**Grado en Ingeniería Informática**

Inteligencia Artificial II

Laboratorio 02: Implementación de modelos neuronales supervisados

Grupo B07:

Álvaro Ramos Morales

Álvaro Delgado Gallego

Fernando Ramirez Fernández

Juan Esteban Bernal Santos

Tabla de contenido

[1. Introducción 3](#_gjdgxs)

[2. Metodología 4](#_30j0zll)

[2.1. Pre-procesamiento 4](#_1fob9te)

[2.2. Tuning de hiperparámetros 5](#_3znysh7)

[2.3. Arquitectura 6](#_2et92p0)

[2.4. Validación 7](#_tyjcwt)

[2.5. Evaluación 8](#_3dy6vkm)

[3. Experimentación y Resultados 10](#_1t3h5sf)

[3.1. Tuning 10](#_hemzenxnlhhk)

[3.2. Validación 12](#_8z5ktl12mvyj)

[3.3. Evaluación 13](#_50f8rpvyaa56)

[4. Cuestiones 15](#_4d34og8)

[4.1. Primera práctica 15](#_fe78t7bnyhq5)

[4.2. Segunda práctica 16](#_8x99nm7nya0l)

[4.3. Tercera práctica 19](#_wmn6j6kn4ojl)

[5. Bibliografía 21](#_2s8eyo1)

# Introducción

El propósito de este laboratorio es explorar el uso de perceptrones para tareas de clasificación, abarcando desde la implementación más básica, con la construcción de un perceptrón lineal, hasta la construcción de modelos multicapa (MLP) por medio de Keras. A lo largo de las prácticas propuestas, se busca desarrollar una comprensión profunda de estos modelos y su aplicabilidad en diferentes contextos.

El uso de los perceptrones tanto simple como multicapa, se justifica en su capacidad para aprender patrones y realizar clasificaciones en conjuntos de datos específicos. Los perceptrones simples son adecuados para problemas más simples y lineales, mientras que los multicapa son la opción con mayor utilidad al poder manipular conjuntos de datos tanto lineales como no lineales. Además, los modelos multicapa permiten delimitar diversas clases en un solo modelo, lo que los hace especialmente útiles en tareas de clasificación donde existen múltiples clases y relaciones complejas entre las variables de entrada y salida.

Por otro lado, estos modelos presentan limitaciones cuando trabajan con datos no etiquetados y clases desbalanceadas. Si las clases están desbalanceadas, el modelo puede desarrollar un sesgo hacia la clase más representada, lo que resulta en una menor precisión al identificar instancias de clases menos comunes, por ello es muy importante asegurarse de que nuestros datos no tengan estos problemas y si los tienen realizar un preprocesamiento adecuado.

En la primera práctica, se aborda la clasificación de distintos tipos de lirios basándose en las medidas de sus pétalos y sépalos. Aquí, el uso de un perceptrón simple permite una separación eficiente de una sola clase respecto a las demás, teniendo en cuenta los patrones presentes en los datos. Es importante destacar que el perceptrón realiza una división lineal del espacio, lo que significa que solo es capaz de clasificar dos clases.

En la segunda práctica, usaremos un perceptrón multicapa para clasificar los distintos tipos de lirios teniendo en cuenta todos los atributos presentes en el dataset. El uso de un MLP para este caso se justifica en su capacidad para clasificar conjuntos de datos más complejos con mayor precisión, aprovechando la posibilidad de emplear funciones de activación no lineales y el uso de múltiples capas y neuronas en el mismo modelo.

Finalmente, en la tercera práctica, nos enfrentamos a un desafío práctico real: predecir el riesgo de sufrir un infarto. En este contexto, las clases posibles son dos, cada una indicando si el paciente presentó un infarto o no. Sin embargo, la cantidad de atributos presentes en el dataset sugiere que una relación lineal simple entre las variables puede ser insuficiente para modelar adecuadamente este fenómeno. Por esta razón, se recurre a un perceptrón multicapa para abordar esta problemática, ya que este tipo de modelo neuronal artificial puede capturar relaciones no lineales entre las características de entrada y la variable objetivo, permitiendo así una mejor representación del complejo comportamiento subyacente en los datos.

En resumen, trabajaremos con perceptrones simples hasta modelos multicapa como MLP para tareas de clasificación, la elección entre un perceptrón simple o multicapa depende de la complejidad del problema; los simples son útiles para problemas lineales y los multicapa para situaciones más complejas que involucran datos lineales y no lineales, así como múltiples clases

# Metodología

## Pre-procesamiento

2.1.1. Antes de introducir los datos al modelo nos aseguramos de que no haya un desbalance de clases utilizando un histograma, en este caso las clases están equilibradas pero nos encontramos con un dataset de tres clases. El perceptrón simple es incapaz de separar más de dos clases, por lo que nos vemos obligados a seleccionar sólo dos de ellas. Así, mediante varios diagramas de dispersión que combinan cada par de atributos entre las tres clases, hemos escogido las dos clases que más se diferencien para facilitar el entrenamiento.

2.1.2. Se han importado los datos mediante la función usada en la práctica 1 llamada “load\_iris”de sklearn para cargar el dataset de lirios. Se ha usado una codificación de etiquetas para las etiquetas categóricas de las especies de lirios para que el modelo las procese de manera más sencilla mediante OneHotEncoder, el cual convierte las etiquetas en vectores binarios para la clasificación de múltiples clases (Ej: Clase 1: [1,0,0], clase 2: [0,1,0]).Posteriormente, mediante la función “train\_test\_split” se ha realizado la separación en conjuntos de entrenamiento y de pruebas en un porcentaje de 80% de datos de entrenamiento y 20% los datos de prueba. Esta separación es vital para que el modelo pueda evaluar y generalizar de manera objetiva después del entrenamiento.

2.1.3. Para prevenir posibles problemas durante el entrenamiento del modelo, son necesarias varias fases. La primera de ellas es el tratamiento de los valores nulos, que se presentan principalmente en el atributo 'bmi'. Se eliminarán los registros que no contengan un valor asignado para asegurar que el modelo se entrene con conjuntos de datos completos, lo cual es especialmente importante en un caso médico. Adicionalmente, se eliminará la columna 'id' del dataset original, ya que no aporta información útil para el estudio del problema. Del mismo modo, los valores 'NSNC' (No Sabe/No Contesta) en el campo uso\_tabaco también serán eliminados, por la misma razón que los valores nulos: es esencial entrenar al modelo con conjuntos de datos completos.

El segundo problema que encontramos en el dataset es el desbalance de clases. Al mostrar el histograma para diferenciar el número de clases, se puede observar una proporción desbalanceada de 95% a 5%.

Una vez detectamos este problema, procedimos a igualar la cantidad de estas utilizando técnicas de undersampling como el oversampling. En el undersampling, redujimos aleatoriamente la cantidad de ejemplos en la clase mayoritaria que en nuestro caso era la de no infartos para igualarla con la clase minoritaria que sería que se produjese un infarto. Por otro lado, en el oversampling, aumentamos la cantidad de ejemplos en la clase minoritaria replicándose aleatoriamente hasta alcanzar un equilibrio con la clase mayoritaria. Esto nos permitirá asegurar que el modelo tenga la misma cantidad de datos de cada clase para entrenar.

Por último, los atributos que contengan dos clases distintas se convertirán a formato binario y aquellos con más de dos clases se transformarán mediante one-hot encoding, para optimizar el procesamiento por parte del MLP. Además, los datos se normalizarán para que se encuentren en un rango uniforme sin perder su estructura inicial. Esto facilitará la detección de patrones por el perceptrón multicapa.

## Tuning de hiperparámetros

2.2.1. Para este modelo los hiperparámetros que hemos escogido son learning rate y threshold. Hemos seleccionado estos dos porque el learning rate determina la velocidad a la que el modelo ajusta sus pesos durante el entrenamiento, lo cual es crucial para asegurar que el modelo converja a una solución óptima sin moverse demasiado lentamente hacia el mínimo. Por otro lado, el threshold establece el límite que decide cuándo una entrada es clasificada como una clase u otra, siendo fundamental para la precisión del modelo al separar las clases.

Para este caso se emplearán valores de learning rate de 0.01 , 0.1 y 0.5 estos valores no son muy elevados ya que queremos un entrenamiento estable sin cambios bruscos buscando así que el error siempre disminuya de manera controlada. Con estos valores disminuimos la posibilidad de que se ajuste demasiado a los datos específicos del entrenamiento, lo que podría llevar a problemas de overfitting.

En cuanto al threshold, para este caso se emplearán valores de 0.01, 0.2 y 0.75. Estos valores nos permiten flexibilidad en la decisión de clasificación del modelo, ajustando la sensibilidad frente a las distintas clases. Un threshold más bajo como 0.01 hace que el modelo sea más inclusivo, considerando más entradas como positivas, mientras que un valor más alto como 0.75 hace que el modelo sea más restrictivo, reduciendo la probabilidad de falsos positivos.

2.2.2. En la práctica de clasificación de lirios utilizando un perceptrón multicapa, la selección adecuada de hiperparámetros es importante para optimizar tanto el rendimiento como la capacidad de generalización del modelo. Los hiperparámetros ajustados incluyen el número de neuronas por capa, la tasa de aprendizaje, el momento (momentum) y el número de capas ocultas.

Para la tasa de aprendizaje, se experimentó con 0.01 y 0.05. Una tasa de aprendizaje baja puede resultar en una convergencia lenta, posiblemente atascandose en mínimos locales, mientras que una tasa alta puede sobrepasar los mínimos deseables, dificultando la estabilización del aprendizaje.

El momentum, probado en valores de 0.0, 0.2 y 0.9, ayuda a acelerar el optimizador en la dirección correcta, suavizando las actualizaciones y potencialmente evitando oscilaciones y divergencias durante el entrenamiento. Juntos, estos parámetros trabajan para encontrar un equilibrio en el aprendizaje que favorezca tanto la eficiencia como la efectividad del entrenamiento.

Para el número de capas ocultas, se evaluaron configuraciones con 5, 7 y 10 capas ocultas para explorar cómo la profundidad del modelo puede afectar su capacidad para capturar patrones complejos en los datos. Un mayor número de capas puede influir en el aprendizaje, necesitando para su máxima efectividad un mayor número de datos, el cual en el dataset de lirios no tenemos.

Para determinar el número óptimo de neuronas por capa, seleccionamos tres configuraciones: 10, 15 y 20 neuronas. La elección de estos valores responde a la necesidad de equilibrar entre la capacidad del modelo para representar de manera adecuada la complejidad de los datos y la eficiencia en el procesamiento computacional. Un número menor de neuronas puede ser insuficiente para capturar la variabilidad y las relaciones no lineales entre las características, mientras que un exceso puede llevar al modelo a aprender detalles específicos del conjunto de entrenamiento que no se generalizan a nuevos datos (overfitting).

2.2.3. Los hiperparámetros que han sido seleccionados para el modelo y debido al dataset son el número de capas ocultas, el número de neuronas por capa y la tasa de aprendizaje del perceptrón. La elección de estos hiperparámetros se justifica por su impacto significativo en el rendimiento y la capacidad de generalización del modelo. El número de capas ocultas y su número de neuronas determinan la complejidad y la capacidad de representación del modelo, lo cual a su vez influye en la precisión con la que este capta los patrones de comportamiento presentes en los datos. Por otro lado, la tasa de aprendizaje controla la rapidez con la que el modelo ajusta sus pesos durante el entrenamiento, lo que puede afectar tanto en la convergencia del modelo como en su capacidad para evitar el sobreajuste. Los valores propuestos para cada hiperparámetro se listan a continuación.

Para el número de capas, se han seleccionado valores de 2, 3 y 5, basándonos en la posibilidad de que el modelo pueda necesitar de una estructura simple como una compleja dependiendo de la facilidad con la que pueda interpretar los datos del dataset. Con dos capas ocultas, el modelo puede capturar relaciones más simples entre las características de entrada y la variable de clase, lo que puede ser útil para conjuntos de datos con patrones más lineales. Por otro lado, al aumentar el número de capas a 3 y 5, se busca explorar la capacidad del modelo para aprender representaciones más profundas y complejas entre los datos.

Para seleccionar el número de neuronas por capas nos hemos guiado por el teorema de aproximación universal. La cantidad de neuronas necesaria depende de la complejidad del problema, el número de ejemplos en el conjunto de entrenamiento y el número de entradas. Aunque el teorema no asegura que ese número sea el más óptimo. Después de realizar este procedimiento el rango que nos salía era entre 22 y 87.

Con toda la información obtenida a partir del teorema, los rangos de neuronas que utilizamos fueron de 20, 70 y 100. Esto se debe a que una mayor cantidad de neuronas contribuye a detectar patrones de manera más eficaz. Sin embargo, un número excesivamente alto de neuronas puede implicar un alto costo computacional y aumentar el riesgo de sobreajuste del modelo. Por lo tanto, buscamos un equilibrio entre el costo y la eficacia/precisión. Cabe destacar que el número de neuronas disminuirá a la mitad en cada capa sucesiva, siguiendo la regla de 2^k, donde k representa el número de la capa actual (por ejemplo, si el número inicial de neuronas por capa es 128 y tenemos 5 capas, se distribuirán de la siguiente manera: 128, 64, 32, 16, 8).

## Arquitectura

2.3.1. La arquitectura del perceptrón consiste en una única neurona con múltiples entradas y una salida. Cada entrada está asociada con un peso, y además, hay un bias que ajusta el nivel de activación necesario para que la neurona se active. Para la primera práctica, identificamos entradas que representan las características de las flores de lirio. Los pesos y el bias se asignan de modo que cada característica influye en la activación de la neurona mediante el uso de un combinador lineal, que es la suma ponderada de las entradas. Utilizamos una función de activación de tipo HardLimiter, que determina si la salida es 0 o 1, es decir, si la flor pertenece a una categoría de lirio u otra. Para entender cómo se activa la neurona, debemos explicar qué es el umbral, esté equivale a un peso ficticio no conectado a ninguna entrada y que se resta del valor de entrada a la neurona. Por lo tanto, la neurona se 'activa' emitiendo una salida de +1 cuando las señales de entrada, adecuadamente ponderadas y ajustadas por el umbral, son lo suficientemente fuertes para superar dicho umbral. De lo contrario, la neurona no se activa y su salida es 0. Este mecanismo permite al perceptrón realizar clasificaciones binarias.

El método de entrenamiento que hemos seguido comienza con la inicialización, donde los pesos y sesgos se inician aleatoriamente o a cero. Luego, en la fase de alimentación hacia adelante, se calcula la salida para cada instancia utilizando la suma ponderada y la función de activación. Si la salida es incorrecta, entramos en la fase de retroalimentación y ajuste, donde los pesos y sesgos se modifican según la diferencia entre la salida esperada y la obtenida, aplicando la tasa de aprendizaje. Este proceso se repite en múltiples iteraciones sobre el conjunto de datos hasta que los ajustes en los pesos y sesgos convergen a una solución estable o hasta completar un número predefinido de iteraciones.

2.3.2. La arquitectura del perceptrón multicapa (MLP) diseñada para la clasificación de especies de lirios es significativamente más compleja que la de un perceptrón simple. El MLP consiste en varias capas de neuronas interconectadas, organizadas en una capa de entrada, varias capas ocultas y una capa de salida.

La capa de entrada recibe las entradas directamente de las características de los lirios, como el largo y ancho de los pétalos y sépalos. Cada neurona en esta capa está asociada con una característica específica del lirio y pasa su valor a la siguiente capa sin realizar ninguna operación de procesamiento

En cuanto a las capas ocultas, el número de las mismas y de neuronas en cada una se define basándose en el tuning de hiperparámetros. Estas capas son el núcleo computacional del MLP, donde cada neurona recibe entradas de todas las neuronas de la capa anterior. Cada entrada es ponderada por un peso y la suma de estas entradas ponderadas más un término de bias constituye el input neto de la neurona. Las capas ocultas utilizan funciones de activación no lineales, como ReLU o Sigmoid, para introducir no linealidades en el modelo, permitiendo al MLP aprender y modelar relaciones complejas entre las entradas.

Finalmente la capa de salida consta de neuronas que corresponden al número de clases a clasificar.

A la hora del entrenamiento el MLP se realiza mediante el método de backpropagation que está formado por varias etapas:

Inicialización: se asignan valores aleatorios a todos los pesos de la red.

Forward pass: cada entrada se propaga se propaga a través de las capas sucesivas de la red, haciendo uso de las funciones de activación definidas, hasta generar una salida en la capa final. (López Herraiz, n.d.)

Retropropagación del error: si la salida generada en la capa final difiere de la salida deseada , el error se propaga de vuelta por toda la red hasta la capa de entrada, ajustando los pesos para minimizar el error.

Finalmente el ajuste de los pesos: los pesos se ajustan haciendo uso de un algoritmo de optimización, como puede ser el SGD o ADAM.

## Validación

2.4.1. No emplea un método de validación para la primera práctica, debido a que con un conjunto de test ya es suficiente para probar que el modelo está generalizando correctamente.

2.4.2. Para la validación del MLP con el dataset de lirios hemos utilizado una división simple de 80% para el conjunto de datos de entrenamiento, que permitirá al modelo aprender y adaptarse a las características de los datos de los lirios, y un 20% para el conjunto de datos de validación, que permitirá evaluar el rendimiento del modelo al no ser utilizados esos datos en la fase de entrenamiento.

Este método de evaluación ayuda a identificar si el modelo ha aprendido a generalizar las características de las flores usando métricas de precisión y pérdida para determinar la efectividad de la clasificación y con la posibilidad de apoyarse en una gráfica.

2.4.3. La estrategia de validación empleada incorpora el uso de validación cruzada, específicamente a través de la técnica K-Fold. En esta aproximación, el conjunto de datos se divide inicialmente en dos partes: una para entrenamiento, que comprende el 80% de los datos, y otra para validación, con el 20% restante. Este proceso permite que el modelo aprenda con la mayor parte de los datos y luego se evalúe su rendimiento en una sección separada que simula datos nuevos y no vistos.

Para reforzar la robustez de la validación, se aplica K-Fold, que divide el conjunto de entrenamiento en n\_splits subconjuntos, en este caso, cinco. El modelo se entrena en cuatro de estos subconjuntos, mientras que el restante se utiliza para la validación. Este proceso se repite cinco veces (el número de 'folds'), con cada subconjunto usado exactamente una vez para la validación. Así, se obtienen múltiples métricas de rendimiento que reflejan cómo el modelo podría funcionar en diferentes partes del conjunto de datos, reduciendo el sesgo y proporcionando unas mejores medidas para su caso.

Finalmente, se cuenta con un conjunto de test separado, que permanece completamente sin usar durante las fases de entrenamiento y validación cruzada, para proporcionar una estimación imparcial del rendimiento del modelo una vez completado todo el proceso de entrenamiento. Este enfoque asegura que las métricas finales sean representativas de la habilidad del modelo para trabajar con datos que no ha encontrado durante su aprendizaje, lo que es crítico en aplicaciones prácticas.

## Evaluación

2.5.1. Debido a que el problema consiste en separar dos clases y existen diversos hiperplanos que puedan llegar a la misma solución, no se realizará un proceso de evaluación

2.5.2. Después de la fase de entrenamiento el modelo se somete a una prueba final con un conjunto de test que ha sido reservado específicamente para medir la precisión del modelo de una manera objetiva.

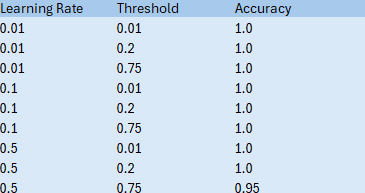
Haremos uso del conjunto de prueba para medir la precisión del modelo, comparando los resultados predichos con los reales. Esto nos permitirá construir la matriz de confusión, que nos permite visualizar el número de predicciones correctas e incorrectas para cada clase de lirio.

2.5.3. Evaluaremos el modelo, una vez entrenado, visualizando las métricas obtenidas en los conjuntos de entrenamiento, validación y prueba. Utilizaremos el conjunto de prueba para medir la precisión del modelo, comparando los resultados predichos con los reales. Esto nos permitirá construir la matriz de confusión y obtener métricas clave como el accuracy, que determina la proporción de predicciones correctas del total de predicciones. También calcularemos la sensibilidad, que refleja la capacidad del modelo para identificar correctamente los casos positivos, es decir, la proporción de pacientes diagnosticados con infarto que el modelo identificó correctamente, y la especificidad, que mide la capacidad del modelo para reconocer correctamente los casos negativos, es decir, la proporción de pacientes no infartados correctamente identificados por el modelo. Adicionalmente, crearemos una tabla que compare estos valores entre los conjuntos de entrenamiento, validación y prueba para verificar si el modelo ha generalizado correctamente, observando la consistencia en los resultados obtenidos en cada conjunto.

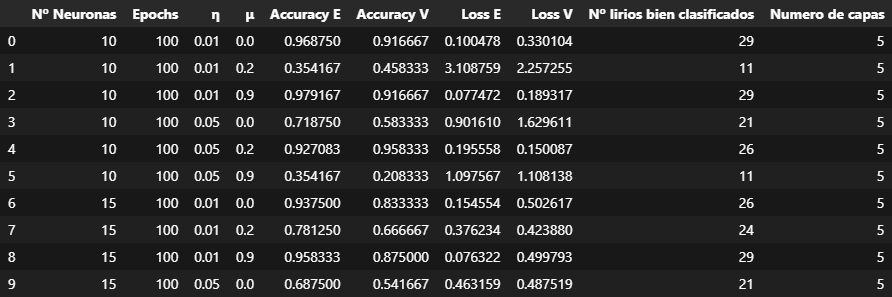
# Experimentación y Resultados

## 3.1. Tuning

3.1.1. Con el fin de construir un algoritmo consistente y que de resultados coherentes, se realizó un estudio sobre el cómo los hiperparámetros del perceptrón afectan en su rendimiento, esto lo hacemos con un grid search, destaca mucho que al ser dos conjuntos que son linealmente separables sin complicación, entonces con la mayoría de configuraciones logramos un accuracy de 1.0.



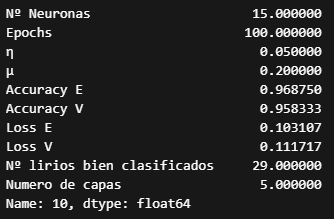
3.1.2. Para la elección de los hiperparámetros óptimos del modelo, hemos realizado un grid search para visualizar de manera clara cómo afecta cada hiperparámetro [elegido](#_3znysh7) a nuestro modelo.



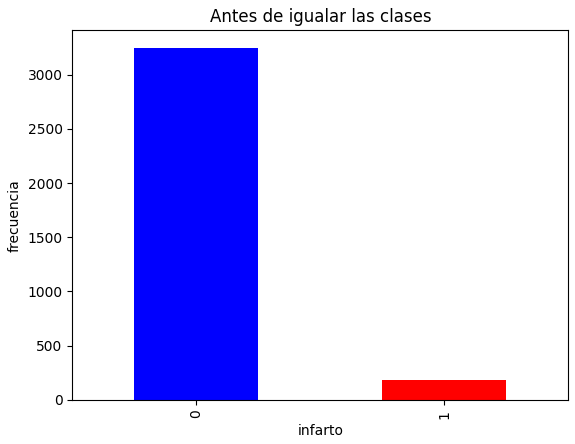
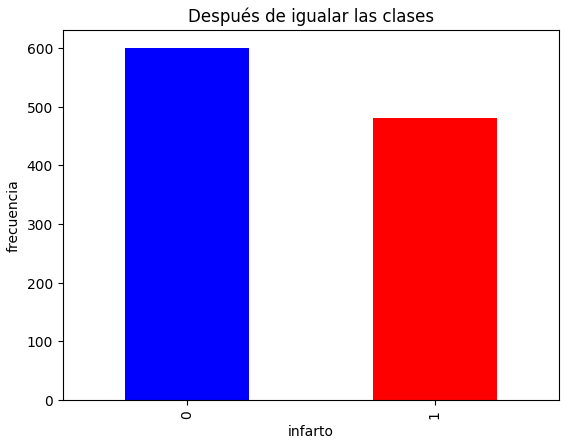
Una vez hecho el grid search obtenemos los hiperparámetros con más accuracy y menos loss, para poder hacer un estudio del más preciso y que tenga el menor overfitting.



También hemos hecho uso de gráficas para medir el mejor momentum y observar cómo afecta al modelo, especificado más [abajo](#_8x99nm7nya0l), finalmente el modelo seleccionado como óptimo para continuar con las validaciones ha sido:



3.1.3 Para igualar las clases realizamos, técnicas de undersampling y oversampling como se comenta en [Pre-procesamiento](#_1fob9te)

.

Para identificar los hiperparámetros que mejor se adaptan a nuestro modelo, realizamos un estudio utilizando Grid Search. Esta técnica permite evaluar y comparar el rendimiento de diferentes combinaciones de hiperparámetros sobre un conjunto de pruebas. Después de probar diversas combinaciones, la configuración más efectiva resultó ser un learning rate de 0.1, cuatro capas ocultas y 45 neuronas en cada una de estas capas. Estos hiperparámetros permitieron alcanzar una precisión del 78%.



La selección de estos hiperparámetros apunta a mejorar tanto la capacidad de aprendizaje del modelo como su precisión.

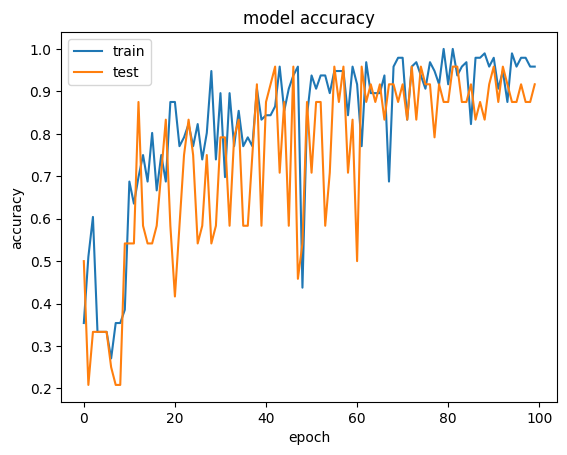
Un learning rate de 0.1 es perfecto porque encuentra un buen equilibrio entre la rapidez y la consistencia del aprendizaje, evitando que el modelo se quede atascado en pequeños errores durante el entrenamiento. Este ritmo asegura que el modelo se ajuste de manera efectiva sin adaptarse demasiado rápido, lo que podría impedirle encontrar la mejor solución.

Por otro lado, el uso de cuatro capas ocultas y 45 neuronas por capa contribuye a que el modelo sea lo suficientemente complejo para detectar los detalles cruciales en los datos médicos, pero sin llegar a ser tan complejo que aprenda de memoria los datos de entrenamiento. Este diseño ayuda al modelo a generalizar mejor, es decir, a funcionar eficazmente con nuevos datos que no ha visto anteriormente, lo que es esencial para predecir con precisión eventos médicos graves como los infartos. Además se verifica el teorema de aproximación universal ya que el número de neuronas que se predijo entra dentro del rango explicado.

Además, se ha decidido entrenar el modelo con 200 épocas. Esta cantidad de épocas permite que el modelo tenga múltiples oportunidades de aprender ajustando sus pesos de forma efectiva a través del tiempo. Es importante, sin embargo, observar el rendimiento del modelo a lo largo del entrenamiento para asegurar que no se está sobre ajustando a medida que se exponen repetidamente a los mismos datos de entrenamiento.

## 3.2. Validación

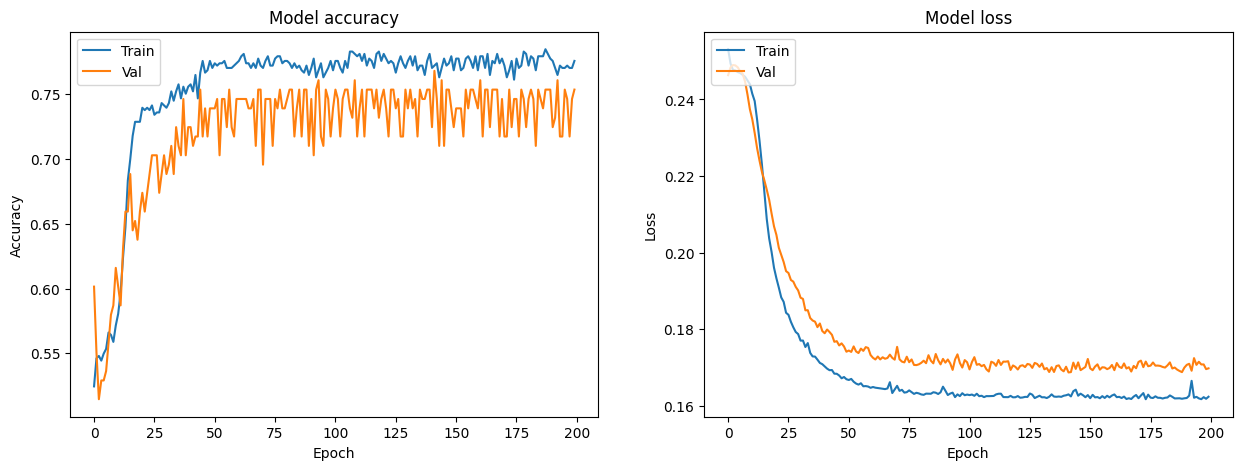
3.2.1. No se utilizó un método de validación para la primera práctica.

3.2.2. Para la validación de nuestro modelo, nos centramos en las gráficas de precisión obtenidas tras el entrenamiento. Se observa que la precisión de entrenamiento supera ligeramente a la de prueba, con una brecha que se hace más notable después de las primeras 25 épocas, lo cual puede sugerir un principio de overfitting. A pesar de la presencia de ruido y patrones en los datos, que el modelo parece haber aprendido, la cercanía entre las curvas de precisión indica que el modelo conserva una capacidad de generalización adecuada. 

3.2.3.Ahora procederemos a la validación de nuestro modelo utilizando el método de validación cruzada, un proceso que evalúa la capacidad del modelo para generalizar más allá de los datos con los que fue entrenado. Generalmente, los modelos tienden a mostrar un mejor rendimiento con datos ya conocidos. Sin embargo, es crucial que el modelo también maneje bien datos no vistos, que pueden presentar características o patrones diferentes. La validación cruzada, al ejecutar el modelo en varias combinaciones de entrenamiento y validación, nos permite identificar y corregir posibles desviaciones o sobreajustes. Este método divide el conjunto de datos original en múltiples subconjuntos, permitiendo que cada parte sea utilizada tanto para entrenamiento como para validación en diferentes iteraciones. Esto garantiza una evaluación más robusta y menos sesgada de las métricas finales, aumentando la su capacidad de generalizar correctamente sobre nuevos datos.

Para verificar una buena validación de los resultados, observamos el accuracy promedio obtenido de las diferentes iteraciones de la validación cruzada. Notamos que, a partir de la época 25, el accuracy del entrenamiento tiende a ser superior al de la validación. Esto sugiere que el modelo podría estar comenzando a sobreajustarse, aunque es común observar cierta desviación entre estos dos valores debido a la naturaleza de los datos no vistos en el conjunto de validación, que presenta características o patrones ligeramente diferentes. Esta diferencia es esencial para evaluar la capacidad de generalización del modelo de manera efectiva.Al igual que con el accuracy, el análisis del loss durante la validación cruzada nos proporciona información valiosa sobre el rendimiento del modelo. En la gráfica que muestra la evolución del loss a lo largo de las épocas, se puede apreciar que tanto el loss de entrenamiento como el de validación disminuyen a medida que avanzan las épocas, lo cual es una señal positiva de que el modelo está aprendiendo. Sin embargo, es importante prestar atención a la posible aparición de variaciones significativas, especialmente después de la época 25 donde parece estabilizarse la diferencia. Al ser una distancia tan mínima no podemos identificar un sobre ajustamiento al revés, al estar tan cercanos podemos intuir que el modelo está generalizando bien y es capaz de mantener un rendimiento consistente en datos no vistos.

#### Accuracy y Loss



## 3.3. Evaluación

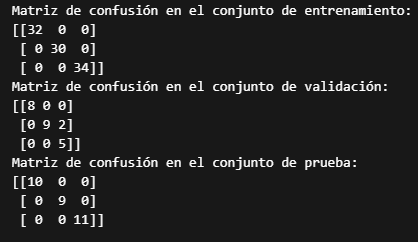
3.3.1. El modelo es demasiado simple y su problema también, en este caso no aplica la evaluación ya que se ve claramente como el hiperplano separa las dos clases.

3.3.2.Para evaluar el modelo hemos hecho uso de matrices de confusión del conjunto de entrenamiento, validación y pruebas. Mediante este proceso podemos predecir las clases de cada instancia de los tres conjuntos de lirios posibles a clasificar.

La matriz de confusión del conjunto de entrenamiento muestra una clasificación perfecta, indicando que el modelo se ha ajustado completamente a los datos de entrenamiento.

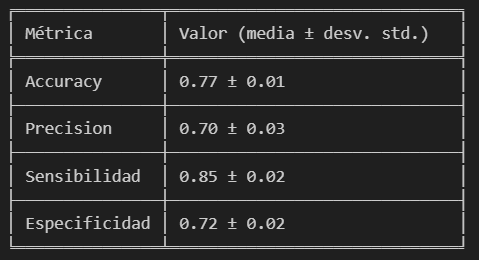
En el conjunto de validación observamos alguna clasificación incorrecta, lo que quiere decir que el modelo puede mejorar, posiblemente con un conjunto de datos mayor el modelo sea capaz de generalizar de mejor manera.

Finalmente, el conjunto de pruebas, el modelo actúa sobre datos no vistos en el conjunto de entrenamiento, no obtiene ningún fallo, lo que nos dice que el modelo tiene buena capacidad de generalización.



Lo que nos indica esta métrica podemos garantizar la precisión del modelo y su gran capacidad para generalizar

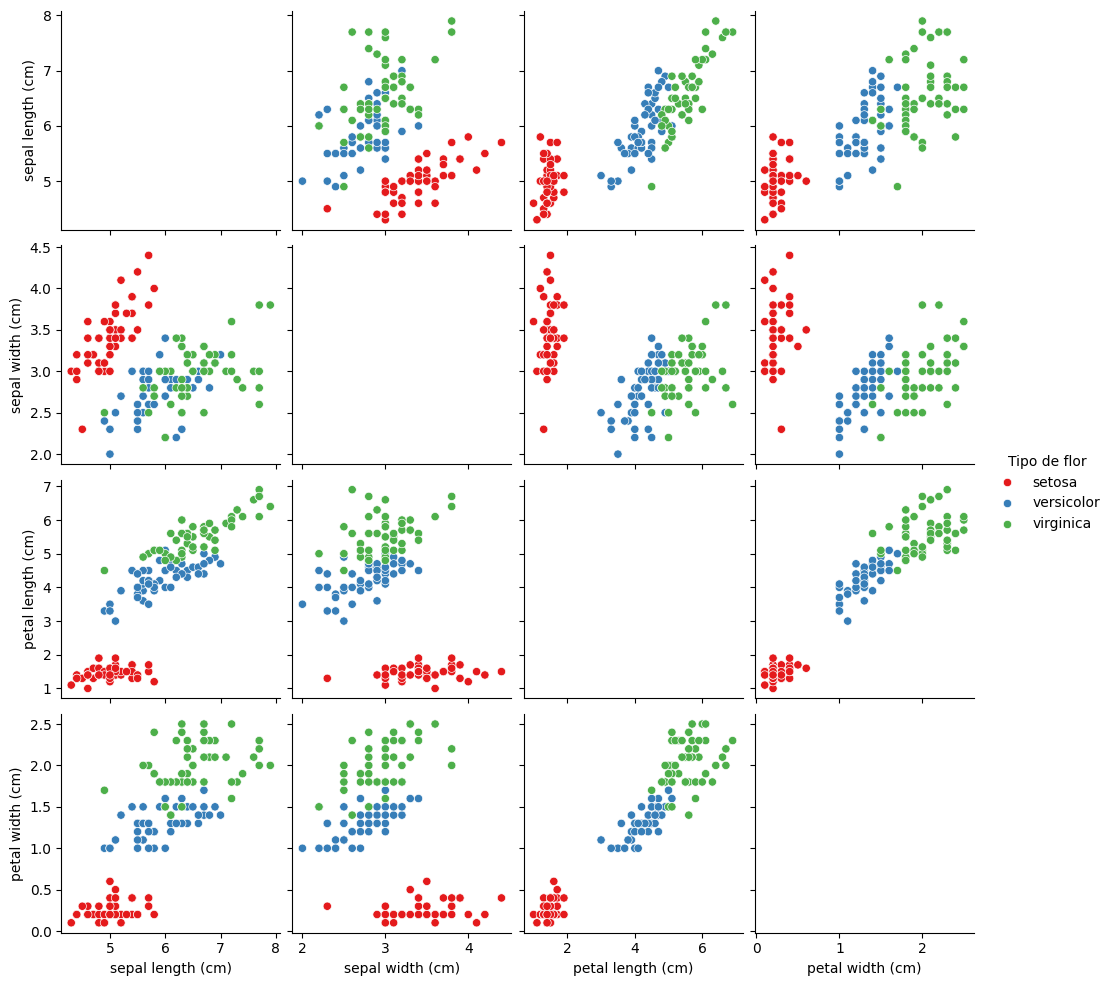
3.3.3. Para evaluar el entrenamiento del modelo, creamos una tabla con las métricas de specificity la tabla de evaluación, en ella vemos las accuracy y las desviaciones estándar por cada clase y conjunto de datos. Estudiando los valores de desviación estándar vemos que todos son bajos (alrededor de 0.02) lo que nos indica que los resultados de los modelos son similares y por lo tanto no están sesgados; viendo las demás métricas no destaca ningún valor entre los demás, son todos valores similares. Los valores de precisión y sensibilidad también son relativamente altos, lo que indica que el modelo es bueno para detectar la clase positiva (sensibilidad) y ser preciso cuando emite una predicción positiva (precisión). La especificidad de 0.72 sugiere que el modelo también es razonablemente bueno en la identificación de los verdaderos negativos, aunque hay más espacio para la mejora en comparación con la sensibilidad. Con una sensibilidad más alta en comparación con la especificidad podemos entender que el modelo está ligeramente inclinado hacia la predicción de la clase positiva.



# Cuestiones

## 4.1. Primera práctica

1. **Haz una gráfica por cada posible pareja de atributos donde representes con puntos de distinto color cada uno de los tres tipos de lirio. Selecciona y justifica que pareja de atributos vas a usar para la clasificación de cada par de tipos de lirios.**

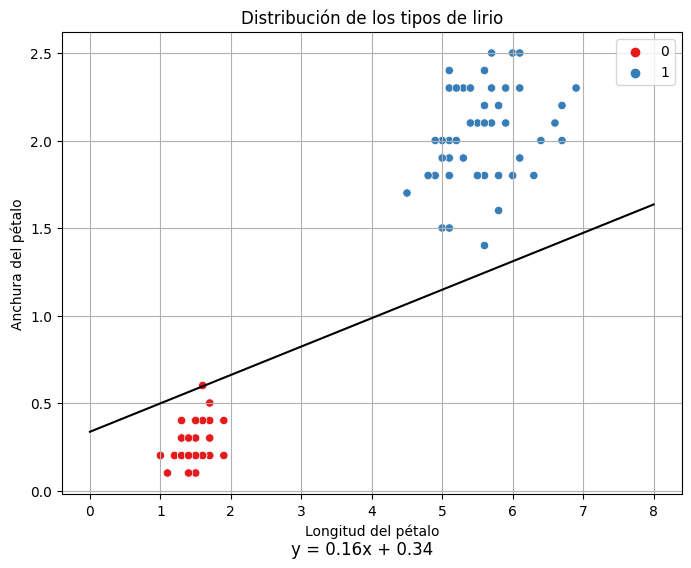


*Figura X. Gráfica de relación entre cada par de atributos del dataset Iris*

Una vez analizados los gráficos de dispersión, que representan los diferentes tipos de lirios de acuerdo con distintos pares de atributos, hemos observado que los atributos más adecuados para la clasificación son el petal length y el petal width. Elegimos estos dos atributos porque, según la gráfica correspondiente, las tres clases se distinguen con mayor claridad en comparación con las otras combinaciones de atributos. Esto nos indica cuáles son las características más distintivas para diferenciar entre las especies de lirios.

1. **Dibuja una gráfica con puntos de distinto color para cada tipo de lirio y dibuja en ella el hiperplano. Escribe de forma explícita la ecuación del hiperplano que se ha generado.**

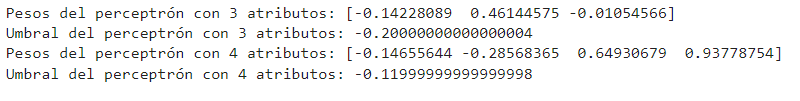
En este apartado, presentamos la distribución de los tipos de lirio según la relación entre los atributos longitud del pétalo (petal length) y anchura del pétalo (petal width). Dado que un perceptrón es un clasificador binario, procedemos a clasificar dos grupos: las setosas y los versicolor. Estos dos grupos están separados por un hiperplano, definido por la ecuación y = 0.16x + 0.34. Esta línea actúa como el límite de decisión que clasifica los lirios setosa, representados en rojo, de los versicolor, mostrados en azul.



1. **Elige uno de los pares de flores y modifica el código para entrenar dos Perceptrones con 3 y 4 atributos respectivamente. Compara los resultados con los obtenidos para ese par en el apartado anterior.**

Para el perceptrón de tres atributos, seleccionamos las características longitud del pétalo (petal length), anchura del pétalo (petal width) y longitud del sépalo (sepal length). Una vez entrenado el modelo, obtenemos una precisión (Accuracy) de 1.00 y un umbral (bias) de -0.20. Para el perceptrón de cuatro atributos, elegimos los cuatro atributos disponibles, que incluyen los tres anteriores más la anchura del sépalo (sepal width). Este modelo también nos proporciona una precisión de 1.00 y un umbral de -0.3. Comparando estos dos modelos con el de dos atributos, que también alcanzó una precisión de 1 y un umbral de -0.17, podemos inferir que el incremento en la cantidad de atributos no afecta la precisión en este caso particular; Esto se debe a que las clases son fáciles de distinguir ya que son linealmente separables, es decir, añadir más atributos no ayuda a que el perceptrón separe mejor porque ya está separando de forma óptima solo con 2 atributos.(En conclusión podemos ver que el umbral varía mínimamente en comparación al entrenamiento con dos atributos por su facilidad de clasificarlos debido a la poca cantidad de datos )

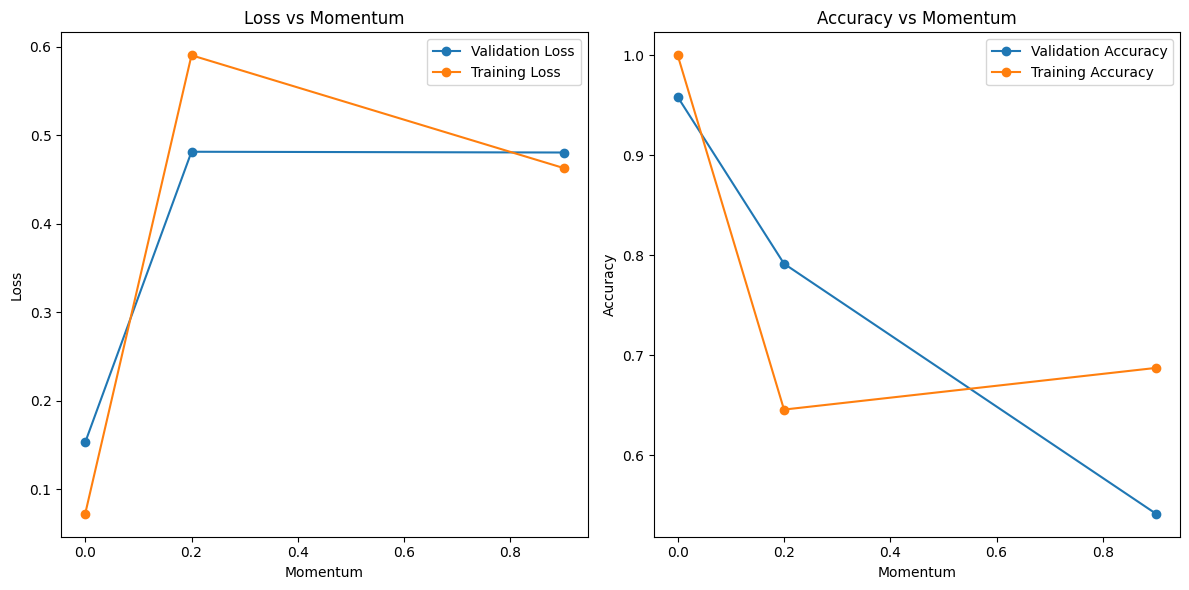




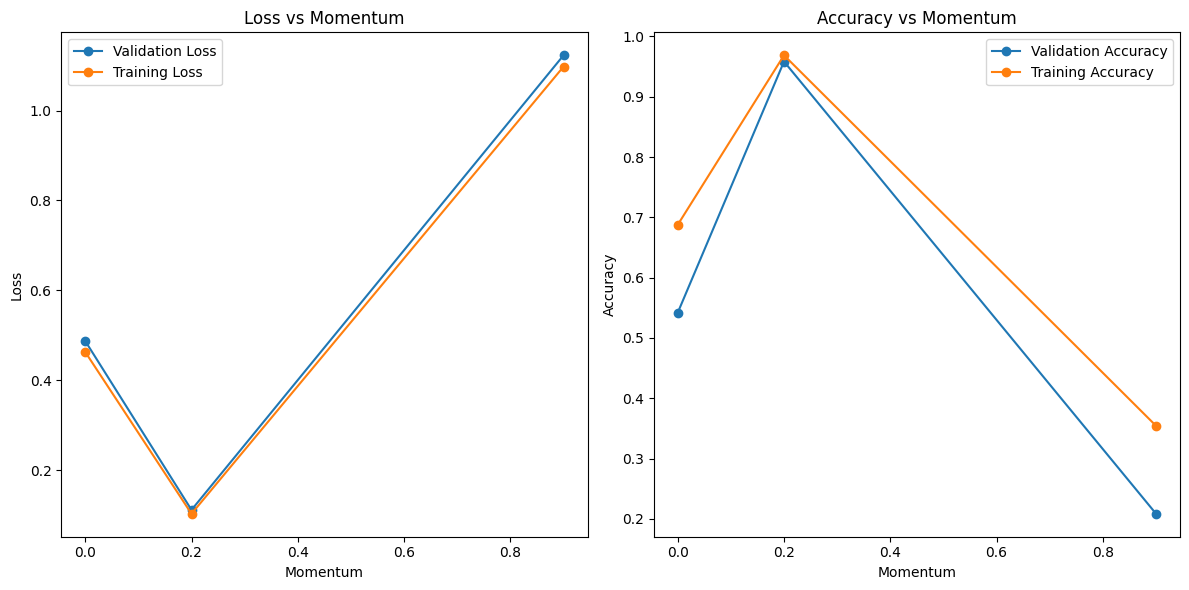
## 4.2. Segunda práctica

1. **¿Cómo influye el parámetro de momentum del optimizador (SGD) durante el entrenamiento? ¿Qué función de pérdida has escogido? Justifica tu respuesta y aporta datos objetivos**

El momentum influye significativamente en el entrenamiento de modelos MLP, ayuda a acelerar el SGD en la dirección correcta, en nuestros entrenamientos podemos observar que el momentum 0.0 es el que obtiene mejores resultados para modelos con 7 capas de profundidad, ya que maximiza el accuracy y disminuye el loss del modelo.



En modelos con 5 capas de profundidad, observamos que se obtienen mejores resultados con un momentum de 0.2, esto es debido a que al tener una menor profundidad necesita un pequeño impulso en el aprendizaje

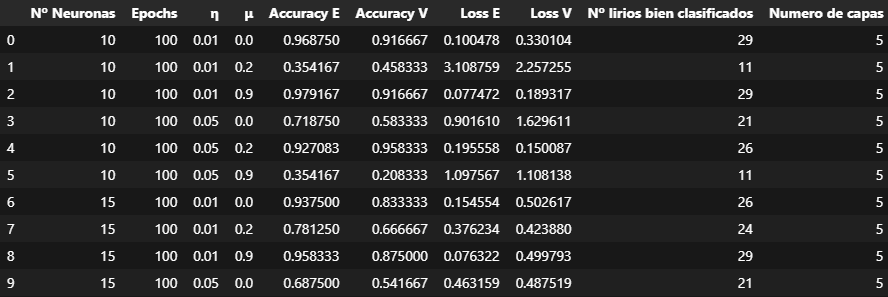


En ambos modelos el momentum debe ser muy bajo o incluso nulo debido a la baja cantidad de datos que nos proporciona la librería “iris”.

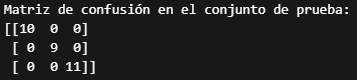
La función de pérdida que hemos usado es categorical\_crossentropy mide el rendimiento del modelo en una salida de clasificación de múltiples clases en la que las etiquetas, codificadas en formato one-hot encoding (la clase 1 podría representarse como [1, 0, 0], la clase 2 como [0, 1, 0], y la clase 3 como [0, 0, 1]) , son mutuamente excluyentes. Es decir, cada entrada de datos debe pertenecer exactamente a una clase.

1. **¿Qué relación hay entre Loss/Accuracy/Nº lirios? ¿Cuántos lirios de cada tipo están bien clasificados? Compara los resultados con los obtenidos de la práctica anterior.**

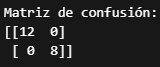
Mediante la tabla del grid search podemos observar una clara relación entre loss/accuracy y número de lirios. El accuracy podemos observar que afecta de manera proporcional, cuanto mayor es el accruacy mayor es el número de lirios bien clasificados. Por otro lado el loss actúa de manera inversamente proporcional al número de lirios bien clasificados, cuanto mayor loss tiene un modelo, peor clasifica, y por tanto menor número de lirios bien clasificados



El mayor número de lirios bien clasificados en los mejores modelos es 30/30, en la tabla se muestra el número 29 por la posición del array de lirios bien clasificados, que significa de igual manera 30 lirios bien clasificados. Esto podemos comprobarlo mediante la matriz de confusión.



Comparándolo con la primera práctica, ambos modelos clasifican bien el 100% de los datos de prueba. Aunque el uso del MLP da la posibilidad de clasificar patrones más complejos, posibilitando en este caso clasificar los tres tipos de lirios posibles en vez de únicamente 2 tipos.

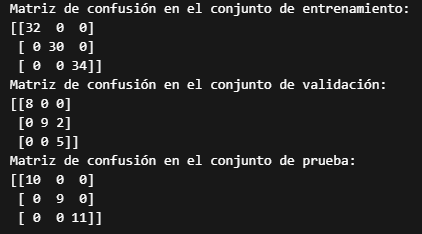


1. **Prueba sobre el conjunto de test y compara los resultados obtenidos con los del entrenamiento y validación. ¿Ha generalizado el modelo?**

Basándonos en las matrices de confusión proporcionadas:

* Conjunto de entrenamiento: El modelo clasificó todos los ejemplos correctamente, lo que sugiere que ha aprendido a identificar las características que distinguen a cada clase dentro del conjunto de datos de entrenamiento.
* Conjunto de validación: En este conjunto, el modelo también realizó un buen trabajo, pero con un pequeño error: dos ejemplos de la segunda clase fueron clasificados incorrectamente en la tercera clase. Esto puede ser indicativo de un comienzo de sobreajuste o de que algunas características de esas dos clases pueden solaparse, haciendo que la distinción entre ellas sea más difícil.
* Conjunto de prueba: Aquí el modelo ha clasificado todos los ejemplos correctamente, al igual que con el conjunto de entrenamiento, lo que sugiere que está generalizando bien a nuevos datos.

Comparando los resultados de las matrices de confusión de entrenamiento, validación y prueba, podemos concluir que el modelo ha generalizado bastante bien, aunque hay una señal de precaución en la confusión entre la segunda y tercera clase en el conjunto de validación.



## 4.3. Tercera práctica

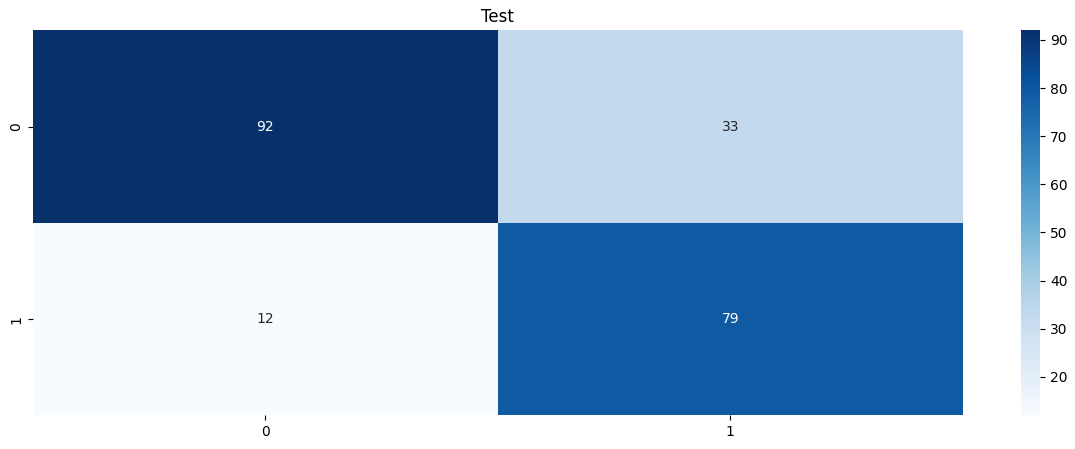
1. **Determina si se ha realizado un buen entrenamiento y el modelo ha generalizado correctamente.**

Para asegurarnos que el entrenamiento ha sido correcto nos apoyamos en la tabla de evaluación, podemos destacar que la desviación estándar de cada métrica es muy baja lo que nos indica que el modelo no está sesgado ya que se han dado resultados similares en todas las métricas del modelo. Si nos centramos en la [matriz de confusión](#_kc23q1d81sbb) podemos ver que hay casos en los que el modelo no ha generalizado bien por ejemplo cuando dice que si es infarto pero no lo es, es decir un falso negativo, también es importante fijarse en la gráfica del [loss](#_wll622u87awx) ya que la línea de validación se muestra un poco por encima de la de train, lo que indica que el modelo no es capaz de generalizar perfectamente pero si lo hace en la mayoría de los casos. En conclusión se aprecia que el modelo aún tiene espacio para mejorar, especialmente en lo que respecta a la generalización en datos no vistos.

1. **Interpreta cada uno de los valores de la matriz de confusión para el problema planteado**

Una vez entrenado el modelo con los hiperparámetros que hemos obtenido en el grid search, realizamos la matriz de confusión que nos sirve para evaluar el rendimiento del modelo de clasificación. La matriz de confusión nos muestra cuántas predicciones fueron correctas y cuántas incorrectas, divididas en cuatro categorías: verdaderos positivos, verdaderos negativos , falsos positivos y falsos negativos. Al observarla, podemos destacar que la cantidad de verdaderos positivos y verdaderos negativos es significativamente mayor en comparación con los falsos positivos y los falsos negativos . Esto sugiere que, en la mayoría de los casos, cuando el modelo predice esta predicción suele ser correcta. Sin embargo, observamos algunos casos en los que el modelo no ha generalizado bien provocando predicciones erróneas. Esto es generado cuando el modelo predice un infarto y en realidad no ocurrió uno, así como en los casos en los que no detecta un infarto que sí ocurrió. Estos errores son importantes ya que nos encontramos ante un caso médico, por ello nos tenemos que fijar en que los falsos negativos son mayores que los falsos positivos , esto indica que hay veces que un paciente puede tener un infarto y nuestro modelo no lo identifica correctamente. En conclusión este modelo puede ayudar a los médicos a guiarse pero siempre teniendo en cuenta que puede fallar con ciertos pacientes, por eso se debe contrastar con más pruebas.

##### Matriz confusión infartos



1. **¿Detectas algún problema en tus datos? En caso de existir, modifica los datos para que esto no ocurra y vuelve a entrenar tu modelo**

Al realizar el pre-procesamiento de datos vimos que las clases estaban desbalanceadas , con un histograma diferenciamos el número de clases, observamos una proporción desbalanceada de 95% a 5%. Una vez detectamos este problema, procedimos a igualar la cantidad de estas utilizando técnicas de undersampling como el oversampling como se explica en el apartado de [Pre-procesamiento](#_1fob9te) , utilizando estas técnicas nos aseguramos de que nuestro modelo no este sesgado, ya que si no hubiésemos detectado este problema el accuracy en el entrenamiento seguiría siendo muy alto porque estaría clasificando todos a la clase mayoritaria que en este caso son los no infartos.

1. **¿Qué variables influyen más a la hora de que un paciente padezca un infarto? ¿Qué variables influyen menos? Describe el método que has llevado a cabo para llegar a esa conclusión y presenta los resultados.**

Las variables que más influyen a la hora de predecir son la edad, que destaca con una influencia del 38.02%, indicando que a mayor edad, mayor es la posibilidad de sufrir un infarto. Le sigue el BMI, que se refiere al índice de masa corporal, con un 9.65% que muestra que cuanto mayor es la tasa, también aumenta la posibilidad de que se produzca un infarto. Por último, la ocupación de menores de edad con un 8.82%, lo que puede reflejar aspectos socioeconómicos o de estilo de vida. Por otro lado, las variables con menor influencia incluyen la ocupación en el sector privado con un 1.35%, la ocupación como funcionario público con un 1.08%, y el tipo de residencia con un 0.96%, lo que sugiere que estos factores tienen un impacto relativamente bajo en la predicción de un evento cardíaco. Hemos obtenido estos parámetros mediante el cálculo de la importancia relativa de cada característica en el modelo. Este proceso lo hemos conseguido obteniendo los pesos del modelo, calculando su valor absoluto obteniendo la magnitud de influencia, normalizar estos valores para que sumen 100% y representarlos. Finalmente, asociamos estos valores con sus variables para identificar cuáles tienen mayor impacto en la predicción de nuestro modelo. (Goodfellow et al., 2016, 21)

# Bibliografía

(n.d.). Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, 65(6), 386–408.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Jiménez Pastor, A. M. (n.d.). Aprendizaje profundo y biomarcadores de imagen en el estudio de enfermedades metabólicas y hepáticas a partir de resonancia magnética y tomografía computarizada. riunet.upv.es. https://riunet.upv.es/handle/10251/202602

López Herraiz, J. (n.d.). Perceptrón Multicapa. Wikipedia. Retrieved April 22, 2024, from https://espanol.libretexts.org/Matematicas/Las\_matematicas\_de\_la\_inteligencia\_artificial/06%3A\_Redes\_Neuronales/6.3%3A\_Perceptron\_Multicapa

Pattern Recognition and Machine Learning. (n.d.). ResearchGate. Retrieved April 22, 2024, from https://www.researchgate.net/publication/221995544\_Pattern\_Recognition\_and\_Machine\_Learning

The Perceptron. (n.d.). GitHub Pages. Retrieved April 22, 2024, from https://com-cog-book.github.io/com-cog-book/features/perceptron.html