Universidad de Valladolid
Escuela de Ingeniería Informática
Grado en Ingeniería Informática
Mención en Computación

Curso 2022/2023

Trabajo de Fin de Grado:

Deep Learning aplicado a la Clasificación del Riesgo en Pacientes con Intoxicación

Autor/a:
Cristina Domingo Redondo

Tutor:
Benjamín Sahelices Fernández

Julio de 2023, Valladolid

Índice general

Índice de figuras

Resumen

En el ámbito sanitario, el uso del Deep Learning se presenta como una innovadora herramienta capaz de transformar la atención médica. Aunque hasta ahora los avances más destacados en este campo se han centrado en el reconocimiento de imágenes, aún quedan horizontes por explorar utilizando otras tipologías de datos. Para contribuir a esta evolución, se ha estudiado el uso de Fastai como instrumento de creación de un modelo de clasificación del nivel de riesgo en pacientes intoxicados de Castilla y León.

Abstract

In the field of healthcare, the use of Deep Learning emerges as an innovative tool able to transform medical care. While the most notable advancements in this field have focused on image recognition, there are still uncharted horizons to explore using other data types. To contribute to this evolution, the use of Fastai has been studied as a tool for creating a risk classification model for intoxicated patients from Castilla y León.

Capítulo 1

Contexto tecnológico

Resulta de vital importancia comprender el origen y la evolución de las redes neuronales en el contexto de la inteligencia artificial y el aprendizaje automático. El entendimiento de la trayectoria histórica de este tipo de modelos de datos es fundamental para apreciar su relevancia y el impacto actual que han logrado en diversos campos, así como para anticipar su futuro desarrollo y explorar su potencial en diversas disciplinas.

1.1. Origen y evolución

Las bases de lo que hoy se conoce como Inteligencia Artificial se remontan a varias décadas atrás. Uno de los pilares de este conocimiento data del siglo XX con el planteamiento del teorema de Bayes en 1812, según el cuál la probabilidad de que un evento ocurra se puede modelar basándose en el conocimiento de las condiciones previas que pudieran estar relacionadas con dicho evento [1].

Si se hace un recorrido por los sucesivos avances a lo largo de la Historia en el ámbito de la ciencia de la computación, se declara como uno de los primeros precursores en este campo al matemático Alan Turing. En 1935, Turing describió una máquina de computación abstracta con memoria ilimitada que puede operar gracias a un escáner que se mueve hacia adelante y hacia atrás a través de una cinta que posee una serie de instrucciones grabadas [2]. Lo sorprendente de la máquina, fue que también era capaz de escribir nuevas instrucciones, modificando y mejorando su propio código, es decir, operando de manera autónoma.

En 1943, el neurobiólogo McCullots y el estadístico Pitts propusieron el primer modelo de neurona artificial [3]. En este sistema, el soma de una neurona se correspondía con el núcleo procesador de los datos de entrada modificados por unos determinados pesos, de forma similar al proceso biológico de la sinapsis neuronal. Las entradas correspondían a las dendritas de la neurona y la salida de dichos datos, se correspondería de esta forma con la suma ponderada de las entradas, propagándose por la red de igual forma que la información se propaga por el axón de una neurona en el cerebro humano. Propusieron tres tipos de neurona: sensitivas, que reciben las entradas directamente, activadoras, que materializan la respuesta o salida, y el resto, que se corresponden con los nodos o interconexiones que transmiten y procesan la información de acuerdo al modelo. Respecto al proceso de la sinapsis, diferenciaron entre sinapsis excitadora, si el peso que afectaba al dato de entrada era de signo positivo, o inhibidora si este era negativo.

En la década de 1950 se producen grandes avances como la Teoría de Convergencia del Perceptrón de Rosenblatt [4]. Se proponía una red formada por varias de las neuronas del modelo de McCullots y Pitts con capacidad de aprendizaje mediante la progresiva modificación de sus parámetros. Este sistema, aportaba muy buenos resultados en el ámbito de la clasificación para muestras de entrada linealmente separables. Basándose en estos sistemas, Marvin Minksy y Dean Edmonds, científicos del Instituto Tecnológico de Massachussets (MIT), crearon un modelo capaz de aprender de sus experiencias anteriores para lograr salir de un laberinto.

Este avance se vio frenado a finales de la década de los 60 en lo que se conoce como "el invierno de la IA" tras la publicación del libro Perceptrones, escrito por Marvin Minsky y Seymour Papert en 1969 [5]. Se demostró mediante un simple experimento con puertas lógicas XOR que el Perceptrón simple propuesto por Rosenblatt, nunca podría tener una capacidad computacional similar al de una máquina de Turing, ya que con muestras de entrada que no fueran linealmente separables era incapaz de encontrar un hiperplano que lo hiciera. No vuelven a producirse avances significativos hasta 1986 con la publicación por parte de McClelland y Rumelhart de Parallel Distributed Processing (Procesamiento Distribuido en Paralelo), en la cual se proponía un modelo de perceptrón multicapa capaz de resolver este problema [6].

El estudio y aplicación de estos sistemas se engloba en la rama informática que hoy se conoce como Aprendizaje Automático. Desde sus inicios y hasta la actualidad, estos sistemas de redes neuronales han ido evolucionando y superando problemas tales como el de la evanescencia del descenso del gradiente o la falta de hardware capaz de comportarse de manera eficiente frente a altos volúmenes computacionales cuando se operan con muchas capas [7][8].

Capítulo 2

Contexto conceptual

En este capítulo se asientan las bases teóricas del estudio y sus implicaciones en relación al uso de las redes neuronales como herramientas de clasificación y predicción. Además, se examinan sus capacidades y limitaciones con el objetivo de comprender mejor su funcionamiento.

2.1. Machine Learning

El aprendizaje automático o *machine learning* (ML) es una disciplina de la ciencia de datos basada en la creación de modelos y algoritmos que buscan patrones y relaciones en los datos a través de la optimización de una función objetivo. Estos modelos se ajustan automáticamente a medida que se les proporciona más información, lo que les permite mejorar su precisión y generalizar a nuevos datos. En lugar de programar explícitamente todas las reglas necesarias para realizar una tarea, se proporciona al programa una gran cantidad de datos de entrenamiento que le permite aprender de forma automatizada. La definición más aceptada en el campo de la informática le corresponde a Tom Michell en [9], según él: "un programa de computadora se dice que aprende de la experiencia E con respecto a una tarea T y una medida de rendimiento P, si su rendimiento en la tarea T, medido por P, mejora con la experiencia E". En otras palabras, el programa ajusta sus parámetros internos a medida que se le proporciona más información y, por lo tanto, puede realizar la tarea con mayor precisión basándose en su propia experiencia.

2.2. Deep Learning

El aprendizaje profundo o *deep learning* (DL) es una rama aún más específica del aprendizaje automático que se engloba dentro de la neurocomputación, una rama de la informática cuya finalidad es crear sistemas inteligentes simulando modelos cercanos al cerebro humano. Estos sistemas extraen y transforman datos para predecir, clasificar o identificar conceptos mediante el uso de capas que se comportan como redes neuronales [7].

2.2.1. Redes Neuronales

Las redes neuronales se corresponden con un conjunto de autómatas celulares denominados neuronas, que transmiten los flujos de información capa tras capa según la topología de interconexiones

de la red. Cada una de estas capas toma sus entradas de las capas anteriores y las transforma progresivamente. Las capas son entrenadas por algoritmos que minimizan sus errores y mejoran su precisión. De esta manera, la red puede aprender a realizar una tarea específica. Una de las características que mejora la eficiencia de estos sistemas respecto a otros modelos, es el uso del paralelismo, ya que se comportan como un conjunto de procesadores simples (neuronas) trabajando en paralelo. Esta circunstancia les aporta gran potencial, pero hace que el resultado esté marcado por la disponibilidad de sus entradas (una neurona no obtiene su salida hasta tener todas sus entradas).

Existen distintos tipos de neurona. El primero que surgió fue el perceptrón simple, desarrollado en las décadas de 1950 y 1960 por el científico Frank Rosenblatt e inspirado en trabajos anteriores de Warren McCulloch y Walter Pitts [4] [3]. Para explicar el funcionamiento básico y la estructura de una red neuronal, se tomará este modelo como punto de partida.

2.2.2. Estructura de una red neuronal

Un perceptrón simple toma varias entradas binarias[1] $x_1, x_2, ..., x_n$, y produce una única salida también binaria y. A cada entrada se le asocian unos pesos $w_1, w_2, ..., w_n$ con $w_i \in \mathbb{R}$ que se utilizarán para calcular la salida de forma que:

$$y = \begin{cases} 0 & \sum x_i \cdot w_i \leq umbral \\ 1 & \sum x_i \cdot w_i > umbral \end{cases} \tag{2.1}$$

Con el objetivo de dar solución a problemas más complejos, esta idea derivó en el perceptrón múltiple. En dicho modelo se superponen perceptrones simples en forma de capas, de manera que las salidas de unos se toman como entradas de los siguientes hasta devolver una única salida en la última capa. Por razones históricas, a estas redes de muchas capas se les llama *multilayer perceptrons*, perceptrones de muchas capas (MLPs), aunque no están formadas por perceptrones sino por neuronas sigmoides. La estructura de este nuevo modelo, tal y como muestra la figura 2.1, se dividirá en neuronas de la capa de entrada, neuronas de la capa de salida y aquellas neuronas que se encuentren en las capas intermedias u ocultas:

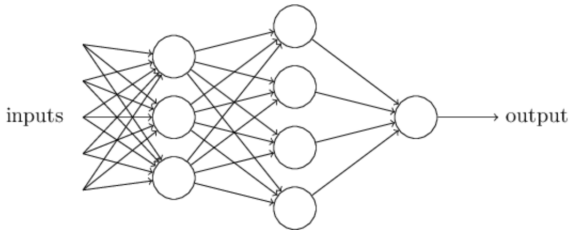


Figura 2.1: Perceptrón múltiple

[1] $x \in [0, 1]$

Para calcular la salida de los perceptrones múltiples, se transforma la ecuación anterior (2.1) en un producto escalar de la forma $w \cdot x \equiv \sum x_i \cdot w_i$ cambiando el umbral por el bias, de tal manera que bias $\equiv$ - umbral (2.2).

$$y = \begin{cases} 0 & \sum x_i \cdot w_i + bias \leq 0 \\ 1 & \sum x_i \cdot w_i + bias > 0 \end{cases} \tag{2.2}$$

El bias o sesgo se considera una medida de la facilidad de la neurona para "dispararse", es decir, para generar una salida con valor 1. En la actualidad, existen algoritmos de aprendizaje que pueden ajustar automáticamente los pesos y sesgos de una red de neuronas artificiales [10].

El problema de los perceptrones, es que un ligero cambio en los pesos o en el sesgo puede hacer que el comportamiento del resto de la red cambie por completo de una manera muy complicada. Esto hace que sea difícil ver cómo modificar los pesos y sesgos gradualmente para que la red se acerque más al comportamiento deseado. Como solución, se introdujo la neurona sigmoide.

Las neuronas sigmoides son similares a los perceptrones, pero modificadas para que pequeños cambios en sus pesos y sesgos causen solo un pequeño cambio en su salida. La principal diferencia respecto al modelo anterior es que sus entradas $x_1, x_2, ..., x_n$ no son binarias, tomando valores continuos comprendidos entre [0,1]. La forma de calcular las salidas también cambia. Para dicho cálculo se empleará la función sigmoide cuya definición se muestra a continuación [11].

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-\sum_i x_i \cdot w_i - bias}} \tag{2.3}$$

Dicha función devuelve un valor entre 0 y 1 para cualquier valor de entrada x. Cuando x es muy negativo (es decir, $x \to -\infty$), la función se acerca a 0. Por otro lado, cuando x es muy positivo (es decir, $x \to \infty$), la función se acerca a 1. En ambos casos, la función sigmoide aproxima al perceptrón, es decir, se comporta de manera análoga. En el punto medio, $x = 0$, la función devuelve un valor de 0.5 y es donde más se aleja del comportamiento del perceptrón. Por lo tanto, el rango de valores que puede tomar la función sigmoide es [0,1], tal y como ilustra la figura 2.2.

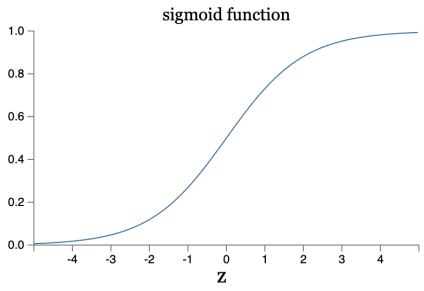


Figura 2.2: Función sigmoide

Este rango limitado es útil para la interpretación y la normalización de las salidas de una red neuronal, ya que asegura que las salidas estén dentro de un rango manejable y consistente y facilita el cálculo de la variación de la salida entre iteraciones mostrado a continuación (2.4).

$$\Delta y \approx \sum_j \frac{dy}{dw_j} \Delta w_j + \frac{dy}{db} \Delta b \tag{2.4}$$

2.2.3. Tipos de redes neuronales

Debido a la importancia y utilidad de este tipo de modelos, las redes neuronales están en constante evolución, lo que hace prácticamente imposible la enumeración de todas ellas. A continuación, se describen las principales y más utilizadas redes neuronales [11]. Cada uno de estos tipos tiene sus propias características y aplicaciones, lo que los hace adecuados para diferentes tipos de tareas de aprendizaje automático:

- **Redes neuronales feedforward.**
  Las redes neuronales feedforward se componen de una capa de entrada, una o más capas ocultas y una capa de salida. Cada capa contiene una o más neuronas, que se conectan con todas las neuronas de la capa siguiente. La información se transmite en una sola dirección, desde la capa de entrada hasta la capa de salida. En cada capa, los valores de entrada se multiplican por los pesos de la capa y se les suma el sesgo. Después, se aplica una función de activación a la suma ponderada para producir los valores de salida de cada neurona. Este proceso se repite capa por capa hasta que se llega a la capa de salida, donde se obtiene la salida final. Las principales diferencias con respecto a otros tipos de redes neuronales es que las feedforward son relativamente simples y eficientes en términos de tiempo de entrenamiento. Sin embargo, no son adecuadas para modelar secuencias o relaciones temporales entre los datos, ya que la red no tiene memoria a largo plazo y no puede recordar estados anteriores. Por lo tanto, no puede capturar la dinámica temporal o las dependencias secuenciales en los datos.

- **Redes neuronales recurrentes.**
  Las redes neuronales recurrentes (RNN) tienen una estructura similar a las anteriores, pero en este contexto la información fluye en bucle porque neuronas de distintas capas están interconectadas, es decir, las salidas de las capas anteriores se retroalimentan en las capas posteriores. De esta manera, cada neurona en una RNN tiene una "memoria" interna que le permite recordar información anterior. En cada iteración, la RNN toma una entrada y produce una salida, así como una nueva "memoria" interna que se calcula utilizando la salida anterior y la entrada actual. En cada paso de tiempo, la red también actualiza los pesos y sesgos basados en el error de la predicción actual. Estos modelos resultan muy útiles para modelar secuencias y relaciones temporales entre los datos, lo que las hace adecuadas para tareas como el procesamiento del lenguaje natural y la predicción de series temporales. Sin embargo, entrenar una RNN puede ser más difícil debido a la retroalimentación recurrente y la propagación del gradiente.

- **Redes neuronales convolucionales.**
  Las redes neuronales convolucionales (CNN) tienen una estructura en la que las neuronas están organizadas en filtros que se aplican a subconjuntos del conjunto entrada. Estos filtros se desplazan sobre la entrada para detectar patrones en diferentes partes, por lo que son el

modelo más utilizado cuando los datos de entrada se corresponden con imágenes. En general, una CNN consta de varias capas, cada una con un propósito específico en el procesamiento de imágenes.

La primera capa de una CNN es la capa de entrada, donde se procesa por primera vez la imagen en forma matricial. La segunda capa es la capa de convolución, que utiliza un conjunto de filtros para realizar la convolución de la imagen de entrada. La convolución es una operación matemática que permite extraer características importantes de la imagen. Cada filtro en la capa de convolución está diseñado para detectar un patrón específico en la imagen. Por ejemplo, un filtro puede estar diseñado para detectar bordes, mientras que otro puede estar diseñado para detectar esquinas. Los filtros se aplican a la imagen de entrada y generan una imagen de características, donde cada píxel se corresponde con un valor numérico que representa la activación de un filtro en una ubicación específica de la imagen.

La siguiente capa en una CNN es la capa de pooling, que reduce la dimensión del espacio de características al muestrear y agrupar píxeles cercanos. Esto ayuda a reducir también el número de parámetros necesarios para procesarla. Después de la capa de pooling, se pueden agregar varias capas de convolución y pooling, que extraen características cada vez más complejas de la imagen. Finalmente, los píxeles de la imagen resultante se aplanan en un vector y se pasan a una capa de salida, donde se realiza la clasificación o predicción correspondiente. En resumen, las CNN son muy útiles en el procesamiento de imágenes porque pueden detectar patrones y características importantes de manera eficiente.

- **Redes LSTM.**
  Las redes LSTM (*Long Short-Term Memory*) son un tipo de RNN que se utilizan para modelar datos secuenciales. A diferencia de las RNN tradicionales, las redes LSTM tienen una estructura más compleja y utilizan celdas de memoria para almacenar información anterior y decidir qué información se transmite hacia adelante. Las celdas de memoria tienen puertas que controlan el flujo de información hacia y desde la celda, lo que permite que la red se adapte a diferentes secuencias y longitudes de secuencia.

  Las redes LSTM tienen tres capas principales: la capa de entrada, la capa de salida y la capa de olvido. La capa de entrada decide cuánta información nueva se agrega a la memoria de corto plazo, la capa de salida decide cuánta información de la memoria de corto plazo se agrega a la salida y la capa de olvido decide cuánta información antigua se mantiene en la memoria de corto plazo.

  Las redes LSTM son útiles para modelar secuencias largas y datos de series temporales, y se han utilizado en aplicaciones como el reconocimiento de voz, la traducción automática y el análisis de sentimientos.

- **Redes GAN.**
  Las redes GAN (*Generative Adversarial Networks*) son un tipo de red neuronal utilizada para generar datos nuevos y realistas a partir de un conjunto de datos de entrenamiento. Las GAN constan de dos redes: un generador y un discriminador. El generador crea muestras falsas y el discriminador decide si cada muestra es real o falsa. El generador mejora su capacidad para generar muestras más realistas en función de los errores del discriminador, mientras que el

discriminador mejora su capacidad para distinguir lo real de lo falso en función de las muestras generadas por el generador. Las redes GAN son útiles para la generación de imágenes, audio y texto, y se han utilizado en aplicaciones como la creación de imágenes de caras realistas, la síntesis de voz y la generación de diálogos de bots.

2.2.4. Funciones de activación

Tal y como se ha descrito anteriormente (2.2.2), en cada iteración del proceso de aprendizaje o entrenamiento de la red neuronal, estas funciones se aplican a la suma ponderada de las entradas y los pesos más el bias de una neurona. Si el resultado de la función de activación es mayor que un cierto umbral, la neurona se activa y pasa su salida a la siguiente capa de la red neuronal. De lo contrario, si el resultado de la función de activación es menor que el umbral, la neurona permanece inactiva y no pasa ninguna salida a la siguiente capa.

Algunas de las funciones de activación más utilizadas son:

- Lineal: esta función simplemente devuelve la suma ponderada de las entradas y los pesos. Aunque esta función no introduce ninguna no linealidad en la red neuronal, todavía se utiliza en algunas situaciones, como en capas de salida para la regresión.

$$f(x) = x \tag{2.5}$$

- Sigmoide: esta función se utiliza comúnmente en las capas ocultas de las redes neuronales. La función sigmoide tiene una forma de "S" y se aproxima a una función de paso en los límites. Cuando la entrada es grande, la función de activación sigmoide se aproxima a 1 y cuando la entrada es pequeña, se aproxima a 0. La función sigmoide es diferenciable[2], lo que la hace útil en la retropropagación del error.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

- ReLU (*Rectified Linear Unit*): esta función se ha vuelto muy popular en los últimos años y se utiliza comúnmente en las capas ocultas. La función ReLU toma el valor máximo entre cero y la suma ponderada de las entradas y los pesos. Cuando la entrada es positiva la función ReLU devuelve la entrada y cuando es negativa devuelve cero. Esta función es muy eficiente en términos computacionales y no sufre del problema de gradiente que se presenta en la función sigmoide.

$$f(x) = \text{máx}(0, x) \tag{2.7}$$

- Softmax: esta función se utiliza comúnmente en la capa de salida para la clasificación multiclase. La función Softmax normaliza la salida de la capa anterior en un vector de probabilidades, donde cada elemento del vector representa la probabilidad de que la entrada pertenezca a una de las clases.

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.8}$$

donde z_i es la entrada a la i-ésima neurona y K es el número total de neuronas en la capa.

[2]Una función se considera diferenciable si existe su derivada en todos los puntos de su dominio.

2.2.5. Aprendizaje

En apartados anteriores (2.2.2) se ha proporcionado información sobre cómo se transmite iterativamente la información entre las capas de una red sencilla, y sobre cómo en estas iteraciones denominadas de entrenamiento se ajustan los pesos y bias (sesgos) empleando los datos de entrada, de manera que el modelo pueda producir la salida deseada para una futura muestra denominada de validación. Durante el entrenamiento, la red aprende a reconocer patrones y relaciones en los datos, lo que le permite hacer predicciones o clasificar las nuevas entradas con precisión.

Esta precisión viene determinada por la capacidad de aprendizaje y generalización de la red. En ella influyen diversos factores como la cantidad y calidad de los datos de entrenamiento que proporcionemos a la red, el tamaño de la misma (tanto en número de capas como de neuronas por capa), la función o funciones de activación utilizadas, o la técnica de optimización empleada durante el entrenamiento [10].

Respecto a la influencia del tamaño de la red en el aprendizaje de la misma, existen diversos estudios como [12] que desmienten el hecho de que a mayor tamaño mejor es su precisión. Si bien es cierto que en algunos modelos concretos relacionados con la clasificación y reconocimiento de imágenes sí que existe una correlación directa, no siempre es así. Por ejemplo, si se busca crear un modelo basado en el reconocimiento de patrones visuales con imágenes, las neuronas de la primera capa podrían aprender a reconocer bordes, y a continuación las neuronas de la segunda capa podrían aprender a reconocer formas más complejas, como triángulos o rectángulos, construidos a partir de bordes. La tercera capa reconocería formas aún más complejas o incluso colores, y así sucesivamente. Es probable que estas múltiples capas de abstracción proporcionen a las redes neuronales una ventaja a la hora de aprender a resolver problemas complejos de reconocimiento de patrones frente a las redes más simples, pero en problemas de estudio con datos de otro tipo como tabulares o series temporales no se ha podido demostrar dicha ventaja [13].

De hecho, en algunos casos, el uso de redes neuronales muy profundas puede llevar a problemas como el sobreajuste, donde la red se ajusta demasiado a los datos de entrenamiento y no puede generalizar bien a nuevos datos. Por lo tanto, es importante encontrar un equilibrio adecuado entre la profundidad de la red y otros factores para lograr un rendimiento óptimo [10].

Descenso de gradiente estocástico

Para entender de forma más precisa cómo aprende una red compleja, conviene entender el algoritmo de descenso de gradiente estocástico por retropropagación. Para ello, resulta necesario comenzar explicando por qué estas redes con mayor número de capas ocultas o intermedias resultan más difíciles de entrenar. A menudo, cuando las últimas capas de la red están aprendiendo bien, se observa que las primeras capas presentan peores métricas durante el entrenamiento y no aprenden casi nada. Más adelante se demuestra que existen razones fundamentales por las que se produce la desaceleración del aprendizaje, relacionadas con el uso de técnicas de aprendizaje basadas en gradientes.

El descenso de gradiente estocástico (SGD) es el algoritmo de optimización de los pesos y el bias que se aplica a una función de coste, la cual mide la diferencia entre la salida producida por la red y la salida deseada para una determinada entrada. Este gradiente se propaga hacia atrás a través de la red con el objetivo de minimizar la función de coste. El gradiente de esta función de coste con respecto a los pesos y el bias de la red nos indica la dirección (creciente o decreciente) en la que se

deben actualizar los parámetros para disminuir el valor de la función de coste (2.9). Es decir, si se aumentan o disminuyen los pesos y bias de forma opuesta al gradiente, la función de coste debería disminuir [14].

$$C(w,b) = \frac{1}{2n}\sum_{x} \parallel y_{real} - y_{pred} \parallel \tag{2.9}$$

Donde w son los pesos, b el bias, n el tamaño del conjunto de entrada de la red, y_{real} el valor real asociado al conjunto de entrada x y y_{pred} el valor predicho asociado a la salida de la red para el conjunto de entrada x.

Por lo tanto, el SGD actualiza los pesos y bias de la red moviéndolos en la dirección opuesta del gradiente de la función de coste. La actualización de los parámetros se realiza iterativamente, en función de un ratio de aprendizaje (learning rate) que determina la magnitud de cada actualización. En el SGD, en lugar de calcular el gradiente de la función de coste para todo el conjunto de entrenamiento, se utiliza una muestra aleatoria (lote o batch) de ejemplos de entrenamiento en cada iteración. Esto se hace por eficiencia computacional y para evitar que el algoritmo caiga en mínimos locales.

El SGD se puede representar matemáticamente mediante la siguiente fórmula para la actualización de los pesos:

$$w_{i,j}^{(k)} = w_{i,j}^{(k)} - \alpha\frac{\partial C(w,b;x,y)}{\partial w_{i,j}^{(k)}} \tag{2.10}$$

$$b_{j}^{(k)} = b_{j}^{(k)} - \alpha\frac{\partial C(w,b;x,y)}{\partial b_{j}^{(k)}} \tag{2.11}$$

Donde $w_{i,j}^{(k)}$ representa el peso que conecta la neurona i de la capa $k-1$ con la neurona j de la capa k, $b_j^{(k)}$ es el bias de la neurona j de la capa k, α es la tasa de aprendizaje, $C(w,b;x,y)$ es la función de coste que se está minimizando, y $\frac{\partial C(w,b;x,y)}{\partial w_{i,j}^{(k)}}$ y $\frac{\partial C(w,b;x,y)}{\partial b_{j}^{(k)}}$ son las derivadas parciales de la función de coste con respecto a los pesos y los bias, respectivamente.

De entre los elementos que intervienen en el cálculo del descenso del gradiente, cabe mencionar algunas características de cada uno a tener en cuenta. El tamaño de lote o *batch size* es el número de ejemplos de entrenamiento que se utilizan en cada iteración para actualizar los pesos [11]. Un tamaño de lote pequeño puede conducir a actualizaciones más rápidas de los pesos, pero puede ser menos preciso que un tamaño de lote grande debido a la variabilidad en la muestra.

El ratio de aprendizaje o *learning rate* (LR) es el tamaño de cada actualización de los pesos. Un ratio grande puede hacer que el algoritmo oscile o diverja, mientras que uno pequeño puede hacer que el algoritmo tarde más en converger.

La función de coste ya mencionada anteriormente es aquella que se utiliza para medir la diferencia entre la salida producida por la red y la salida deseada. Tal y como se explicó anteriormente, el SGD busca minimizar esta función variando los pesos y bias en la dirección opuesta al gradiente.

Respecto a la inicialización de los pesos, es importante inicializar sus valores de manera adecuada, ya que un mal ajuste puede hacer que el algoritmo tarde más en converger o incluso que no converja en absoluto. Una forma común de inicialización es aquella que utiliza una distribución normal centrada en cero con varianza inversamente proporcional al número de entradas de una capa.

En conclusión, el descenso del gradiente estocástico es un algoritmo de optimización comúnmente utilizado en el entrenamiento de redes neuronales. Al actualizar los pesos de la red en función del gradiente de la función de coste, el SGD busca minimizar el valor de dicha función y encontrar los valores óptimos de los pesos y bias de la red. Sin embargo, el SGD puede presentar problemas como el problema del tamaño de paso o el estancamiento en mínimos locales, que se pueden abordar mediante técnicas como la regularización o el momento [14].

Tipos de aprendizaje en redes neuronales

Existen diferentes enfoques de aprendizaje en las redes neuronales, incluyendo el aprendizaje supervisado, no supervisado y por refuerzo, cada uno con sus propias técnicas y aplicaciones.

- Aprendizaje supervisado: la red se entrena utilizando pares de entrada y salida previamente etiquetados, de manera que la red pueda aprender a producir la salida correcta para una entrada determinada.
- Aprendizaje no supervisado: la red se entrena para identificar patrones y estructuras en los datos de entrada sin necesidad de etiquetas previas.
- Aprendizaje por refuerzo: la red aprende a tomar decisiones en un entorno dinámico, recibiendo refuerzos positivos o negativos en función de sus acciones.

2.2.6. Limitaciones

Las redes neuronales como modelo computacional poseen numerosas ventajas, como su capacidad de generalización o su tolerancia a fallos, ya que pueden seguir funcionando incluso si algunos de los nodos o conexiones de la red fallan, su capacidad de procesamiento paralelo, pudiendo procesar múltiples entradas al mismo tiempo, o su capacidad de modelado no lineal, ya que las redes neuronales pueden modelar funciones no lineales complejas, lo que significa que pueden representar relaciones entre variables que son difíciles de capturar con otros modelos lineales [11].

Aunque todas estas características las convierten en una buena elección a la hora de construir un modelo de datos, también poseen ciertas limitaciones.

Para entrenar una red neuronal de manera efectiva, se requiere una gran cantidad de datos de entrenamiento o entrada etiquetados, es decir, la calidad de la salida viene condicionada por la disponibilidad de datos de entrada.

Además, entrenar una red neuronal puede ser un proceso lento y computacionalmente costoso, especialmente para redes grandes y complejas, ya que poseen un mayor número de parámetros a ajustar.

Otro problema reseñable es que algunos de estos modelos pueden ser difíciles de interpretar, lo que significa que no siempre es fácil entender cómo la red está tomando decisiones o qué característi-

cas está utilizando para hacer sus predicciones.

Una de sus grandes limitaciones derivada del entrenamiento es el sobreajuste [15]. Este fenómeno ocurre cuando la red neuronal se ajusta demasiado a los datos de entrenamiento y no generaliza bien a los datos de prueba. Cuanto más tiempo entrene, mejor será su precisión en el conjunto de entrenamiento; la precisión del conjunto de validación también mejorará durante un tiempo, pero comenzará a empeorar a medida que el modelo comience a memorizar el conjunto de entrenamiento, en lugar de encontrar patrones subyacentes generalizables en los datos.

Para solventar dicho problema existen diversas técnicas como la regularización [15]. Se trata de una técnica que, agregando un parámetro de penalización a la función de coste, impide que los pesos de la red tomen valores grandes. Dos formas comunes de regularización son la regularización L1 y la regularización L2. La principal diferencia entre la regularización L1 y L2 es la forma en que penalizan los pesos grandes, mientras que la primera añade un término proporcional a la suma de los valores absolutos de los pesos (también llamada norma L1), la regularización L2 añade un término proporcional a la suma de los cuadrados de los pesos (también llamada norma L2). La elección de una u otra viene determinada por el problema en cuestión y las características de los datos.

En cuanto a los problemas derivados del uso del algoritmo SGD, uno de los más comunes es el problema del tamaño del ratio de aprendizaje (LR). Si dicho parámetro toma valores demasiado grandes, el algoritmo puede oscilar o divergir, mientras que si son demasiado pequeños, el algoritmo puede tardar mucho en converger. Por lo tanto, se debe encontrar un LR adecuado a cada modelo mediante prueba y error.

Otro problema común es el estancamiento en mínimos locales. El SGD puede atascarse en un mínimo local no óptimo y no ser capaz de llegar al mínimo absoluto. Una forma de abordar este problema es utilizar el momento, técnica que agrega una fracción del gradiente anterior a la actualización actual de los pesos, lo que suaviza la dirección de la actualización y puede ayudar a evitar oscilaciones.

En general, a pesar de estas desventajas, el uso de redes neuronales como modelo computacional sigue siendo una buena elección en muchos problemas de aprendizaje automático debido a sus numerosas ventajas y a los continuos avances en la investigación y la tecnología.

Capítulo 3

Software

Para el caso práctico objeto de estudio de este documento, se van a utilizar principalmente dos herramientas, RStudio y Fastai. La primera de ellas se emplea para los procedimientos de análisis, limpieza y separación de los conjuntos de datos, y la segunda para la creación de un modelo de clasificación.

3.1. RStudio

RStudio es un entorno de desarrollo integrado (IDE) de código abierto diseñado específicamente para el lenguaje de programación R [16]. R es un lenguaje de programación estadística que se originó a mediados de los 90s como un proyecto académico y de investigación liderado por los estadísticos Ross Ihaka y Robert Gentleman de la Universidad de Auckland, Nueva Zelanda [17].

En su origen, R se desarrolló como una alternativa a otros programas estadísticos como SAS o SPSS que eran privados y cuyas licencias presentaban un coste económico demasiado elevado para estudiantes e investigadores. Sin embargo, a medida que R se fue popularizando, se convirtió en una herramienta fundamental en el análisis de datos en diferentes áreas, desde la biología y la medicina hasta las finanzas y el marketing. A medida que R se fue convirtiendo en una herramienta más popular, se hizo evidente la necesidad de una interfaz de usuario que facilitase su uso. Por ello, en 2010 el científico de datos JJ Allaire y su equipo desarrollaron RStudio.

Para el caso práctico que ocupa este estudio, se ha decidido emplear Rstudio para los procedimientos de limpieza y análisis de datos, ya que es una herramienta compatible con muchas otras tecnologías del ecosistema de la ciencia de datos, incluyendo Git, LaTeX, Jupyter Notebooks y muchos otros, que además posee librerías específicas para la realización de dichas tareas.

3.2. Fastai

Se trata de una librería de DL de código abierto implementada sobre PyTorch, otra popular librería de aprendizaje profundo [18]. Fue creada por Jeremy Howard y Rachel Thomas en 2018 con el objetivo de hacer que el DL fuese más accesible y fácil de usar para una audiencia más amplia, incluyendo aquellos sin experiencia previa en programación o en aprendizaje automático. En su nuevo

enfoque, la mayoría de procedimientos de alta complejidad que se realizan durante la construcción de un modelo de redes neuronales, permanecen ocultos al usuario tras el uso de funciones más sencillas que los engloban.

Otra ventaja importante de Fastai es su amplia gama de características y herramientas para la creación y entrenamiento de modelos de DL, incluyendo una variedad de arquitecturas de redes neuronales predefinidas y herramientas de visualización de datos, lo que hace que la creación de estos modelos sea mucho más fácil y rápida. Además, Fastai también implementa algunas técnicas avanzadas de DL, como la transferencia de aprendizaje y la regularización automática.

El hecho de que esta librería se implemente sobre PyTorch tampoco es trivial, ya que Fastai se creó pensando en su utilización en modelos con imágenes, lo que requiere un procesamiento de datos específico para trabajar con ellas y suponía hasta el momento un coste computacional bastante elevado para las librerías existentes. Esta necesidad derivó en la idea de utilizar PyTorch y no otras librerías de Python, ya que ofrece un alto nivel de paralelismo y capacidad de cómputo distribuido, otorgándole una capacidad para aprovechar el hardware de GPU que el resto de librerías no presentaban. Este paralelismo resulta especialmente útil para modelos que involucran grandes conjuntos de datos o requieren una alta capacidad de procesamiento, como los modelos con imágenes. Además, proporciona automáticamente entrenamiento con normalización de batch optimizado para transferencia de aprendizaje, congelamiento de capas[1] y tasas de aprendizaje discriminatorias. Fastai también se integra con la librería cuDF de NVIDIA, proporcionando procesamiento de datos y entrenamiento de modelos optimizado para GPU de extremo a extremo, tal y como ilustra la figura 3.1. Fastai es el primer framework de deep learning que se integra con cuDF de esta manera.

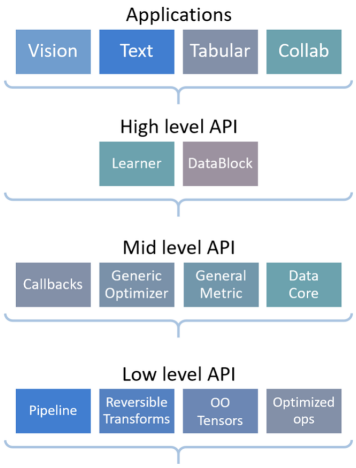


Figura 3.1: Arquitectura Fastai

La utilización de Fastai para la resolución del caso de estudio se ha llevado a cabo utilizando Google Colab [19], un entorno de desarrollo en la nube basado en Jupyter Notebook que permite

[1] Mantener las capas de una red neuronal pre-entrenada fijas durante el entrenamiento de la red en una tarea específica.

escribir y ejecutar código Python sin necesidad de configurar un entorno local. Además, proporciona acceso a recursos computacionales como GPU.

Todos los modelos, tanto de clasificación como de regresión, construidos con Fastai siguen una estructura similar compuesta por objetos predefinidos que aporta la propia librería [20]. Cada uno de estos objetos posee un constructor genérico, pero Fastai también ofrece la posibilidad de utilizar constructores ya adaptados al tipo de dato de entrada del modelo. Siguiendo el orden lógico de creación de un modelo se distinguen los siguientes constructores y pasos:

Transformaciones previas

Cuando no se dispone de un conjunto de datos previamente preparado para la creación del modelo, se deben aplicar a los datos de entrada una serie de procedimientos previos. La librería Fastai proporciona funciones específicas para obtener, separar y etiquetar conjuntos de datos [18]. Estas funciones son compatibles con las transformaciones genéricas que se pueden aplicar en Python.

Para la obtención de datos posee variantes del comúnmente utilizado *get*. Si se desean leer datos de una ruta específica se puede utilizar la función *getFiles* o sus variantes para imágenes *getImageFiles* y texto *getTextFiles*.

Una vez se posee un conjunto de datos, se puede realizar una separación en subconjuntos de entrenamiento y validación personalizada de los mismos. Fastai ofrece objetos que los separan automáticamente utilizando muestreo aleatorio, tal y como se explica más adelante, pero si se busca una separación con características más específicas como el tamaño de la muestra o el tipo del muestreo, es recomendable acudir a las funciones *split*. Las más utilizadas son *RandomSplitter*, la cual te devuelve un conjunto de entrenamiento a partir de los datos de entrada que le proporciones así como del tamaño del conjunto de validación que le indiques. Resulta muy útil si se desea utilizar muestreo estratificado. También se utilizan *TrainTestSplitter*, con funcionalidad idéntica a la función que proporciona la comúnmente utilizada función de la librería *sklearn* de Python, y *IndexSplitter* para separar por índices entre otras.

Si se necesitan etiquetar los datos de entrada posee algunas funciones útiles como *parentlabel*, para etiquetar directamente con el nombre del directorio padre del que se extraen los datos en una ruta, o *ColReader*, para leerlas de la columna de un DataFrame directamente.

Datablock / Dataloader

Cuando los datos de entrada se encuentran debidamente preparados para la creación del modelo, el primer paso antes de crearlo es cargarlos. Para hacerlo con Fastai, existen dos opciones [18].

Si los datos no están en el formato adecuado, se debe usar *Datablock* y después convertirlo a *Dataloader*. *Datablock* es un objeto de la biblioteca Fastai que permite personalizar completamente la forma en que los datos se transforman y se combinan para crear un conjunto de datos. Una vez se haya definido un *Datablock*, se puede reutilizar en diferentes proyectos sin tener que volver a escribir el mismo código, permitiendo también dividir la preparación de datos en bloques más pequeños y personalizables que se pueden combinar y reutilizar para diferentes conjuntos de datos.

Si por el contrario los datos ya presentan un formato adecuado, se puede utilizar directamente *Dataloader*, objeto que se utiliza para cargar los datos y crear los lotes de entrenamiento y validación necesarios para el entrenamiento del modelo. Este objeto también incluye la técnica mini-batch[2]. Es decir, si por ejemplo se desea entrenar un modelo de clasificación de imágenes, *Datablock* se utilizaría para definir la transformación de los datos de entrada, como la normalización y el redimensionamiento de las imágenes. También se puede utilizar para dividir los datos en conjuntos de entrenamiento y validación y para definir los tipos de datos de entrada y salida.

Por otro lado, *Dataloader* se utilizaría para cargar los datos preprocesados, crear lotes y preparar los datos para su uso en la fase de entrenamiento. Además, permite personalizar los parámetros de carga de los datos, como el tamaño de lote o el tipo de cada variable del modelo. El objeto *Dataloader* se podría entender como un objeto que automáticamente construye "cargadores" de datos de validación y entrenamiento. Esto facilita que, gracias a que ambos conjuntos de datos se integran en un único objeto, Fastai pueda mostrar de forma predeterminada métricas durante el entrenamiento usando el conjunto de validación.

Según el tipo de dato de entrada también encontramos variantes de este objeto como *LMDataLoader* para procesamiento del lenguaje y texto.

Learner

El *Learner* es el objeto que reúne arquitectura, optimizador y datos. Además, automáticamente elige una función de pérdida apropiada cuando sea posible. Se considera el núcleo de Fastai y es el que controla todo el proceso de entrenamiento, conectando los datos y las métricas para que el modelo pueda aprender de manera efectiva. También permite personalizar el entrenamiento con diferentes parámetros y técnicas.

A este objeto se le pasan como argumentos el objeto de tipo *Dataloader*, la estructura de las capas (número de capas y neuronas por capa) y la métrica que se desea utilizar (*accuracy*, MSE...) entre otros. Por defecto, construye redes completamente conectadas o RNN. Este objeto también posee variantes en función del tipo de dato de entrada, como *tabularLearner* para datos tabulares o *textLearner* para texto. Cada tipo de *Learner* tiene sus propios métodos y atributos específicos, pero todos ellos comparten algunos métodos y atributos básicos, como *fit* para entrenar el modelo, *validate* para validar el modelo en un conjunto de datos de validación, y *predict* para hacer predicciones sobre nuevos datos.

Además, los objetos *Learner* en Fastai pueden ser personalizados de varias maneras, como por ejemplo, cambiando la arquitectura del modelo mediante la función *model* o la clase *createCnn* (crear convolución) para imágenes. También permite cambiar el optimizador mediante el método *optFunc*, cambiar la función de pérdida mediante el método *lossFunc*, personalizar el proceso de entrenamiento mediante el método *fitOneCycle* o la clase *callbacks*, muy útil para ahorrar coste computacional gracias a la función *EarlyStoppingCallback*, que termina el entrenamiento cuando a partir de un número determinado de epoch[3] no mejora la métrica. Otra utilidad que presenta es la posibilidad de guardar los mejores modelos o la modificación de las tasas de aprendizaje durante el entrenamiento, pudiendo calcular cuál es la tasa óptima con la función *lrFind*.

[2]Cálculo de las actualizaciones de los parámetros del modelo utilizando pequeños lotes en vez de todo el conjunto de datos de entrenamiento, lo que permite una actualización más frecuente de los parámetros.

[3]Iteración completa a través de todo el conjunto de datos de entrenamiento.

Una vez se tiene construido el objeto *Learner*, lo habitual es entrenarlo empleando la función *fitOneCycle* para el número de epoch que se desee, en vez de utilizar la habitual función *fit*. El método *fitOneCycle* es más avanzado, ya que utiliza un ciclo de aprendizaje con un LR que cambia dinámicamente durante el entrenamiento. La idea detrás del ciclo de aprendizaje es que comienza con un LR bajo, aumenta gradualmente durante la primera mitad del entrenamiento y luego disminuye gradualmente durante la segunda mitad. Esto ayuda a evitar el sobreajuste y a mejorar el rendimiento general del modelo. Además, implementa otras técnicas como el congelamiento gradual de capas y el ajuste de los pesos de las mismas.

Después del entrenamiento, se puede consultar la arquitectura de la red empleando la función *summary*.

Una vez el modelo es construido y entrenado, se puede evaluar su calidad proporcionándole un conjunto de datos de validación y llamando a continuación a la función *getPreds* y pasando esas predicciones a la función de métrica que se desee aplicar, por ejemplo, *accuracy*. Cuando se considera que la calidad es adecuada, se puede realizar una predicción de datos sin etiquetar llamando a la función *predict*, que devolverá la correspondiente clasificación o predicción para la muestra de entrada. Es importante que los datos de validación no hayan sido utilizados durante el entrenamiento de la red para evaluar la capacidad de generalización correctamente.

Capítulo 4

Análisis

En este capítulo, se realiza un análisis exhaustivo del caso de estudio, donde se examinan los puntos de partida, los objetivos principales y la metodología utilizada para su desarrollo. El objetivo de este análisis es proporcionar una visión clara y fundamentada sobre el contexto en el que se ha llevado a cabo la investigación, estableciendo las bases necesarias para comprender los resultados y conclusiones presentados posteriormente.

4.1. Descripción del problema

El conjunto de datos empleado en este estudio consiste en datos médicos anonimizados de pacientes reales que han sufrido algún tipo de emergencia médica relacionada con una intoxicación. Este conjunto de datos tabulares cuenta con más de 100 mediciones relacionadas con diversos aspectos de la salud de los pacientes, abarcando desde indicadores biomédicos hasta variables demográficas y antecedentes médicos relevantes.

Partiendo de dicho conjunto de datos, se pretende desarrollar un clasificador capaz de predecir cuatro variables específicas que, en conjunto, formarán lo que se denomina un índice de riesgo. Este índice se corresponde con un número entero con un rango de [0,15], es decir, los números en los que se traduce la combinación de las cuatro variables binarias de salida. Estas variables representan aspectos críticos para la evaluación de la salud de los pacientes y son de suma importancia para la toma de decisiones médicas informadas. La capacidad de predecir con precisión el índice de riesgo permitirá una mejor identificación de los pacientes que requieren una atención médica más intensiva o un seguimiento más cercano.

Además, este estudio busca avalar la posible utilidad de la librería Fastai en el contexto de datos tabulares multicategóricos. Aunque Fastai es ampliamente reconocida por su eficacia en la clasificación de imágenes, su aplicación en problemas de clasificación tabular es menos común. Esta elección se basa en la necesidad de explorar el potencial de Fastai en un dominio de aplicación diferente, como es la clasificación de datos tabulares médicos, donde las características y los desafíos difieren de los problemas de clasificación de imágenes.

La mayoría de los usos de Fastai se han centrado tradicionalmente en problemas de clasificación de imágenes debido a su capacidad para aprovechar las arquitecturas de las CNNs y las técnicas

de aprendizaje profundo asociadas. Estas arquitecturas se han mostrado altamente efectivas para extraer características y patrones de imágenes, lo que ha llevado a resultados destacados en aplicaciones como reconocimiento de objetos, segmentación y detección. Sin embargo, es importante destacar que la aplicación de técnicas de aprendizaje profundo en la clasificación de datos tabulares también puede ser altamente beneficiosa. La utilización de Fastai en este contexto permitirá evaluar su eficacia y generalidad en la resolución de problemas de clasificación tabular multicategoría, expandiendo así su aplicabilidad y contribuyendo al avance de la investigación en inteligencia artificial en el campo de la medicina.

4.2. Planificación de objetivos y metodología

El caso de estudio se basa en la necesidad de desarrollar un clasificador utilizando una red neuronal para la predicción de variables en un conjunto de datos médicos tabulares. Este problema surge de la importancia de contar con herramientas que permitan una evaluación más precisa y temprana del riesgo asociado a la salud de los pacientes. A través de la clasificación de datos tabulares y la construcción de un índice de riesgo, se busca proporcionar información valiosa para la toma de decisiones médicas y mejorar la calidad de la atención brindada.

El segundo objetivo principal del estudio pretende evaluar la efectividad de la biblioteca Fastai en la clasificación tabular multicategoría, así como comparar el rendimiento de un modelo de clasificación multicategórico con el de otros modelos de clasificación individual para las variables de salida. Además, se busca identificar las configuraciones óptimas de hiperparámetros que maximicen la precisión y el rendimiento del modelo de clasificación.

Para alcanzar ambos objetivos, se ha decidido apostar por una metodología de trabajo iterativa. Esta metodología se caracteriza por su naturaleza flexible y adaptativa, ya que permite aprender y mejorar a medida que se obtienen resultados y se van ajustando las variables. Resulta especialmente útil en casos de estudio como este, donde existen múltiples variables o factores que pueden influir en los resultados. Además, gran parte de la experimentación se basará en el cumplimiento o no de una serie de hipótesis. En el caso de cumplir las hipótesis, se asumen suposiciones sobre los resultados esperados y las siguientes decisiones se toman en función de si los resultados experimentales refutan o no esas hipótesis.

Capítulo 5

Dataset

Tal y como se ha mencionado en el capítulo anterior, el objetivo principal de este proyecto es la creación de un clasificador multiclase capaz de predecir el índice de riesgo de pacientes que presentan algún tipo de intoxicación, además de otras patologías. Este índice hace referencia a la predicción de los valores de un subconjunto de cuatro variables binarias consideradas las variables dependientes. Para calcularlo se emplean otra serie de variables, independientes, que se detallan a continuación.

Todas estas variables corresponden a mediciones anonimizadas llevadas a cabo por profesionales del ámbito sanitario que acudieron en ambulancia a atender una urgencia médica. Las distintas mediciones se realizaron sobre los mismos pacientes, siendo algunas tomadas al llegar al lugar del aviso con el instrumental y herramientas correspondientes a una ambulancia, y otras tomadas al llegar al hospital o durante el ingreso del paciente en el mismo (en el caso de que el paciente ingrese).

El conjunto de datos inicial ha sido facilitado por el Servicio de Salud de Castilla y León (Sacyl), contando con la ayuda y supervisión de un médico del Hospital Río Hortega de Valladolid. A dicho conjunto de datos se le han aplicado una serie de procedimientos de limpieza y transformación para adecuar los datos a la creación del modelo requerido. El conjunto de datos definitivo se separó finalmente en un subconjunto de datos para entrenar el modelo, y en otro subconjunto para validarlo.

5.1. Descripción del conjunto de datos

El conjunto de datos inicial está formado por 108 mediciones tomadas sobre 618 individuos, es decir, se tienen 108 variables y 618 observaciones. Los datos están anonimizados, por lo que no presentan ninguna información o valor que permitiese identificar al paciente. Esto permite el uso del resto de información sin comprometer la privacidad de los individuos involucrados.

Las variables se dividen en dos subconjuntos: dependientes o respuesta e independientes.

Las variables dependientes son cuatro, todas ellas binarias, y corresponden a los valores que debe predecir el modelo. Juntas forman el índice de riesgo del paciente:

- **Mortalidad**
  Indica si el paciente fallece o no.

- **Hospital**
  Indica si el paciente es hospitalizado o no.

- **UVI**
  Indica si la hospitalización es en la Unidad de Vigilancia Intensiva.

- **VMH**
  Indica si el paciente precisa de ventilación mecánica una vez está en el hospital.

La predicción de estas cuatro variables implica 2^4 posibles combinaciones de valores, es decir, 16 categorías si las variables se predicen conjuntamente.

El subconjunto de 104 variables independientes se divide en diferentes categorías según el tipo de medición:

- **Temporales**
  En este grupo encontramos la variable *fecha*, que hace referencia al día en que se realizó el avisó en formato YYYY-MM-DD, relacionada con otras cuatro variables epidemiológico demográficas correspondientes a las horas en formato HH:mm de: activación del aviso, llegada del equipo médico al lugar, traslado y llegada al hospital. También engloba las variables de fecha del alta y fecha del fallecimiento (en caso de existir). Dentro de esta categoría aunque de carácter cuantitativo y no en formato temporal, encontramos las variables *días exitus*, que indica cuántos días ha estado ingresado el paciente hasta su fallecimiento, y *días ingreso*, que indica cuántos días ha estado ingresado el paciente hasta que se le da el alta, y *días UCI*, que indica el número de días que ha estado ingresado en la Unidad de Cuidados Intensivos.

- **Edad**
  Variable numérica que indica la edad que posee el paciente cuando es atendido. El rango de edad de los pacientes va desde los 18 años hasta los 95.

- **Sexo**
  Variable categórica que indica el género del paciente, siendo “1” varón y “2” mujer.

- **Constantes vitales de forma basal**
  En esta categoría encontramos 10 mediciones numéricas relacionadas con la toma de valores como la presión sistólica y diastólica, la cantidad de O_2 en respiración o el GCS (Score Coma Glasgow [1] entre otras). Estas medidas se toman dos veces sobre cada paciente, una en la ambulancia al llegar al lugar del aviso, y otra una vez se llega al hospital, por lo que en realidad representan 20 variables.

- **Valores analíticos en ambulancia**
  Estas variables se corresponden con 24 mediciones numéricas como el pH, la glucosa en sangre o el nivel de urea, todas ellas fruto de análisis realizados en la ambulancia.

[1] Herramienta de evaluación neurológica que se utiliza para medir el nivel de conciencia de una persona después de una lesión cerebral o trauma.

- **VM**
  Variable binaria que indica si el paciente ha precisado o no ventilación mecánica en el lugar de asistencia.

- **Medicación**
  Este subconjunto de variables engloba 7 categóricas que indican el medicamento o medicamentos que se ha proporcionado al paciente, y una octava variable binaria que indica si se le ha proporcionado un antídoto o no.

- **Descripciones**
  Se corresponde con 11 variables: una de ellas numérica, que identifica en un rango del 1 al 9 el tipo de intoxicación que presenta el paciente, y otras 10 categóricas que describen diversas lesiones que además presenta o circunstancias específicas a tener en cuenta.

- **Comorbilidades**[2]
  Este subconjunto está formado por 19 variables binarias que identifican la presencia o ausencia de diversas patologías previas que puede presentar el paciente antes de ser atendido por intoxicación. También pertenece a este subconjunto la variable numérica *CACI* (*Charlson Comorbidity Index*), un índice basado en una lista de 19 condiciones médicas crónicas, como la diabetes o el cáncer entre otras, que asigna un valor numérico a cada una en función de su gravedad y su impacto en la mortalidad.

- **Triaje**
  Variable numérica que mide la prioridad que presenta el paciente en urgencias, siendo 1 la máxima y 3 la mínima.

- **Vasoactivos**
  En este subconjunto se engloban tres variables, una binaria que mide la presencia o no de vasoactivos, una categórica que indica de qué vasoactivo se trata y una numérica con la dosis correspondiente al vasoactivo.

- **Mortalidades**
  Además de la variable respuesta *Mortalidad*, el conjunto de datos presenta otras dos variables binarias, una que indica si el paciente fallece en el hospital y otra que indica si lo hace fuera del mismo.

5.2. Análisis del conjunto de datos

El análisis del conjunto de datos se ha realizado con la herramienta RStudio [17]. En el análisis de datos, es común clasificar los datos en dos categorías principales: categóricos y numéricos. Los datos categóricos son aquellos que representan características o atributos que no tienen un valor numérico, mientras que los datos numéricos son aquellos que representan valores cuantitativos, tanto enteros como continuos.

En el análisis preliminar, se analizan estos dos tipos de datos por separado, ya que requieren diferentes técnicas de análisis. En el caso de los datos categóricos, se aplican procedimientos de análisis estadístico descriptivo, como la frecuencia y el porcentaje, para resumir y visualizar los datos. Además, se pueden utilizar pruebas estadísticas como el χ^2 (Chi-cuadrado) para determinar si

[2]Presencia simultánea de dos o más enfermedades o trastornos en una misma persona.

existe una relación significativa entre dos o más variables categóricas. Por otro lado, para los datos numéricos, se utilizan otros estadísticos como la media y la desviación estándar que permiten resumir los datos y analizar su distribución. Además, se pueden utilizar otras medidas estadísticas como la correlación para detectar relaciones entre dos o más variables numéricas.

Para el análisis categórico del conjunto de datos se ha decidido prescindir de algunas variables redundantes que no aportaban información y dificultaban el tratamiento de los datos. Es el caso de algunas de las variables pertenecientes a los grupos de medicaciones o descripciones. En el caso de las medicaciones, en vez de seleccionar los nombres de todos los medicamentos, se ha decidido conservar únicamente la variable binaria que indicaba si se trataba o no de un antídoto, ya que las otras variables no tenían un rango de categorías limitado y además, al ser datos tomados manualmente en formato texto, presentaban desigualdades para valores que se supone deberían ser idénticos. En el caso de las descripciones la problemática era similar, repitiéndose los datos en muchas de las observaciones o suponiendo valores ausentes en otras, por lo que finalmente se decide mantener solo la variable "tipo de intoxicación", variable categórica con un rango limitado del 1 al 9.

Análisis variables categóricas

El subconjunto definitivo de variables categóricas se resume en *Sexo*, *VM*, *Antidoto*, *Tipo de intoxicación*, *Triaje*, *GCS.0.1*, *ICC*, *IAM*, *EVP*, *ACV*, *Hemiplejia*, *EPOC*, *DM*, *DM.con.lesion*, *Enf Renal*, *Enf Hepa Leve*, *Enf Hepa Grave*, *Úlcera*, *SIDA*, *Linfoma*, *Leucemia*, *Metástasis*, *Cáncer no metastásico*, *Enf T.Conectivo*, *Demencia*, *CACI*, *Vasoactivosh*, *VMH*, *UVI*, *Mortalidad* y *Hospital*. A continuación se describen las variables junto a sus respectivas distribuciones:

- Sexo: indica el género del paciente.

Varón	Mujer
307	311

Cuadro 5.1: Frecuencias por sexo

Se observa un mayor número de mujeres intoxicadas que de hombres, pero la diferencia es mínima, por lo que para dicha variable los resultados no se encontrarán sesgados.

- Ventilación mecánica (VM): indica si el paciente requiere de asistencia respiratoria en el momento en el que es atendido.

No VM	Sí VM
557	61

Cuadro 5.2: Frecuencias por VM

La mayoría de pacientes atendidos no requieren ventilación mecánica antes de llegar al hospital.

- Antídoto: indica si el paciente ha requerido de la ingesta de algún antídoto para tratar los efectos de su intoxicación.

No antídoto	Sí antídoto
313	305

Cuadro 5.3: Frecuencias por Antídoto

Hay un menor número de pacientes que requieren tomar un antídoto que aquellos no lo requieren.

- Tipo de intoxicación: indica el identificador numérico correspondiente a la categoría de intoxicación que sufre el paciente, en un rango de 1 a 9.

1	2	3	4	5	6	7	8	9
227	36	148	17	34	28	24	25	79

Cuadro 5.4: Frecuencias por Tipo de intoxicación

Se observa que la mayoría de los pacientes sufren intoxicación de tipo 1.

- Triaje: identificador numérico de la gravedad que reviste el paciente al ser atendido en la ambulancia que permite determinar su grado de prioridad al llegar a urgencias. Los valores habituales de esta escala tienen un rango de [1-5], pero en este contexto se entiende que los dos primeros grados no requieren del uso de ambulancia y que el paciente puede llegar a urgencias por su propio pie, por lo que no están presentes en el conjunto de datos utilizado en el que todos los pacientes requieren de una ambulancia.

1	2	3
51	146	421

Cuadro 5.5: Frecuencias por Triaje

Se observa que la mayoría de pacientes no revisten una gravedad prioritaria en su llegada a urgencias.

- GCS.0.1: escala de coma de Glasgow diseñada para evaluar de manera práctica el nivel de estado de alerta en los seres humanos y su reacción frente a estímulos. En el caso de esta variable concreta, se refiere al grado de apertura ocular del paciente frente a estímulos.

1	2	3	4
71	45	130	372

Cuadro 5.6: Frecuencias por GCS.0.1

La mayoría de pacientes presentan un valor 4 en la escala mencionada, lo que implica la no reacción frente a estímulos.

- ICC: marcador que indica si el paciente presenta insuficiencia cardíaca o no.

Sí	No
25	593

Cuadro 5.7: Frecuencias por ICC

La mayoría de pacientes no presentan esta patología.

- Patologías previas: diversas patologías previas que presentan o no los pacientes intoxicados y que pueden agravar su situación.

Patología	Sí	No
IAM (infarto de miocardio)	30	588
EVP (enfermedad vascular periférica)	41	577
ACV (accidente cerebrovascular)	16	602
Heiplejia	7	611
EPOC (enfermedad pulmonar obstructiva crónica)	91	527
DM (diabetes mellitus)	37	581
DM con lesión	21	597
Enfermedad renal	14	604
Enfermedad hepática leve	35	583
Enfermedad hepática grave	37	581
Úlcera	47	571
SIDA	18	600
Linfoma	2	616
Leucemia	2	616
Metástasis	5	613
Cáncer no metastásico	38	580
Enfermedad del trastorno conectivo	30	588
Demencia	19	599

Cuadro 5.8: Frecuencias por patologías previas

Se observa que estas categorías tienden a estar desbalanceadas.

- Ingesta de vasoactivosH: indicador de la necesidad de proporcionar al paciente intoxicado algún tipo de vasodilatador o vasoconstrictor en función del tipo de intoxicación cuando llega al hospital.

Sí	No
28	590

Cuadro 5.9: Frecuencias por VasoactivosH

- CACI (índice de comorbilidad de Charlson): índice que relaciona la mortalidad del paciente a largo plazo con la comorbilidad que presenta.

0	1	2	3	4	7	otro
343	106	58	41	24	11	35

Cuadro 5.10: Frecuencias por CACI

Respecto a las cuatro variables respuesta, resulta de vital importancia el estudio de las frecuencias de cada clase, ya que si las clases están desbalanceadas, puede influir directamente en el comportamiento del modelo:

- Mortalidad: variable que indica si el paciente fallece o no antes de abandonar el hospital.

Fallece	Sobrevive
598	20

Cuadro 5.11: Frecuencias por Mortalidad

La mayoría de pacientes sobreviven a la intoxicación.

- VMH (Ventilación mecánica en hospital): variable que indica si el paciente va a necesitar o no de un aparato para suplir su función respiratoria de manera artificial.

No	Sí
552	66

Cuadro 5.12: Frecuencias por VMH

La mayoría de pacientes no requiere de ventilación mecánica una vez llegan al hospital.

- Hospital: variable que indica si los pacientes van a necesitar o no ser ingresados en el hospital al llegar a urgencias.

No	Sí
406	212

Cuadro 5.13: Frecuencias por Hospital

La mayoría de pacientes no requiere ingreso en el hospital.

- UVI (unidad de vigilancia intensiva): variable que indica si los pacientes van a necesitar o no ser ingresados en el área de vigilancia intensiva del hospital debido a una mayor gravedad de su estado.

No	Sí
523	95

Cuadro 5.14: Frecuencias por UVI

La mayoría de pacientes no ingresan en UVI.

Como se puede observar, las clases están desbalanceadas en todas las variables respuesta, por lo que el modelo puede tener dificultades para distinguir las clases minoritarias y, por lo tanto, puede tener un sesgo hacia las clases mayoritarias. En otras palabras, el modelo puede ser más preciso para predecir la clase mayoritaria, mientras que la predicción de la clase minoritaria puede ser menos precisa. Este problema se abordará más adelante.

Para analizar las posibles relaciones entre las variables independientes categóricas y, en caso de ser redundantes poder decidir si conviene o no añadirlas a un futuro modelo, se va a aplicar un análisis de correspondencias múltiples (ACM), una técnica exploratoria de análisis multivariante que permite analizar simultáneamente múltiples variables categóricas y descubrir patrones y relaciones entre ellas. El cálculo del ACM implica la creación de una tabla de contingencia que muestra la frecuencia con la que aparecen combinaciones de categorías de las diferentes variables. A partir de esta tabla, se puede calcular la matriz de inercia, que es una medida de la variabilidad total de los datos. La matriz de inercia se utiliza para calcular los valores propios y los vectores propios, que se utilizan para determinar las dimensiones principales del ACM. Los valores de las coordenadas de las categorías de las variables en cada dimensión se calculan como la suma ponderada de las frecuencias de las categorías en cada dimensión. Para la explicación de las relaciones entre estas variables, se han seleccionado únicamente las dos primeras dimensiones, ya que estas representan las relaciones más importantes.

La primera dimensión explica la mayor cantidad de variabilidad en los datos, de forma que los valores más altos en la primera dimensión indican una asociación positiva entre las variables, mientras que los valores más bajos indican una asociación negativa.

La segunda dimensión representa la relación secundaria más importante entre las variables, después de la primera dimensión. Los valores más altos en la segunda dimensión indican una asociación positiva entre las variables que no se captura en la primera dimensión, mientras que los valores más bajos indican una asociación negativa.

	Dim 1	Dim 2
Sexo	0.017163065	2.792582e-02
VM	0.225006787	3.494160e-01
Antidoto	0.018874163	7.054874e-02
Tipo.de.intoxicación	0.202515016	1.621228e-01
Triaje	0.229818528	3.531951e-01
GCS.0.1	0.231563736	3.402929e-01
ICC	0.135819107	1.293727e-01
IAM	0.096850224	4.064665e-02
EVP	0.068046219	5.299594e-03
ACV	0.066361387	1.017660e-01
Hemiplejia	0.034440423	3.315944e-02
EPOC	0.146672543	1.000410e-02
DM	0.015332757	1.404429e-04
DM.con.lesion	0.079493416	1.214177e-01
Enf..Renal	0.103789851	7.567377e-02
Enf..Hepa.leve	0.032857551	1.583738e-02
Enf.Hepa.grave	0.185774380	1.254634e-03
Úlcera	0.072740611	1.090093e-02
SIDA	0.187320428	4.460347e-02
Linfoma	0.009772159	4.259327e-02
Leucemia	0.066883490	3.479869e-02
Metástasis	0.025765905	5.904001e-05
Cáncer.no.metastásico	0.072517371	9.431123e-02
Enf..T..conectivo	0.034363901	1.325846e-02
Demencia	0.031324395	8.351584e-02
CACI	0.726152894	3.344924e-01
VasoactivosH	0.148427784	1.025754e-01

Figura 5.1: Asociaciones entre las variables categóricas independientes

Tal y como se puede observar en la figura 5.1 el valor más alto en la Dimensión 1 corresponde a la variable *CACI*, mientras que el valor más alto en la Dimensión 2 corresponde a la variable *Triaje*. Por lo tanto, se podría decir que la variable *CACI* tiene una fuerte asociación con la Dimensión 1, mientras que la variable *Triaje* tiene una fuerte asociación con la Dimensión 2. En términos más generales, un valor alto en una dimensión indica una mayor asociación entre las categorías de las variables incluidas en esa dimensión. En otras palabras, las categorías de las variables que tienen valores similares en una dimensión están más relacionadas entre sí que las categorías de las variables que tienen valores diferentes en esa dimensión. Por ejemplo, si se lleva a cabo un análisis diferenciando por las categorías de cada variable, se puede ver que las categorías de la variable *VM* tienen valores similares en ambas dimensiones, lo que indica una asociación fuerte entre estas categorías. En contraste, las categorías de la variable *Sexo* tienen valores bajos en ambas dimensiones, lo que indica una menor asociación entre estas categorías.

Esta información se puede ver de manera más clara también gráficamente.

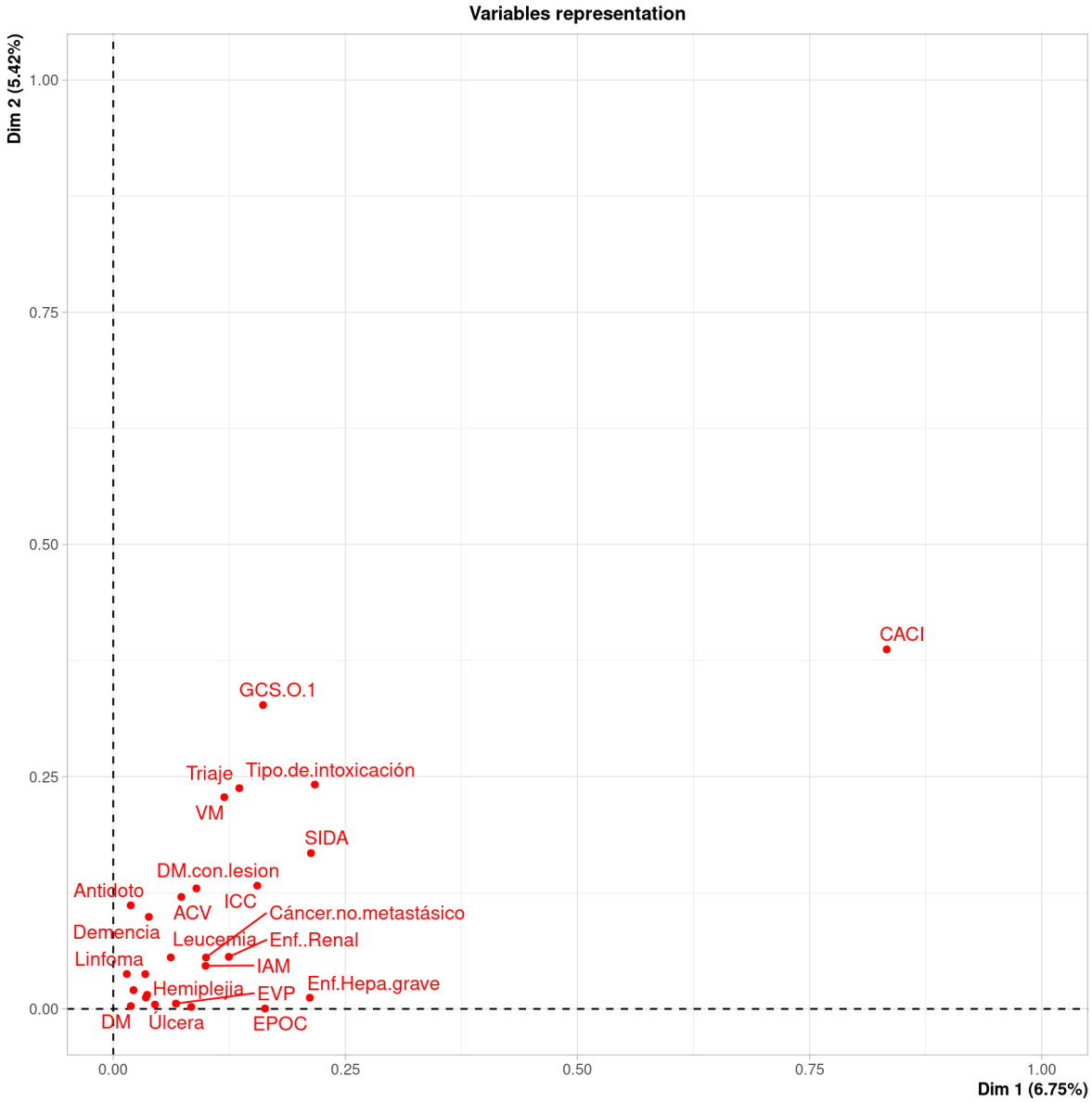


Figura 5.2: ACM variables

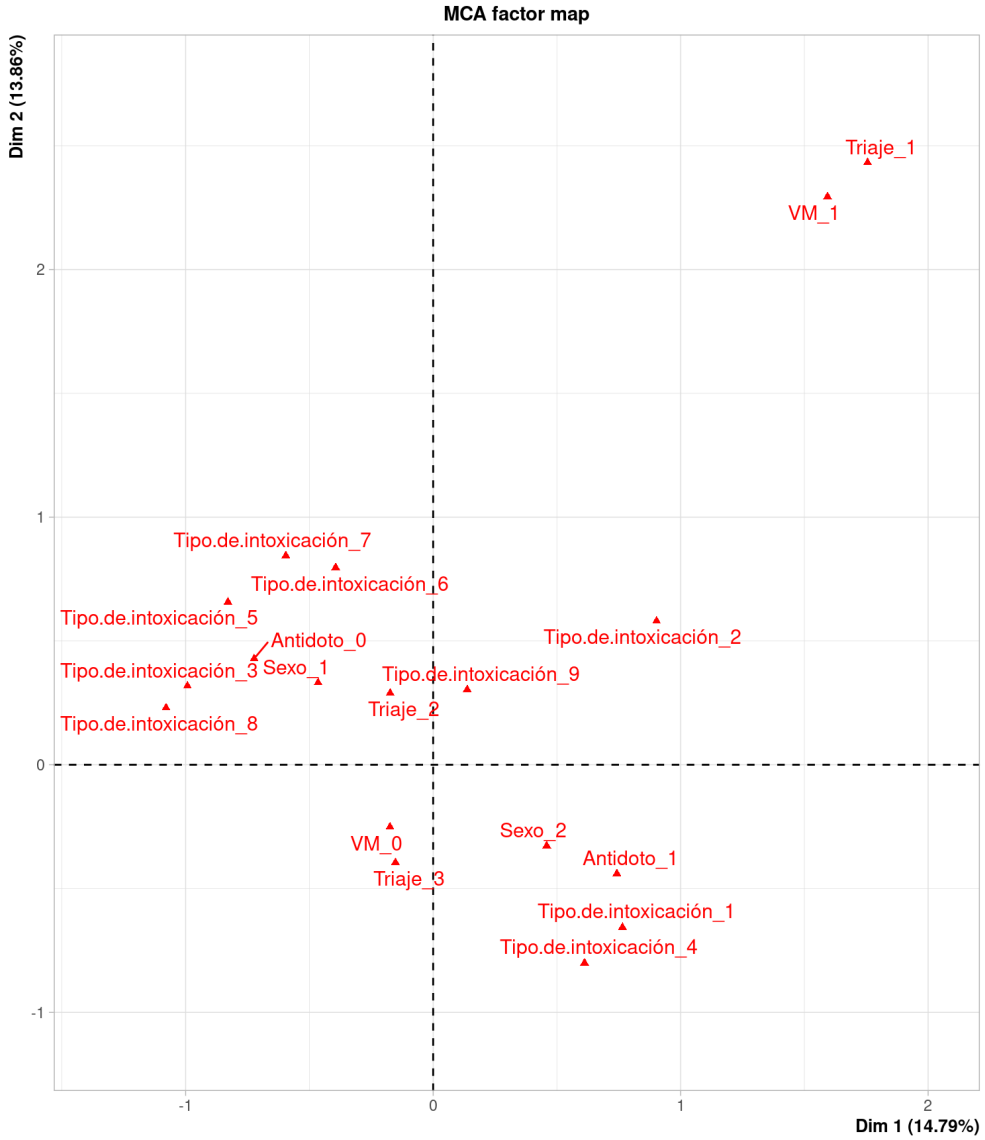
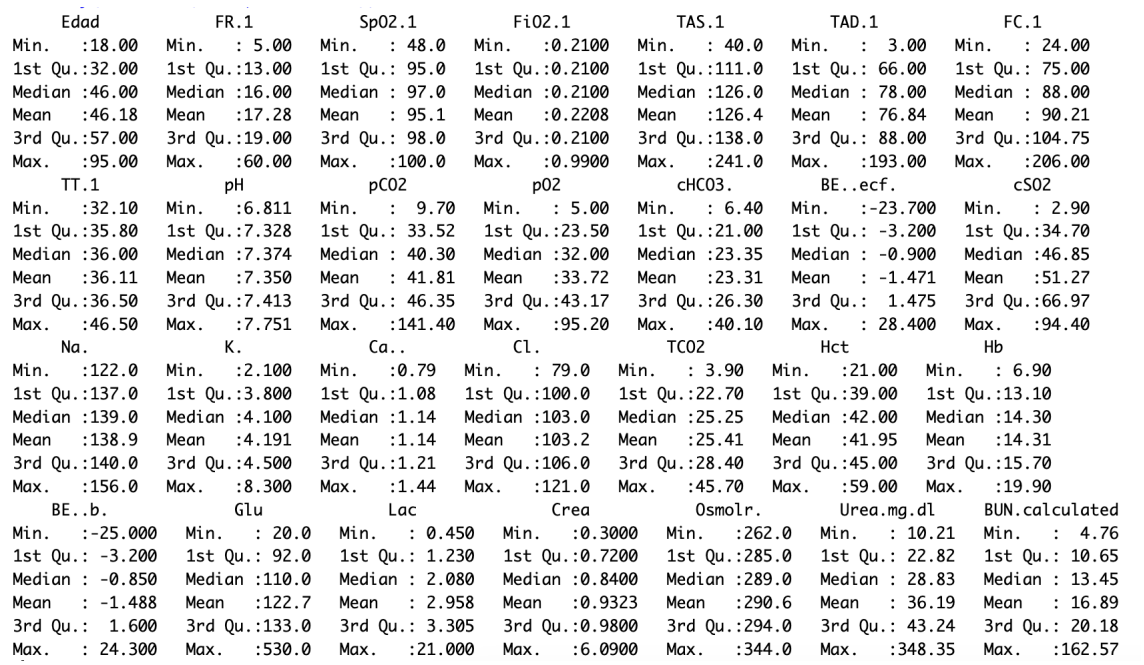


Figura 5.3: ACM variables por categorías

Tal y como se había indicado anteriormente, en las figuras 5.2 y 5.3 se puede observar que las variables *VM* y *Triaje* presentan los valores más altos en la Dimensión 2, lo que sugiere que están más relacionadas entre sí en términos de sus categorías. Por otro lado, la variable *Metástasis* tiene el valor más bajo en la Dimensión 2, lo que sugiere que tiene una relación más débil con las otras variables categóricas.

Análisis variables numéricas

A continuación, se centra el análisis exploratorio en las variables numéricas, comenzando por un estudio de sus distribuciones. En la figura 5.4 se observa una muestra de dichas distribuciones.



```
      Edad            FR.1           SpO2.1          FiO2.1           TAS.1            TAD.1             FC.1
 Min.   :18.00   Min.   : 5.00   Min.   : 48.0   Min.   :0.2100   Min.   : 40.0   Min.   :  3.00   Min.   : 24.00
 1st Qu.:32.00   1st Qu.:13.00   1st Qu.: 95.0   1st Qu.:0.2100   1st Qu.:111.0   1st Qu.: 66.00   1st Qu.: 75.00
 Median :46.00   Median :16.00   Median : 97.0   Median :0.2100   Median :126.0   Median : 78.00   Median : 88.00
 Mean   :46.18   Mean   :17.28   Mean   : 95.1   Mean   :0.2208   Mean   :126.4   Mean   : 76.84   Mean   : 90.21
 3rd Qu.:57.00   3rd Qu.:19.00   3rd Qu.: 98.0   3rd Qu.:0.2100   3rd Qu.:138.0   3rd Qu.: 88.00   3rd Qu.:104.75
 Max.   :95.00   Max.   :60.00   Max.   :100.0   Max.   :0.9900   Max.   :241.0   Max.   :193.00   Max.   :206.00
      TT.1             pH             pCO2             pO2            cHCO3.         BE..ecf.            cSO2
 Min.   :32.10   Min.   :6.811   Min.   :  9.70   Min.   : 5.00   Min.   : 6.40   Min.   :-23.700   Min.   : 2.90
 1st Qu.:35.80   1st Qu.:7.328   1st Qu.: 33.52   1st Qu.:23.50   1st Qu.:21.00   1st Qu.: -3.200   1st Qu.:34.70
 Median :36.00   Median :7.374   Median : 40.30   Median :32.00   Median :23.35   Median : -0.900   Median :46.85
 Mean   :36.11   Mean   :7.350   Mean   : 41.81   Mean   :33.72   Mean   :23.31   Mean   : -1.471   Mean   :51.27
 3rd Qu.:36.50   3rd Qu.:7.413   3rd Qu.: 46.35   3rd Qu.:43.17   3rd Qu.:26.30   3rd Qu.:  1.475   3rd Qu.:66.97
 Max.   :46.50   Max.   :7.751   Max.   :141.40   Max.   :95.20   Max.   :40.10   Max.   : 28.400   Max.   :94.40
      Na.              K.              Ca..            Cl.            TCO2            Hct              Hb
 Min.   :122.0   Min.   :2.100   Min.   :0.79   Min.   : 79.0   Min.   : 3.90   Min.   :21.00   Min.   : 6.90
 1st Qu.:137.0   1st Qu.:3.800   1st Qu.:1.08   1st Qu.:100.0   1st Qu.:22.70   1st Qu.:39.00   1st Qu.:13.10
 Median :139.0   Median :4.100   Median :1.14   Median :103.0   Median :25.25   Median :42.00   Median :14.30
 Mean   :138.9   Mean   :4.191   Mean   :1.14   Mean   :103.2   Mean   :25.41   Mean   :41.95   Mean   :14.31
 3rd Qu.:140.0   3rd Qu.:4.500   3rd Qu.:1.21   3rd Qu.:106.0   3rd Qu.:28.40   3rd Qu.:45.00   3rd Qu.:15.70
 Max.   :156.0   Max.   :8.300   Max.   :1.44   Max.   :121.0   Max.   :45.70   Max.   :59.00   Max.   :19.90
     BE..b.            Glu             Lac              Crea           Osmolr.        Urea.mg.dl     BUN.calculated
 Min.   :-25.000   Min.   : 20.0   Min.   : 0.450   Min.   :0.3000   Min.   :262.0   Min.   : 10.21   Min.   :  4.76
 1st Qu.: -3.200   1st Qu.: 92.0   1st Qu.: 1.230   1st Qu.:0.7200   1st Qu.:285.0   1st Qu.: 22.82   1st Qu.: 10.65
 Median : -0.850   Median :110.0   Median : 2.080   Median :0.8400   Median :289.0   Median : 28.83   Median : 13.45
 Mean   : -1.488   Mean   :122.7   Mean   : 2.958   Mean   :0.9323   Mean   :290.6   Mean   : 36.19   Mean   : 16.89
 3rd Qu.:  1.600   3rd Qu.:133.0   3rd Qu.: 3.305   3rd Qu.:0.9800   3rd Qu.:294.0   3rd Qu.: 43.24   3rd Qu.: 20.18
 Max.   : 24.300   Max.   :530.0   Max.   :21.000   Max.   :6.0900   Max.   :344.0   Max.   :348.35   Max.   :162.57
```

Figura 5.4: Distribución de muestra de variables numéricas

Los valores que resultan de mayor utilidad en este caso son los de la media y la desviación típica, ya que ayudan a detectar posibles valores atípicos o errores en el conjunto de datos, tal y como se mostrará más adelante.

De manera análoga a la búsqueda de relaciones entre las variables categóricas, se pretende detectar estos patrones en las numéricas. Para ello, se realiza un análisis de correlaciones. De forma previa a dicho análisis, se decide comprobar si las variables cumplen la hipótesis de normalidad, ya que esto determinará el coeficiente a utilizar para el cálculo de las correlaciones.

Respecto a la hipótesis de normalidad, se contrasta utilizando el estadístico de Shapiro-Wilk:

$$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5.1}$$

donde $x_{(i)}$ es la i-ésima observación ordenada, a_i son los coeficientes de la distribución normal estandarizada y $\bar{x}$ es la media de las observaciones.

Las hipótesis del test de normalidad para las variables independientes numéricas son:

H_0 : La muestra sigue una distribución normal
H_1 : La muestra no sigue una distribución normal

El p-valor se calcula a través de la aproximación de la distribución del estadístico del test por una distribución chi-cuadrado con k grados de libertad, donde k es el número de observaciones.

Entonces, el p-valor se calcula como:

$$pvalor = 1 - \sum_{i=1}^{n} a_i \left(\frac{\sum_{j=1}^{n} c_{ij} x_j}{s} \right)^2 \tag{5.2}$$

donde n es el tamaño de la muestra, x_j son los valores ordenados de menor a mayor, a_i y c_{ij} son coeficientes obtenidos previamente y s es la desviación estándar de la muestra.

Si se aplica dicho test a las variables numéricas del conjunto de datos, se observa que el pvalor es menor que 0.05 (nivel de confianza) en todos los casos, pudiendo rechazar la hipótesis nula para todas ellas. Para tratar de hacer que los datos aproximen a una distribución Normal, se han realizado diversas transformaciones típicamente utilizadas para este fin.

Transformación logarítmica:

$$f(x) = \log(x) \tag{5.3}$$

Transformación inversa:

$$f(x) = \frac{1}{x} \tag{5.4}$$

Transformación raíz cuadrada:

$$f(x) = \sqrt{x} \tag{5.5}$$

Transformación Box-Cox:

$$f(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{si } \lambda \neq 0 \\ \log(x) & \text{si } \lambda = 0 \end{cases} \tag{5.6}$$

Donde x son los datos originales y λ es el parámetro a estimar por el método de Box-Cox.

Tras cada una de las transformaciones, se repite el test de Shapiro-Wilk, obteniéndose un resultado similar al anterior. Dado que la hipótesis de normalidad no se cumple, se decide elegir un coeficiente que permita calcular la correlación entre las variables numéricas sin necesidad de que estas sigan una distribución Normal. El método más utilizado en estos casos es la correlación de rango, una medida de la fuerza y dirección de una relación entre dos variables. A diferencia de la correlación de Pearson, que se basa en la covarianza de las variables, la correlación de rango se basa en la comparación de los rangos de las observaciones de cada variable. Es decir, se basa en la clasificación de los valores de cada variable en orden ascendente o descendente, para su posterior comparación entre las posiciones relativas de cada valor. El coeficiente de correlación de rango más comúnmente utilizado es el coeficiente de correlación de rango de Spearman (5.7).

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)} \tag{5.7}$$

Donde ρ es el coeficiente de correlación de Spearman, n es el número de observaciones, d_i son las diferencias de los rangos de las observaciones para cada pareja de variables.

Si se realiza dicho análisis de correlaciones, tanto numérica como gráficamente, se detectan una serie de relaciones entre las variables. Debido al elevado número de variables se ha elegido una muestra representativa de los resultados (figuras 5.5, 5.6).

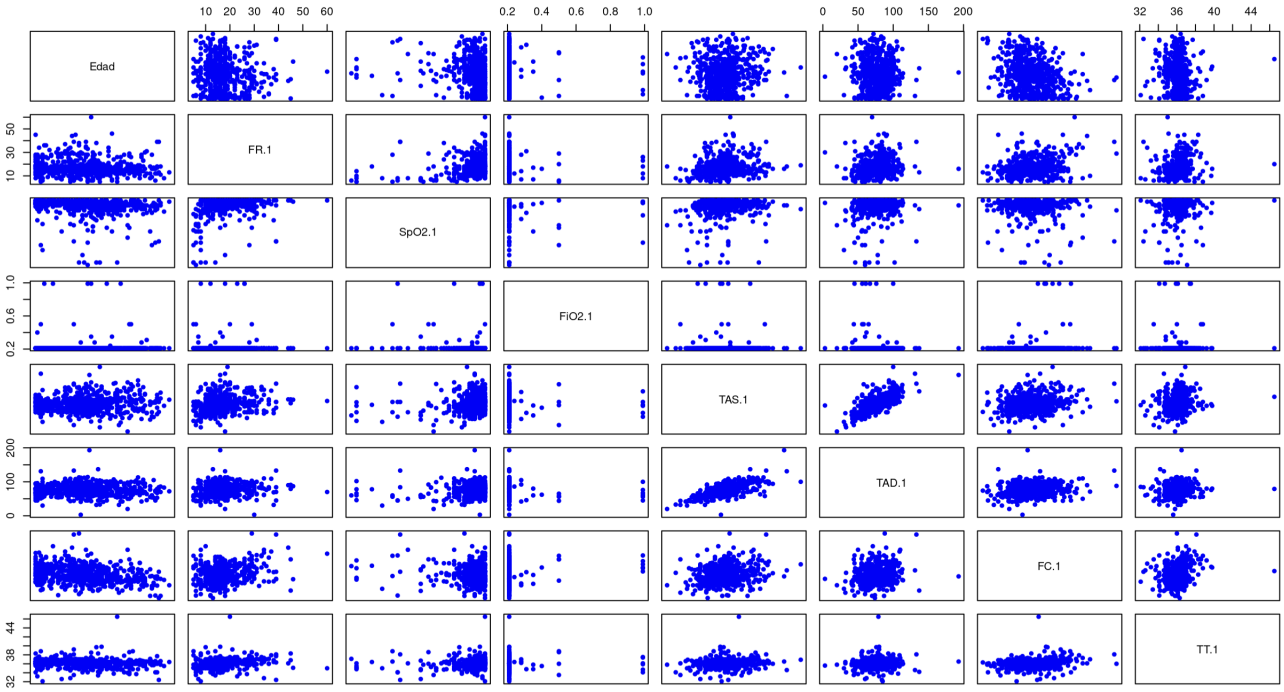


Figura 5.5: Muestra de correlaciones - gráfico

Crea	1.0000000000	0.17860498	0.0187116255	0.19798237	0.198079414	0.479035305	0.479035305	-0.172564450
Agap	0.1786049844	1.00000000	0.6657641436	0.98699978	0.303807283	0.119310792	0.119310792	0.026782417
Agap.Uri.	0.0187116255	0.66576414	1.0000000000	0.67852199	0.390462003	0.018445615	0.018445615	0.026641407
AGapK	0.1979823743	0.98699978	0.6785219935	1.00000000	0.311733718	0.140252424	0.140252424	0.028750065
Osmolr.	0.1980794144	0.30380728	0.3904620032	0.31173372	1.000000000	0.313295984	0.313295984	0.217915345
Urea.mg.dl	0.4790353051	0.11931079	0.0184456154	0.14025242	0.313295984	1.000000000	1.000000000	0.718399567
BUN.calculated	0.4790353051	0.11931079	0.0184456154	0.14025242	0.313295984	1.000000000	1.000000000	0.718399567
BUN.Crea	-0.1725644500	0.02678242	0.0266414070	0.02875006	0.217915345	0.718399567	0.718399567	1.000000000

Figura 5.6: Muestra de correlaciones - matriz de correlaciones

En el gráfico de pares (figura 5.5), cada punto representa la relación entre dos variables. La posición en el eje x representa la primera variable y la posición en el eje y representa la segunda variable. Cuando la nube de puntos es aleatoria y no sigue ninguna forma, esto indica que no hay una relación clara entre las dos variables. En este caso, el coeficiente de correlación entre las variables será cercano a cero, es decir, las variables son incorreladas. Si la nube de puntos se agrupa entorno

a una línea recta con una pendiente positiva, eso indica una relación positiva entre las dos variables. Esto significa que a medida que aumenta una variable, la otra también tiende a aumentar. Si por el contrario la nube de puntos forma una línea recta con una pendiente negativa, esto indica una relación negativa entre las dos variables.

Una vez aplicado dicho análisis al conjunto de datos del estudio, se puede observar cómo gráficamente (5.5) ciertas variables presentan correlación positiva. Es el caso de variables como *TAS.1* y *TAD.1*, ya que una mide la tensión sistólica y la otra la diastólica, dicha relación se explica por la naturaleza de los datos. Numéricamente (5.6) también se pueden encontrar correlaciones señalables, como es el caso de las variables *BUN.calculate* y *BUN.Crea*, fruto también del origen de las mismas, ya que el cálculo de ambas se realiza utilizando un valor común. Similar es el caso de las variables *Agap* y *Agap.Uri*. Este análisis se tendrá en cuenta a la hora de seleccionar las variables que se incluirán en el modelo, evitando así redundancias y disminuyendo el coste computacional.

Una vez analizados los datos, se posee la información necesaria para llevar a cabo la limpieza del conjunto de datos inicial, así como para seleccionar únicamente aquellas variables que aporten información a un futuro modelo y que no resulten redundantes.

5.3. Limpieza y procesamiento del conjunto de datos

La limpieza de datos es un proceso crítico en el análisis de datos y modelado. Los datos pueden contener errores, inconsistencias, valores atípicos y valores ausentes, entre otros problemas, que pueden comprometer la calidad y la precisión de cualquier análisis o modelo posterior. Por lo tanto, limpiar y preprocesar los datos es una tarea esencial antes de aplicar cualquier técnica de análisis o modelo. Este proceso incluye la identificación y eliminación de datos incompletos, inconsistentes o incorrectos, pero también puede involucrar la transformación de datos en formatos y tipos adecuados para el análisis posterior.

El primer paso para seleccionar observaciones que realmente aporten información y consistencia a un modelo es eliminar aquellas con un alto porcentaje de valores ausentes. En la figura 5.7 se puede observar dicho porcentaje para cada variable del conjunto de datos.

Fecha	AMBULANCIA	Edad	Sexo	H..Act.	H..Asi.
0.000000000	1.000000000	0.000000000	0.000000000	0.000000000	0.000000000
H..Tra.	H..Lle.	FR.1	SpO2.1	FiO2.1	TAS.1
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
TAD.1	FC.1	TT.1	GCS.O.1	GCS.V.1	GCS.M.1
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
pH	pCO2	pO2	cHCO3.	BE..ecf.	cSO2
0.000000000	0.001618123	0.000000000	0.000000000	0.000000000	0.000000000
Na.	K.	Ca..	Cl.	TCO2	Hct
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
Hb	BE..b.	Glu	Lac	Crea	Agap
0.001618123	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
Agap.Uri.	AGapK	Osmolr.	Urea.mg.dl	BUN.calculated	BUN.Crea
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
VM	Antidoto	Med..1	Med..2	Med..3	Med..4
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
Med..5	Med..6	Med..7	Dg..1	Descripción..1	Dg..2
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.313915858
Descripción..2	Dg..3	Descripción..3	Tipo.de.intoxicación	Mecanismo.lesional	HOSPITAL
0.000000000	0.891585761	0.000000000	0.000000000	0.000000000	1.000000000
ICC	IAM	EVP	ACV	Hemiplejia	EPOC
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
DM	DM.con.lesion	Enf..Renal	Enf..Hepa.leve	Enf.Hepa.grave	Úlcera
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
SIDA	Linfoma	Leucemia	Metástasis	Cáncer.no.metastásico	Enf..T..conectivo
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
Demencia	CACI	Triaje	FR.H	SpO2.H	FiO2.H
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
TAS.H	TAD.H	FC.H	TT.H	GCS.O.H	GCS.V.H
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
GCS.M.H	VMH	VasoactivosH	Tipo	Dosis	Descripción..1.1
0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
Descripción..2.1	Descripción..3.1	Hospital	UVI	Días.UCI	Fecha.Alta
0.000000000	0.000000000	0.000000000	0.000000000	0.846278317	0.000000000
Mort..Hospi	Mort..Extr.	Fecha.exitus	Dias.exitus	Mortalidad	Días.ingreso
0.000000000	0.000000000	0.000000000	0.967637540	0.000000000	0.000000000

Figura 5.7: Porcentaje valores ausentes por variable

Una vez calculado el porcentaje de valores ausentes en cada variable, se decide eliminar aquellas en las que estos valores suponen un 100 % : *Ambulancia* y *Hospital*. Por su cercanía a dicho valor, se prescinde también de *Días.exitus*. Respecto al resto de variables que presentan valores ausentes, aunque en menor cantidad, como *pCO2* o *Hb* se opta por aplicar otra solución. Aunque algunas observaciones presentan valores ausentes en ellas, eliminarlas supondría una pérdida de información innecesaria, por lo que en su lugar se ha optado por sustituir esos valores ausentes por la media de la variable, evitando de esta forma posibles sesgos o desviaciones en los datos.

En el caso de las variables textuales que también presentaban valores ausentes, *Dg.1*, *Dg.2*, *Dg.3* (correspondientes a descripciones), se ha optado por eliminarlas directamente, ya que sus valores no forman parte de un conjunto finito de categorías. Además, los valores han sido tomados manualmente, por lo que también existe inconsistencia en los formatos. Únicamente permanecerán en el conjunto de datos aquellas variables descriptivas que llevan asociada el tipo de intoxicación y que sí pertenecen a un conjunto finito de categorías identificadas numéricamente.

Respecto al formato y codificación de algunas variables, se han detectado más inconsistencias en los datos. Es el caso de algunas variables numéricas como *BUN.Crea* y *TT.H*, en las que algunos valores continuos separaban la parte decimal con coma y otros con punto, dificultando los análisis

y generando errores en los mismos. Para solucionar este problema, se decide unificar el formato tomando como separador decimal el carácter punto.

La siguiente técnica aplicada para detectar posibles inconsistencias en las variables numéricas es la búsqueda de valores atípicos, ya que en algunos casos pueden suponer errores a la hora de transcribir los datos. Antes de analizar sus valores, se estandarizan las variables, reduciendo así el efecto de los valores extremos en la media y la varianza de los datos.

$$z = \frac{x - \bar{x}}{s} \tag{5.8}$$

donde z representa la variable estandarizada, x es la variable original, μ es la media de la variable x y σ es la desviación típica de la variable x.

El estándar estadístico más utilizado es suponer que una observación es atípica cuando se desvía 4 veces la desviación típica. Si se aplica esta condición a los datos estandarizados de todas las variables numéricas, se detectan varios valores que cumplen con ella. Se extrapolan los índices de estos valores a los datos originales sin normalizar, para poder comprobar en un contexto sanitario si se consideran cifras anómalas o no. Una vez contrastadas con el Doctor responsable del conjunto de datos y especialista en la materia, se determina que dichos valores sí que tienen sentido en un contexto de intoxicación, por lo que no se elimina ninguna observación.

Una vez se tienen las observaciones definitivas, es decir, depuradas, se decide realizar un análisis preliminar de la importancia que tienen el conjunto de variables actual sobre las variables respuesta: *Mortalidad*, *Hospital*, *VMH*, *UVI*.

Para ello, se aplica el algoritmo RFE (*Recursive Feature Elimination*), una técnica utilizada en el aprendizaje automático para seleccionar un subconjunto óptimo de variables de un conjunto de datos. Su objetivo es reducir la dimensión del conjunto de variables y mejorar la precisión y la eficiencia del modelo. El proceso del algoritmo RFE comienza con la construcción de un modelo utilizando todas las variables disponibles. A continuación, se asigna una puntuación de importancia para cada una de ellas y aquellas menos importantes se eliminan y se construye un nuevo modelo con el subconjunto restante. Este proceso se repite de forma recursiva hasta que se obtiene la precisión óptima.

```
> predictors(res1) #mortalidad
 [1] "BE..b."         "pH"             "Crea"           "BE..ecf."       "CACI"           "Urea.mg.dl"     "BUN.calculated"
 [8] "VasoactivosH"   "Lac"            "pCO2"           "TCO2"           "cHCO3."         "FiO2.1"         "pO2"
[15] "Hb"             "TAD.1"          "Enf..Renal"
> predictors(res2) #UVI
 [1] "VM"                   "Triaje"               "VasoactivosH"         "GCS.O.1"              "pH"
 [6] "SpO2.1"               "pCO2"                 "cHCO3."               "pO2"                  "FR.1"
[11] "BE..ecf."             "BE..b."               "TCO2"                 "ACV"                  "cSO2"
[16] "TT.1"                 "BUN.calculated"       "Urea.mg.dl"           "Crea"                 "Lac"
[21] "Tipo.de.intoxicación" "FC.1"                 "Na."                  "Enf..T..conectivo"    "Edad"
[26] "Enf..Hepa.leve"       "TAD.1"                "CACI"                 "EVP"                  "TAS.1"
[31] "DM.con.lesion"        "Antidoto"             "Cl."                  "K."                   "Enf..Renal"
[36] "Osmolr."              "Hb"                   "FiO2.1"               "Glu"                  "Cáncer.no.metastásico"
[41] "Sexo"                 "Leucemia"             "IAM"                  "DM"                   "Demencia"
[46] "Linfoma"              "Hemiplejia"           "Hct"                  "Metástasis"           "Enf.Hepa.grave"
[51] "SIDA"                 "EPOC"                 "Ca.."                 "Úlcera"               "ICC"
> predictors(res3) #VMH
[1] "VM"          "Triaje"      "VasoactivosH" "pH"          "SpO2.1"      "GCS.O.1"     "BE..b."      "pCO2"
[9] "BE..ecf."
> predictors(res4) #Hospital
[1] "VM"                   "Triaje"               "SpO2.1"               "Demencia"             "Tipo.de.intoxicación"
[6] "Lac"                  "VasoactivosH"
```

Figura 5.8: Variables independientes seleccionadas para cada variable respuesta

En la figura 5.8 se puede observar la selección de variables obtenida al aplicar el algoritmo RFE sobre cada variable respuesta por separado. Como a la hora de obtener una predicción para la variable respuesta *UVI* (res2) la mayoría de variables son importantes, se decide conservar todas en el conjunto de datos con el que se construirá el modelo.

5.4. Separación del conjunto de datos

Una vez se tiene el conjunto de datos definitivo, se divide en dos subconjuntos, uno para el entrenamiento del modelo y otro para la validación del mismo. Aunque la práctica habitual es dedicar un 20 % del conjunto completo al subconjunto de validación, como en este caso no se posee un gran número de observaciones, se ha decidido reducir ese valor, destinando únicamente 90 pacientes a dicha finalidad.

Se ha llevado a cabo una separación aleatoria partiendo del conjunto inicial completo, en el que se toma como semilla el valor 3 y se escoge una muestra aleatoria indicando el tamaño de validación. Finalmente se obtienen los dos conjuntos: el de entrenamiento con 528 observaciones y 59 variables, y el de validación, con 90 observaciones y 59 variables.

Capítulo 6

Experimentación

En este capítulo, se presenta la experimentación llevada a cabo para evaluar la creación de un modelo de clasificación de datos tabulares, concretamente una red neuronal, utilizando la librería Fastai. El objetivo principal es determinar la posible utilidad de Fastai en experimentos multicategoría con datos tabulares, para lo cuál se decide comparar la calidad de las predicciones obtenidas con modelos que predicen simultáneamente las cuatro variables respuesta en diferentes hipótesis de búsqueda de hiperparámetros, teniendo en cuenta diversos aspectos como el tamaño de la red (número de capas y neuronas), el ratio de aprendizaje, el tamaño de batch, la aplicación de técnicas de regularización, así como la elección de distintas funciones de pérdida y activación. Es decir, este capítulo se centra en la experimentación y el análisis de los resultados, buscando identificar las configuraciones de hiperparámetros más efectivas para el problema de clasificación.

El diseño experimental se ha estructurado en torno a la selección de un modelo óptimo basado en el contraste de estas hipótesis. Para ello, se han llevado a cabo múltiples iteraciones, ajustando los hiperparámetros mencionados y evaluando el desempeño del modelo resultante en términos de métricas de clasificación pertinentes. Una vez seleccionado el modelo óptimo, se procede a comparar los resultados de la clasificación multicategoría con aquellos obtenidos a través de modelos de clasificación que predicen cada variable de salida de manera individual. Esta comparación permitirá evaluar si la utilización de una red neuronal para clasificación general con todas las variables de salida en conjunto ofrece ventajas significativas en términos de rendimiento y precisión en comparación con los modelos de clasificación individual.

6.1. Búsqueda de hiperparámetros en modelos multicategóricos

Anteriormente, se detalló el proceso de tratamiento, limpieza y preparación del conjunto de datos tabulares definitivo que iba a ser utilizado para la creación de un modelo de clasificación capaz de predecir el valor de las cuatro variables categóricas que conforman el índice de riesgo. Dicho conjunto de datos se dividió en dos subconjuntos, entrenamiento y validación, cada uno de ellos utilizado en una fase diferente del proceso de experimentación que se describe a continuación.

Tal y como se ha mencionado anteriormente, la experimentación es un proceso iterativo que se

divide en hipótesis. En cada una de estas hipótesis se probarán diversos modelos que refutarán o no la posible mejora de los resultados de los anteriores respecto a una serie de modificaciones. Cada uno de estos modelos, se entrenará con el subconjunto de datos de entrenamiento y se validará con el de validación.

A lo largo de todo el estudio se empleará la versión 2.7.12 de la librería Fastai.

```
import fastai
print(fastai.__version__)
```

```
2.7.12
```

Figura 6.1: Versión Fastai

En cada hipótesis se llevan a cabo una serie de pasos comunes que se repiten en la refutación de todas ellas. El primero de ellos es la carga de los datos.

```
[ ] datos_red = pd.read_csv('train_norm.csv')
    datos_red = datos_red.drop(datos_red.columns[[0]], axis='columns')
    display(datos_red)
```

	Edad	Sexo	FR.1	SpO2.1	FiO2.1	TAS.1	TAD.1	FC.1	TT.1	GCS.0.1	...	Cáncer.no.metastásico
0	-0.448628	1	-1.492510	-1.962031	-0.133374	-2.117611	-2.007326	-0.643449	-1.723544	1	...	0
1	-0.503443	2	-1.056950	-0.431228	-0.133374	-1.747054	-1.710748	-0.431865	0.520227	1	...	0
2	2.182473	1	-0.040643	0.403756	-0.133374	0.146904	1.670244	-0.981984	0.199688	4	...	0
3	-0.229370	2	0.104544	0.403756	-0.133374	-0.594210	0.009406	0.414472	-0.120850	3	...	0
4	-0.010111	2	-0.476203	-6.276113	-0.133374	-1.088286	-0.643066	-0.389548	-0.014004	1	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
523	-0.393814	2	0.685291	0.403756	-0.133374	0.682152	0.661878	0.626057	0.413381	4	...	0
524	0.154332	1	-0.621390	0.125428	-0.133374	0.599806	1.017772	0.372155	0.627073	4	...	0
525	-1.325662	2	-0.185830	0.403756	-0.133374	-0.594210	0.009406	0.329839	-0.014004	4	...	0
526	-1.325662	1	-0.476203	0.542919	-0.133374	-0.429518	-0.524435	0.202888	0.947612	4	...	0
527	2.292102	1	-0.040643	0.125428	-0.133374	1.135055	-0.583750	-1.320519	-2.150929	3	...	0

528 rows × 59 columns

Figura 6.2: Conjunto de entrenamiento

Por un lado se cargan los datos con los que se va a entrenar el modelo, mostrándose una muestra de dicho conjunto para asegurar la correcta lectura de los datos. También ayuda a comprobar que la dimensión es la deseada, observándose que el subconjunto de entrenamiento está compuesto de las 55 variables dependientes y las cuatro independientes, con un total de 528 pacientes. En la figura 6.2 se puede observar una muestra del conjunto de entrenamiento. El conjunto de validación se carga en un dataframe diferente.

Una vez se tienen cargados los datos, se define un array de nombres en función del tipo de cada variable (figura 6.3). Esto es, un array para las variables numéricas (cont_names), otro para las categóricas (cat_names) y otro específico para las variables dependientes (y_names). Este paso surge de la necesidad de Fastai de conocer la tipología de cada variable para decidir qué procesamiento aplicar a cada una de ellas a la hora de construir y entrenar un modelo de manera más eficiente.

```
y_names = ['Mortalidad', 'Hospital','UVI','VMH'] #dependientes
cat_names = ['Sexo', 'VM', 'Antidoto', 'Tipo.de.intoxicación', 'Triaje', 'GCS.O.1',"ICC","IAM" ,"EVP","ACV",
               "Hemiplejia","EPOC","DM","DM.con.lesion","Enf..Renal","Enf..Hepa.leve","Enf.Hepa.grave","Úlcera",
               "SIDA","Linfoma","Leucemia","Metástasis","Cáncer.no.metastásico",
               "Enf..T..conectivo","Demencia","CACI","VasoactivosH"] #categoricas
cont_names = ['Edad', 'FR.1', 'SpO2.1', 'FiO2.1','TAS.1', 'TAD.1','FC.1', 'TT.1',
              'pH', 'pCO2', 'pO2', 'cHCO3.', 'BE..ecf.', 'cSO2', 'Na.', 'K.', 'Ca..', 'Cl.','TCO2', 'Hct',
              'Hb', 'BE..b.', 'Glu',  'Lac',  'Crea', 'Urea.mg.dl', 'BUN.calculated', 'Osmolr.'] #continuas
```

Figura 6.3: Arrays de nombres de variables

Los pasos consecutivos a los mencionados también se repiten en varias de las hipótesis, pero sufriendo ligeras variaciones, tal y como se muestra a continuación.

Hipótesis 1 - Tamaño de la red

El número de capas y el número de neuronas influyen en la capacidad de representación de una red neuronal, ya que un mayor número de capas y más neuronas pueden capturar relaciones más complejas y abstracciones de los datos. Anteriormente (sección 2.3) se ha demostrado que esto no siempre es así, ya que una red neuronal con demasiadas capas y neuronas puede ser propensa al sobreajuste. Al probar con diferentes tamaños, se puede determinar la capacidad necesaria para modelar correctamente el problema en cuestión.

El siguiente paso correspondiente a la creación de un modelo, será común a todas las iteraciones de esta hipótesis, ya que no está relacionado con el tamaño de la red. Se trata de la creación del objeto *Dataloader*, mencionado en la sección 3.2. Debido a la naturaleza de los datos del experimento en el que las categorías correspondientes a las variables dependientes se encuentran desequilibradas, es conveniente llevar a cabo muestreo estratificado, una opción no disponible en la función de Fastai *DataLoader*. Por ello, se separarán los datos de entrenamiento mediante muestreo estratificado utilizando *RandomSplitter* y reservando un 20 % del conjunto para la fase de test de cada iteración del entrenamiento del modelo. A continuación, se crea un objeto de carga de datos de tipo *TabularPandas*, en el que se indica que se desean cargar los datos de entrenamiento, a los que se aplicarán procesamientos de relleno de valores ausentes y categorización, el tipo de cada variable, y la forma de separar el subconjunto en cada iteración de entrenamiento del modelo. Después, dicho objeto se convierte a tipo *DataLoader* para poder ser empleado con Fastai, indicándole que el tamaño de lote deseado en cada iteración sera de 32 observaciones (valor por defecto). Para comprobar que la carga de datos es correcta, se puede imprimir, por ejemplo, el número de variables salida, tal y como se muestra en la figura 6.4.

```
procs = [Categorify, FillMissing]
splits = RandomSplitter(valid_pct=0.2)(range_of(datos_red))
to = TabularPandas(datos_red, procs=procs,
                   cat_names = cat_names,
                   cont_names = cont_names,
                   y_names=y_names,
                   splits=splits)
data = to.dataloaders(bs=32)
num_outputs = data.c
num_outputs
```

```
4
```

Figura 6.4: Construcción DataLoader

Una vez el objeto de carga de datos está preparado, se puede comenzar a construir modelos de diferentes tamaños y a entrenarlos después. En todos ellos se indicarán el objeto de carga de datos a utilizar, el número de capas y de neuronas por capa y la métrica empleada. En este caso se utiliza como métrica *accuracy_multi*, una medida que se emplea para evaluar la precisión de clasificación en problemas de clasificación multiclase como el estudiado. Es una métrica que calcula la precisión de predicción de la red neuronal para cada clase individualmente y luego promedia esas precisiones.

El modelo se entrena en 500 epoch. A continuación, se muestra un ejemplo del entrenamiento los modelos evaluados y un resumen de los resultados de cada uno de ellos en cuanto a su precisión (figuras 6.5 y 6.6). Para evaluar dicha precisión, se realizan las predicciones (preds) de las cuatro variables dependientes respecto a las independientes empleando el conjunto de validación. Con esas predicciones se comparan los valores de salida reales (targs) con los predichos y se calcula el porcentaje de aciertos sobre el total (figura 6.6).

```
[ ] m = tabular_learner(data, layers=[55,256,128,32], metrics=accuracy_multi)
    m.fit_one_cycle(500)
```

Figura 6.5: Ejemplo de entrenamiento de la red

```
[ ] dl = m.dls.test_dl(test_df)
    preds, targs = m.get_preds(dl=dl)
    accuracy_multi(preds, targs)

    TensorBase(0.4806)
```

Figura 6.6: Ejemplo del cálculo de la precisión

Modelo	Tamaño	Precisión
1	[55,256,128,32]	0.4806
2	[55,1000,512,32]	0.3778
3	[55,1000,512]	0.3889
4	[55,256,128]	0.4222
5	[55,256,128,32,16]	0.3389
6	[55,128,256,32]	0.4444
7	[256,128,128,32]	0.4667
8	[256,128,32]	0.4750
9	[55,256]	0.3472
10	[200,100]	0.4250
11	[55,200,100]	0.4750

En base a los resultados obtenidos, se concluye que, en relación al tamaño de red preliminar adecuado a los datos de estudio, no se puede rechazar que el tamaño óptimo sea el del modelo 1. Este modelo posee 5 capas intermedias y una de salida con tamaños 55, 256, 128, 32 y 4 respectivamente.

Hipótesis 2 - Tamaño de batch

El tamaño del batch tiene un impacto directo en la eficiencia computacional durante el entrenamiento de la red neuronal. Un tamaño de batch más grande permite procesar más muestras en paralelo, lo que puede acelerar el entrenamiento y hacer un uso más eficiente de los recursos computacionales disponibles, pero a su vez también puede consumir más memoria y ralentizar el proceso de entrenamiento. Por lo tanto, es necesario encontrar un equilibrio que permita un entrenamiento eficiente sin comprometer el rendimiento del modelo. Además, un tamaño de batch grande puede ayudar a estimar mejor el gradiente y proporcionar una dirección más precisa para las actualizaciones de los pesos de la red neuronal, lo que conlleva una mejor generalización y un modelo más estable. Sin embargo, un tamaño de batch más pequeño puede introducir más variabilidad en las actualizaciones de los pesos y permitir que el modelo explore diferentes regiones del espacio de búsqueda de manera más efectiva.

En la experimentación de esta hipótesis y debido a que el tamaño del conjunto de datos del que se dispone para entrenar el modelo no es excesivamente grande, no resulta lógico experimentar con demasiados valores ni con valores mayores del que Fastai toma por defecto (32). Tomando como válida esta suposición, se eligen dos múltiplos de 2 que se encuentren por debajo de dicho valor y se reevaluan los resultados, todo ello partiendo del tamaño de modelo elegido en la hipótesis anterior. La precisión correspondiente al tamaño de batch 32 se corresponde con la del modelo elegido en el apartado anterior (0.4806). En la figura 6.7 se ejemplifica la variación del tamaño de batch durante el entrenamiento.

```
[ ] data = to.dataloaders(bs=16)
    m2 = tabular_learner(data, layers=[55,256,128,32], metrics=accuracy_multi)
    m2.fit_one_cycle(500)
```

```
[ ] dl = m2.dls.test_dl(test_df)
    preds, targs = m2.get_preds(dl=dl)
    accuracy_multi(preds, targs)

    TensorBase(0.4139)
```

Figura 6.7: Ejemplo tamaño batch 16

A continuación, se analizan los resultados con los diferentes tamaños de batch.

Modelo	Tamaño batch	Precisión
1	32	0.4806
2	16	0.4139
3	8	0.1500

Tras el empeoramiento de la precisión fruto de la disminución del tamaño de batch, se decide mantener el tamaño de batch óptimo en 32.

Hipótesis 3 - Ratio de aprendizaje óptimo

Se sabe que utilizar un ratio de aprendizaje muy elevado durante el entrenamiento de una red neuronal implica que los pesos y los sesgos se ajustarán de manera más drástica en cada iteración del algoritmo de optimización, lo que puede llevar a una convergencia más rápida pero también a una mayor inestabilidad. En consecuencia, el rendimiento de la red neuronal puede ser deficiente y los resultados pueden ser impredecibles.

Durante la fase inicial del entrenamiento, Fastai realiza un ajuste automático del ratio de aprendizaje aplicando *cyclical learning rate finder*, es decir, explorando diferentes valores en las distintas iteraciones de la fase de entrenamiento y evaluando cómo se comporta la función de pérdida del modelo de manera dinámica. Luego, para la siguiente iteración, elige un valor inicial adecuado basado en ese análisis.

En esta hipótesis se pretende contrastar el funcionamiento de la elección de dicho valor de manera automática por Fastai, frente al funcionamiento de la función *lr_find*, una función específica que realiza un barrido a través de una amplia gama de valores de ratio de aprendizaje y evalúa cómo se comporta la función de pérdida del modelo en cada valor. Luego, genera una gráfica (llamada gráfica de búsqueda de ratio de aprendizaje) que muestra la variación de la función de pérdida en función del ratio de aprendizaje. Esta gráfica ayuda a identificar los rangos de ratio de aprendizaje donde la función de pérdida disminuye rápidamente (buenos valores) y donde se produce inestabilidad o se estanca (valores inadecuados). A partir de esta gráfica, se puede seleccionar un valor inicial adecuado para el ratio de aprendizaje antes de comenzar el entrenamiento.

Se comienza eligiendo el valor del ratio óptimo de aprendizaje antes del entrenamiento, es decir, aplicando *lr_find* tal y como se observa en la figura 6.8.

```
[15] to = TabularPandas(datos_red, procs=procs,
                        cat_names = cat_names,
                        cont_names = cont_names,
                        y_names=y_names,
                        splits=splits)
     data = to.dataloaders(bs=32)
     m = tabular_learner(data, layers=[55,256,128,32], metrics=accuracy_multi)

     m.lr_find()
     lr = m.recorder.lrs[np.argmin(m.recorder.losses)]
     print(lr)
```

Figura 6.8: Búsqueda del ratio de aprendizaje óptimo

Para este procedimiento, se selecciona de todo el rango de valores que se han probado aquel que minimice la función de pérdida.

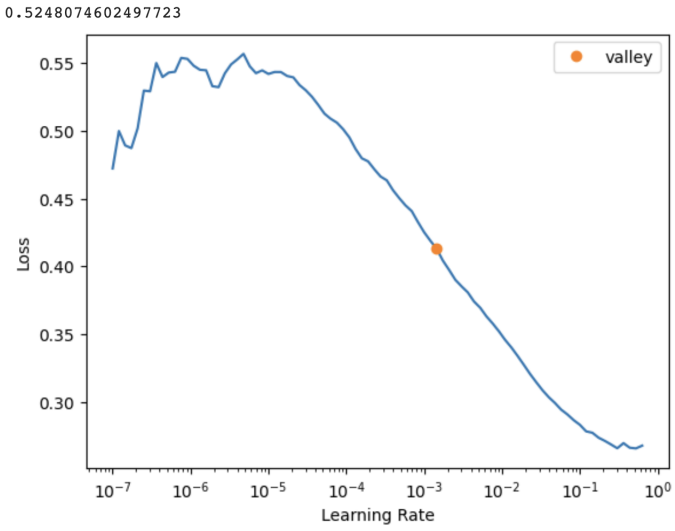


Figura 6.9: Evolución lr y loss

En este caso, en la figura 6.9 se observa que el valor obtenido es 0.524, un valor que, a priori, puede parecer elevado. El punto naranja denominado "valley" indica un valor de referencia de LR en el que la pérdida disminuye rápidamente y de manera más estable. Es decir, alrededor de este punto, el modelo converge de manera más eficiente hacia una solución óptima. Si se entrena el modelo con el valor seleccionado por Fastai (0.524), es fácilmente observable tanto en el valor de la función de pérdida como en la precisión la inestabilidad que presenta el modelo (figura 6.10).

```
[16] epochs = 500
     m.fit_one_cycle(epochs, lr)

     m.recorder.plot_loss()
```

41	0.347191	49721607454720.000000	0.133333	00:00
42	0.290428	57716973568.000000	0.573810	00:00
43	0.243767	1623893999616.000000	0.735714	00:00
44	0.206852	7561885722471301120.000000	0.361905	00:00
45	0.176664	21442725157011456.000000	0.164286	00:00
46	0.150816	4117949413588992.000000	0.692857	00:00
47	0.131485	2354960667527610368.000000	0.754762	00:00
48	0.117951	67143352587386880.000000	0.245238	00:00

Figura 6.10: Inestabilidad por elevado ratio de aprendizaje

Además, si se entrena el modelo utilizando este valor de ratio de aprendizaje de 0.524 se detecta un empeoramiento de la precisión del modelo a la hora de predecir frente a la selección dinámica de Fastai, ya que con el uso de *lr_find* esta disminuye hasta el valor 0.3167, mientras que con el modelo preliminar se obtenía un 0.48.

Modelo	**Función de búsqueda**	**Precisión**
1	*cyclical learning rate finder*	0.4806
2	*lr_find*	0.3167

Por tanto, se decide asumir que la selección dinámica que realiza el objeto de Fastai de manera automática repercute de manera positiva en los resultados frente a la selección de un valor preliminar. Esto significa que el modelo elegido en la hipótesis anterior continúa siendo el que pasa como óptimo a la siguiente fase de la experimentación.

Hipótesis 4 - Regularización

El dropout es una técnica de regularización que introduce aleatoriedad en la red neuronal durante el entrenamiento. En lugar de penalizar directamente los pesos, el dropout apaga aleatoriamente un subconjunto de neuronas en cada actualización de los parámetros. Esto evita que las unidades dependan demasiado entre sí y promueve una representación más robusta y generalizable, reduciendo a su vez el sobreajuste del modelo. Al regularizar la penúltima capa con dropout, se espera reducir la tendencia del modelo a sobreajustar los datos de entrenamiento y, en cambio, aprender características más relevantes y transferibles. Por ello, se pretende contrastar la eficiencia de una red que presente esta regularización frente a una que no.

En la figura 6.11 se ilustra cómo se decide establecer la probabilidad de dropout en la penúltima capa en un 10 %, es decir, que durante el entrenamiento cada neurona de la penúltima capa tiene

una probabilidad del 10 % de ser "apagada"(de que sus salidas se establezcan en cero) en cada actualización de los parámetros.

```
m2 = tabular_learner(data, layers=[55,256,128,32], metrics=accuracy_multi)
m2.model.layers[-1] = nn.Linear(32, to.c) # Cambiar última capa lineal
m2.model.layers[-2].p = 0.1 # Regularización dropout en penúltima capa

m2.fit_one_cycle(500)
```

Figura 6.11: Ejemplo modelo con dropout

Modelo	Método	Precisión
1	Sin regularización	0.4806
2	Con regularización	0.5000

Al aplicar esta técnica de regularización se obtiene una precisión del 0.5000, por lo que se concluye que los resultados del modelo mejoran y por tanto se asume este modelo como el nuevo óptimo.

Hipótesis 5 - Función de activación diferente

Es importante adaptar la función de activación en la capa de salida según el tipo de problema que se esté abordando. En el caso de una regresión, no se aplica una función de activación específica en la capa de salida, ya que se espera una predicción continua sin restricciones. Sin embargo, para problemas de clasificación, se suelen utilizar diferentes funciones de activación en la capa de salida para generar una distribución de probabilidad sobre las clases. Al adaptar la función de activación en la capa de salida, se pueden obtener salidas adecuadas para el tipo de problema y se puede mejorar el rendimiento del modelo.

En el modelo seleccionado en la hipótesis anterior, la capa de salida era lineal, es decir, no se aplicaba ninguna otra transformación a los valores de salida de la red. Sabiendo que según la naturaleza del problema es más adecuado emplear unas funciones de activación u otras en esta capa de salida, se decide evaluar cómo se comporta la red si se le aplican diferentes funciones. Si bien en problemas de clasificación binaria, la capa de salida puede requerir una función de activación sigmoide para generar una salida en el rango de $[0, 1]$ que se interpreta como una probabilidad de pertenecer a la clase positiva, en problemas de clasificación multiclase la capa de salida puede requerir una función de activación softmax para generar una distribución de probabilidad sobre todas las clases. Estas premisas no siempre se cumplen, por lo que habrá que comprobar qué función se adapta mejor al caso de estudio concreto. En la figura 6.12 se ilustra un ejemplo de modificación de la función de activación.

```
[7] model_2 = tabular_learner(data, layers=[55,256,128,32], metrics=accuracy_multi)
    model_2.model.layers[-1].act = nn.Sigmoid()
    model_2.fit_one_cycle(400)
```

Figura 6.12: Ejemplo modificación función de activación

Modelo	Función de activación	Precisión
1	Lineal	0.4806
2	Sigmoide	0.1333
3	Softmax	0.2167

En la primera iteración se decide tomar como función de activación de la capa de salida la función sigmoide, observando que la precisión de la red empeora. Este empeoramiento de los resultados resulta lógico, ya que si se trata de un problema de clasificación y se cambia la función de activación lineal a sigmoide en la capa de salida, es posible que la red neuronal no pueda modelar eficazmente la naturaleza discreta y categórica de las clases objetivo. Además, la función de activación sigmoide produce salidas en el rango de [0, 1], lo que se interpreta como probabilidades de pertenecer a una clase en problemas de clasificación binaria. Si el problema no se ajusta a esta interpretación de probabilidades o si las clases no tienen una distribución equilibrada como en este caso, utilizar la función de activación sigmoide puede no ser adecuado.

Si se utiliza una función de activación softmax, se observa que nuevamente la precisión de la red empeora. De forma análoga a la iteración anterior, esto puede deberse al desequilibrio de clases que presentan las variables salida. Como el problema de clasificación multiclase tiene un desequilibrio significativo en la distribución de las clases, la utilización de softmax puede amplificar este desequilibrio. La función softmax asigna probabilidades a las clases, y si hay una clase dominante o subrepresentada, puede resultar en una mayor probabilidad para la clase mayoritaria y una menor probabilidad para las clases minoritarias, lo que afecta negativamente a la precisión del modelo.

Por todo ello, se decide mantener como modelo óptimo aquel con capa de salida lineal.

Hipótesis 6 - Funciones de pérdida

Otra estrategia útil a la hora de intentar mejorar los resultados de un modelo es probar diferentes funciones de pérdida, ya que según el caso específico se requieren diferentes enfoques. En algunos problemas, la función de pérdida más adecuada es el error cuadrático medio (MSE), que penaliza las diferencias al cuadrado entre las predicciones y los valores reales. En cambio, en otros problemas como los de clasificación binaria, la entropía cruzada suele ser la función de pérdida más apropiada. También conviene recordar que algunas funciones de pérdida pueden ser más sensibles a ciertos tipos de errores que otras. Por ejemplo, la función de pérdida de entropía cruzada penaliza de manera más significativa las predicciones incorrectas cercanas a la clase correcta, lo que puede ser beneficioso en problemas donde la precisión en la clase correcta es crítica.

Tal y como muestra la figura 6.13, en el modelo elegido hasta el momento la función utilizada por defecto por Fastai es el error cuadrático medio (MSE), concretamente *FlattenedLoss of MSE-Loss*, una función modificada para aplanar los datos de entrada antes de aplicar el cálculo del error cuadrático medio. Esto significa que si los datos de entrada tienen una dimensión adicional, como un batch de muestras, esta función los aplana antes de calcular la pérdida. Esto es útil cuando se trabaja con datos en lotes y se quiere obtener una única puntuación de pérdida para todo el batch.

```
[ ] m = tabular_learner(data, layers=[55,256,128,32], metrics=accuracy_multi)
    print(m.loss_func) #por defecto

    FlattenedLoss of MSELoss()
```

Figura 6.13: Función de pérdida por defecto

Se decide aplicar varias funciones de pérdida, comenzando por la de entropía categórica cruzada (*Categorical Cross-Entropy*), utilizada cuando las etiquetas de las variables salida están codificadas como categóricas y por tanto adecuada para problemas de clasificación multiclase. Esta opción tuvo que ser descartada ya que, por la naturaleza del problema en cuestión, generaba diversos errores de interpretación de la salida, puesto que clasifica de manera que cada muestra puede pertenecer única y exclusivamente a una clase. En el problema objetivo de este estudio, cabe recordar que las clases no son excluyentes, pudiendo activarse más de un 1 en la salida de la misma muestra. Un problema similar aparece cuando se intenta aplicar *LabelSmoothingCrossEntropy*, otro tipo de entropía cruzada que en lugar de codificar las etiquetas verdaderas como variables categóricas, las suaviza distribuyendo la probabilidad en todas las clases. También se aplica únicamente para clases excluyentes, no siendo apropiada para el objetivo de este estudio.

Dado que las variables de salida del modelo objetivo son todas binarias, se decide abordar la elección de la función de pérdida teniendo en cuenta esta característica. Se opta por probar la función *BCEWithLogitsLossFlat*, una función de pérdida de entropía cruzada binaria (*binary cross-entropy*) con logits aplanados. Se trata de una adaptación de *BCEWithLogitsLoss*, función de PyTorch.

La entropía cruzada binaria es una función de pérdida comúnmente utilizada en problemas de clasificación binaria. Toma como entrada los logits (es decir, los valores antes de aplicar una función de activación) y los valores de destino binarios (0 o 1) y calcula la pérdida basada en la diferencia entre las predicciones del modelo y los valores reales.

Al aplicar esta función, se observa que en cada iteración de la fase de entrenamiento los resultados presentan una mejora considerable respecto a los entrenamientos anteriores. Si se extrae la precisión media durante el entrenamiento, es decir la calculada utilizando el conjunto de validación que utiliza el objeto *Learner* de Fastai cuando se le pasa el objeto *Dataloader*, y no el conjunto de validación que se reserva para la predicción posterior al entrenamiento, se obtiene un resultado significativamente mejor que en iteraciones anteriores. Este resultado se puede observar en la figura 6.14.

```
[39] m_9.validate()[1]

     0.8999999761581421
```

Figura 6.14: Precisión con función de pérdida de entropía cruzada binaria

Sin embargo, si se realiza una predicción sobre el conjunto de validación nunca visto como en las hipótesis anteriores, se obtiene un valor mucho menor, por lo que la capacidad de generalización de este modelo disminuye considerablemente respecto al anteriormente considerado como óptimo.

Modelo	Función de pérdida	Precisión
1	*FlattenedLoss of MSELoss*	0.4806
2	*BCEWithLogitsLossFlat*	0.1333

Tras el análisis de estos resultados, se decide seguir considerando como óptimo el elegido en la hipótesis anterior.

6.1.1. Red neuronal definitiva

Tras los contrastes de hipótesis llevados a cabo, se decide considerar como proceso de modelado óptimo aquel que construye un modelo empleando un objeto de carga de datos que emplea lotes o batchs con un tamaño de 32 observaciones cada uno, para entrenar una arquitectura de 5 capas, siendo 4 de ellas ocultas con 55, 256, 128 y 32 neuronas respectivamente, y una última capa de salida de 4 neuronas. En esta arquitectura se aplica regularización dropout en la penúltima capa y se emplea una función de activación lineal para la capa de salida. Además, la evolución de dicho entrenamiento se estudia en base a una función de pérdida que calcula el error cuadrático medio de los datos aplanados y utilizando como medida de eficiencia la precisión de la salida multivariable (*accuracy_multi*), que comprueba que todos los valores predichos para las variables coinciden con las etiquetas reales. En este proceso de entrenamiento, el ratio de aprendizaje se ajusta dinámica y automáticamente en cada iteración en función de los resultados obtenidos. Este modelo presentaba una precisión de aproximadamente el 50 %.

La arquitectura que construye Fastai para este modelo se divide en varias subestructuras, algunas de ellas ocultas al usuario durante su creación. Si se imprime la salida de dicha estructura se observa que, antes de comenzar a crear las capas que el usuario indica en la creación del objeto *Learner*, Fastai genera una serie de capas denominadas embeddings. Los embeddings son capas utilizadas para representar variables categóricas en forma de vectores densos de números reales. Estas capas permiten que el modelo aprenda representaciones numéricas significativas para las categorías presentes en las variables categóricas. Cada capa de *embedding* tiene un número determinado de neuronas, que son elegidos automáticamente por Fastai según el número de categorías únicas presentes en cada variable categórica. El número de neuronas en cada capa de *embedding* se determina en base a una regla general recomendada. Por lo general, se elige un número de neuronas igual al tamaño de la categoría más grande dividido entre 2, pero con un límite superior. Por ejemplo, si una variable categórica tiene 10 categorías únicas, se podría elegir 5 neuronas para su capa de *embedding*. Esta regla ayuda a encontrar un equilibrio entre una representación significativa y una dimensión manejable.

```
TabularModel (Input shape: 32 x 27)
==========================================================================
Layer (type)          Output Shape          Param #    Trainable
==========================================================================
                      32 x 3
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
__________________________________________________________________________
                      32 x 6
Embedding                                   60         True
__________________________________________________________________________
                      32 x 3
Embedding                                   12         True
__________________________________________________________________________
                      32 x 4
Embedding                                   20         True
__________________________________________________________________________
                      32 x 3
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
Embedding                                   9          True
__________________________________________________________________________
                      32 x 7
Embedding                                   98         True
```

Figura 6.15: Capas *embedding*

En la figura 6.15 se puede observar que la entrada del modelo consta de 32 observaciones derivadas del tamaño de batch elegido, por 27, valor correspondiente a las 27 variables categóricas presentes en los datos de entrada. También se observa que hay 6 capas de *embedding* aunque el número de variables categóricas es 27. Esto se debe a que algunas de las variables categóricas tienen un gran número de categorías, frente a otras que son binarias. El uso de múltiples capas de este tipo puede permitir al modelo capturar de manera más efectiva las relaciones no lineales entre las categorías de las variables categóricas. Cada capa de *embedding* se encarga de transformar una variable categórica en una representación numérica de dimensión reducida, que luego se combina con las demás características para la etapa de procesamiento posterior.

	32 x 3		
Embedding		9	True
Dropout			
BatchNorm1d		56	True
	32 x 55		
Linear		6435	True
ReLU			
BatchNorm1d		110	True
	32 x 256		
Linear		14080	True
ReLU			
BatchNorm1d		512	True
	32 x 128		
Linear		32768	True
ReLU			
BatchNorm1d		256	True
	32 x 32		
Linear		4096	True
ReLU			
BatchNorm1d		64	True
	32 x 4		
Linear		132	True

Figura 6.16: Capas ocultas y capa de salida

La siguiente capa que se observa en la figura 6.16 combina *embedding*, *dropout* y *batchnorm*. Esta capa está diseñada para codificar una variable categórica en una representación numérica más densa, aplicar regularización a través de *dropout* y normalizar los valores antes de pasarlos a la siguiente capa en la red neuronal. El uso de este tipo de capas es habitual en modelos que combinan diferente tipología de variables y se utilizan para mejorar la capacidad de generalización y disminuir el sobreajuste.

El resto de capas ocultas (figura 6.16) presentan un diseño común mediante la combinación de las funciones de activación y procedimientos *Linear*, *Relu* y *BatchNorm*. Esta capa es una capa lineal completamente conectada, es decir, cada neurona de la capa anterior está conectada a todas las neu-

ronas de la capa actual y por tanto cada salida de la capa anterior se conecta como entrada a cada neurona de la capa siguiente. Esta capa realiza una transformación lineal de los datos de entrada empleando la función de activación *ReLU*, que se aplica después de la capa lineal para introducir no linealidad en el modelo. A continuación, se aplica normalización por lotes para normalizar las salidas de la capa anterior antes de pasarlas a la siguiente capa y acelerar así el entrenamiento del modelo.

La capa de salida no realiza transformaciones sobre los datos, es lineal, y consta de cuatro neuronas, cada una de las cuales devuelve la salida para cada una de las variables dependientes.

```
Total params: 58,906
Total trainable params: 58,906
Total non-trainable params: 0

Optimizer used: <function Adam at 0x7f34c6865b40>
Loss function: FlattenedLoss of MSELoss()
```

Figura 6.17: Información adicional del modelo

En la figura 6.17 se puede observar que la red presenta 58906 parámetros entrenables, es decir, hay 58906 valores que se ajustarán durante el proceso de entrenamiento de la red neuronal para minimizar la función de pérdida empleando descenso del gradiente (optimizador Adam). Estos parámetros hacen referencia a los pesos y sesgos asociados a cada conexión entre las neuronas de la red.

Para indagar un poco más en la capacidad de predicción y generalización del modelo se decide complementar la precisión calculada automáticamente por Fastai con el cálculo manual de la matriz de confusión y las tasas de acierto y error. Además, dado que Fastai calcula los aciertos y errores en función de si la salida completa es correcta o no, se decide analizar la precisión del modelo para cada variable salida por individual, puesto que puede estar comportándose de forma incorrecta a la hora de predecir una de las categorías, pero dicha capacidad predictiva puede ser muy buena con otras.

Se comienza calculando la tasa de acierto sobre el conjunto de validación, es decir, se extraen los valores predichos y se comparan con las etiquetas reales de los datos para dichas observaciones (figura 6.18). Después, se calcula el porcentaje de acierto sobre el total de valores predichos. Antes de llevar a cabo la comparación, se convierten los valores que se devuelven como predicción correspondientes a las probabilidades de pertenencia a la categoría a valor entero, es decir, si la probabilidad de que sí pertenezca a la categoría es mayor que 0.5 le asignamos la categoría (figura 6.19).

```
# Predicciones para el conjunto de validación
dl = modelo.dls.test_dl(test_df)
preds, targets = modelo.get_preds(dl=dl)

# Convertimos las predicciones y objetivos a etiquetas numéricas
preds_label = (preds > 0.5).int().numpy()
targets = targets.numpy() #valores reales de salidas validacion
```

Figura 6.18: Cálculo de predicciones

```
[25] correct_preds = np.sum(np.all(preds_label == targets, axis=1))
     total_preds = len(targets)
     accuracy = correct_preds / total_preds
     misclassified_indices = np.where(preds_label != targets)[0]
     misclassified_samples = test_df.iloc[misclassified_indices]
```

```
[28] accuracy

     0.7444444444444445
```

Figura 6.19: Cálculo de tasa de acierto

Se observa que este valor es bastante más optimista respecto a la capacidad predictiva del modelo que el realizado por Fastai. Para analizar el comportamiento de la red a la hora de predecir, se decide crear una matriz de confusión de tamaño 16x16, en la que se muestra para cada categoría de las 16 totales cuántos individuos se clasifican correctamente y cuántos se clasifican en otra categoría y en cuál. Esta opción no está disponible en la librería Fastai, ya que su función *confusion_matrix()* solo se puede emplear en problemas con dos categorías. Por ello, la creación de esta matriz será manual.

```
[[52  0  0  0  8  0  1  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [15  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 2  0  0  0  0  0  1  4  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]]
```

Figura 6.20: Matriz de confusión 16 categorías

Tal y como muestra la figura 6.20, los individuos clasificados correctamente para las 4 variables salida aparecen en la diagonal de la matriz. Resulta lógico que la categoría en la que mejor se clasifican sea la primera, aquella en la que todas las variables toman valor 0, ya que también es la categoría mayoritaria en el conjunto de datos utilizado para entrenar la red. En la figura 6.21 se analiza la tasa de acierto de cada categoría.

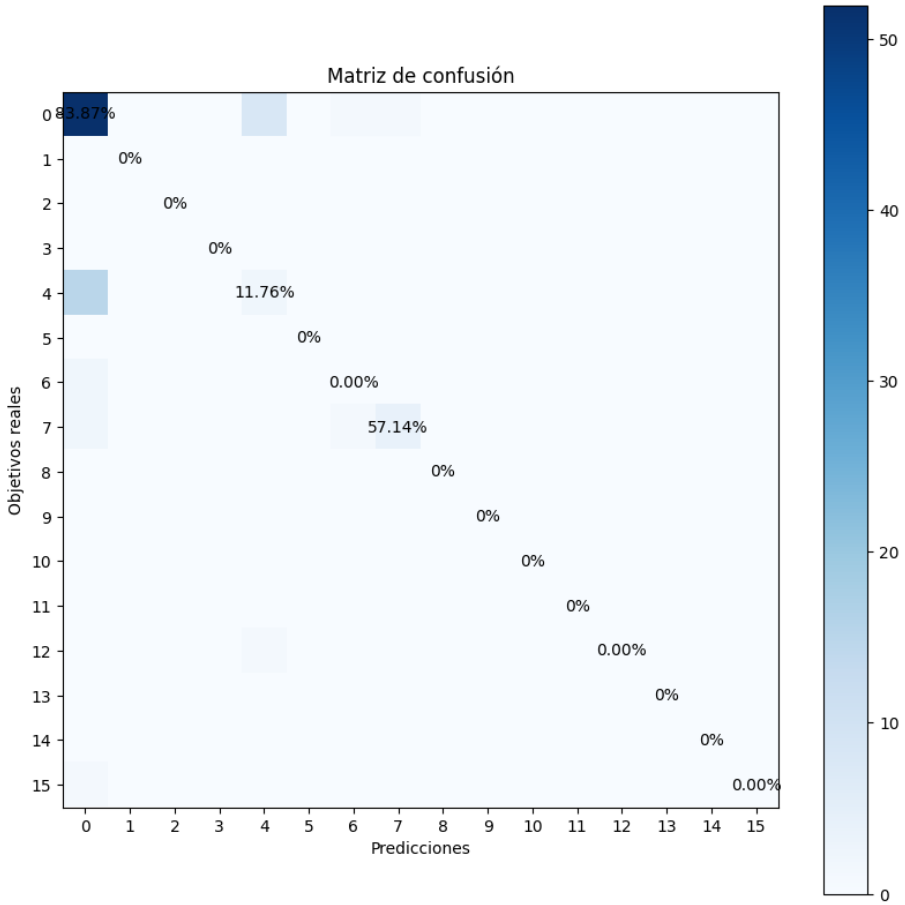


Figura 6.21: Matriz de confusión, diagonal

Cabe destacar que algunas de las categorías no existen en el conjunto de datos, por lo que no presentan tasa de clasificación correcta ni incorrecta. La existencia de otras de las combinaciones, por el contexto sanitario del caso de estudio, tampoco tendría sentido que existieran. Por ejemplo, que se ingrese en *UVI* conlleva que se ingresa en el hospital, por lo que la aparición de un 1 en la segunda de las categorías y no en la primera, no podría darse.

Estos resultados se agrupan en conteos de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Estos resultados se calculan manualmente aplicando las siguientes fórmulas y se muestran en la figura 6.22.

$$VP = \sum_{i=1}^{n} \mathbb{I}(y^{(i)}_{\text{verdadero}} = 1 \ \wedge \ y^{(i)}_{\text{predicho}} = 1) \tag{6.1}$$

$$VN = \sum_{i=1}^{n} \mathbb{I}(y^{(i)}_{\text{verdadero}} = 0 \ \wedge \ y^{(i)}_{\text{predicho}} = 0) \tag{6.2}$$

$$FP = \sum_{i=1}^{n} \mathbb{I}(y^{(i)}_{\text{verdadero}} = 0 \ \wedge \ y^{(i)}_{\text{predicho}} = 1) \tag{6.3}$$

$$FN = \sum_{i=1}^{n} \mathbb{I}(y^{(i)}_{\text{verdadero}} = 1 \ \wedge \ y^{(i)}_{\text{predicho}} = 0) \tag{6.4}$$

	True Positive	False Positive	False Negative	True Negative
0000	52	20	10	8
0001	0	0	0	90
0010	0	0	0	90
0011	0	0	0	90
0100	2	9	15	64
0101	0	0	0	90
0110	0	2	2	86
0111	4	1	3	82
1000	0	0	0	90
1001	0	0	0	90
1010	0	0	0	90
1011	0	0	0	90
1100	0	0	1	89
1101	0	0	0	90
1110	0	0	0	90
1111	0	0	1	89

Figura 6.22: Resumen de cálculos por categoría

Tras los resultados obtenidos, se decide repetir este conteo para cada variable individualmente, con el fin de averiguar con qué variables se comporta mejor el modelo (figura 6.23).

```
Variable de salida 'Mortalidad'
[[88  0]
 [ 2  0]]

Variable de salida 'Hospital'
[[54  8]
 [18 10]]

Variable de salida 'UVI'
[[79  1]
 [ 3  7]]

Variable de salida 'VMH'
[[82  0]
 [ 3  5]]
```

Figura 6.23: Matrices de confusión individuales

A continuación se calcula la tasa de verdaderos positivos para cada variable, ya que por el contexto sanitario del problema, resulta prioritario predecir qué pacientes sí pertenecen a la categoría frente a aquellos que no. Los valores críticos en este contexto resultan los verdaderos positivos y los falsos negativos.

$$TPR = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{6.5}$$

- Mortalidad: 97.78 %
- Hospital: 75.00 %
- UVI: 96.34 %
- VMH: 96.47 %

Se observa que, analizando cada variable individualmente, el modelo presenta una precisión bastante elevada, pero dichos resultados se ven enmascarados cuando al fallar en una de ellas se clasifica como fallo global. Este resultado hace pensar que, en este contexto, tal vez sea más óptimo realizar un modelo predictivo de regresión para cada una de las variables salida por separado.

6.2. Modelos individuales

En este apartado, se repiten los procedimientos de carga de datos de forma análoga a los anteriores, pero centrándose en una única variable objetivo cada vez. En la figura 6.24 se ilustra como el vector que indicaba los nombres de las variables dependientes ahora estará formado por una única variable, y por tanto el número de neuronas de la capa de salida se reduce también a un valor. Además, las otras tres variables categóricas que antes se consideraban objetivo y en la nueva red individual no, pasan a considerarse como independientes y por tanto se incluyen en el vector de

variables categóricas. Cabe destacar que, tanto el tamaño de cada una de las cuatro nuevas redes como las funciones de pérdida utilizadas y el tamaño de batch, se han decido siguiendo el mismo planteamiento iterativo que con el modelo multicategórico, según el cual se escogía el hiperparámetro que mejor precisión implicaba en el modelo.

```
y_names = ['Mortalidad']
cat_names = ['Sexo', 'VM', 'Antidoto', 'Tipo.de.intoxicación', 'Triaje', 'GCS.O.1',"ICC","IAM" ,"EVP","ACV",
               "Hemiplejia","EPOC","DM","DM.con.lesion","Enf..Renal","Enf..Hepa.leve","Enf.Hepa.grave","Úlcera",
               "SIDA","Linfoma","Leucemia","Metástasis","Cáncer.no.metastásico",
               "Enf..T..conectivo","Demencia","CACI","VasoactivosH", 'Hospital','UVI','VMH'] #categoricas
cont_names = ['Edad', 'FR.1', 'SpO2.1', 'FiO2.1','TAS.1', 'TAD.1','FC.1', 'TT.1',
              'pH', 'pCO2', 'pO2', 'cHCO3.', 'BE..ecf.', 'cSO2', 'Na.', 'K.', 'Ca..', 'Cl.','TCO2', 'Hct',
              'Hb', 'BE..b.', 'Glu',  'Lac',  'Crea', 'Urea.mg.dl', 'BUN.calculated', 'Osmolr.'] #continuas
```

```
procs = [Categorify, FillMissing]
splits = RandomSplitter(valid_pct=0.2)(range_of(datos_red))
to = TabularPandas(datos_red, procs=procs,
                   cat_names = cat_names,
                   cont_names = cont_names,
                   y_names=y_names,
                   splits=splits)
data = to.dataloaders(bs=32)
num_outputs = data.c
num_outputs
```

```
1
```

Figura 6.24: Ejemplo carga de datos para variable individual

6.2.1. Resumen de modelos elegidos

Se aplica nuevamente la metodología iterativa basada en la validación de hipótesis a cada una de las cuatro redes neuronales que predicen las variables respuesta de forma individual. Las hipótesis a validar son las mismas que las empleadas en la búsqueda de hiperparámetros del clasificador multicategórico. Al finalizar el proceso, se han obtenido como óptimas a la hora de predecir cada variable respuesta las siguientes configuraciones.

Variable Respuesta	Tamaño red	Tamaño batch	Regularización
Mortalidad	[1000,256,128,32]	32	Sí
Hospital	[1000,256,128,32]	32	Sí
UVI	[256,128,32]	32	Sí
VMH	[58,256,128,32]	32	Sí

Cuadro 6.1: Tamaño red, tamaño batch, regularización

Variable Respuesta	Función de activación	Función de pérdida
Mortalidad	Lineal	*BCEWithLogitsLossFlat*
Hospital	Lineal	*BCEWithLogitsLossFlat*
UVI	Lineal	*FlattenedLoss of MSELoss*
VMH	Lineal	*FlattenedLoss of MSELoss*

Cuadro 6.2: Función de activación y función de pérdida

6.2.2. Resultados

En vista de los resultados obtenidos al utilizar modelos de regresión individuales para predecir las variables salida individualmente, se puede decir que, para aquellas más desbalanceadas en el conjunto de datos inicial como *Mortalidad*, no existe una mejora a la hora de hacer predicciones. Sin embargo, en otras como *VMH*, se obtienen mejores resultados que, en el modelo conjunto, quedaban ocultos por los resultados de las demás variables. Se concluye así que en el caso de las variables *VMH* y *UVI* sí que merecería la pena utilizar modelos individuales para construir el índice de riesgo.

Variable Respuesta	**Precisión**
Mortalidad	0.2333
Hospital	0.3333
UVI	0.7556
VMH	0.9111

Cuadro 6.3: Precisión de modelos de predicción individuales

6.3. Data Augmentation

Tal y como se expuso en el capítulo 2 sobre el contexto de este estudio, el éxito de los modelos depende en gran medida de la disponibilidad y calidad de los datos utilizados para entrenarlos. Sin embargo, en muchos casos los conjuntos de datos son limitados en tamaño o presentan desequilibrios, lo que puede afectar la capacidad del modelo para generalizar y obtener resultados precisos. En este sentido, las técnicas de aumento de datos o *data augmentation* son una solución efectiva para abordar estos desafíos y mejorar el rendimiento de los modelos. El *data augmentation* se refiere al proceso de generar nuevas muestras de datos a partir de las muestras existentes mediante la aplicación de transformaciones y manipulaciones controladas. Estas técnicas permiten aumentar la diversidad y cantidad de datos disponibles para el entrenamiento, evitando así el sobreajuste y mejorando la capacidad del modelo para capturar las características subyacentes de los datos.

En problemas relacionados con clasificación de imágenes y visión por computadora, el *data augmentation* ha demostrado ser especialmente efectivo [21]. Las transformaciones como la rotación, el volteo, el recorte y el cambio de brillo se aplican a las imágenes existentes, generando nuevas variantes que preservan la información esencial de la imagen original. Esto ayuda al modelo a ser más robusto a variaciones en la iluminación, la orientación y la posición de los objetos, mejorando su capacidad de generalización.

Sin embargo, aunque el *data augmentation* destaca en el campo de la imagen, no se limita únicamente a este. En el caso de los datos tabulares, también es posible aplicar técnicas similares para generar nuevas muestras que enriquezcan el conjunto de datos original. Por ejemplo, una técnica común es la adición de ruido gaussiano a los datos numéricos, lo que introduce variabilidad y ayuda al modelo a aprender patrones más robustos. Esto puede ser especialmente beneficioso en conjuntos de datos tabulares como el de este estudio, donde no se poseen demasiados ejemplos y estos pertenecen en mayor porcentaje a una categoría que a otra. Por ello y con el objetivo de mejorar la precisión obtenida para el modelo multicategórico que construía el índice de riesgo para las 4 variables salida simultáneamente, se decide aplicar esta técnica para incrementar la variabilidad de los datos y mejorar así la precisión y estabilidad del modelo.

Para aplicar la técnica de adición de ruido gaussiano al conjunto de datos original, se decide aplicar *bootstrapping*, una técnica estadística que consiste en generar múltiples muestras aleatorias con reemplazo a partir de un conjunto de datos existente. El reemplazo permite que una misma observación pueda ser seleccionada múltiples veces en diferentes muestras sintéticas. Debido al elevado coste computacional que puede suponer un aumento demasiado elevado del conjunto de datos y tras varios experimentos en los que se varía el tamaño de las muestras artificiales a generar, se opta por generar 5 muestras del tamaño de la muestra original, cada una de ellas en una iteración. En una iteración se selecciona una muestra del tamaño del conjunto original pero con reemplazamiento, permitiendo la repetición de algunas de las observaciones. Una vez que se han seleccionado las observaciones del conjunto de datos existente, se les agrega ruido aleatorio siguiendo una distribución normal con media cero y desviación estándar de 0.1. Las observaciones sintéticas generadas en cada iteración se combinarán con el conjunto de datos originales al terminar el bucle. Este procedimiento se ilustra en la figura 6.25.

```
# Número de observaciones sintéticas a generar
num_synthetic_samples = 5

# Bootstrapping
df_synthetic = pd.DataFrame()
for _ in range(num_synthetic_samples):

    bootstrap_sample = d_a.sample(n=len(d_a), replace=True)
    noisy_sample = bootstrap_sample.apply(lambda x: x + np.random.normal(0, 0.1), axis=1)
    df_synthetic = pd.concat([df_synthetic, noisy_sample], ignore_index=True)

df_combined = pd.concat([d_a, df_synthetic], ignore_index=True)
```

```
[12] df_synthetic.shape, df_combined.shape

     ((2640, 59), (3168, 59))
```

Figura 6.25: Generación ruido gaussiano

Si se repite el proceso de entrenamiento y validación del modelo multicategórico utilizando los mismos hiperparámetros y características, en la figura 6.26 se observa que, efectivamente, el resultado mejora considerablemente. Se confirma así que, en el caso de estudio, la utilización de técnicas que aumenten la cantidad y variabilidad de los datos contribuyen a un aumento de la capacidad de generalización del modelo.

```
✓ 0s [10] dl = m.dls.test_dl(test_df)
          preds, targs = m.get_preds(dl=dl)
          accuracy_multi(preds, targs)

          TensorBase(0.6056)
```

Figura 6.26: Precisión tras aplicar data augmentation

Capítulo 7

Conclusiones

A lo largo de este estudio se ha ilustrado el procedimiento de construcción de una red neuronal como modelo de clasificación empleando la herramienta Fastai, tomando un conjunto de datos sanitarios de tipo tabular como ejemplo de aplicación. Cabe recordar que de entre las numerosas opciones de implementación disponibles en la actualidad, Fastai destaca como una de las alternativas más innovadoras y eficientes para desarrollar este tipo de proyectos. La continua exploración del uso de esta librería en diversos campos de aplicación y en relación a diferentes tipos de datos, no solo limitados a imágenes, contribuye al progreso de la inteligencia artificial. Durante este proceso, se han asentado las bases teóricas de este tipo de modelos con el fin de entender y mejorar su funcionamiento.

El estudio persigue dos metas principales, ambas directamente relacionadas. La primera de ellas corresponde a la creación de un clasificador multiclase para datos tabulares como método de construcción de un índice de riesgo utilizando la librería Fastai. La segunda de ellas pretende demostrar la competitividad de Fastai en este tipo de problemas no relacionados con imágenes frente a otras herramientas existentes.

Para llevar a cabo este proceso se decidió emplear una metodología iterativa basada en hipótesis. Queda demostrado que esta metodología resulta más eficiente que seleccionar los hiperparámetros de la red de manera aleatoria, ya que al rechazar o aceptar hipótesis en función del rendimiento del hiperparámetro, se puede ajustar y mejorar gradualmente el modelo a medida que se realizan más iteraciones. De esta manera, se evita el desperdicio de recursos computacionales en configuraciones que no son prometedoras.

Respecto al primer objetivo, al finalizar la experimentación siguiendo esta metodología se ha logrado alcanzar una precisión máxima del 60 % gracias a la aplicación de técnicas data augmentation a los datos de origen. Si se comparan estos resultados con los obtenidos al utilizar modelos de regresión para predecir las variables salida del índice individualmente, se observa que para la predicción individual de algunas de ellas la precisión alcanzada ascendía a un 90 %. Por tanto, se concluye que utilizar modelos más sencillos de regresión puede resultar más provechoso, ya que el desequilibrio entre las categorías de las variables respuesta penaliza los resultados de la red multicategórica en su conjunto, problema que se podría mitigar al predecir cada categoría individualmente.

Por otro lado, del segundo objetivo se puede concluir que, para datos tabulares con la morfología del conjunto de datos empleado en este estudio, es posible que existan otras herramientas capaces

de ofrecer similares o incluso mejores resultados con menor consumo de recursos computacionales. Es decir, así como Fastai destaca en el ámbito del reconocimiento y clasificación de imágenes, en base a los resultados de este estudio no se puede decir que ocurra lo mismo con datos tabulares.

Como posibles mejoras futuras para los resultados obtenidos, se plantea la posibilidad de aumentar el tamaño de la base de datos utilizada para entrenar la red neuronal. Esta medida resultaría beneficiosa al incrementar la variabilidad en los datos, lo que permitiría abordar de manera más efectiva el desequilibrio existente entre las diferentes categorías. Al contar con una mayor cantidad de muestras, se podrían seleccionar las que presenten un reparto más equitativo de los individuos de cada clase, por lo que el modelo desarrollado con Fastai para datos tabulares podría tener una capacidad de generalización más sólida y, en consecuencia, mejoraría su precisión. Además, resultaría interesante explorar el comportamiento de modelos similares y con el mismo conjunto de datos, pero empleando otras librerías como *keras*, o incluso comparar sus resultados con los de otros modelos más sencillos ajenos al DL, como la regresión logística.

Bibliografía

[1] Thomas Bayes. *An Essay towards solving a Problem in the Doctrine of Chances.* Self-published, 1763.

[2] Alan Turing. *On Computable Numbers, with an Application to the Entscheidungsproblem.* Londres: Proceedings of the London Mathematical Society, 1936.

[3] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[4] Frank Rosenblatt. *The Perceptron: A Perceiving and Recognizing Automaton.* Cornell Aeronautical Laboratory, 1958.

[5] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, 1969.

[6] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, 1986.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[8] Simon Haykin. *Neural Networks and Learning Machines.* Pearson Education, 2009.

[9] Tom M. Mitchell. Machine learning. *ACM Computing Surveys*, 29, 1997.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016.

[11] Michael A. Nielsen. *Neural Networks and Deep Learning.* Deterministic Press, 2015.

[12] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018.

[13] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *1312.6098*, 2013.

[14] Sebastian Ruder. An overview of gradient descent optimization algorithms. *1609.04747*, 2016.

[15] Andrew Y Ng, Michael I Jordan, and Yair Weiss. Feature selection, l1 vs. l2 regularization, and rotational invariance. *Journal of Machine Learning Research*, 5(Aug):1157–1182, 2004.

[16] Hadley Wickham, Jim Hester, and Winston Chang. Rstudio: Integrated development environment for r. *Journal of Statistical Software*, 54(1):1–37, 2015.

[17] RStudio Team. *RStudio: Integrated Development Environment for R.* RStudio, PBC., Boston, MA, 2020.

[18] Fastai Community. Fastai documentation. `https://docs.fast.ai/`, Última consulta: Junio 2023.

[19] Ekaba Bisong and Ekaba Bisong. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 59–64, 2019.

[20] Jeremy Howard and Sylvain Gugger. *Deep Learning for Coders with fastai and PyTorch.* O'Reilly Media, 2020.

[21] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *International Conference on Learning Representations (ICLR)*, pages 1–9, 2014.