para los valores perdidos. Sin embargo, este modelo tiene varias desventajas que tienden a superar las ventajas. En primer lugar, como los valores sustituidos se predijeron a partir de otras variables, tienden a encajar «demasiado bien», por lo que el error estándar se desinfla. También hay que suponer que existe una relación lineal entre las variables utilizadas en la ecuación de regresión, cuando puede que no la haya.

2.5.6 Imputación múltiple

En la imputación múltiple se imputan las entradas que faltan de los conjuntos de datos incompletos m veces ($m = 3$ en la figura). Es importante en cuenta que los valores imputados se extraen de una distribución. La simulación de extracciones aleatorias no incluye la incertidumbre en los parámetros del modelo. Un enfoque mejor es utilizar la simulación Markov Chain Monte Carlo (MCMC). Este paso da como resultado m conjuntos de datos completos.

A continuación se analiza cada uno de los m conjuntos de datos completos, y por último se integran los m resultados del análisis en un resultado final.

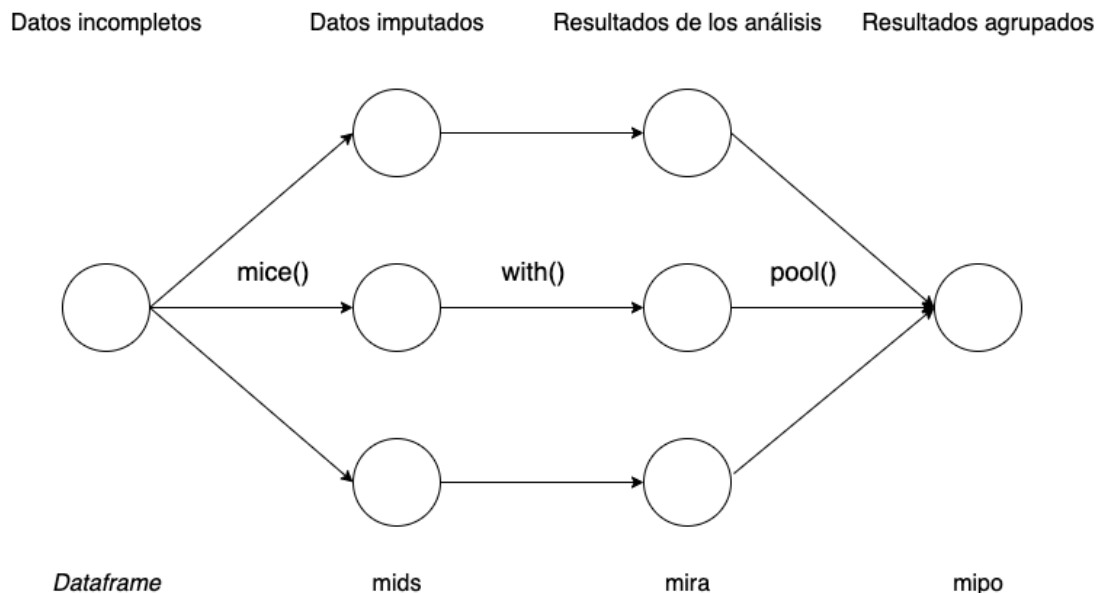


Figura 2-17 Esquema del funcionamiento de la imputación múltiple

Este es, con mucho, el método preferido para la imputación por su facilidad de uso y porque no hay sesgos (si el modelo de imputación es correcto).

2.5.7 KNN (K-Nearest Neighbors)

Existen otras técnicas de aprendizaje automático como XGBoost y Random Forest para la imputación de datos, pero se hablará de KNN porque es muy utilizado (Faisal, S., y Tutz, G., 2017). En este método, se eligen «k» vecinos en función de alguna medida de distancia y su media se utiliza como estimación de la imputación. El método requiere la selección del número de vecinos más cercanos y una métrica de distancia. KNN puede predecir tanto atributos discretos (el valor más frecuente entre los k vecinos más cercanos) como continuos (la media entre los k vecinos más cercanos)

La métrica de distancia varía según el tipo de datos:

- Datos continuos: Las métricas de distancia comúnmente utilizadas para los datos continuos son la euclidiana, la de Manhattan y la del coseno (Zakka, K., 2016).
- Datos categóricos: En este caso se suele utilizar la distancia de Hamming. Toma todos los atributos categóricos y para cada uno, cuenta uno si el valor no es el mismo entre dos puntos. La distancia de Hamming es entonces igual al número de atributos cuyo valor es diferente (Zakka, K., 2016).

Una de las características más atractivas del algoritmo KNN es que es sencillo de entender y fácil de implementar. La naturaleza no paramétrica de KNN le da una ventaja en ciertos escenarios en los que los datos pueden ser muy «inusuales» (Brownlee, J., 2020).

Uno de los inconvenientes obvios del algoritmo KNN es que consume mucho tiempo cuando se analizan grandes conjuntos de datos porque busca instancias similares en todo el conjunto de datos. Además, la precisión del KNN puede verse gravemente degradada con datos de gran dimensión porque hay poca diferencia entre el vecino más cercano y el más lejano (Brownlee, J., 2020).

2.5.8 Cuál emplear

Entre todos los métodos mencionados anteriormente, la imputación múltiple y el KNN son ampliamente utilizados, y la imputación múltiple, al ser más simple, es generalmente preferida.

3. Especificación de requisitos

3.1 Descripción del sistema

El médico inicia sesión con su número de facultativo y su contraseña.

Una vez que el médico inicia sesión correctamente, es dirigido a la página principal del *dashboard* de la aplicación donde puede ver el listado de los pacientes con una pequeña información para cada uno de ellos y la predicción de días hasta su alta.

Cuando se selecciona uno de estos pacientes, el médico ve información más detallada sobre cada uno de ellos: información general del paciente (peso, altura, edad, etc.) y sus constantes vitales y otras mediciones (basófilos, bicarbonato, ALT-GPT, CO₂, etc.).

En lo relativo a la introducción de los datos de nuevos pacientes en el sistema, es el personal sanitario el encargado de realizar esta tarea mediante una segunda aplicación cuya única finalidad es la de introducir nuevos pacientes. Esta tarea se lleva a cabo de forma manual, mediante la introducción de los datos del paciente en un formulario; o de forma automática, a través de un fichero CSV que contenga los datos de los pacientes.

3.2 Modelado

En lo relativo al modelado en UML del sistema, se muestran los diagramas más importantes y relevantes, sin entrar en los detalles de casos en los que mostrar mensajes, excepciones, etc., pues todas estas situaciones podrán observarse con un mayor detenimiento en el archivo de Visual Paradigm adjunto con el resto del código.

3.2.1 Casos de uso

El subsistema relativo a la aplicación de lectura de los datos relativos al estado de los pacientes presenta los siguientes casos de uso: consultar la información general del hospital en cuanto a los pacientes afectados por el SARS-CoV-2, listar los pacientes y mostrar brevemente información de cada uno de ellos y consultar un paciente en concreto para ver con mayor detalle sus datos.

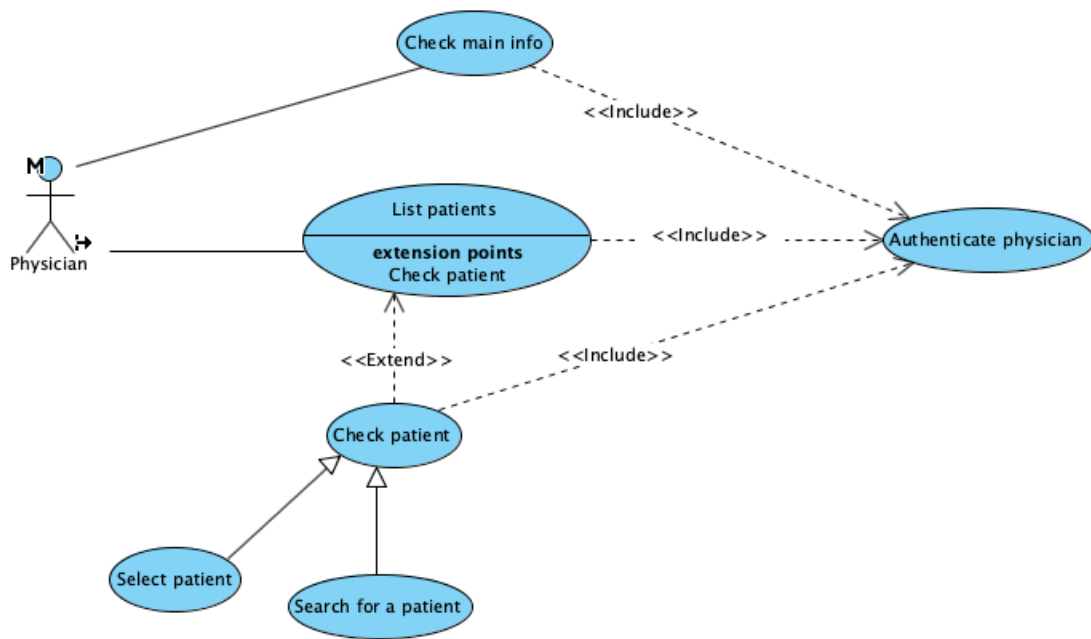


Figura 3-1 Casos de uso del subsistema dashboard

En lo relativo al subsistema de la aplicación de escritura de los datos de los pacientes, existe un único caso de uso, siendo este el de añadir nuevos pacientes al sistema.

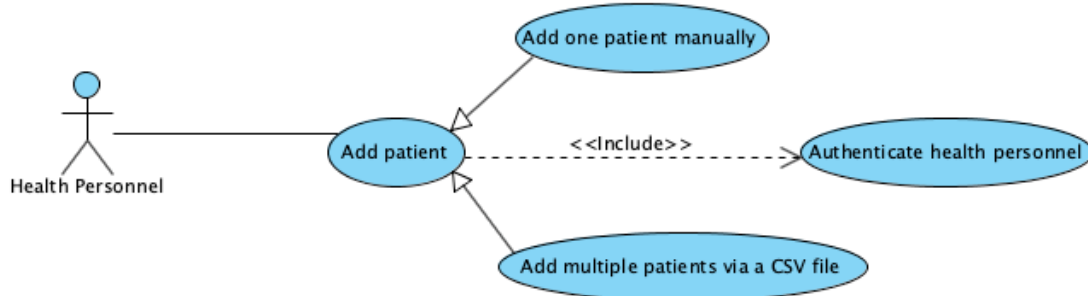


Figura 3-2 Casos de uso del subsistema alta de pacientes

3.2.2 Base de datos

Se empleará una base de datos NoSQL, debido a que los requerimientos de los datos no están claros y es muy probable que no estén estructurados. Esto se debe a que cada hospital mantendrá los datos de una forma diferente y no necesariamente tienen por qué recoger las mismas variables. Por lo tanto, la base de datos no puede tener un esquema predefinido.

La base de datos NoSQL a emplear será DynamoDB, debido a su gran flexibilidad y su facilidad para acceder a índices secundarios, lo que permitirá gestionar fácilmente datos anidados y acceder rápidamente a cualquier atributo (como por ejemplo la información de cada paciente). Además, es muy fácil integrarla dentro de la arquitectura en AWS basada en el patrón CQRS que el sistema tiene.

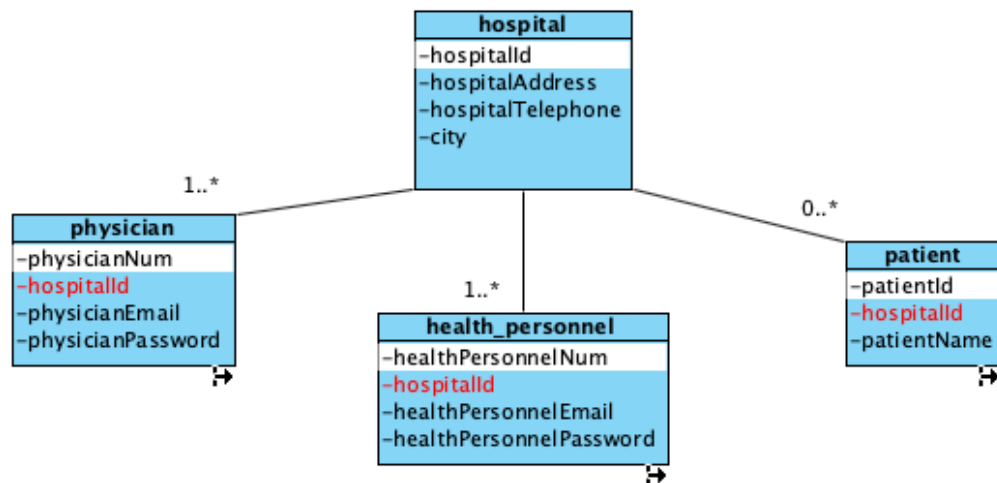


Figura 3-3 Base de datos de Health Twin

3.2.3 Diagramas de actividades

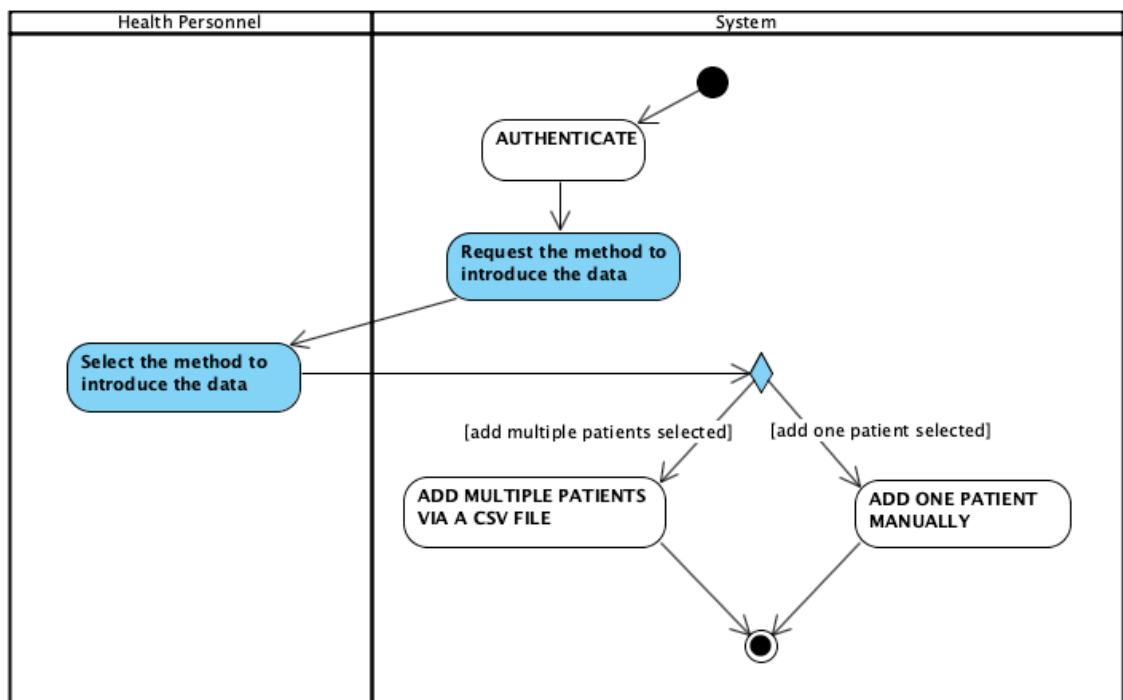


Figura 3-4 Diagrama de actividades para añadir un nuevo paciente al sistema

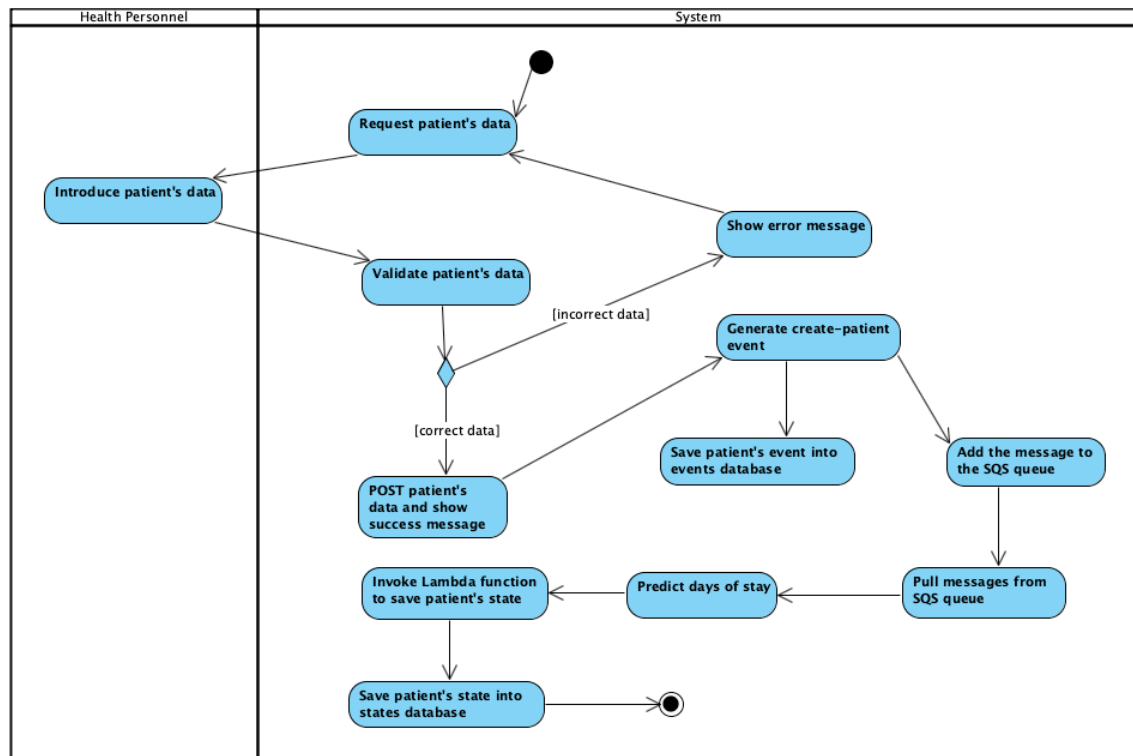


Figura 3-5 Diagrama de actividades para añadir un nuevo paciente manualmente

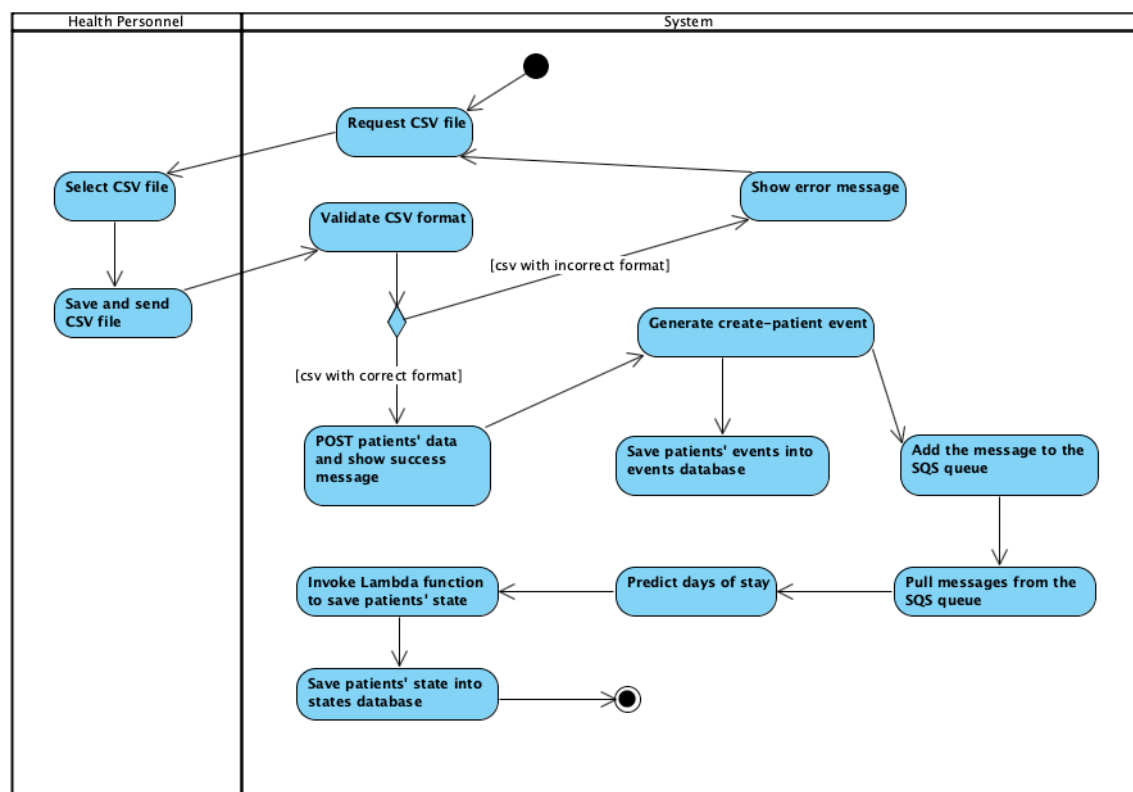


Figura 3-6 Diagrama de actividades para añadir varios pacientes por medio de un fichero CSV

3.2.4 Diagramas de secuencias

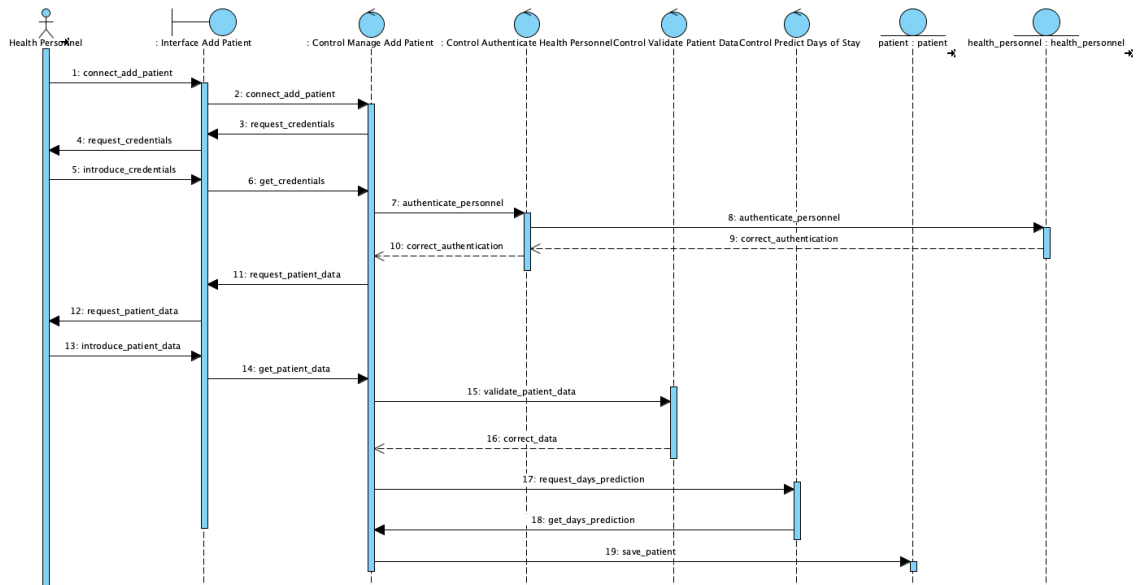


Figura 3-7 Diagrama de secuencias de análisis para añadir un paciente manualmente

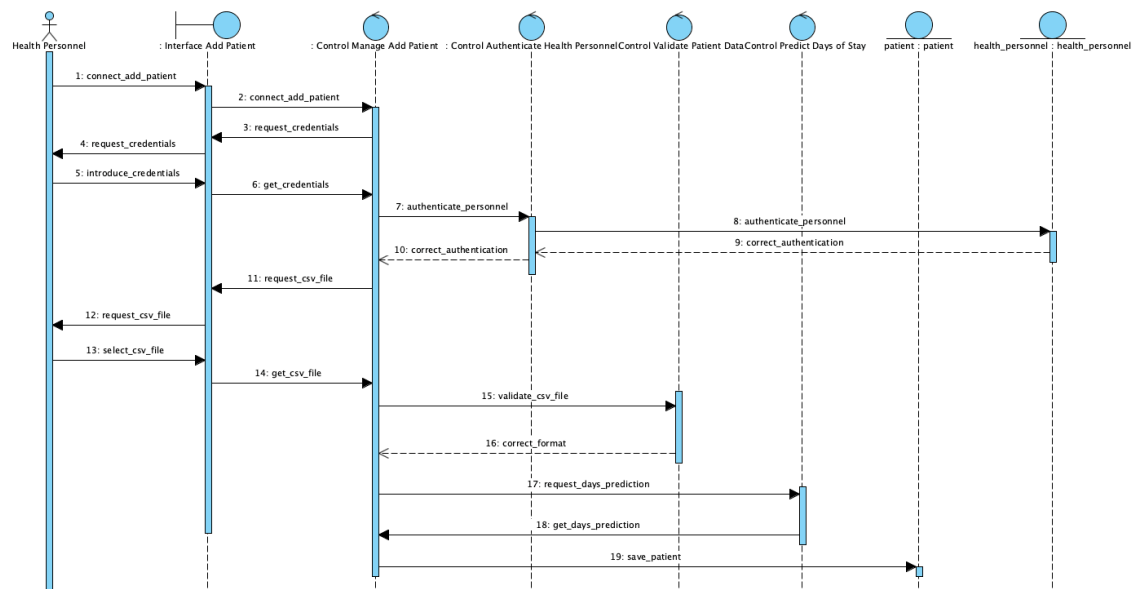


Figura 3-8 Diagrama de secuencias de análisis para añadir varios pacientes por medio de un fichero CSV

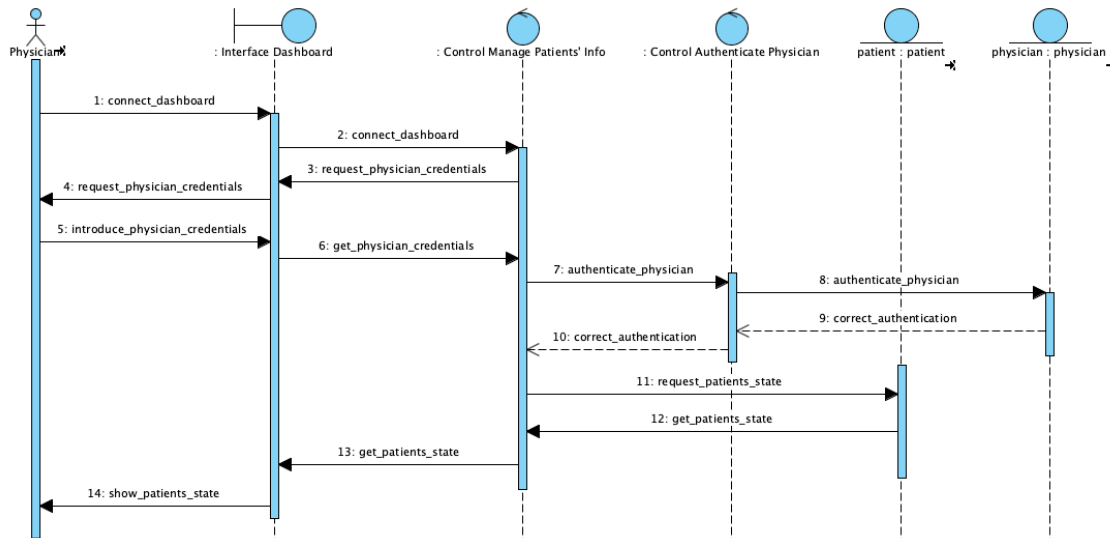


Figura 3-9 Diagrama de secuencias de análisis para obtener el listado del estado de los pacientes

3.3 Modelo de arquitectura

Para el modelo de arquitectura del proyecto se emplea CQRS

El patrón es el siguiente:

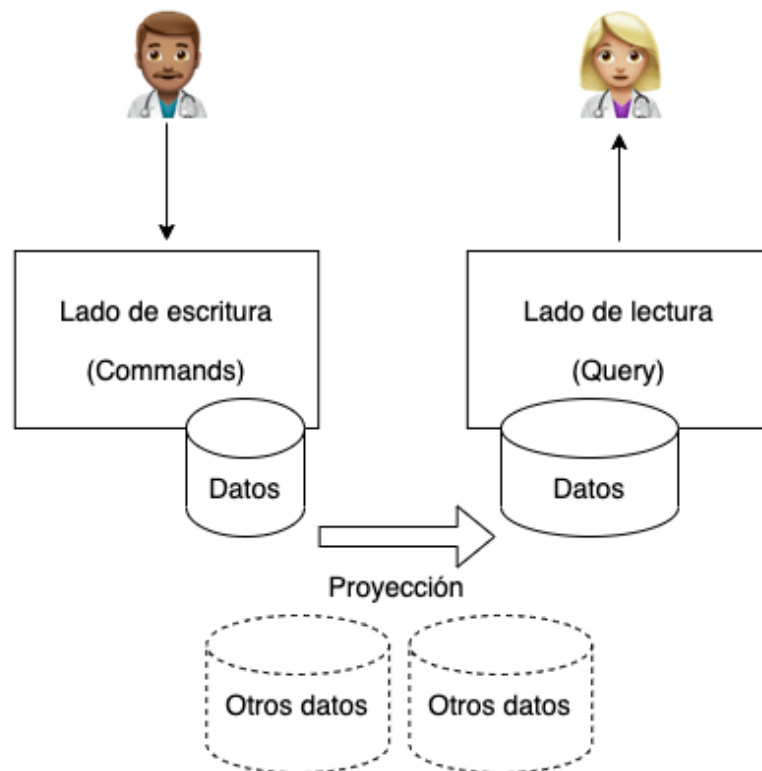


Figura 3-10 Representación del patrón CQRS en Health Twin

3.3.1 Un modelo de escritura, N modelos de lectura

La idea de CQRS es permitir que una aplicación (en sentido amplio) trabaje con diferentes modelos:

- Un modelo interno con el que se escribe: el modelo de escritura, alterado por los comandos.
- Uno o varios modelos de lectura de los que las aplicaciones leen.

Los modelos de lectura pueden ser leídos por *frontends* o por APIs, no importa.

3.3.2 Escalabilidad

CQRS permite escalar un sistema de forma independiente: a menudo es necesario manejar muchas más lecturas que escrituras, de ahí que la escalabilidad sea diferente.

Es deseable que las lecturas se accedan en un tiempo $O(1)$ (constante): no se quiere que tarden más en buscar una entidad porque tenga más enlaces con otras entidades: es preferible evitar los JOIN explosivos. La solución es precalcular el resultado cuando los datos cambian, antes de que nadie los solicite. Haciendo esto se utilizan menos recursos cuando se produce una petición, se reduce la latencia y se hace predecible y estable. De esta forma se está cambiando tiempo por espacio (Melnik, G., Dominguez, J., Cazzulino, D., Simonazzi, F., Subramanian, M., de Lahitte, H.,...Michell, N., 2012).

3.3.3 Requisitos del negocio

Separar el modelo de escritura de los modelos de lectura ayuda a separar los aspectos complejos de un dominio (quién necesita qué, quién es responsable de qué) y a aumentar la flexibilidad de la solución. Es posible adaptarse de forma más sencilla a los cambios en los requisitos del negocio (Melnik, G., Dominguez, J., Cazzulino, D., Simonazzi, F., Subramanian, M., de Lahitte, H.,...Michell, N., 2012).

Esto se debe a que se piensa más en las responsabilidades: ¿quién muta los datos? ¿cuál es el caso de uso? ¿es necesario poseer realmente estos datos? ¿se actúa sobre ellos? ¿no es responsabilidad de otra aplicación? ¿quién necesita solo leer los datos? ¿debe ser fuertemente consistente? etc. CQRS suele estar vinculado al DDD (diseño guiado por el dominio) por esta forma de pensar.

3.3.4 Concurrencia

Técnicamente, CQRS también puede simplificar la gestión de la concurrencia y el bloqueo (bases de datos transaccionales), al revelarlos.

Cuando se trabaja con un patrón CQRS asíncrono, a menudo se habla de que los datos son eventualmente consistentes, de los ciclos de vida de los datos, de las propiedades de los datos, de los requisitos de negocio y mucho sobre la modelización: los límites transaccionales de las entidades y los invariantes que siempre se deben tener. De nuevo, esta es la razón por la que CQRS suele estar orientado a DDD: los datos forman agregados que deben ser definidos con mucho cuidado (Melnik, G., Dominguez, J., Cazzulino, D., Simonazzi, F., Subramanian, M., de Lahitte, H.,...Michell, N., 2012).

3.3.5 No hay semántica de «lee tus propios escritos»

Los datos obsoletos deben ser tratados explícitamente. Si un recurso es alterado (enviando un comando) y este mismo recurso es leído inmediatamente, no se verán los cambios. Async CQRS no proporciona la semántica de «leer sus propias escrituras».

Los *frontends* pueden simularlo haciendo Concurrencia Optimista: se puede incrustar algún conocimiento y suponer que la mutación que se pidió está bien, por lo que muestra lo que cree que es la respuesta antes de obtener la real. En caso de discrepancias, se adapta.

Cuando se trata de CQRS sincrónico es posible obtener esta semántica: se escriben los dos modelos en tablas diferentes, en la misma transacción. Siempre está sincronizado. CQRS rara vez es síncrono porque se quiere trabajar o escalar con diferentes recursos, tipos de bases de datos, usar una infraestructura de mensajería: rara vez es posible hacer una transacción distribuida entre recursos (Melnik, G., Dominguez, J., Cazzulino, D., Simonazzi, F., Subramanian, M., de Lahitte, H.,...Michell, N., 2012).

3.3.6 Comandos/Escrituras: efectos secundarios

Lo que causa las escrituras en el modelo de escritura se denomina comando. Es un término genérico para designar lo que siempre se ha trabajado: algo para cambiar el estado del sistema (una actualización de cualquier tipo).

- Un comando puede ser tratado de forma sincrónica o asincrónica.
- Un comando puede pasar por un bus de mensajes o no.
- Un comando puede ser una simple llamada a la API.
- Un comando puede ser una superclase si hace OOP o no.
- Un comando puede ser una simple llamada a una función.

Todos estos conceptos son ortogonales a lo que es un comando. Se ordena que algún estado cambie de alguna manera. Un comando es una intención (y no un hecho) y causa efectos secundarios (en un recurso). Se dirige a un destino particular (no se emite) y se define en términos del dominio del consumidor (el codominio) (Melnik, G., Dominguez, J., Cazzulino, D., Simonazzi, F., Subramanian, M., de Lahitte, H.,...Michell, N., 2012).

Generalmente se define como «VerboAlgo»: CreateOrder, ShipProduct, ComputePrice, GiveMoney.

Un comando está centrado en el comportamiento y no en los datos. Se trata de la intención de cambiar algo, no se corresponde con el formato de un recurso (como los DTOs u objetos de transferencia de datos en una API). Puede contener datos que ayuden a procesar la intención, pero eso es todo.

También se habla de sistemas basados en tareas y no en recursos. La parte de escritura no acepta nuevos recursos o parches de recursos existentes: aceptan tareas, es decir, comandos.

3.3.7 Flujo

El flujo de manejo de un comando es siempre el mismo (Melnik, G., Dominguez, J., Cazzulino, D., Simonazzi, F., Subramanian, M., de Lahitte, H.,...Michell, N., 2012):

- Si es asíncrono, es guardado en una infraestructura de mensajería y devuelve el OK a quien lo ha llamado (no puede hacer mucho más).
- Al manejarlo (sync o async) según su forma (API, mensaje, llamada a función), llama al Manejador de Comandos correcto (una función).
- Este manejador debe determinar si es posible procesarlo:
 - Recupera el estado actual si debe actuar sobre él (desde una base de datos o usando Event Sourcing).
 - Utiliza algunas reglas de negocio para saber si puede conceder o denegar el comando sobre el estado.
- Si se concede, aplica el comando sobre el estado (puede generar Eventos o no).
- Si no se concede, devuelve un error (sync o async).

- Se guarda el nuevo estado.
- Si es *sync*, devuelve el OK a la persona que llama (o una información mínima como un ID pero no todo el estado, que es el modelo de escritura).
- Si es *async*, se envía el mensaje a la infraestructura de mensajería.

3.3.8 Ventajas del modelo de arquitectura CQRS

- Permite controlar la complejidad del sistema de forma muy sencilla.
- Se logra la consistencia a través de sistemas independientes.
- Permite la escalabilidad de forma independiente, debido a que, por ejemplo, si las lecturas en la aplicación superan ampliamente las escrituras (que es normalmente lo que sucede), con este enfoque se puede replicar el lado de lectura con más frecuencia y se pueden tener vistas independientes que son actualizadas empleando el sistema de almacenamiento de eventos y esto puede estar en vistas de memoria o incluso en bases de datos independientes.
- Si el lado de lectura escala independientemente, siempre estará disponible incluso si el lado de escritura falla. De tal forma que si el lado de escritura o el sistema de almacenamiento de eventos está caído, es posible leer, al menos, la última actualización de ese lado de lectura independiente. Esto hace que sea una muy buena opción para los sistemas altamente escalables.

3.3.9 Desventajas del modelo de arquitectura CQRS

- Añade una complejidad innecesaria si las aplicaciones tienen operaciones CRUD sencillas, que se pueden conseguir con los estilos arquitectónicos tradicionales.
- Al requerir modelos separados para lectura y escritura, la duplicación de código es inevitable.
- En el caso de dos bases de datos separadas para lectura y escritura, la base de datos de escritura necesita actualizar la base de datos de lectura, lo que podría resultar en Vistas Eventualmente Consistentes.

3.4 Implementación de la arquitectura del sistema en AWS

3.4.1 Qué es AWS (Amazon Web Services)

En términos simples, AWS se considera un proveedor de la nube, lo que significa que AWS proporciona varios recursos de Tecnología de la Información basados en la nube a sus consumidores de la nube. En la actualidad, AWS es uno de los principales proveedores de servicios en la nube en todo el mundo.

3.4.2 Funcionamiento de AWS

Hay una variedad de servicios proporcionados por AWS que pueden ser configurados según el requisito de los usuarios. Sobre la base de la demanda y los requisitos, los usuarios pueden consultar las ubicaciones geográficas para el servidor individual con las opciones de configuración previstas en ese lugar en particular. A continuación, se explican brevemente los servicios que han sido empleados, siguiendo la documentación oficial de Amazon Web Services (s.f.).

3.4.3 Servicios de AWS empleados

AWS Lambda

AWS Lambda es un servicio de AWS que permite a las personas ejecutar funciones en la nube sin necesidad de aprovisionar o administrar servidores. Además, al utilizar AWS Lambda, las

personas tienen que pagar el importe solo cuando se ejecuta la función definida. Por lo tanto, utilizando Lambda, el código puede ser ejecutado sin una aplicación o servicio *backend* (Amazon Web Services, s.f.).

AWS EC2

Elastic Compute Cloud o EC2 es un servidor virtual que ayuda a los usuarios a ejecutar numerosas aplicaciones en la infraestructura de la nube de AWS.

Con AWS EC2 se obtienen instancias con diferentes configuraciones de recursos de CPU, memoria, almacenamiento y red. Cada tipo está disponible en diferentes tamaños para que sea posible atender la carga de trabajo según sea necesario.

Las instancias provienen de imágenes de máquina de Amazon (AMI). Estas imágenes de máquina actúan como una plantilla que configura un sistema operativo y determina el entorno operativo del usuario (Amazon Web Services, s.f.).

AWS IAM

AWS Identity Access Management (IAM) es un servicio de AWS que permite a los individuos administrar el acceso a varios servicios y recursos de AWS de forma segura. Por lo tanto, con la implementación de AWS IAM, las personas pueden crear y administrar varios usuarios y grupos. Además, los usuarios y grupos creados reciben varios permisos para permitir y denegar el acceso a los recursos de AWS. IAM es un servicio que no requiere ningún tipo de cargo adicional (Amazon Web Services, s.f.).

AWS SQS

Básicamente, SQS es un servicio web que da acceso a una cola de mensajes que puede utilizarse para almacenar mensajes mientras se espera que un consumidor los procese.

SQS es un sistema de colas distribuidas y una cola es un depósito temporal para los mensajes que están esperando ser procesados (Amazon Web Services, s.f.).

AWS SNS

AWS SNS, siglas de Simple Notification Service, es un servicio en la nube que gestiona y facilita la transmisión o distribución de mensajes a los puntos finales o clientes que se suscriben. Esto ofrece a los desarrolladores una capacidad altamente escalable, rentable y versátil para publicar los mensajes de una aplicación y distribuirlos a otras aplicaciones.

Sigue el paradigma de mensajería de publicar-suscribir (pub-sub) con la notificación que se entrega al cliente utilizando un mecanismo de empuje que elimina la necesidad de comprobar o sondear periódicamente para obtener nueva información y actualizaciones (Amazon Web Services, s.f.).

AWS DynamoDB

Amazon DynamoDB es un servicio NoSQL clave-valor administrado con una fuerte consistencia y un desempeño predecible que protege a los usuarios de las complejidades de configurarlo manualmente (Amazon Web Services, s.f.).

AWS API Gateway

AWS API Gateway es un servicio totalmente administrado que facilita a los desarrolladores la creación, publicación, mantenimiento, monitorización y seguridad de las API. La API actúa como una puerta de entrada para que la aplicación acceda a los datos, la lógica empresarial o la funcionalidad de los servicios de *backend*. Se encarga de todas las tareas relacionadas con la

aceptación y el procesamiento de cientos o miles de llamadas concurrentes a la API, incluida la gestión del tráfico, la autorización, el control de acceso, la supervisión y la gestión de la API (Amazon Web Services, s.f.).

3.4.4 Ventajas de AWS

Las principales ventajas que se pueden atribuir a Amazon Web Services (s.f.) son:

- AWS permite a las organizaciones utilizar modelos de programación, sistemas operativos, bases de datos y arquitecturas que ya conoce el usuario.
- Amazon Web Services puede ser muy rentable, debido a que el individuo que está utilizando los servicios en la nube tiene que pagar solo por lo que utiliza. Además, existe una capa gratuita muy generosa con la que la mayoría de proyectos en fase de desarrollo pueden funcionar sin incurrir en ningún tipo de gasto.
- Los gastos de creación, implementación y mantenimiento de los centros de datos no son necesarios cuando se utiliza AWS.
- A los usuarios se les facilita la ampliación y reducción de los recursos asignados según la demanda de recursos.
- AWS ofrece un despliegue rápido que permite a las personas obtener una satisfacción óptima del usuario.
- AWS permite implementar la aplicación en diferentes regiones de una forma muy sencilla.

3.4.5 Desventajas de AWS

Tras consultar la documentación de Amazon Web Services (s.f.), se puede afirmar que sus principales desventajas son:

- Al utilizar los servicios de AWS, el individuo debe pagar para obtener asistencia inmediata.
- Los recursos de AWS pueden diferir de una región a otra, ya que no todos los servicios de AWS se proporcionan a todas las regiones.
- Algunos problemas como la desaparición de archivos y el problema de la no sincronización del servidor pueden surgir mientras se trabaja con AWS.
- Sin acceso a Internet, no se puede acceder a los datos presentes en la nube.

3.4.6 Implementación del patrón CQRS en AWS

Tal y como se indica en el perfectamente detallado documento de Amazon Web Services (2019), implementar el patrón CQRS haciendo uso de los servicios de AWS es perfectamente posible y una práctica muy enriquecedora, tanto a nivel profesional como para desarrollar una mejor aplicación.

El patrón CQRS implementado con los servicios de AWS propuesto es el siguiente:

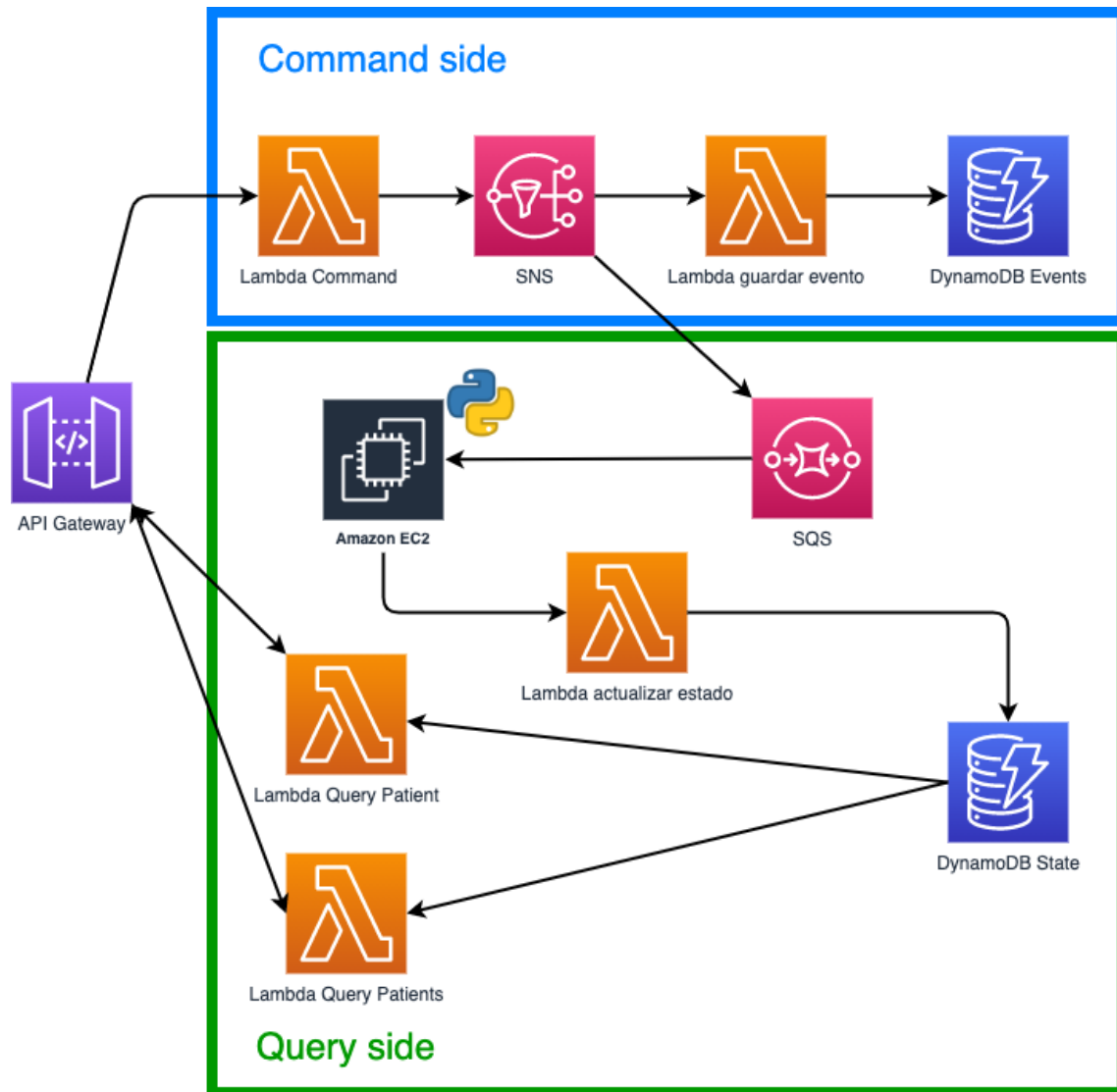


Figura 3-11 Diagrama que describe el sistema que implementa un patrón CQRS en AWS en Health Twin

La aplicación del personal sanitario únicamente realizará peticiones POST, por lo que el servicio API Gateway dirigirá estas peticiones hacia el lado de comandos. Una vez aquí, una función lambda se encargará de crear el evento que posteriormente SNS publicará.

La función lambda que guarda el evento en la base de datos de eventos y una cola SQS están suscritos a este servicio SNS, con lo que cada vez que se produzca la publicación de un nuevo evento (cada vez que el personal sanitario realice una petición POST), reciben este evento. La función lambda únicamente guarda este evento en la base de datos de eventos, mientras que la cola SQS almacena este evento (y todos los que puedan llegar) hasta que otro servicio lo consuma.

El porqué de incluir una cola SQS antes de la instancia del servicio EC2 es para evitar la pérdida de información si esta instancia se cae o deja de funcionar. Imagínese que no se hubiera incluido una cola de mensajes entre el servicio SNS y el servicio EC2. En esta situación, si la instancia deja de funcionar, pierde toda la información relativa a los eventos que sean publicados mientras dure el problema. Sin embargo, al incluir una cola de mensajes, la instancia no pierde información, pues es un *script* dentro de esta instancia quien se encarga de consumir

los mensajes (los eventos que han sido publicados por el servicio SNS y que el servicio SQS ha recibido). Por lo tanto, cuando la instancia no está funcionando, los mensajes simplemente se irán acumulando en la cola SQS hasta que la instancia vuelva a funcionar y el *script* siga consumiendo estos mensajes.

El *script* contenido en esta instancia de EC2 no solo se encarga de consumir los mensajes almacenados en la cola SQS, sino también de ejecutar la *pipeline* que permitirá predecir el rango de días en los que el paciente recibirá el alta de la UCI e invocar una función lambda que recibe toda la información del paciente (incluyendo ahora la variable predicha).

Posteriormente, la función lambda que es invocada por el *script* contenido en la instancia de EC2 se encarga de guardar el estado del paciente (es decir, únicamente los datos sin la información relativa al evento que se generó previamente) en la base de datos de estados.

La aplicación que los médicos emplean se encarga únicamente de realizar peticiones GET, con lo que el servicio API Gateway las dirige al lado de consultas, en el que una función lambda se encarga de consultar la información en la base de datos de estados y devolverla.

3.4.7 Despliegue de la arquitectura del entorno local a AWS

Para desplegar la arquitectura en AWS se ha hecho uso de un fichero de configuración YAML, de forma que simplemente es necesario ejecutar el comando *serverless deploy* y toda la arquitectura (los servicios, sus configuraciones y las conexiones entre ellos) que ha sido programada y configurada en local, se creará automáticamente sin necesidad de configurar todo manualmente. Este hecho supone un nivel de automatización y facilidad de implementación que supera por mucho al proceso manual de creación de estos servicios a través del sitio web de AWS, pues simplemente es necesario ejecutar un comando para que todo el sistema esté en funcionamiento, mientras que haciéndolo manualmente se tardaría mucho tiempo y es muy probable que se cometieran fallos, debido a que no solo hay que crear los servicios, sino también asignarle permisos y comunicarlos entre sí.

Se ha hecho uso de YAML en lugar de JSON porque es más legible. Además, YAML soporta comentarios, comandos en varias líneas y presenta más ventajas.

3.5 Metodologías ágiles

A continuación, se detalla la organización que se ha seguido en el desarrollo del proyecto aplicando metodologías ágiles, recogiendo características de Scrum y Kanban.

Debido a que el Trabajo de Fin de Grado ha sido realizado sin la colaboración de ningún equipo, todos los roles típicos de una metodología ágil (Product Owner, Scrum Master y el equipo de desarrollo) han sido asumidos por la misma persona.

Si se desea comprobar el tablero de Trello en el que se ha registrado el desarrollo del proyecto siguiendo metodologías ágiles, se puede acceder a través del siguiente enlace: <https://trello.com/b/vPaYXF63>

El director del Trabajo de Fin de Grado fue añadido al tablero de Trello desde el primer momento para que pudiera seguir correctamente el desarrollo del proyecto.

3.5.1 Temas

La tarjeta con el nombre «Temas» hace referencia a las diferentes categorías de trabajo que se consideran lo suficientemente independientes entre sí para poder ser desglosadas. Cada categoría de trabajo presenta un nombre descriptivo.

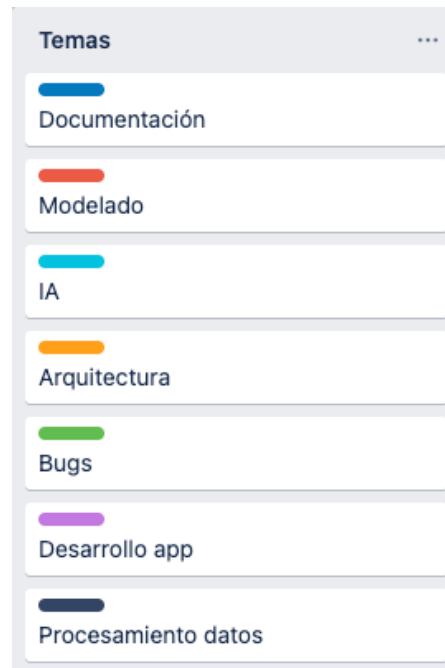


Figura 3-12 Tarjeta de Trello que contiene los temas del del desarrollo de Health Twin

3.5.2 Product Backlog

Siguiendo la metodología de Scrum, es en el Product Backlog donde se listan todas las tareas de cada tema, con sus respectivas Checklist. Además, se fija una fecha concreta (la fecha de finalización del Trabajo de Fin de Grado), que, siguiendo los principios de Scrum, puede variar si así lo desea el Product Owner ante necesidades del equipo de desarrollo.



Figura 3-13 Tarjeta de Trello que contiene los distintos ítems a completar en el desarrollo de Health Twin

3.5.3 Sprints Backlog

Tras la reunión del Sprint Planning se delimitan cuáles de las tareas del Product Backlog se estima que se pueden desarrollar durante el próximo Sprint, además de decretar su orden, debido a que en algunos casos la finalización de una determina el inicio de otra. Por último, se decreta aquí qué integrantes del equipo Scrum desarrollan las tareas indicadas en la tarjeta.

Un ejemplo de la *checklist* de una de las tareas sería el siguiente:

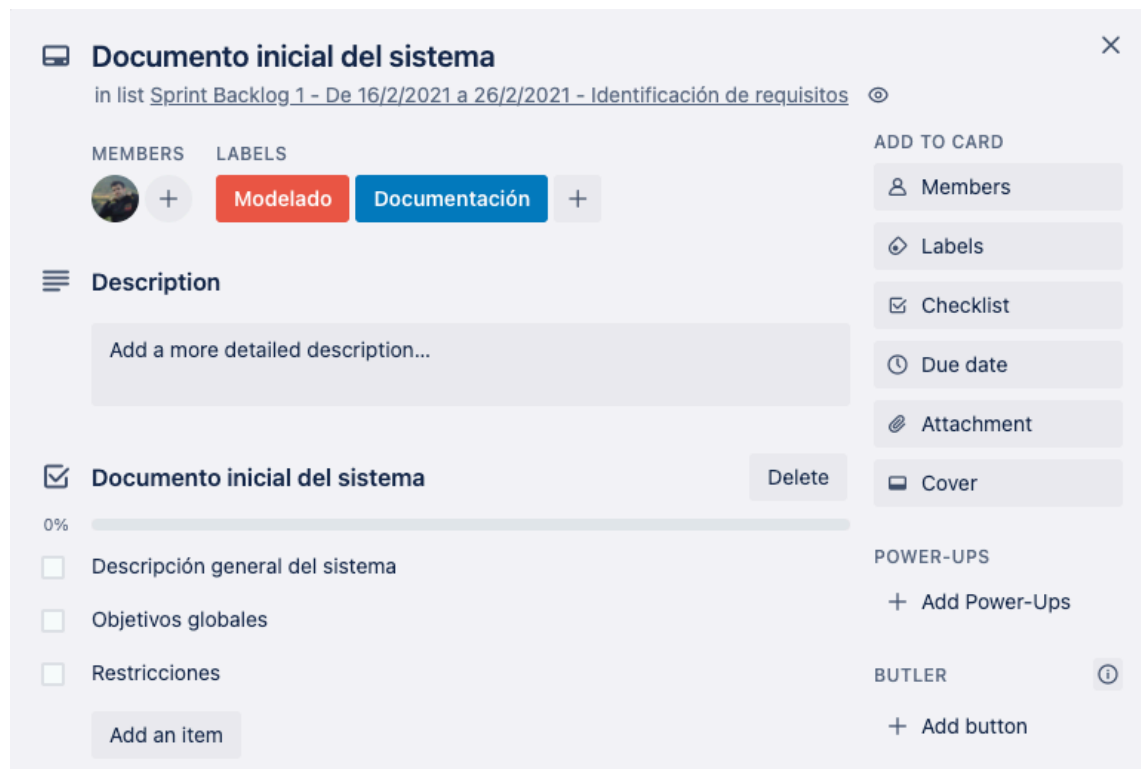


Figura 3-14 Información de uno de los ítems a completar en el desarrollo de Health Twin

3.5.4 En Progreso

Describe las tareas del Sprint actual que están siendo desarrolladas y no han sido finalizadas.

3.5.5 Revisando

Describe las tareas del Sprint que están siendo realizadas, una entrada de la *checklist* positiva indicará qué partes de la tarea han sido realizadas.

3.5.6 Sprints finalizados

Se crean listas con la información relativa a las tareas designadas en Sprints anteriores. Tras el Sprint Review se documenta qué ha sido completado correctamente, así como seguir el Burn-Up y Burn-Down por si fuera necesario realizar sus respectivos gráficos.

Al llegar a este paso se vuelve a realizar un Sprint Planning para determinar qué tareas se llevan a cabo, así como la duración del Sprint.

3.5.7 Procedimiento

Al comienzo del proyecto, el Product Owner, con la aceptación del Scrum Master y el equipo de desarrollo, fija los ítems en el Product Backlog. A continuación, se lleva a cabo el Sprint Planning, decretando los ítems a desarrollar en los próximos días.

En los proyectos reales de Scrum se debe realizar una reunión diaria (Daily meeting) de duración máxima de 15 minutos. En esta reunión se concreta lo que se va a realizar durante el día, problemas durante el proceso, etc. En este proyecto esta reunión diaria se ha realizado individualmente, debido a la naturaleza del Trabajo de Fin de Grado.

4. Desarrollo

4.1 Implementación del modelo de Machine Learning

4.1.1 Preparación para el modelado

Para comenzar esta sección, primero se eliminan del *dataframe* todas las columnas que no sean útiles para la elaboración de modelos predictivos, además de características que no estarían presentes en el momento de la admisión del paciente, evitando así la fuga de datos.

Posteriormente se realiza un análisis para comprobar el porcentaje de datos que faltan por cada columna del *dataframe*, con el que se obtiene que la altura falta en un 78.2 % de los pacientes, los valores de la Troponina en un 78.6 % y los valores de PVA_med_max, PVA_med_mean y PVA_med_min en un 36.4 %. Existen datos faltantes en la mayoría de las columnas, pero en menor proporción.

Teniendo en cuenta que solo se cuenta con un *dataframe* de 93 pacientes únicos, la mejor forma de tratar los datos faltantes es mediante el uso de la técnica de imputación múltiple explicada previamente en el capítulo relativo a los fundamentos teóricos. Gracias a esta técnica, se consigue que no haya ningún valor faltante en todo el *dataframe*. Para ello se hace uso de la librería que Scikit-Learn (s.f.) proporciona.

```
iter_imputer = IterativeImputer(estimator=KNeighborsRegressor(),
n_nearest_features=None, imputation_order='ascending')
iter_imputer.fit(data_no_date)
imputed_df = iter_imputer.transform(data_no_date)
imputed_df = pd.DataFrame(imputed_df, columns=data_no_date.columns)
```

Figura 4-1 Código que describe la imputación múltiple para solucionar el problema de los valores NA

Es necesario obtener el número de días que los pacientes han estado en UCI hasta recibir el alta médica. Para ello se obtiene la diferencia de días entre la fecha de la última observación del paciente más dos días (pues en la descripción de los datos se indica que el paciente recibe el alta médica dos días después de la fecha de su última observación) y la fecha entre la primera observación del paciente.

```
m1 = ~survived_patient_vitals_df.duplicated(["NHC"], keep="last")
m2 = survived_patient_vitals_df.duplicated(["NHC"], keep=False)
m = m1 & m2
survived_patient_vitals_df.loc[m, "discharge_date"] =
survived_patient_vitals_df[m]["Dia"] + timedelta(days=2)
```

Figura 4-2 Código que describe cómo obtener la fecha de alta médica de los pacientes

```
fechas_df["discharge_date"] =
pd.DataFrame(survived_patient_vitals_df[survived_patient_vitals_df["dis
charge_date"].notnull()]["discharge_date"]).reset_index().drop("index",
1)
fechas_df["days_of_stay"] = fechas_df["discharge_date"] -
fechas_df["admit_date"]
fechas_df["days_of_stay"] = fechas_df["days_of_stay"].dt.days
```

Figura 4-3 Código que describe cómo obtener los días de estancia en UCI de los pacientes

Debido a que el *dataframe* contiene las distintas observaciones de los pacientes en distintos instantes de tiempo y no se va a trabajar con series temporales, se opta por aplicar un estadístico a los datos relativos a las constantes vitales de estos pacientes, de forma que tras este proceso el *dataframe* contenga una única fila por cada paciente.

Se utiliza la indexación de cadenas para un subconjunto particular de características. Para el grupo de edad, la indexación de cadenas se lleva a cabo debido al hecho de que hay una ordinalidad inherente a las categorías dentro de estas características. Por ejemplo, el grupo de edad «0 a 17» es un grupo de edad más joven que «70 o más». Por ello, tiene sentido codificar el grupo de edad más joven «0 a 17» con un 1, y los grupos de edad más grandes con números enteros cada vez mayores.

```
bins = [0, 4, 14, 20, 29, 39, 49, 59, 69, 79, 89, 120]
labels = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11"]
survived_patients_vitals_length_of_stay_df["age_range"] =
pd.cut(survived_patients_vitals_length_of_stay_df.Edad, bins, labels =
labels, include_lowest = True)
```

Figura 4-4 Código que describe la transformación del atributo edad en etiquetas

La indexación de cadenas en características particulares como esta puede ser extremadamente beneficiosa, debido a que evita el aumento de la dimensionalidad del conjunto de datos y permite que los modelos aprendan la ordinalidad presente en las categorías de características. Sin embargo, es importante tener cuidado con el uso de la indexación de cadenas en las características que no tienen una ordinalidad inherente.

A continuación, siguiendo las recomendaciones recogidas en el *paper* de Baek, H., Cho, M., Kim, S., Hwang, H., Song, M., y Yoo, S. (2018), se estudia más en profundidad la característica de predicción, la duración de la estancia. La duración de la estancia va de 1 a 80 (existiendo algunos casos aislados con un valor superior) y solo toma valores enteros. Tras varias pruebas, se decide tratar la predicción de la duración de la estancia como un problema de clasificación multiclase

(en lugar de regresión). Esto se debe a que predecir el número de días de estancia de manera exacta es muy complejo por dos motivos:

- No se disponen datos para realizar una serie temporal (siendo esto lo más sensato). Es necesario realizarlo con una regresión clásica (lineal - no lineal), que falla mucho.
- Si se analiza la variabilidad de días de estancia en este grupo tan pequeño, es posible observar que es muy grande y que además no sigue una distribución normal, por lo que o bien se sobreajusta a la muestra o bien el resultado será muy malo.

Como se puede observar en la siguiente figura, los datos de la variable que representa los días que cada paciente necesita para recibir el alta presentan un sesgo a la derecha.

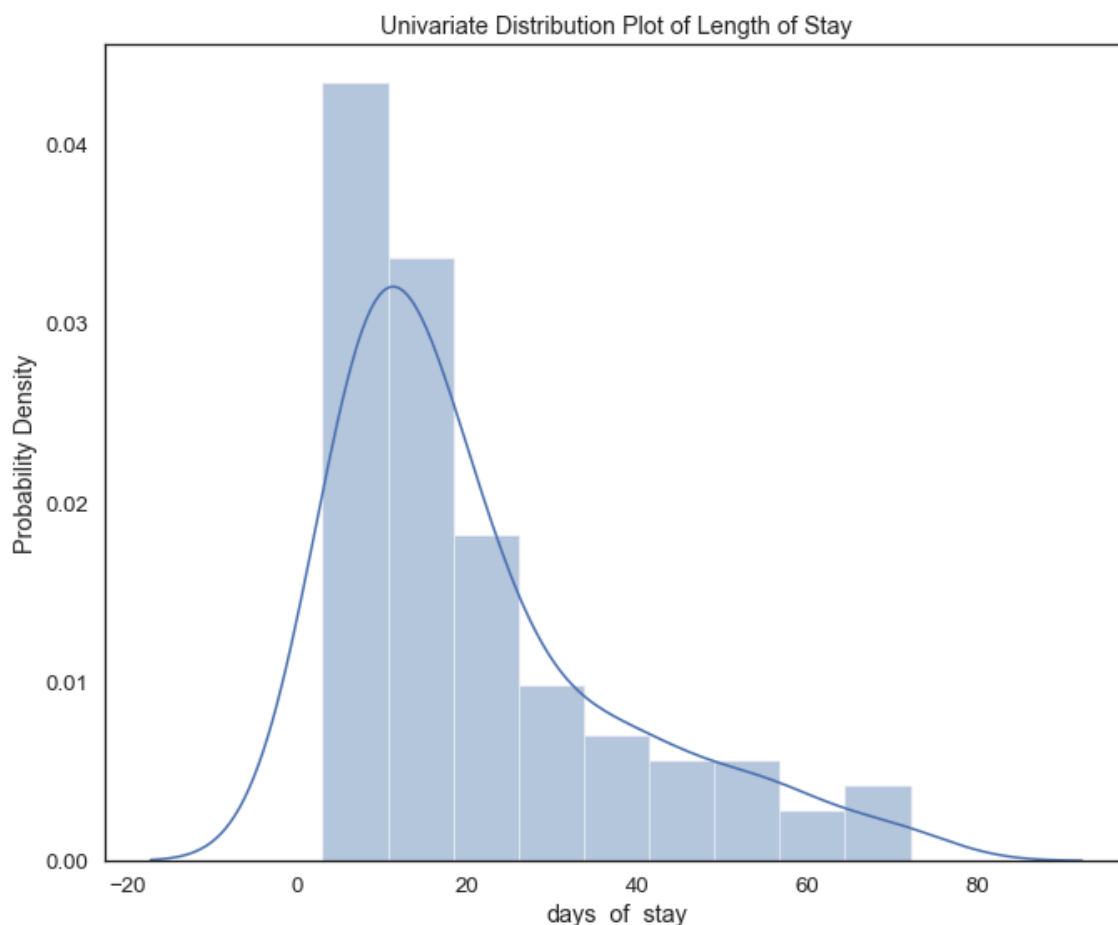


Figura 4-5 Histograma de la distribución de la variable que representa los días hasta el alta

La clasificación multiclase ofrece una gran ventaja en comparación con la regresión empleada en un principio: la posibilidad de definir manualmente los intervalos de clase para controlar manualmente la especificidad de la predicción.

En lugar de tratar la duración de la estancia como si existieran 120 clases diferentes, es posible agrupar estos valores en intervalos que tengan más sentido para las predicciones sin una pérdida significativa en la especificidad de las predicciones.

Tras la exploración de múltiples opciones para estos intervalos, que implicaban un equilibrio entre la utilidad del modelo (si los intervalos son demasiado grandes, las predicciones dejan de ser útiles) y la precisión del modelo, finalmente se optó por el siguiente formato de intervalo: 1–5 días, 6–10 días, 11–20 días, 21–30 días, 31–50 días y 51–120+ días.

```
bins = [0,5,10,20,30,50,120]
labels = [5,10,20,30,50,120]
survived_patients_vitals_length_of_stay_df['stay_bin']=pd.cut(x =
survived_patients_vitals_length_of_stay_df['days_of_stay'],
                        bins = bins)
survived_patients_vitals_length_of_stay_df['stay_label']=pd.cut(x =
survived_patients_vitals_length_of_stay_df['days_of_stay'],
                        bins = bins,
                        labels = labels)
survived_patients_vitals_length_of_stay_df['stay_bin'] =
survived_patients_vitals_length_of_stay_df['stay_bin'].apply(lambda x:
str(x).replace(',',' -'))
survived_patients_vitals_length_of_stay_df['stay_bin'] =
survived_patients_vitals_length_of_stay_df['stay_bin'].apply(lambda x:
str(x).replace('120','120+')) #make this bin more descriptive
display(survived_patients_vitals_length_of_stay_df)
```

Figura 4-6 Código que describe la creación de los rangos de días de estancia en UCI de los pacientes

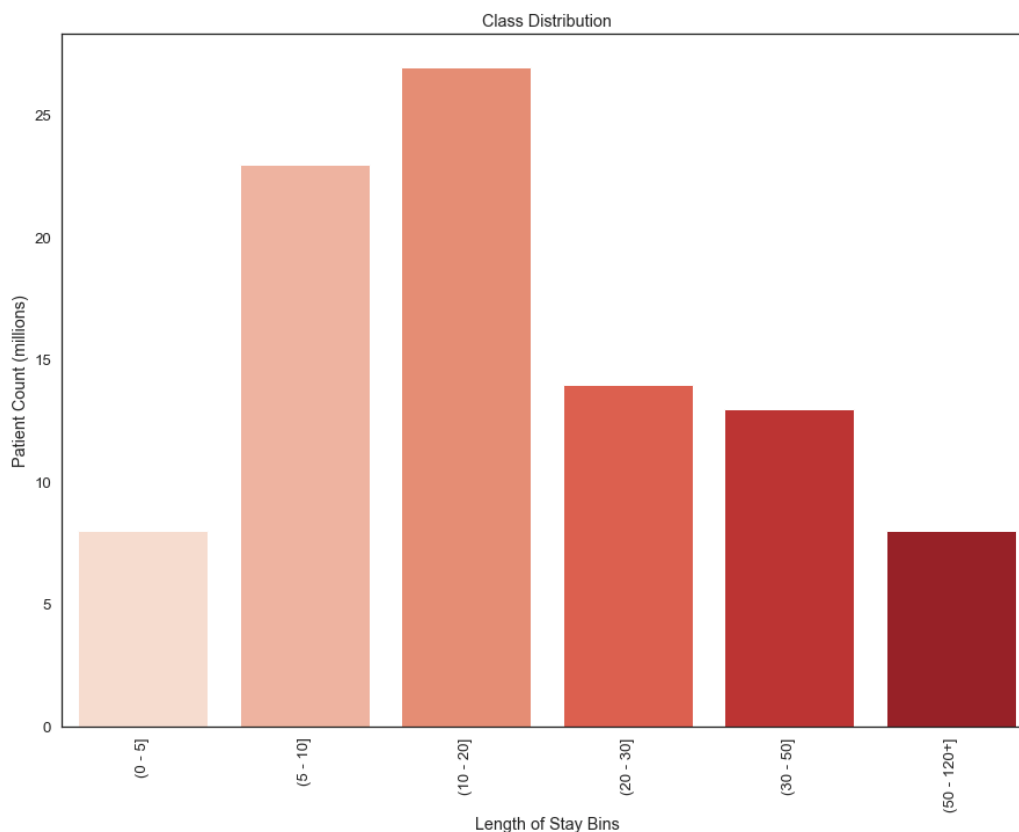


Figura 4-7 Histograma de la distribución de los rangos de días de alta

De esta visualización se desprende un desequilibrio de clases. Este desequilibrio en los datos debe manejarse con cuidado, pues puede conducir a puntuaciones de precisión engañosas. El desequilibrio de clases puede llevar a un modelo a predecir en exceso la clase que aparece con

más frecuencia, debido a que la simple predicción de esta clase para la mayoría de las instancias de datos, independientemente de sus características, conduce a la mayor puntuación de precisión global. Por lo tanto, un modelo que pretenda optimizar la puntuación de precisión puede caer en esta trampa.

4.1.2 Cómo hacer frente al desequilibrio de las clases

El submuestreo de la clase más frecuente y la asignación de penalizaciones por exceso de predicción de clases son dos de los métodos más comunes para tratar el desequilibrio de clases, y ambos se explotan en este conjunto de datos. Finalmente, se descubre que la asignación de penalizaciones es el método más eficaz, porque da lugar a una mayor precisión del modelo que el método de submuestreo, a la vez que evita los riesgos asociados al desequilibrio de clases como cuando se realiza el submuestreo. Para asignar estas penalizaciones a las clases, se hace uso del parámetro "peso de la clase" en las funciones de Machine Learning de Scikit-learn (s.f.), y se establece este parámetro como «equilibrado». Esto asigna efectivamente un peso a cada clase inversamente proporcional a la frecuencia con la que aparece.

```
RandomForestClassifier(n_estimators=100, max_depth=15,  
class_weight='balanced')
```

Figura 4-8 Código que describe cómo solucionar el desequilibrio de las clases

4.1.3 Modelado de Machine Learning

PCA

Antes de entrenar cualquier modelo, se realiza un PCA en los datos después de utilizar la función `StandardScaler()` para normalizar los conjuntos de datos de entrenamiento y de prueba. El PCA es una poderosa herramienta que permite reducir la dimensionalidad de un conjunto de datos, lo que puede ser extremadamente beneficioso para conjuntos de datos con bastantes variables, como el que se está tratando aquí. Además, elimina cualquier multicolinealidad en los datos. Es importante normalizar siempre los datos antes de aplicar el PCA porque este proyecta los datos brutos en direcciones que maximizan la varianza calculando las distancias relativas entre los puntos de una característica. Por lo tanto, cuando las características se encuentran en diferentes escalas, el PCA puede ser rechazado.

A continuación, se traza el porcentaje de varianza explicada frente al número de componentes del PCA que se utilizan. A partir de este gráfico, es posible decidir cuántos componentes utilizar para el resto del análisis con el fin de mantener un determinado porcentaje de varianza explicada en los datos. En este caso, se elige mantener 24 componentes, pues es el número mínimo de componentes necesario para explicar el 95 % de la varianza de los datos.

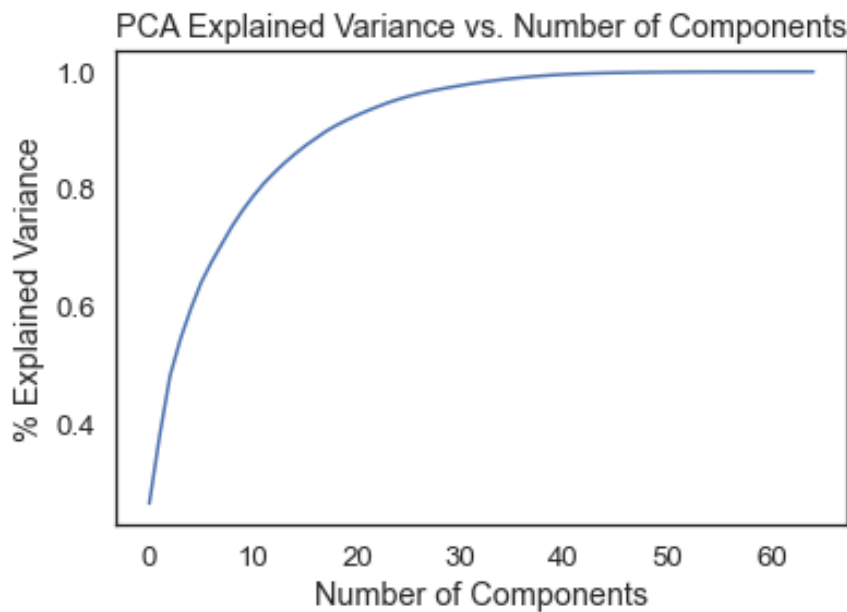


Figura 4-9 Gráfico que representa el porcentaje de varianza explicada frente al número de componentes

Modelos empleados

Finalmente se emplean los algoritmos random forest y AdaBoost, que suelen funcionar muy bien en problemas de clasificación multiclase y son intuitivos de entender. Sin embargo, antes de crear el modelo, se optimizan primero los hiperparámetros. Para ello, se puede realizar una búsqueda aleatoria en un rango de combinaciones de valores de hiperparámetros definidos por un diccionario. A partir de esta búsqueda, es posible devolver los valores óptimos encontrados. Esta búsqueda de hiperparámetros se habilita a través de la función de Scikit-learn (s.f.) «RandomizedSearchCV». En este caso se opta por optimizar los parámetros «max_depth» y «max_leaf_nodes», debido a que son parámetros clave para evitar el sobreajuste, una propiedad a la que los árboles de decisión son especialmente propensos.

El algoritmo Random Forest es el que arroja un mejor desempeño con un valor AUC de 0.78, seguido del algoritmo AdaBoost con un valor AUC de 0.64.

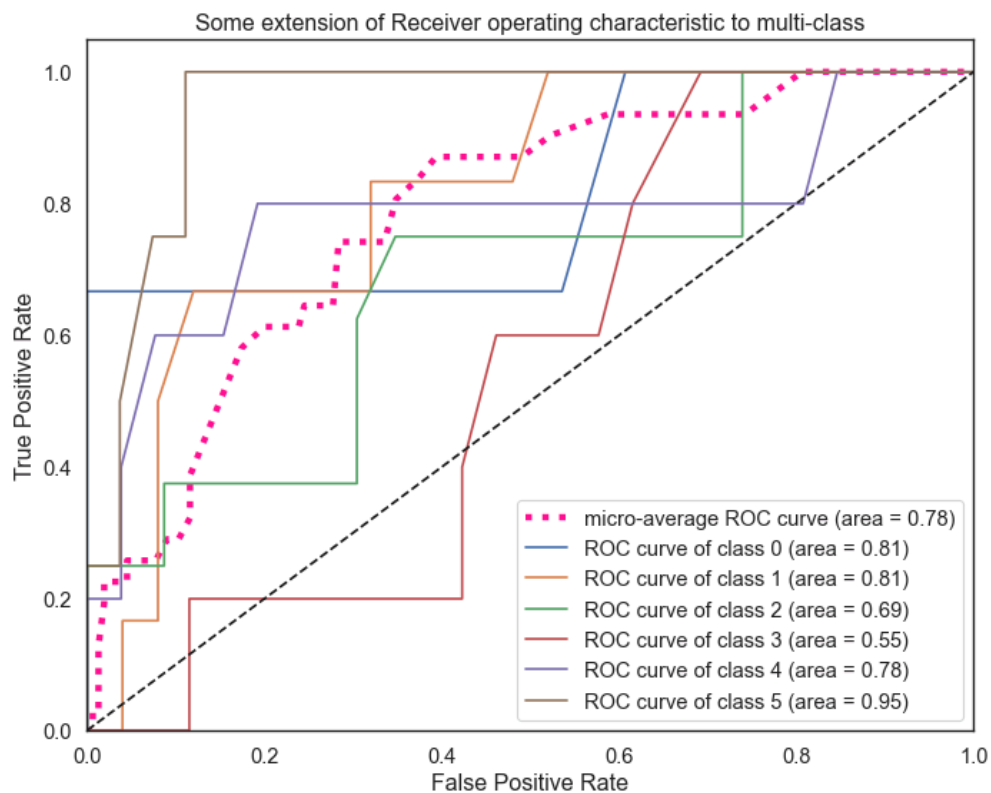


Figura 4-10 Curvas ROC para random forest con sus respectivos valores AUC para las distintas clases

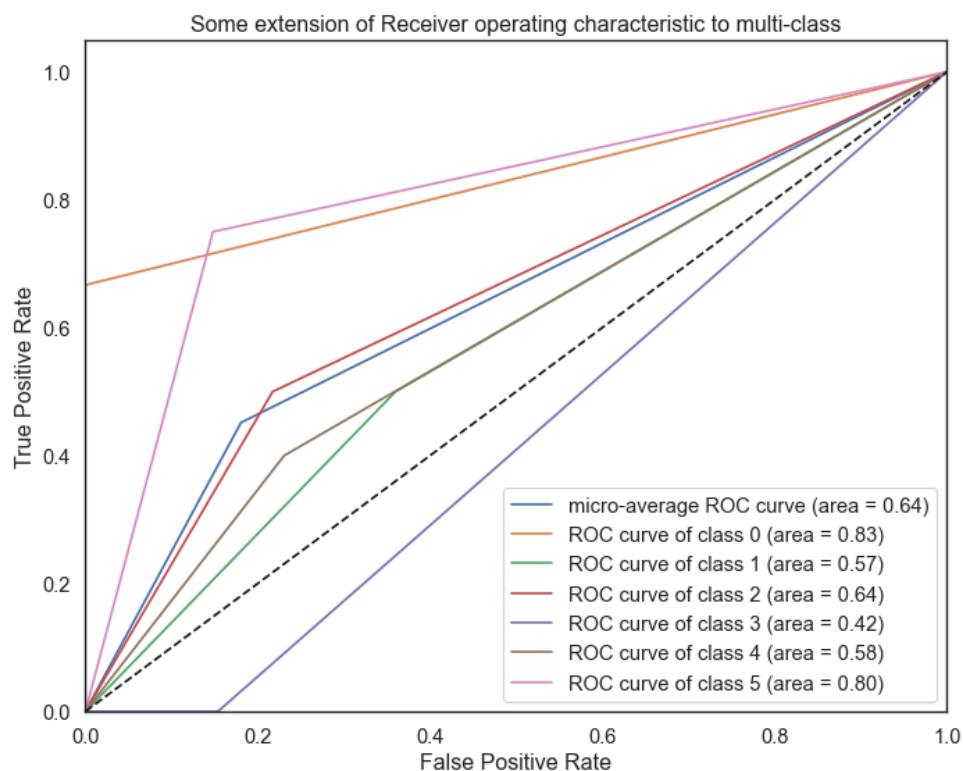


Figura 4-11 Curvas ROC para AdaBoost con sus respectivos valores AUC para las distintas clases

Como se puede observar, la clase 3 es en la que el modelo muestra peor desempeño a la hora de separarla del resto, siendo su valor AUC de un 0.55, lo que significa que el modelo prácticamente no presenta capacidad de discriminación. Sin embargo, como se puede observar, la media de todas las clases (línea rosa punteada) presenta un valor AUC de 0.78 para el modelo que emplea random forest, lo cual es un valor muy bueno para un modelo con tan pocos datos y sin haber aplicado series temporales.

4.1.4 Interpretabilidad del modelo

El uso de los valores SHAP arroja los siguientes resultados en cuanto al impacto medio de las variables en el resultado del modelo para cada una de las clases:

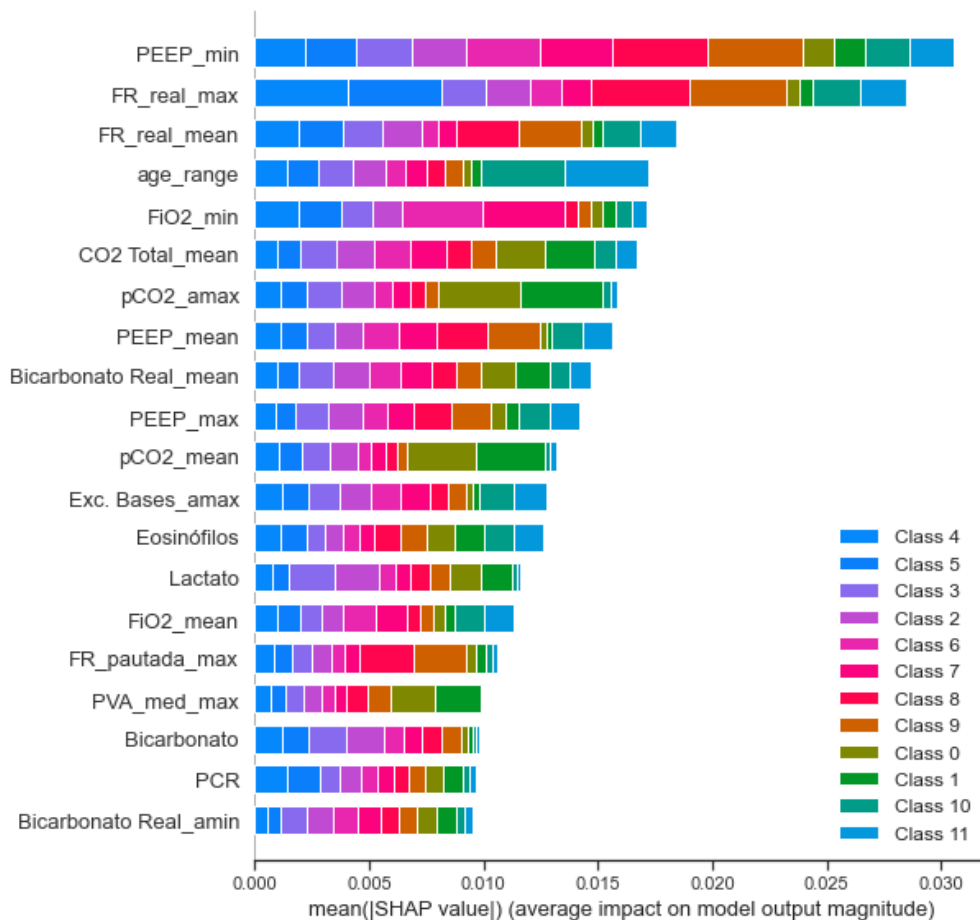


Figura 4-12 Representación gráfica del impacto medio de las distintas variables en el resultado del modelo

Como se puede observar, la variable PEEP_min (presión de Final de Expiración Positiva) es la que impacta en mayor medida a todas las clases. Este hecho confirma que el modelo está realmente realizando unas predicciones sobre las que impactan las mismas variables a las que un médico daría prioridad a la hora de realizar el seguimiento de un paciente que padece SARS-CoV-2, como se puede comprobar en el artículo publicado por Perier, F., Tuffet, S., Maraffi, T., Alcalá, G., Victor, M., Haudebourg, A.F.,...Dessap, A.M. (2020).

4.1.5 Implementación de la pipeline

Una *pipeline* de Machine Learning se utiliza para ayudar a automatizar los flujos de trabajo. Estas funcionan permitiendo transformar una secuencia de datos y correlacionarlos en un modelo que puede probarse y evaluarse para obtener un resultado, ya sea positivo o negativo.

Las *pipelines* de Machine Learning constan de varios pasos para entrenar un modelo. Las cadenas de aprendizaje automático son iterativas, porque cada paso se repite para mejorar continuamente la precisión del modelo y lograr un algoritmo exitoso. Para crear mejores modelos de aprendizaje automático y obtener el máximo valor de ellos, es imprescindible contar con soluciones de almacenamiento accesibles, escalables y duraderas.

Tal y como indican O'Brien, L., Guilley, S., Coulter, D., Lu, P., Martens, J., Brockschmidt, K.,...Bye, T. (2021), el principal objetivo de tener una *pipeline* adecuada para cualquier modelo de Machine Learning es ejercer el control sobre ella. Una *pipeline* bien organizada hace que la implementación sea más flexible. Es como tener una vista desglosada de un ordenador en la que se pueden coger las piezas defectuosas y sustituirlas (en este caso, sustituir una parte del código).

Para poder implementar una *pipeline* personalizada, es necesario implementar y sobrescribir los métodos «*fit*» y «*transform*» de los distintos «*transformers*» empleados en la *pipeline*. Estos *transformers* son empleados para, como su nombre indica, realizar transformaciones sobre los datos que recibe la *pipeline*. Para el problema que Health Twin resuelve, es necesario realizar transformaciones sobre el atributo edad, para los valores NA (valores que faltan) y para normalizar los datos.

```
class AgeToLabelTransformer(BaseEstimator, TransformerMixin):

    def __init__(self, bins_age, labels_age):
        self.bins_age = bins_age
        self.labels_age = labels_age

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy()

        X["age_range"] = pd.cut(X.Edad, self.bins_age, labels =
self.labels_age, include_lowest = True)
        X.drop("Edad", axis=1, inplace=True)

        return X
```

Figura 4-13 Código de la función que describe el funcionamiento de un transformer personalizado

Posteriormente, simplemente es necesario incluir todos los *transformers* creados dentro de un objeto «Pipeline».

```
stay_range_pipe = Pipeline(steps=[
    ("ImputeNATransformer", ImputeNATransformer(imputer=imputer)),
    ("AgeToLabelTransformer", AgeToLabelTransformer()),
    ("CustomMinMaxScaler", CustomMinMaxScaler(scaler=scaler))
])
```

Figura 4-14 Código que describe la creación de la pipeline

Una vez que la *pipeline* ha sido creada, su método «*transform*» permite realizar las transformaciones necesarias sobre los datos para poder llevar a cabo la predicción.

Como se puede observar, en ocasiones es necesario pasar a los *transformers* algunos parámetros que han sido previamente entrenados sobre el conjunto de datos.

4.2 Implementación del sistema en AWS

Una vez que toda la parte relativa al modelo de Machine Learning que predecirá el rango de días hasta el alta del paciente ha sido finalizada, el siguiente paso es implementar el sistema con los servicios de AWS, siguiendo las recomendaciones de la documentación de Amazon Web Services (2019).

4.2.1 Implementación del API Gateway

Los *endpoints* de los recursos de la aplicación son creados en el fichero YAML en el momento en el que se crean las funciones Lambda de comandos y consultas.

La API y sus *endpoints* pueden ser consultados en la siguiente documentación, que ha sido creada con Swagger (s.f.), haciendo uso de OpenAPI (s.f.): https://app.swaggerhub.com/apis-docs/gon99martin/HT_API_OAS3.0/1.0.0-/

Responses		
Code	Description	Links
200	Éxito en la obtención de la información del estado de los pacientes.	No links
<div>Media Type</div> <div>application/json</div> <div>Controls Accept Header</div> <div>Example Value Schema</div> <div><pre>{ "id": "4703d0a0-ca32-11eb-bb0d-a59243dcc879", "nhc": 5343, "icu_admission_date": "2020-04-17", "age": 57, "sex": 1, "height": 176, "weight": 74, "alt_gpt": 962, "ast_got": 259, "bshapils": 0.2, "beecf_amax": 6, "beecf_amin": 3, "beecf_mean": 0.33, "bicarb": 3.07, "bicarb_real_amax": 30, "bicarb_real_amin": 24, "bicarb_real_mean": 26, "bilineubin_t": 0.8, "chlorine": 95, "ck": 149, }</pre></div> <div>SXX</div> <div>Error inesperado.</div> <div>No links</div>		

Figura 4-15 Ejemplo que describe el formato de la documentación de la API de Health Twin creada

4.2.2 Implementación de las funciones Lambda

Se han creado dos tipos de funciones Lambda, las relativas a comandos y las relativas a las consultas.

Todas las funciones Lambda deben ser creadas para poder trabajar con ellas, no es suficiente con programarlas y hacer el *deploy* a AWS, porque esto provoca un error. Para crearlas, se hace uso de las siguientes sentencias en YAML:

```
functions:
  commandCreatePatient:
    handler: commands.createPatient
    events:
      - http: 'POST /patients'
  queryGetPatients:
    handler: queries.getPatients
    events:
      - http: 'GET /patients'
  queryGetPatient:
    handler: queries.getPatient
    events:
      - http: 'GET /patients/{id}'
  communicationSaveEventCreatePatient:
    handler: communications.saveEventCreatePatient
    events:
      - sns:
          topicName: create-patient
          displayName: "Patient create events"
  communicationSaveStatePatient:
    handler: communications.saveStatePatient
    events:
      - sns:
          topicName: create-patient
          displayName: "Patient create events"
  communicationSaveStatePredictedPatient:
    handler: communications.saveStatePredictedPatient
```

Figura 4-16 Anotaciones YAML que describen la configuración de las funciones Lambda

Lambdas comandos

La función Lambda Command CreatePatient permite dar de alta pacientes. Crea un evento «create-patient» que publicará el servicio SNS y que permitirá a otros servicios suscribirse a él. Esta función es activada mediante un *endpoint* POST.

```
const createPatientEvent = {
  type: 'create-patient', timestamp: new Date().getTime(),
  id: uuid.v1();, payload: { . . . } };
```

Figura 4-17 Código de la función que crea el evento con la información del paciente

```
bus.publish(createPatientEvent,
  msg => {
    callback(null, {
      statusCode: 200,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(createPatientEvent.payload)
    })
  },
  err => {
    callback(null, {
      statusCode: 500,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({"error": err})
    })
  }
);
```

Figura 4-18 Código de la función que emite el evento a SNS con la información del paciente

El método «bus» ha sido creado para poder publicar el evento en el servicio SNS. Su código es el siguiente:

```
const createSnsConnection = () => IS_OFFLINE ? new
AWS.SNS({ endpoint: "http://127.0.0.1:4002", region: REGION }) :
  new AWS.SNS();

const sns = createSnsConnection();

const buildTopicArn = (topicName) => {
  const accountId = IS_OFFLINE ? OFFLINE_AWS_ACCOUNT_ID :
  AWS_ACCOUNT_ID;
  return `arn:aws:sns:${REGION}:${accountId}:${topicName}`;
};

const createEventBusClient = () => {
  return {
    publish: (event, success, failure) => {
      let message = compose(JSON.stringify, assoc("default",
"json"))(event);
      sns.publish({
        Message: message, TopicArn: buildTopicArn(event.type)
      }, error =>
        error ? failure(error) : success(message)
      );
    }
  };
};
```

Figura 4-19 Código que describe cómo SNS recibe información y la convierte en un evento

Lambda Save Event create-patient guarda el evento generado previamente en la base de datos de eventos para poder disponer de un histórico de los mismos. El lanzamiento de un evento create-patient provoca la activación de esta función.

```
module.exports.saveEventCreatePatient = (message, context, callback) =>
{
  console.log("saveEventCreatePatient "+JSON.stringify(message));
  const event = parseEvent(message);

  if (event.type !== "create-patient") return;

  db.save(process.env.EVENTS_PATIENTS_TABLE, event,
    entry => callback(null, entry),
    error => callback(error));
};
```

Figura 4-20 Código de la función que guarda el evento del paciente en la base de datos de eventos

Para guardar el estado actual de cada paciente se hace uso de la función Lambda Save State Predicted Patient, que es invocada desde el *script* contenido en el servicio EC2. Esta función simplemente recibe un *payload* generado por el *script* (en el que ya se incluye la predicción del rango de los días para recibir alta) en Python y lo almacena en la base de datos de estados.

```
module.exports.saveStatePredictedPatient = (message, context, callback)
=> {
  console.log("saveStatePatient "+JSON.stringify(message));

  const event = message

  let patient = event.payload;

  db.save(process.env.STATE_PATIENTS_TABLE, patient,
    entry => callback(null, entry),
    error => callback(error));
};
```

Figura 4-21 Código de la función que guarda el estado del paciente en la base de datos de estados

Como se puede observar, al guardar el estado del paciente no es necesario comprobar el tipo de evento, debido a que esta función necesita ser invocada por el *script* en Python, que previamente ha recibido datos procedentes de un evento de tipo create-patient.

Para que esta función pueda ser invocada por el *script*, es necesario asignarle permisos de invocación de la siguiente forma:

```
AllowInvokeSavePatientStateLambda:
  Type: AWS::Lambda::Permission
  Properties:
    Action: lambda:InvokeFunction
    FunctionName: "arn:aws:lambda:eu-west-3:${ssm:accountId}:function:sless-cqrs-patients-dev-communicationSaveStatePredictedPatient"
    Principal: ${ssm:accountId}
```

Figura 4-22 Anotaciones YAML que describen cómo otorgar permisos de invocación a una función Lambda

Lambdas consultas

Se ha creado únicamente una función lambda de comandos, Lambda Query `getPatients`, que permite, mediante un *endpoint* GET, consultar el estado actual de los pacientes.

```
module.exports.getPatients = (event, context, callback) => {
  db.all(process.env.STATE_PATIENTS_TABLE,
    res => callback(null, {
      statusCode: 200,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(res),
    }),
    err => callback(err)
  );
};
```

Figura 4-23 Código de la función que permite consultar el estado de los pacientes

```
db.find(process.env.STATE_PATIENTS_TABLE, event.pathParameters.id, ...
```

Figura 4-24 Código de la función que permite consultar el estado de un paciente en concreto

4.2.3 Implementación del SNS

Implementar el servicio SNS es muy sencillo con las anotaciones YAML. Simplemente es necesario crear un recurso, indicando el nombre que se le quiere dar al servicio SNS.

```
Resource:
  - "arn:aws:sns:eu-west-3:${ssm:accountId}:create-patient"
```

Figura 4-25 Anotaciones YAML que describen la creación del servicio SNS

4.2.4 Implementación de la cola SQS

Como se explicó anteriormente, el uso de una cola SQS tiene sentido en este problema. Sin embargo, no es una práctica común en cuanto a la implementación de un patrón CQRS, sino que se trata, por lo tanto, de un sistema más complejo. Sin embargo, consultando la documentación ofrecida por Andrews, P. (2020), se ha implementado una cola SQS que logra su objetivo de forma totalmente satisfactoria.

Para poder implementar la cola SQS que comunica el servicio SNS con EC2, es necesario asignar una serie de permisos además de crearla.

```
# Definición de la cola SQS que conecta SNS con Elastic Beanstalk
CreatePatientQueue:
  Type: "AWS::SQS::Queue"
  Properties:
    QueueName: sqs-queue-create-patient
# Definición de la suscripción que permite a la cola SQS recibir
eventos de SNS
Subscription:
  Type: 'AWS::SNS::Subscription'
  Properties:
    TopicArn: "arn:aws:sns:eu-west-3:${ssm:accountId}:create-
patient"
    Endpoint: !GetAtt
      - CreatePatientQueue
      - Arn
    Protocol: sqs
    RawMessageDelivery: 'true'
```

Figura 4-26 Anotaciones YAML que describen cómo crear una cola SQS y suscribirla al servicio SNS

```
# Asignación de permisos a la conexión SNS -> SQS
AllowSNS2SQSPolicy:
  Type: AWS::SQS::QueuePolicy
  Properties:
    Queues: [!Ref 'CreatePatientQueue']
    PolicyDocument:
      Version: '2008-10-17'
      Id: PublicationPolicy
      Statement:
        - Sid: Allow-SNS-SendMessage
          Effect: Allow
          Principal:
            AWS: '*'
          Action: ['sqs:SendMessage']
          Resource: !GetAtt [CreatePatientQueue, Arn]
          Condition:
            ArnEquals:
              aws:SourceArn: "arn:aws:sns:eu-west-
3:${ssm:accountId}:create-patient"
```

Figura 4-27 Anotaciones YAML que describen cómo otorgar permisos a la conexión entre SNS y SQS

4.2.5 Implementación de las base de datos DynamoDB

Se implementan dos bases de datos, una para los eventos que permiten la creación de los pacientes y otra para los estados actuales de los pacientes.

```
EventsPatientsDynamoDBTable:
  Type: 'AWS::DynamoDB::Table'
  Properties:
    TableName: '${self:provider.environment.EVENTS_PATIENTS_TABLE}'
    AttributeDefinitions:
      -
        AttributeName: id
        AttributeType: S
    KeySchema:
      -
        AttributeName: id
        KeyType: HASH
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
```

Figura 4-28 Anotaciones YAML que describen cómo configurar las base de datos DynamoDB de eventos

```
StatePatientsDynamoDBTable:
  Type: 'AWS::DynamoDB::Table'
  Properties:
    TableName: '${self:provider.environment.STATE_PATIENTS_TABLE}'
    AttributeDefinitions:
      -
        AttributeName: id
        AttributeType: S
    KeySchema:
      -
        AttributeName: id
        KeyType: HASH
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
```

Figura 4-29 Anotaciones YAML que describen cómo configurar las base de datos DynamoDB de estados

4.3 Desarrollo de las aplicaciones Electron

El programa presenta dos aplicaciones Electron que permiten al personal sanitario trabajar de forma sencilla con los servicios proporcionados, debido a que todo este proyecto sería prácticamente inutilizable si no se les proporcionara una interfaz con la que puedan trabajar apropiadamente.

Las aplicaciones han sido desarrolladas haciendo uso del *framework* Electron porque ofrece varias ventajas: es fácil desarrollar aplicaciones Electron empleando conocimientos del desarrollo web, son aplicaciones de muy fácil distribución y no ofrecen problemas de compatibilidad (Fratila, C., 2018). La principal desventaja de este *framework* es que las aplicaciones son bastante pesadas (Fratila, C., 2018). Debido a las ventajas que ofrece y a que el

tamaño de las aplicaciones no es muy grande, el *framework* Electron es una opción perfecta para Health Twin.

El desarrollo de estas aplicaciones cumple con todos los requisitos de experiencia de usuario con el objetivo de ofrecer una solución totalmente intuitiva y que no genere dudas al personal sanitario sobre su uso.

La primera de estas aplicaciones, Health Twin Admission, permite al personal sanitario introducir los datos de los pacientes (uno por uno o varios al mismo tiempo mediante un fichero CSV) y, de esta forma, generar los eventos que desencadenarán el funcionamiento de la aplicación.

La primera pantalla de esta aplicación permite al personal sanitario registrarse o iniciar sesión para comenzar a introducir información relativa de los pacientes.

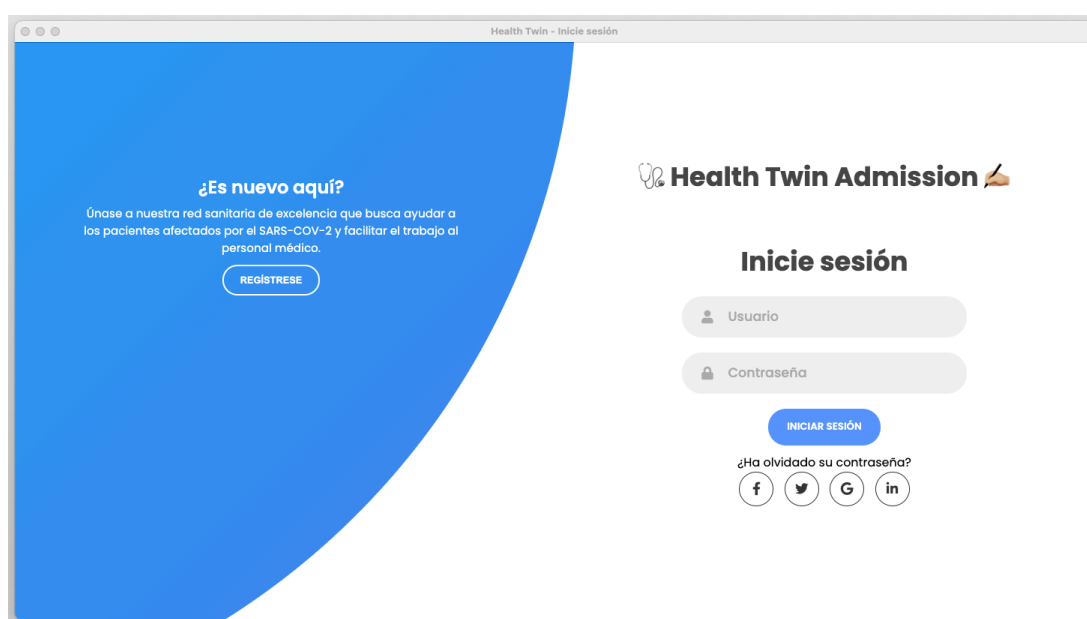


Figura 4-30 Pantalla de inicio de sesión de la aplicación Health Twin Admission

Una vez se inicia sesión, la primera pantalla de la aplicación permite seleccionar el método de introducción de los datos.

Si se elige introducir los datos manualmente, se creará un nuevo paciente tras introducir todos los datos solicitados en el formulario. Este formulario consta de dos pantallas. La primera de ellas permite introducir los datos generales del paciente y en la segunda pantalla se introducen su constantes vitales y otras mediciones.



Figura 4-31 Pantalla de selección del método de introducción de los datos de los pacientes

Health Twin - Patient Admission

←

Formulario de admisión del paciente en UCI

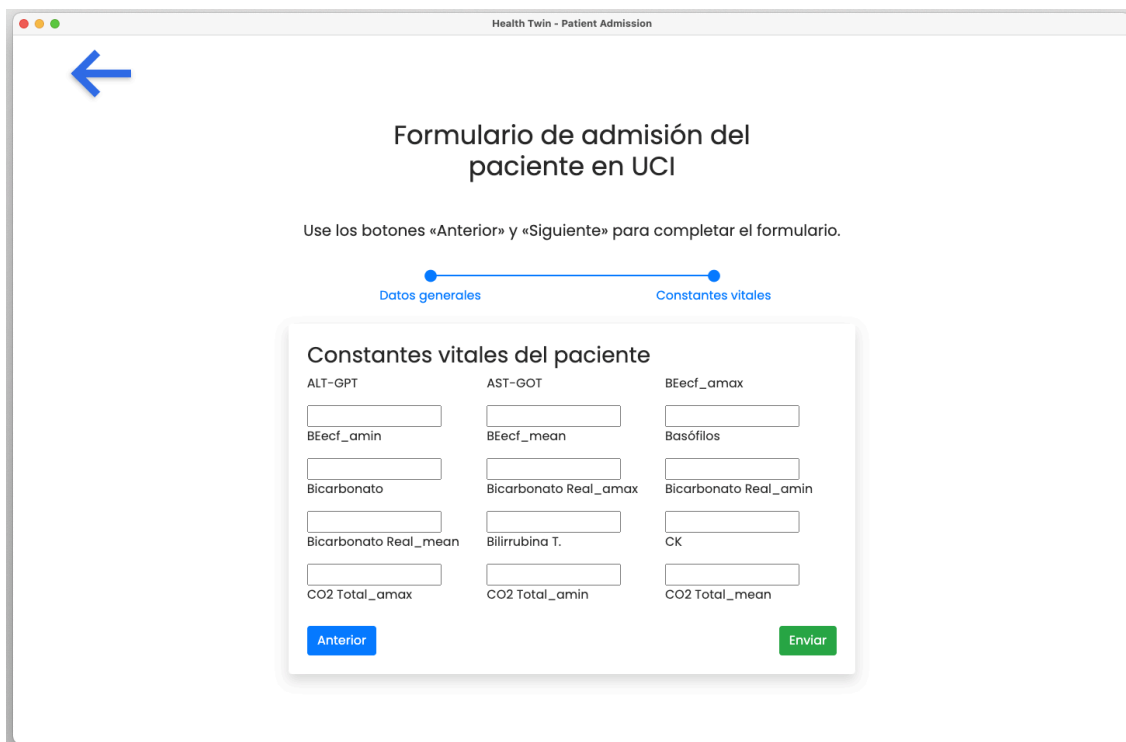
Use los botones «Anterior» y «Siguiente» para completar el formulario.

Datos generales Constantes vitales

Datos generales del paciente

NHC	Fecha de ingreso en UCI
<input type="text"/>	<input type="text" value="dd/mm/yyyy"/>
Edad	Sexo del paciente
<input type="text"/>	<input type="button" value="Hombre"/> <input type="button" value="Mujer"/>
Altura (cm)	Peso (kg)
<input type="text" value="ej: 176"/>	<input type="text" value="ej: 70"/>

Figura 4-32 Pantalla de la primera parte del formulario de introducción de datos del paciente



Health Twin - Patient Admission

←

Formulario de admisión del paciente en UCI

Use los botones «Anterior» y «Siguiente» para completar el formulario.

Datos generales Constantes vitales

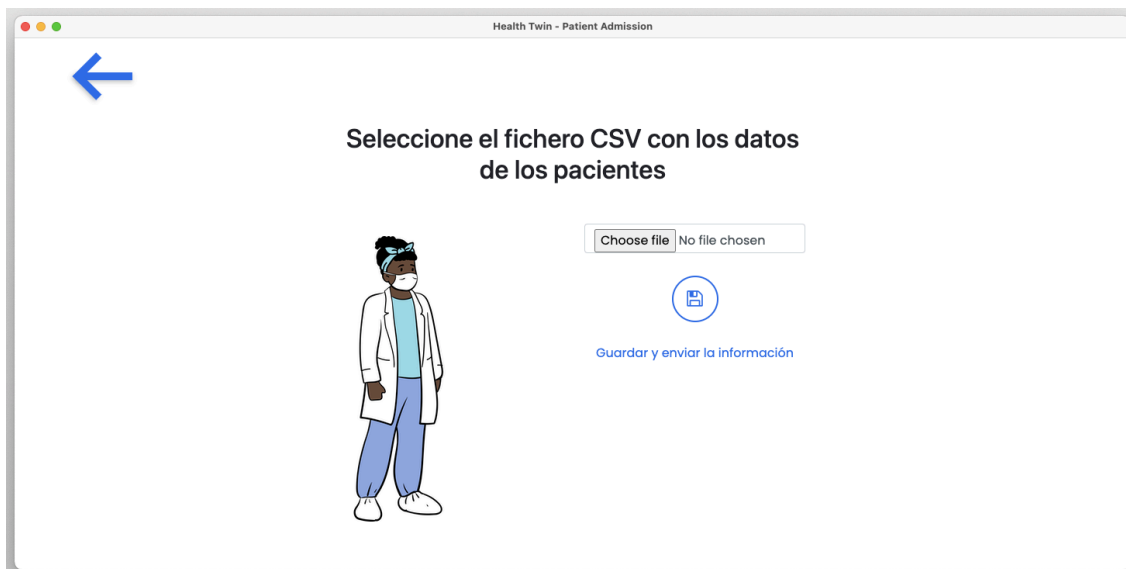
Constantes vitales del paciente

ALT-GPT	AST-GOT	BEcf_amax
<input type="text"/>	<input type="text"/>	<input type="text"/>
BEcf_amin	BEcf_mean	Basófilos
<input type="text"/>	<input type="text"/>	<input type="text"/>
Bicarbonato	Bicarbonato Real_amax	Bicarbonato Real_amin
<input type="text"/>	<input type="text"/>	<input type="text"/>
Bicarbonato Real_mean	Bilirrubina T.	CK
<input type="text"/>	<input type="text"/>	<input type="text"/>
CO2 Total_amax	CO2 Total_amin	CO2 Total_mean
<input type="text"/>	<input type="text"/>	<input type="text"/>

Anterior Enviar

Figura 4-33 Pantalla de la segunda parte del formulario de introducción de datos del paciente


Si por el contrario se elige introducir los datos de los pacientes mediante un fichero CSV, simplemente será necesario seleccionar el fichero que contenga estos datos y la aplicación se encargará del resto.




Health Twin - Patient Admission

←

Seleccione el fichero CSV con los datos de los pacientes



Choose file No file chosen



Guardar y enviar la información

Figura 4-34 Pantalla de introducción de datos de los pacientes a través de un fichero CSV

En cuanto a la aplicación que emplean los médicos para comprobar el estado de los pacientes, Health Twin, existe un página de inicio de sesión y una vez se inicia sesión se puede visualizar el listado del estado de los pacientes en UCI.

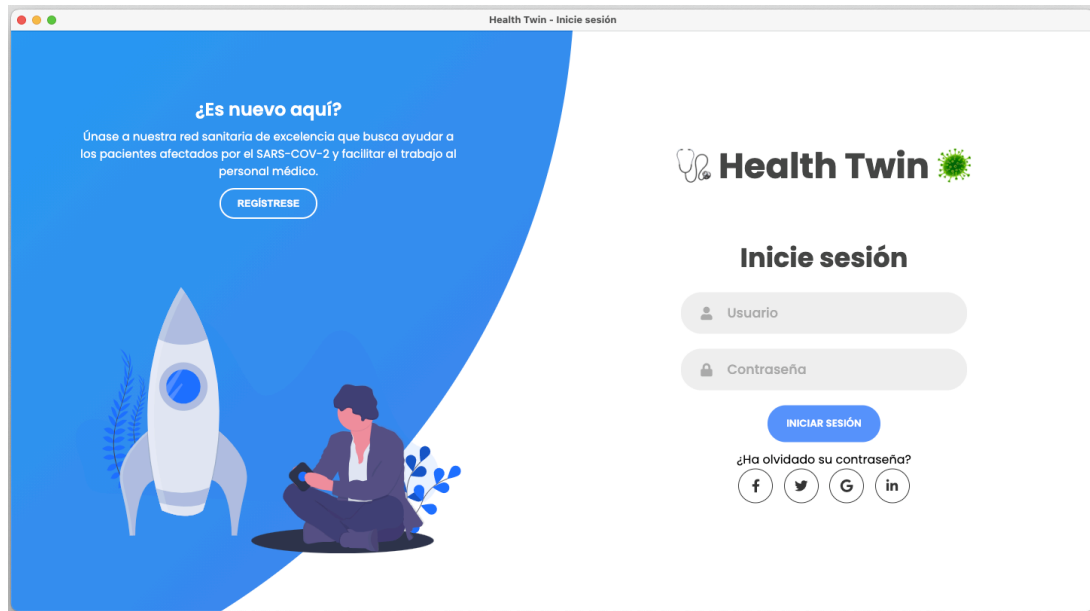


Figura 4-35 Pantalla de inicio de sesión de la aplicación Health Twin



Figura 4-36 Pantalla en la que se visualiza el listado de todos los pacientes

Como se puede observar en la figura anterior, no siempre todas las variables tienen que tener un valor, debido a que en muchas ocasiones variables como la altura o mediciones complejas no se toman porque quizás se tuvo que ingresar al paciente de forma rápida, el hospital estaba

colapsado, etc. Es en situaciones como estas en las que se puede ver las ventajas de emplear una base de datos NoSQL, como es DynamoDB, pues no existe un esquema fijo de datos.

Las siguientes figuras muestran el código responsable de realizar las peticiones POST y GET de las aplicaciones.

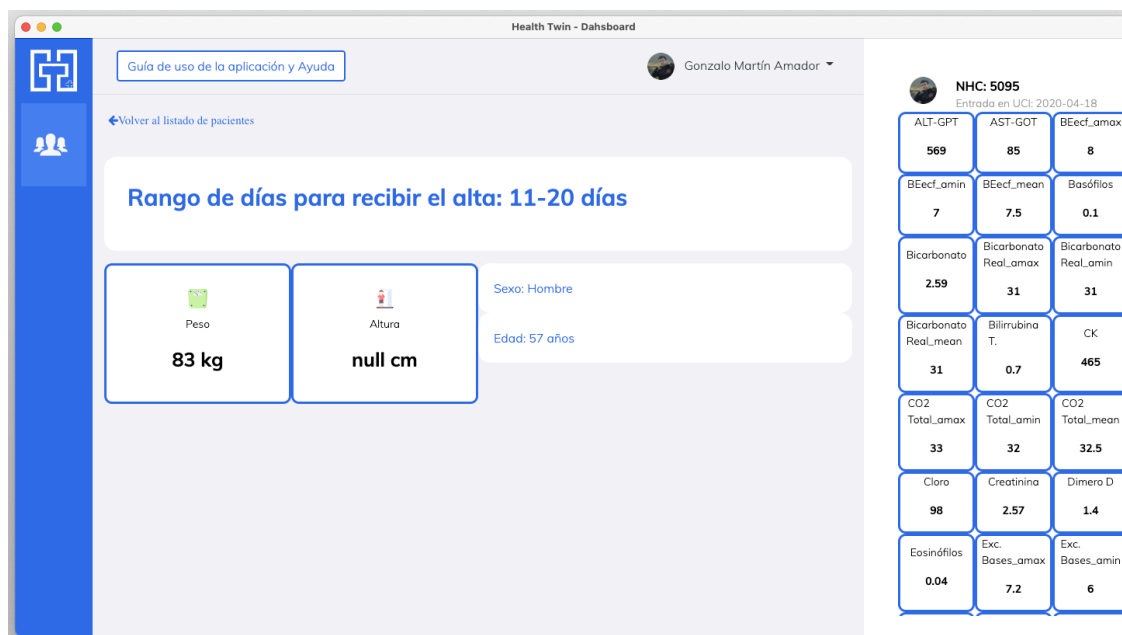


Figura 4-37 Pantalla en la que se visualiza la información relativa a un paciente en concreto

También se llevan a cabo pruebas de usabilidad con personal sanitario real que pueden ser consultadas en el siguiente enlace:

https://drive.google.com/drive/folders/1r8zQANfTD_0scXB_9hW8Vt6OpD9lwPf?usp=sharing

```
fetch('https://ivqcop773j.execute-api.eu-west-3.amazonaws.com/dev/patients', {
  method: 'POST',
  body: patientsArrayJson,
  headers: { 'Content-type': 'application/json; charset=UTF-8' }
}).then(function (response) {
  if (response.ok) {
    return response.json();
  }
  return Promise.reject(response);
}).then(function (data) {
  console.log(data);
}).catch(function (error) {
  console.warn(error);
});
```

Figura 4-38 Código de la petición POST que permite introducir nuevos pacientes en las bases de datos

```
fetch('https://ivqcop773j.execute-api.eu-west-3.amazonaws.com/dev/patients', {
  method: 'GET',
  headers: {
    "Authorization": ". . ."
  }
}).then(function (response) {
  if (response.ok) {
    return response.json();
  }
  return Promise.reject(response);
}).then(function (data) {
  console.log(data);
}).catch(function (error) {
  console.warn(error);
});
```

Figura 4-39 Código de la petición GET que permite recuperar los datos del estado de los pacientes

```
fetch('https://ivqcop773j.execute-api.eu-west-3.amazonaws.com/dev/patients/'+patientId, {
  method: 'GET',
  headers: {
    "Authorization": ". . ."
  }
}).then(function (response) {
  if (response.ok) {
    return response.json();
  }
  return Promise.reject(response);
}).then(function (data) {
  console.log(data);
  return data;
}).catch(function (error) {
  console.warn(error);
});
```

Figura 4-40 Código de la petición GET que permite recuperar los datos del estado de un paciente en concreto

5. Resultados obtenidos y conclusiones

5.1 Resultados

Se ha obtenido un modelo con un valor AUC de 0.78, es decir, hay un 78 % de posibilidades de que el modelo sea capaz de distinguir entre las distintas clases. Por lo tanto, se puede afirmar que el modelo es capaz de separar las clases con un gran desempeño. Es necesario destacar que el modelo presenta sobreajuste, debido a que la cantidad de datos empleada es muy pequeña para un problema de estas características.

Las aplicaciones Electron permiten satisfacer las necesidades del personal médico, pues no solo son mucho más sencillas de usar que una línea de comandos, sino que de por sí son unas aplicaciones que cumplen los requisitos de experiencia de usuario, usabilidad e interacción persona-ordenador necesarios para lograr que los usuarios sean capaces de emplearlas sin incurrir en dudas ni frustración durante su trabajo.

Los objetivos de estas aplicaciones eran permitir la escritura de nuevos pacientes que entraran en UCI y la visualización del estado de los mismos junto a la predicción del rango de días hasta su alta. Ambos han sido cumplidos satisfactoriamente.

5.2 Conclusiones

5.2.1 Conclusiones generales

Tras la realización de este Trabajo de Fin de Grado se ha logrado entender perfectamente el proceso de investigación y desarrollo de una solución informática a un problema real.

El principal problema con el que ha sido necesario lidiar es el hecho de disponer de tan pocos datos y no perfectamente formateados o, al menos, de una forma que permita su fácil manipulación. Además, todos los autores de los *papers* relacionados con el SARS-CoV-2 y soluciones en las que se aplique Machine Learning son muy reacios a compartir resultados, datos o cualquier tipo de ayuda que pueda ser empleada por otra persona. Se entiende que esto se debe a que hay mucha gente que ha obtenido o está en proceso de obtener propiedades intelectuales relacionadas con soluciones informáticas para el problema de la COVID-19, por lo que es muy probable que no compartan los datos ni los algoritmos empleados hasta que pase un tiempo después del fin de la pandemia.

Se ha logrado ver el potencial de hacer uso de las metodologías ágiles, pues a mitad de la investigación y desarrollo del modelo de Machine Learning fue necesario cambiar el planteamiento del problema, debido a que se observó que la mejor aproximación para resolverlo no era planteando un problema de regresión lineal, sino uno de clasificación multiclase. Gracias a la flexibilidad y versatilidad de las metodologías ágiles, este cambio pudo realizarse rápidamente y reutilizando la mayor parte de la investigación y del código ya implementados.

Además, el hecho de trabajar con datos de pacientes reales ha permitido mejorar notoriamente las habilidades relacionadas con el preprocesamiento de los datos, siendo esta una habilidad de vital importancia para la correcta creación de modelos de Machine Learning.

También vale la pena destacar la gran cantidad de conocimientos adquiridos gracias a la decisión de emplear los servicios de Amazon Web Services, pues no solo se ha logrado entender perfectamente cómo funciona una arquitectura *cloud* y *serverless*, sino que se han adquirido conocimientos en lo que a día de hoy es la herramienta más importante en cuanto a lo que computación en la nube se refiere.

Gracias a la complejidad del problema que se ha resuelto, ha sido posible aprender patrones de arquitectura antes desconocidos, como el patrón CQRS, teniendo este un increíble potencial y una excelente riqueza formativa, pues ha permitido reforzar los conocimientos relativos a la arquitectura de software (y en general la arquitectura de sistemas) y adquirir una gran cantidad de nuevos conocimientos, como por ejemplo los relacionados con el *event sourcing*, la persistencia de los datos o la consistencia de los mismos, entre otros.

En definitiva, este Trabajo de Fin de Grado ha logrado cumplir los objetivos propuestos en un comienzo, es decir, desarrollar una aplicación que facilite el trabajo del personal sanitario en su lucha contra el SARS-CoV-2 de forma que los pacientes puedan recibir una atención más cercana y personalizada. También se ha logrado desarrollar un modelo de Machine Learning que arroja unos resultados más que satisfactorios teniendo en cuenta los pocos datos de los que se dispone. Y por último, pero no menos importante, todo este sistema está integrado en una arquitectura *cloud* y *serverless* pensada al detalle, tolerante a fallos y con una enorme escalabilidad.

5.2.2 Los riesgos éticos que se esconden tras los algoritmos

La frecuencia con la que los algoritmos de Machine Learning son usados en el campo de la medicina varía en función de los distintos entornos, pero cada vez se utilizan más para la atención clínica, como la lectura de radiografías e imágenes para diagnosticar afecciones como enfermedades oculares o cáncer de piel. También se utilizan desde el punto de vista empresarial para analizar los historiales médicos y las reclamaciones de seguros con el fin de aumentar la eficiencia de la organización y reducir los costes.

Muchas veces, la gente no es consciente de cómo funcionan este tipo de algoritmos ni de cómo son usados para determinar su cuidado.

Este hecho puede no parecer un problema a simple vista, sin embargo, los algoritmos de Machine Learning pueden conducir en ocasiones a desigualdad en la atención sanitaria.

Por ejemplo, tal y como indica en su artículo Nordling, L. (2019), en la Universidad de Medicina de Chicago hay un excelente grupo de análisis de datos, y una de las cosas que hacen es crear algoritmos para analizar los datos de los registros clínicos electrónicos. Uno de los proyectos en los que trabajaron es ayudar a reducir la duración de la estancia de los pacientes. La idea es que si es posible identificar a los pacientes con más probabilidades de ser dados de

alta antes, se les puede asignar un gestor de casos para asegurar que no haya más bloqueos o barreras que puedan impedirles abandonar el hospital a tiempo.

El grupo de análisis de datos desarrolló inicialmente los algoritmos basándose en datos clínicos, y luego descubrió que añadir el código postal donde vive el paciente mejoraba la precisión del modelo para identificar a las personas que tendrían una estancia más corta. El problema es que cuando se añade el código postal, si se vive en un barrio pobre o en un barrio predominantemente afroamericano, es más probable que la estancia sea más larga. Así, el algoritmo habría conducido al resultado paradójico de que el hospital proporcionara recursos adicionales de gestión de casos a una población predominantemente blanca, más educada y más acomodada, para que salieran antes del hospital, en lugar de a una población de mayor riesgo social que realmente debería ser la que recibiera más ayuda.

Así que, con ese ejemplo local, queda claro que no se trata únicamente de una cuestión abstracta y teórica. Está sucediendo aquí y probablemente en muchos otros lugares.

Debido a casos como este, es necesario discutir clara y explícitamente las implicaciones éticas de cómo el software puede analizar y utilizar los datos con el fin de crear modelos de Machine Learning.

Nordling, L. (2019) indica que se puede pensar en tres factores que causan los problemas. Uno son los propios datos, el segundo son los algoritmos, y el tercero es cómo se utilizan los algoritmos.

Hay varias opciones para tratar de solucionar este problema. En primer lugar, los datos pueden ser un problema. ¿Se está trabajando con conjuntos de datos válidos? ¿Se tienen datos incompletos de la población de riesgo? ¿Se están utilizando datos basados en diagnósticos y etiquetas erróneas?

El segundo paso es examinar cómo se desarrollan realmente los modelos. En este caso, hay formas técnicas de diseñar los algoritmos para promover principios específicos de justicia ética. Se pueden crear algoritmos que garanticen la igualdad de resultados entre dos poblaciones, y se puede asegurar que el rendimiento técnico del modelo sea justo. Así, si hay un problema en el que los algoritmos no diagnostican a los afroamericanos por alguna condición, se pueden alterar los parámetros de las fórmulas para hacerlas más precisas.

Otra forma de promover la justicia es ajustar las fórmulas para que haya una asignación equitativa de los recursos. El ejemplo anterior sobre la asignación de gestores de casos para ayudar a las personas a salir antes del hospital es un buen ejemplo. Se pueden modificar los umbrales para determinar quiénes cumplen los requisitos en estas fórmulas para igualar la asignación de recursos reales a los distintos grupos.

Es de vital importancia tener como objetivos explícitos la equidad y la mejora de la salud de todos, y luego construir los sistemas para alcanzar esos objetivos. Es necesario incluir pasos específicos en los que haya una oportunidad de autorreflexión: ¿Lo que se está haciendo favorece la equidad para todos o se han empeorado involuntariamente las cosas?

6. Futuras líneas de investigación

6.1 Entorno de clúster de Spark

Debido al grandísimo tamaño que este tipo de datos puede llegar a alcanzar, algunos aspectos del modelado, como la optimización de los hiperparámetros, llegan a tardar mucho en ejecutarse. Por ello, sería interesante incluir en futuras versiones de la aplicación el uso del *framework* Apache Spark. Con este *framework* se puede conseguir un aumento significativo de la velocidad del código si se aprovecha junto a los clústeres de AWS paralelizando muchas tareas computacionales intensivas. Uno de los posibles inconvenientes al utilizar Spark para este proyecto puede ser la falta del parámetro de peso de la clase en las funciones del modelo de Spark ML.

Algunas de las ventajas de Spark son: velocidad, facilidad de uso, posibilidad de análisis avanzados, naturaleza dinámica, se puede usar con varios lenguajes de programación, Apache Spark es potente, mayor acceso a Big Data.

Este *framework* también presenta algunas desventajas: no hay proceso de optimización automático, su sistema de gestión de archivos, existen menos algoritmos que en otros *frameworks*, problemas con los archivos pequeños, su criterio de la ventana y no es adecuado para un entorno multiusuario.

6.2 Trabajo con series temporales

Una serie temporal es una colección de observaciones en orden cronológico. Pueden ser los precios diarios de cierre de las acciones, las cifras semanales de los inventarios, las ventas anuales o un sinfín de cosas más.

El análisis de series temporales, por tanto, no es más que analizar (trazar, identificar patrones, etc.) las series.

Una previsión de series temporales consiste en tomar esas observaciones pasadas y hacer predicciones sobre lo que ocurrirá en el futuro si se mantienen los mismos patrones.

El análisis y la previsión de las series temporales se encuentran entre las técnicas cuantitativas más comunes empleadas por las empresas y los investigadores en la actualidad.

Las tres principales ventajas de realizar análisis de series temporales son: ayuda a identificar patrones, ofrece la oportunidad de limpiar los datos y la previsión de series temporales puede «predecir» el futuro.

6.3 Inclusión de un conjunto de validación

Debido al reducido número de datos del que se dispone (93 pacientes) no ha sido posible emplear un conjunto de validación durante la fase de entrenamiento del modelo junto a los conjuntos de entrenamiento y de prueba. Sin embargo, cuando se recopile la información de más pacientes con la apertura de las fuentes de datos relativas al SARS-CoV-2 que a día de hoy se mantienen en privado en su mayoría, será posible crear un conjunto de validación, mejorando así el modelo.

6.4 Investigación sobre técnicas empleadas para reducir el sobreajuste en modelos con pocos datos

Tal y como se puede observar en los resultados del Trabajo de Fin de Grado, se ha obtenido un modelo que presenta sobreajuste. Como se ha comentado, esto es debido a la pequeña cantidad de datos con la que se ha entrenado el modelo. Sin embargo, existen algunas técnicas avanzadas (que sobrepasan los objetivos de este Trabajo de Fin de Grado) que permiten reducir este sobreajuste. Estas técnicas se explican en el *paper* de Sampedro, J., Dorado, P.I., Vicente, V., Sánchez, A., Jiménez, M., San Román, J.A.,...Fernández, F. (2020), en el que se trata un problema similar al que resuelve Health Twin en cuanto a lo que el desarrollo del modelo de Machine Learning se refiere.

En este *paper*, los investigadores emplean una validación cruzada anidada de la siguiente forma:

Todos los modelos se entrenaron y evaluaron con el mencionado esquema de validación cruzada de 10 pliegues. También se ajustaron los hiperparámetros para mejorar el rendimiento de los clasificadores ML de validación cruzada anidada, realizando 9 validaciones cruzadas con cada subconjunto de entrenamiento de la validación cruzada externa. Este enfoque es costoso desde el punto de vista informático debido a su naturaleza altamente iterativa, pero es asequible cuando se trata de conjuntos de datos pequeños y permite utilizar todo el conjunto de datos sin retener una parte para probar el error de generalización. (Sampedro, J., Dorado, P.I., Vicente, V., Sánchez, A., Jiménez, M., San Román, J.A.,...Fernández, F., 2020, p. 9)

6.5 Inclusión de nuevos casos de uso

Sería interesante incluir casos de uso que permitan añadir funcionalidades a la aplicación de lectura de los datos del estado de los pacientes.

Visualizar un calendario en el que se indique en color rojo oscuro, rojo, naranja o verde cada uno de los días según la ocupación de las camas UCI sea completa, casi completa, media o baja, respectivamente.

Recibir alertas cuando la predicción de días de alta o la predicción del estado del paciente cambien. El nivel de importancia de estas alertas dependerá de la gravedad del cambio. Cuando un paciente sufra un cambio muy brusco en la predicción de los días que le quedan para el alta, el médico será notificado mediante una notificación emergente en la aplicación (si está utilizando la aplicación en ese momento) o mediante una notificación del sistema (si no está utilizando la aplicación) y tras clicar la notificación será dirigido a la lista de pacientes y una ventana a modo de pop-up le mostrará aquellos que necesitan atención.

6.6 Sustitución de DynamoDB por Aurora para la base de datos de estados de los pacientes

Una de las ventajas de hacer uso del patrón CQRS es la posibilidad de aprovechar al máximo los puntos fuertes de los lados de lectura y escritura como si fueran unidades independientes.

Por ello, es interesante hacer uso de bases de datos SQL en el lado de lectura, de forma que sea posible configurar el almacenamiento de los datos relativos al estado de los pacientes de forma que su lectura sea lo más rápida posible. Es decir, almacenar los datos de forma desnormalizada, lo que podría implicar la repetición de los datos, al estilo *datawarehouse*. Esto no sería un problema, debido a que el principal interés del lado de lectura es lograr la mayor velocidad posible a la hora de recuperar los datos de la base de datos.

Para lograr este objetivo, la base de datos SQL que AWS ofrece, Aurora, es la solución perfecta, pues permite una fácil integración con el resto de servicios implementados en AWS y es una base de datos que cumple los más altos estándares.

Bibliografía

- [1] Amazon Web Services. (s.f.). Recuperado de: <https://docs.aws.amazon.com/>
- [2] Amazon Web Services. (Octubre de 2019). *Modern Application Development on AWS. Cloud-Native Modern Application Development and Design Patterns on AWS*. Recuperado de: <https://d1.awsstatic.com/whitepapers/modern-application-development-on-aws.pdf>
- [3] Andrews, P. (8 de junio de 2020). How To Create An AWS SQS Queue With Serverless. *Gitconnected*. Recuperado de: <https://levelup.gitconnected.com/how-to-create-an-aws-sqs-queue-with-serverless-2a538fe2413a>
- [4] Baek, H., Cho, M., Kim, S., Hwang, H., Song, M., y Yoo, S. (2018). Analysis of length of hospital stay using electronic health records: A statistical and data mining approach. *PloS one*, 13(4), e0195901. Recuperado de <https://doi.org/10.1371/journal.pone.0195901>
- [5] Brownlee, J. (2020). K-Nearest Neighbors for Machine Learning [Mensaje en un blog]. *Machine Learning Mastery*. Recuperado de: <https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/>
- [6] Cutler, A., Cutler, D.R., y Stevens, J.R. (2012). Random Forests. En Zhang, C., y Ma, Y. (Eds.), *Ensemble Machine Learning* (pp. 157-175). Boston, Estados Unidos: Springer. DOI: https://doi.org/10.1007/978-1-4419-9326-7_5
- [7] Faisal, S., y Tutz, G. (2017). Nearest Neighbor Imputation for Categorical Data by Weighting of Attributes. *ArXiv*. Recuperado de: <https://arxiv.org/abs/1710.01011>
- [8] Fratila, C. (Febrero de 2018). Building Desktop Applications with Electron. Recuperado de: <https://www.jfokus.se/jfokus18/preso/Building-Desktop-Applications-with-Electron.pdf>
- [9] Hossin, M., y Sulaiman, M.N. (2015). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining and Knowledge Management Process*, 5 (2). Recuperado de: https://www.researchgate.net/profile/Mohammad-Hossin/publication/275224157_A_Review_on_Evaluation_Metrics_for_Data_Classification_Evaluations/links/57b2c95008ae95f9d8f6154f/A-Review-on-Evaluation-Metrics-for-Data-Classification-Evaluations.pdf
- [10] Lundberg, S., y Lee, S. I. (2017). A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems*, 4768–4777.
- [11] Melnik, G., Dominguez, J., Cazzulino, D., Simonazzi, F., Subramanian, M., de Lahitte, H.,...Michell, N. (2012). CQRS Journey. *Microsoft*. Recuperado de: [https://docs.microsoft.com/en-us/previous-versions/msp-n-p/jj554200\(v=pandp.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/jj554200(v=pandp.10)?redirectedfrom=MSDN)
- [12] Montgomery, D.C., y Peck, E.A. (2012). Simple Linear Regression. En *Introduction to Linear Regression Analysis* (pp. 12-68). Hoboken, Estados Unidos: John Wiley and Sons, Inc.
- [13] Narkhede, S. (26 de junio de 2018). Understanding AUC-ROC curve. *Towards Data Science*. Recuperado de: <https://www.48hours.ai/files/AUC.pdf>
- [14] Nordling, L. (25 de septiembre de 2019). A fairer way forward for AI in health care. *Nature*, 573(7775), 103-105. DOI: <https://doi.org/10.1038/d41586-019-02872-2>

- [15] O'Brien, L., Guilley, S., Coulter, D., Lu, P., Martens, J., Brockschmidt, K.,...Bye, T. (2021). What are Azure Machine Learning Pipelines? *Microsoft*. Recuperado de: <https://docs.microsoft.com/en-us/azure/machine-learning/concept-ml-pipelines>
- [16] OpenAPI. (s.f.). Recuperado de: <https://oai.github.io/Documentation/>
- [17] Perier, F., Tuffet, S., Maraffi, T., Alcalá, G., Victor, M., Haudebourg, A.F.,...Dessap, A.M. (2020). Effect of Positive End-Expiratory Pressure and Proning on Ventilation and Perfusion in COVID-19 Acute Respiratory Distress Syndrome. *American Journal of Respiratory and Critical Care Medicine* 202(12), 1713-1717. DOI: <https://doi.org/10.1164/rccm.202008-3058LE>
- [18] Sampedro, J., Dorado, P.I., Vicente, V., Sánchez, A., Jiménez, M., San Román, J.A.,...Fernández, F. (2020). Machine Learning To Predict Stent Restenosis Based On Daily Demographic, Clinical And Angiographic Characteristics. *Canadian Journal of Cardiology*. DOI: <https://doi.org/10.1016/j.cjca.2020.01.027>
- [19] Schapire, R.E. (2013). Explaining AdaBoost. En Schölkopf, B., Luo, Z., y Vovk, V. (Eds.), *Empirical Inference* (pp. 37-52). Heidelberg, Alemania: Springer. DOI: https://doi.org/10.1007/978-3-642-41136-6_5
- [20] Scikit-Learn. (s.f.). Recuperado de <https://scikit-learn.org/stable/>
- [21] Soley-Bori, M. (2013). Dealing with missing data: Key assumptions and methods for applied analysis. Boston University.
- [22] Swagger. (s.f.). Recuperado de: <https://swagger.io/docs/>
- [23] Zakka, K. (2016). A Complete Guide to K-Nearest-Neighbors with Applications in Python and R [Mensaje en un blog]. *Kevin Zakka's Blog*. Recuperado de <https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/>