



MEMORIA LABORATORIO II

Julián Prieto Velasco, Pedro José Paniagua Falo,
Guillermo Ramírez Cárdenas, Javier Muñoz Rojas





Índice

1	Introducción	6
2	Práctica 1.....	6
2.1	Perceptrón.....	6
2.2	Puerta AND.....	6
2.2.1	Implementación	7
2.2.2	Entrenamiento	7
2.2.3	Cuestiones AND:.....	9
2.3	Puerta XOR	11
2.3.1	¿Por qué se ha implementado así?	11
2.3.2	Cuestiones XOR:	13
3	Práctica 2.....	16
3.1	Implementación:	16
3.2	Modelos:.....	17
4	Práctica 3.....	20
4.1	Implementación	20
4.2	Cuestiones	23
4.2.1	Cuestión 1.....	23
4.2.2	Cuestión 2.....	23
4.2.3	Cuestión 3.....	24
4.2.4	Cuestión 4.....	29
4.2.5	Cuestión 5.....	33
5	Bibliografía	35
6	Anexo I.....	38
7	Anexo II.....	39

Índice de Ilustraciones

Ilustración 1 Código de Iniciación del Perceptrón [4]	7
Ilustración 2 Código de Entrenamiento de la Neurona Perceptrón AND [16]	8
Ilustración 3 Separatriz del Perceptrón [3]	10
Ilustración 4 Variación del error de la Red [9]	11
Ilustración 5 Esquema Electrónico de la Puerta XOR creada a partir de las puertas AND, OR y NAND [14]	12
Ilustración 6 Red neuronal Perceptrón para crear usarla como una Puerta XOR [15]	13
Ilustración 7 Variación del Error de la red Neuronal.....	15
Ilustración 8 Código de entrenamiento del MLP	17
Ilustración 9 Salida del programa del MLP para 8 neuronas y 10000 épocas	18



Ilustración 10 Salida del Programa del MLP para 4 neuronas y 5000 épocas.....	18
Ilustración 11 Función de Pérdida según el número de épocas	19
Ilustración 12 Importación de las librerías.....	20
Ilustración 13 Creación de carpetas y lectura del CSV	21
Ilustración 14 Código para el Preprocesamiento y Normalización de los datos.....	21
Ilustración 15 Código de entrenamiento del MLP	22
Ilustración 16 Modelo del MLP para 2 neuronas dentro de la capa oculta	24
Ilustración 17 Modelo del MLP para 4 neuronas dentro de la capa oculta	24
Ilustración 18 Modelo del MLP para 8 neuronas dentro de la capa oculta	24
Ilustración 19 Matriz de confusión para 2 neuronas y 25 épocas	24
Ilustración 20 Matriz de confusión para 2 neuronas y 50 épocas	25
Ilustración 21 Matriz de confusión para 2 neuronas y 100 épocas	25
Ilustración 22 Matriz de confusión para 4 neuronas y 25 épocas	25
Ilustración 23 Matriz de confusión para 4 neuronas y 50 épocas	26
Ilustración 24 Matriz de confusión para 4 neuronas y 100 épocas	26
Ilustración 26 Matriz de confusión para 8 neuronas y 50 épocas	27
Ilustración 25 Matriz de confusión para 8 neuronas y 25 épocas	27
Ilustración 27 Matriz de confusión para 8 neuronas y 100 épocas	27
Ilustración 28 Modificación de los datos	29
Ilustración 29 Matriz de confusión para 2 neuronas y 25 épocas	29
Ilustración 30 Matriz de confusión para 2 neuronas 50 épocas después de modificar los datos	30
Ilustración 32 Matriz de confusión para 4 neuronas 25 épocas después de modificar los datos	30
Ilustración 31 Matriz de confusión para 2 neuronas 100 épocas después de modificar los datos	30
Ilustración 33 Matriz de confusión para 4 neuronas 50 épocas después de modificar los datos	31
Ilustración 34 Matriz de confusión para 4 neuronas 100 épocas después de modificar los datos	31
Ilustración 35 Matriz de confusión para 8 neuronas 25 épocas después de modificar los datos	31
Ilustración 36 Matriz de confusión para 8 neuronas 50 épocas después de modificar los datos	32
Ilustración 37 Matriz de confusión para 8 neuronas 100 épocas después de modificar los datos	32

Índice de Tablas

Tabla 1 Tabla de Verdad de la Puerta AND [3].....	7
Tabla 2 Tabla de Variación de pesos en las Iteraciones de la Neurona Perceptrón con la Función AND [10]	10
Tabla 3 de Karnaugh de XOR [12].....	11
Tabla 4 Tabla de Verdad de la Puerta AND	12
Tabla 5 Tabla de Verdad de la Puerta OR.....	12
Tabla 6 Tabla de Verdad de la Puerta NAND.....	12
Tabla 7 Iteraciones de la neurona Perceptrón para la función XOR	14



Tabla 8 Variaciones de la Red según Número de neuronas de la capa Oculta, Iteraciones, salida y error.....	18
Tabla 9 Variación de resultados del MLP según las neuronas de la capa oculta	23
Tabla 10 componentes con sus variables y cómo influyen	33

Índice de Ecuaciones

Ecuación 1 Suma ponderada de las entradas [5]	8
Ecuación 2 Función de activación Límite aplicada a la Suma ponderada [6].....	8
Ecuación 3 Comportamiento de la Función límite según el theta. [7]	8
Ecuación 4 Calculo del error de la salida de la neurona [8]	8
Ecuación 5 Reajuste de la matriz de pesos de la neurona [9]	8
Ecuación 6 Ecuación de la puerta XOR del mapa de Karnaugh [2]	11
Ecuación 7 Ecuación añadiendo puertas AND a cada lado para simplificar [2]	12
Ecuación 8 Ecuación tras simplificar añadiendo Puertas AND a las Puertas XOR [2]	12
Ecuación 9 Ecuación final de la puerta XOR completamente Simplificada para usarla con una puerta AND y otra OR [2]	12





1 Introducción

La siguiente memoria describe el desarrollo de una práctica de laboratorio en la que se implementaron diferentes modelos de redes neuronales de aprendizaje supervisado utilizando Python y las librerías *NumPy*, *pandas*, *scikit-learn*, *Matplotlib*, *Keras* y *TensorFlow*. El objetivo de la práctica es desarrollar distintas implementaciones de redes neuronales que permitan resolver diferentes casos de uso, comenzando con la implementación más sencilla del funcionamiento de una neurona sencilla y terminando con la construcción de un MLP capaz de hacer predicciones para un caso concreto planteado.

En la primera práctica, se implementó un Perceptrón para resolver funciones lógicas usando un modelo neuronal simple. Se programó un Perceptrón cuyos pesos iniciales tendrían un valor al azar entre -1.0 y 1.0, ambos incluidos, y que se entrenaría usando la Ley de Hebb. Se resolvieron las funciones lógicas AND y XOR, probando diferentes learning rates y umbrales para encontrar el óptimo. Los resultados se entregaron en el archivo L2P1-Perceptron.csv.

En la segunda práctica, se utilizó la librería *Keras/TensorFlow2* para construir y entrenar un MLP que resolviera problemas no lineales. Se creó un modelo neuronal *fully connected* y secuencial que resolviera la función XOR, probando con diferentes números de neuronas en la capa oculta y varios valores de épocas, usando como optimizador 'Adam', funciones de activación 'ReLU' y 'sigmoide', y función de error '*Mean Squared Error*'. Los resultados se registraron en tablas distintas para cada una de las dos funciones de activación propuestas.

En la tercera práctica, se utilizó la librería *Keras/TensorFlow2* y una arquitectura MLP para predecir si una persona se contratará o no un producto bancario. Se realizó un análisis exploratorio de datos y se implementó el modelo MLP, se ajustaron los hiperparámetros y se evaluó su desempeño con varias métricas. Los resultados se presentaron en una tabla comparativa para evaluar el mejor modelo.

2 Práctica 1

2.1 Perceptrón

Un perceptrón es un tipo de modelo de aprendizaje automático de redes neuronales artificiales que se utiliza para clasificar objetos en diferentes categorías. Fue desarrollado por el científico informático estadounidense Frank Rosenblatt en 1957. [1]

El perceptrón consta de una capa de entrada que recibe los datos de entrada, una capa de salida que produce la salida del modelo y una capa de peso que realiza la multiplicación de los datos de entrada por los pesos asociados a cada entrada. El perceptrón puede tener una o varias neuronas, cada una de las cuales representa una categoría diferente en la que se pueden clasificar los datos de entrada. [2]

Durante el entrenamiento del modelo, se ajustan los pesos de las conexiones entre las neuronas para minimizar el error de clasificación. Una vez que el modelo ha sido entrenado, puede utilizarse para clasificar nuevos datos de entrada en una de las categorías previamente definidas.

El perceptrón es un modelo relativamente simple, pero puede ser muy efectivo en tareas de clasificación binaria y problemas de clasificación multiclase con datos linealmente separables.

2.2 Puerta AND

En esta práctica se ha pedido realizar una neurona de tipo Perceptrón que se comporte como una puerta AND, para ello se debe estudiar la tabla de verdad de esta puerta.



Entrada X	Entrada Y	Salida
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 1 Tabla de Verdad de la Puerta AND [3]

Como vemos en la Tabla1 las entradas varían entre 0 y 1 y la salida solo se activará si ambas entradas son igual a 1.

Para implementar esto en código inicializamos una matriz de entradas y la matriz de salidas esperadas tal y como se muestra en la tabla de verdad. Creamos una matriz llamada salidas donde se guardarán cada una de las salidas de esta neurona. Además, se inicializa una matriz de pesos aleatorios entre 0 y -1, se solicita al usuario que introduzca por teclado la variable theta, el coeficiente de aprendizaje y el número de épocas. Por último, se inicializa la ruta en la que se guardarán las imágenes de las gráficas para ver cómo aprende la neurona.

2.2.1 Implementación

```
#Iniciación del Perceptrón
entradas = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

salidas_esperadas = np.array([0, 0, 0, 1])

salidas = []
#Vector aleatorio de datos entre 0 y 1
matriz_pesos = np.round(np.random.uniform(low = -1, high = 1, size= (2,)), 2)

teta = float(input("Introduzca el Teta: "))

lr = float(input("Introduzca el Learning Rate: "))

epocas = int(input("Introduzca el numero de repeticiones: "))

cont = 0

ruta = "Evolucion_AND/imagen" + str(cont) + ".png"
```

Python

Ilustración 1 Código de Iniciación del Perceptrón [4]

2.2.2 Entrenamiento

Para entrenar la neurona del perceptrón se ha implementado el código que se muestra en la Ilustración 2.

En primer lugar, inicializamos un array de error histórico en el que guardaremos los errores que se van produciendo durante el entrenamiento. Se han realizado dos bucles anidados, uno que abarca de 0 al número de iteraciones que haya introducido el usuario por teclado. Cada vez que se realiza una iteración del primer bucle se inicializa un array salidas en el que se almacenan las salidas que va sacando la neurona. El segundo bucle abarca de 0 a 4, este cuatro es por el número de posibilidades que existen en la neurona.

En cada iteración del bucle, se calcula la salida que tendrá la neurona con la suma ponderada como se muestra en la ecuación 1:



$$S = \sum_{i=1}^n \omega_i * x_i$$

Ecuación 1 Suma ponderada de las entradas [5]

Después se calcula la salida real gracias a la función de activación, en este caso se usa la función límite.

$$Y = F(S, \theta) = f\left(\sum_{i=1}^n w_i * x_i - \theta\right)$$

Ecuación 2 Función de activación Límite aplicada a la Suma ponderada [6]

$$F(S, \theta) = \begin{cases} +1 & \text{si } S > \theta \\ 0 & \text{si } S \leq \theta \end{cases}$$

Ecuación 3 Comportamiento de la Función límite según el theta. [7]

A continuación se calcula el error que se produce al calcular la salida de la neurona. Para ello se realiza la siguiente operación:

$$E = y_i - d_i$$

Ecuación 4 Calculo del error de la salida de la neurona [8]

Si el error es distinto de cero, es decir, la salida no ha sido la esperada se procede a corregir los pesos de la neurona. Para ello, se aplica la siguiente ecuación:

$$w_{ij}^{t+1} = w_{ij}^t + (E * x_i * \eta)$$

Ecuación 5 Reajuste de la matriz de pesos de la neurona [9]

Por último, cada vez que se produce un ajuste en la matriz de pesos se imprimen por pantalla las regiones que se encuentran en ese instante en el problema y la posición de la recta que las separa y se guarda en la ruta especificada. Además, se muestra cada módulo de 10 iteraciones datos de cómo aprende la neurona.

```
Entrenamiento

errorHistorico = []
periodo = 0
for periodo in range(epocas):
    salidas = []
    for i in range(4):
        s = matriz_pesos[0] * entradas[i][0] + matriz_pesos[1] * entradas[i][1]
        salida_real = limite(s, teta)
        salidas.append(salida_real)
        error = salidas_esperadas[i] - salidas[i]
        if error != 0:
            matriz_pesos = matriz_pesos + (lr * error * entradas[i])
            imprimir_regiones(entradas, salidas_esperadas, matriz_pesos, teta, ruta)
            cont += 1
        ruta = "Evolucion_AND/imagen" + str(cont) + ".png"
    errorHistorico.append(abs(error).sum())
    errorTotal = salidas_esperadas - salidas

    if periodo % 10 == 0:
        print("-----")
        print("Iteración ", periodo)
        print("Error total: ", errorTotal)
        print("matriz_pesos: ", matriz_pesos)
        print("Entradas: ", entradas[i])
        print("Salidas esperadas: ", salidas_esperadas)
        print("salidas: ", salidas)
        print("Error: ", error)
        print("Entradas: ", 5)
        print("Error total: ", errorTotal)
        print("Learning Rate: ", lr)
        print("Theta: ", teta)
        print("-----")

    encontrado = 0
    for k in range(4):
        if errorTotal[k] == 0:
            encontrado += 1

    if encontrado == 4:
        print("-----")
        print("EPOCA: ", epocas)
        print("Entrenamiento terminado con " + str(epocas) + " epocas")
        print("-----")
        print("matriz_pesos: ", matriz_pesos)
        print("Entradas: ", entradas[i])
        print("Salidas esperadas: ", salidas_esperadas)
        print("salidas: ", salidas)
        print("Error: ", error)
        print("Entradas: ", 5)
        print("Error total: ", errorTotal)
        print("Learning Rate: ", lr)
        print("Theta: ", teta)
    else:
        print("No he aprendido con: ", epocas)
        print("Learning Rate: ", lr)
        print("Theta: ", teta)
```

Ilustración 2 Código de Entrenamiento de la Neurona Perceptrón AND [34]



2.2.3 Cuestiones AND:

El modelo ha sido entrenado con distintos valores para comprobar cuáles son los óptimos para el entrenamiento del modelo. Tras mucho entrenamiento se ha llegado a la conclusión de que los valores óptimos son: **Eta: 0.1. Número de Iteraciones: 100 Learning Rate: 0.1**

En este apartado se muestra la tabla de Iteraciones de la Neurona Perceptrón para la Puerta AND. Se ha decidido mostrar las 4 primeras y las 4 últimas para ver los más significativos.

Iteración	Entradas		Pesos Iniciales		Salida Real	Salida Esperada	Error	Pesos Finales	
	X1	X2	W1	W2					
0	0	0	-0.06	-0.62	0	0	0	-0.06	-0.62
	0	1	-0.06	-0.62	0	0	0	-0.06	-0.62
	1	0	-0.06	-0.62	0	0	0	-0.06	-0.62
	1	1	-0.06	-0.62	0	1	1	0.04	-0.52
1	0	0	0.01	-0.44	0	0	0	0.01	-0.44
	0	1	0.04	-0.52	0	0	0	0.04	-0.52
	1	0	0.04	-0.52	0	0	0	0.04	-0.52
	1	1	0.04	-0.52	0	1	1	0.14	-0.42
2	0	0	0.14	-0.42	0	0	0	0.14	-0.42
	0	1	0.14	-0.42	0	0	0	0.14	-0.42
	1	0	0.14	-0.42	1	0	-1	0.14	-0.32
	1	1	0.14	-0.32	0	1	1	0.14	-0.32
3	0	0	0.14	-0.32	0	0	0	0.14	-0.32
	0	1	0.14	-0.32	0	0	0	0.14	-0.32
	1	0	0.14	-0.32	1	0	-1	0.04	-0.32
	1	1	0.04	-0.32	0	1	1	0.14	-0.22
4	0	0	0.14	-0.22	0	0	0	0.14	-0.22
	0	1	0.14	-0.22	0	0	0	0.14	-0.22
	1	0	0.14	-0.22	1	0	-1	0.04	-0.22
	1	1	0.04	-0.22	0	1	1	0.14	-0.12
5	0	0	0.14	-0.12	0	0	0	0.14	-0.12
	0	1	0.14	-0.12	0	0	0	0.14	-0.12
	1	0	0.14	-0.12	1	0	-1	0.04	-0.12
	1	1	0.14	-0.02	0	1	1	0.14	-0.02
6	0	0	0.14	-0.02	0	0	0	0.14	-0.02
	0	1	0.14	-0.02	0	0	0	0.14	-0.02
	1	0	0.14	-0.02	1	0	-1	0.04	-0.02
	1	1	0.14	0.08	0	1	1	0.14	0.08
7	0	0	0.14	0.08	0	0	0	0.14	0.08



7	0	1	0.14	0.08	0	0	0	0.14	0.08
	1	0	0.14	0.08	1	0	-1	0.04	0.08
	1	1	0.04	0.08	0	1	1	0.04	0.08
8	0	0	0.04	0.08	0	0	0	0.04	0.08
	0	1	0.04	0.08	0	0	0	0.04	0.08
	1	0	0.04	0.08	0	0	0	0.04	0.08
	1	1	0.04	0.08	1	1	0	0.04	0.08

Tabla 2 Tabla de Variación de pesos en las Iteraciones de la Neurona Perceptrón con la Función AND [10]

Como vemos en la Tabla2, con la octava iteración la red ya soluciona la función AND sin cometer errores. En el notebook se muestran las demás iteraciones, pero en aras de no recargar mucho la memoria se ha decidido tan solo analizar las 8 primeras. Como vemos cada vez que se comete un error se modifican las matrices de pesos para corregir el error.

En la carpeta Evolución_And de la entrega se puede observar un vídeo con la evolución completa del aprendizaje de la red, además adjuntamos la imagen de una de las iteraciones correctas.

Con esto, podemos afirmar que la ecuación del hiperplano es:

$$0.04 * x_1 + 0.08 * x_2 + \theta = 0$$

Ecuación 6 Ecuación del Hiperplano generado para la función AND [11]

Siendo x_1 y x_2 las entradas de la neurona perceptrón.

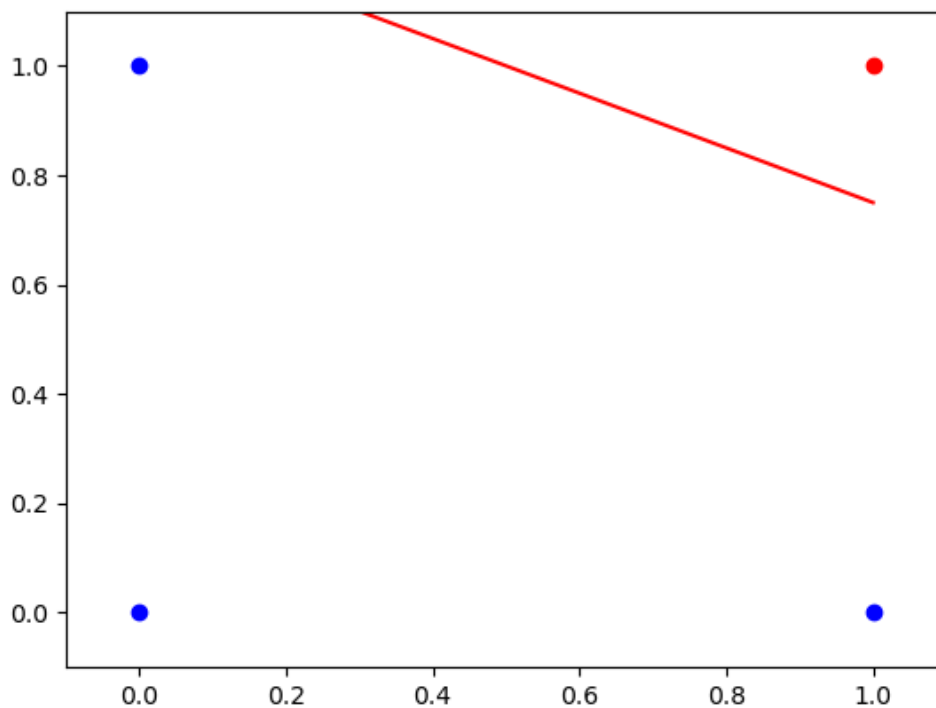


Ilustración 3 Separatriz del Perceptrón [3]



Además, se ha decidido implementar una función para mostrar la variación del error a lo largo del aprendizaje. Como vemos a las 7 iteraciones la red ya ha aprendido.

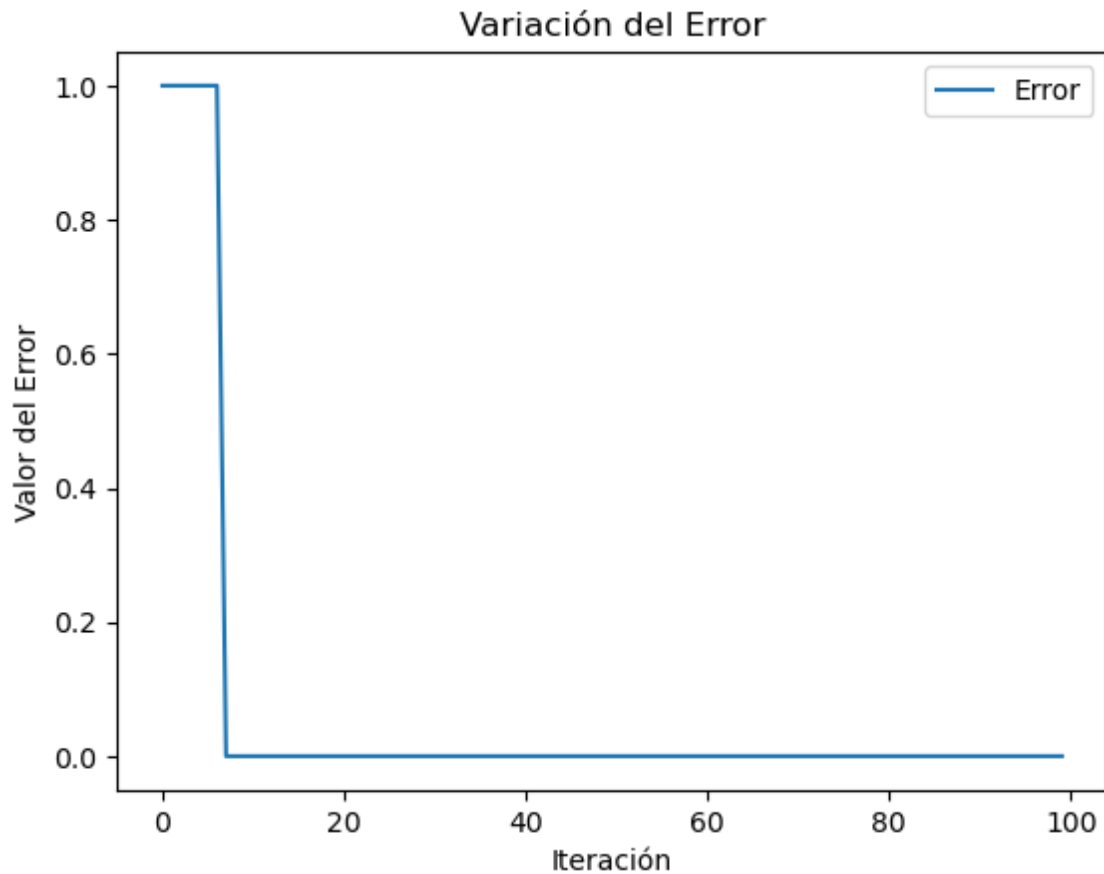


Ilustración 4 Variación del error de la Red [9]

2.3 Puerta XOR

2.3.1 ¿Por qué se ha implementado así?

No podemos entrenar una red básica de perceptrón para usarla como una XOR por el principio de separabilidad lineal, pero podemos crear dos perceptrones para resolver la salida de un XOR. Para ello haremos uso de las tablas de verdad de la puerta AND, la puerta OR y la puerta NOT. [12]

B	A	0	1
0	0	0	1
1	1	1	0

Tabla 3 de Karnaugh de XOR [13]

Como observamos en la Tabla 1, el mapa de Karnaugh, no se cumple el principio de la separabilidad lineal. Si calculamos las ecuaciones del Mapa de Karnaugh la expresión sería la siguiente:

$$XOR = \bar{A}B + \bar{B}A$$

Ecuación 7 Ecuación de la puerta XOR del mapa de Karnaugh [14]

Si desarrollamos esta ecuación y la intentamos simplificar con las leyes de Morgan vamos simplificando la ecuación de la siguiente manera:



$$\bar{A}B + \bar{B}A + \bar{A}A + \bar{B}B$$

Ecuación 8 Ecuación añadiendo puertas AND a cada lado para simplificar [14]

$$\overline{(\bar{A}B)}A + B\overline{(\bar{A}B)}$$

Ecuación 9 Ecuación tras simplificar añadiendo Puertas AND a las Puertas XOR [14]

$$\overline{(\bar{A}B)}(A + B)$$

Ecuación 10 Ecuación final de la puerta XOR completamente Simplificada para usarla con una puerta AND y otra OR [14]

Como vemos en la ecuación de la puerta de XOR se puede simplificar con una puerta AND, otra NAND y otra XOR. Por lo tanto, analizamos las siguientes tablas de verdad:

Entrada X	Entrada Y	Salida
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 4 Tabla de Verdad de la Puerta AND [15]

Entrada X	Entrada Y	Salida
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 5 Tabla de Verdad de la Puerta OR [16]

Entrada X	Entrada Y	Salida
0	0	1
1	0	1
0	1	1
1	1	0

Tabla 6 Tabla de Verdad de la Puerta NAND [17]

Por lo tanto, podemos construir con estas puertas una XOR, el esquema electrónico quedaría de la siguiente forma:

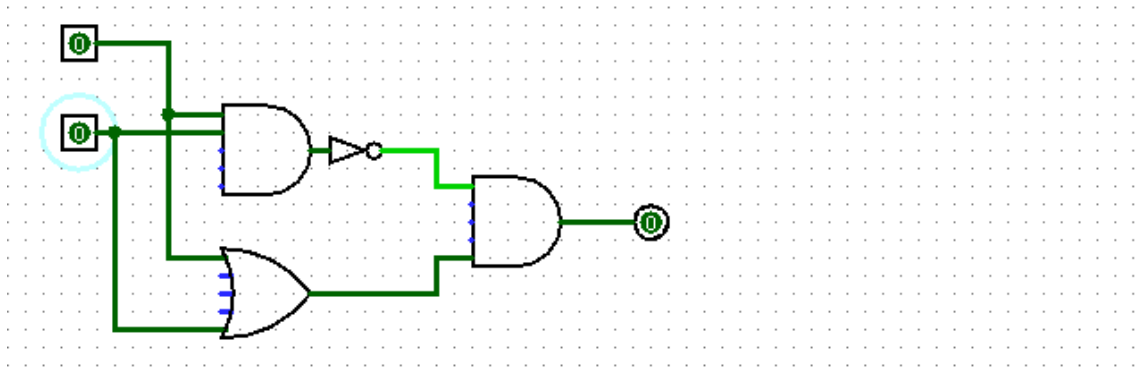


Ilustración 5 Esquema Electrónico de la Puerta XOR creada a partir de las puertas AND, OR y NAND [18]



Para replicar este esquema electrónico en nuestra red neuronal se programará una neurona perceptrón para cada puerta. La red quedaría de la misma forma:

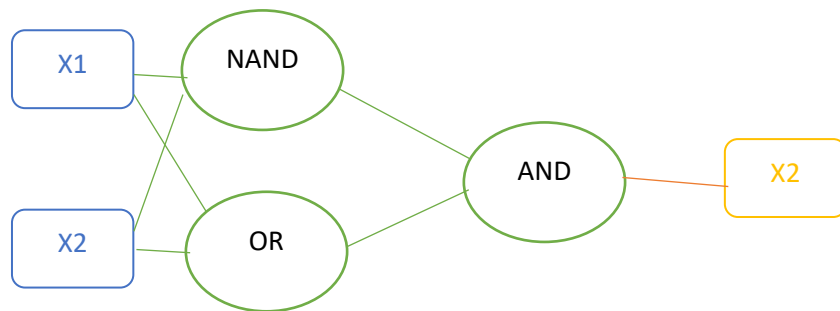


Ilustración 6 Red neuronal Perceptrón para crear usarla como una Puerta XOR [19]

En la Ilustración 3, vemos como está diseñada la red neuronal que se ha programado.

En lo que atañe al aprendizaje y la codificación de esta red se ha realizado exactamente igual que en lo visto anteriormente en la puerta AND, con la diferencia de que se han entrenado 3 neuronas comunicadas entre sí en vez de una.

2.3.2 Cuestiones XOR:

El modelo ha sido entrenado con distintos valores para comprobar cuáles son los óptimos para el entrenamiento del modelo. Tras mucho entrenamiento se ha llegado a la conclusión de que los valores óptimos son: **Eta Puerta OR: 0.1. Eta Puerta NAND: -0.1. Eta Puerta AND: 0.3**
Número de Iteraciones: 100 Learning Rate: 0.1

En este apartado se muestra la tabla de Iteraciones de la Neurona Perceptrón para la Puerta AND. Se ha decidido mostrar las 4 primeras y las 4 últimas para ver los más significativos.

Iteración	Entradas		Pesos Iniciales		Salida Real	Salida Esperada	Error	Pesos Finales	
	X1	X2	W1	W2					
0	0	0	0.33	-0.7	0	0	0	0.33	-0.7
	0	1	0.33	-0.7	0	0	0	0.33	-0.7
	1	0	0.33	-0.7	1	0	-1	0.23	-0.7
	1	1	0.23	-0.7	0	1	1	-0.33	-0.6
1	0	0	-0.33	-0.6	0	0	0	-0.33	-0.6
	0	1	-0.33	-0.6	0	0	0	-0.33	-0.6
	1	0	-0.33	-0.6	1	0	-1	0.23	-0.6
	1	1	0.23	-0.6	0	1	1	0.33	-0.5
2	0	0	0.33	-0.5	0	0	0	0.33	-0.5
	0	1	0.33	-0.5	0	0	0	0.33	-0.5
	1	0	0.33	-0.5	1	0	-1	0.23	-0.5
	1	1	0.23	-0.5	0	1	1	0.33	-0.4
	0	0	0.33	-0.4	0	0	0	0.33	-0.4
	0	1	0.33	-0.4	0	0	0	0.33	-0.4



3	1	0	0.33	-0.4	1	0	-1	0.23	-0.4
	1	1	0.23	-0.4	0	1	1	0.33	-0.3
4	0	0	0.33	-0.3	0	0	0	0.33	-0.3
	0	1	0.33	-0.3	0	0	0	0.33	-0.3
	1	0	0.33	-0.3	1	0	-1	0.23	-0.3
	1	1	0.23	-0.3	0	1	1	0.33	-0.2
5	0	0	0.33	-0.2	0	0	0	0.33	-0.2
	0	1	0.33	-0.2	0	0	0	0.33	-0.2
	1	0	0.33	-0.2	1	0	-1	0.33	-0.1
	1	1	0.33	-0.1	0	1	1	0.23	-0.1
6	0	0	0.23	-0.1	0	0	0	0.23	-0.1
	0	1	0.23	-0.1	0	0	0	0.23	-0.1
	1	0	0.23	-0.1	0	0	0	0.23	-0.1
	1	1	0.23	-0.1	0	1	1	0.33	-2.7e-17
7	0	0	0.33	-2.7e-17	0	0	0	0.33	-2.7e-17
	0	1	0.33	-2.7e-17	0	0	0	0.14	0.08
	1	0	0.33	-2.7e-17	1	0	-1	0.33	0.1
	1	1	0.33	0.1	0	1	1	0.23	0.1
8	0	0	0.23	0.1	0	0	0	0.23	0.1
	0	1	0.23	0.1	0	0	0	0.23	0.1
	1	0	0.23	0.1	1	0	0	0.23	0.1
	1	1	0.23	0.1	1	1	0	0.23	0.1

Tabla 7 Iteraciones de la neurona Perceptrón para la función XOR [20]

Como vemos en la Tabla 7, con la octava iteración la red ya soluciona la función AND sin cometer errores. En el notebook se muestran las demás iteraciones, pero en aras de no recargar mucho la memoria se ha decidido tan solo analizar las 8 primeras. Como vemos cada vez que se comete un error se modifican las matrices de pesos para corregir el error.

Con esto, podemos afirmar que la ecuación del hiperplano es:

$$0.23 * x_1 + 0.1 * x_2 + \theta = 0$$

Ecuación 11 Ecuación del Hiperplano que se genera con la función XOR [21]

Como la capa final de ambas redes está compuesta por el mismo tipo de neurona podemos afirmar que ambas aprenden con el mismo número de iteraciones.

Siendo x_1 y x_2 las entradas de la neurona perceptrón.

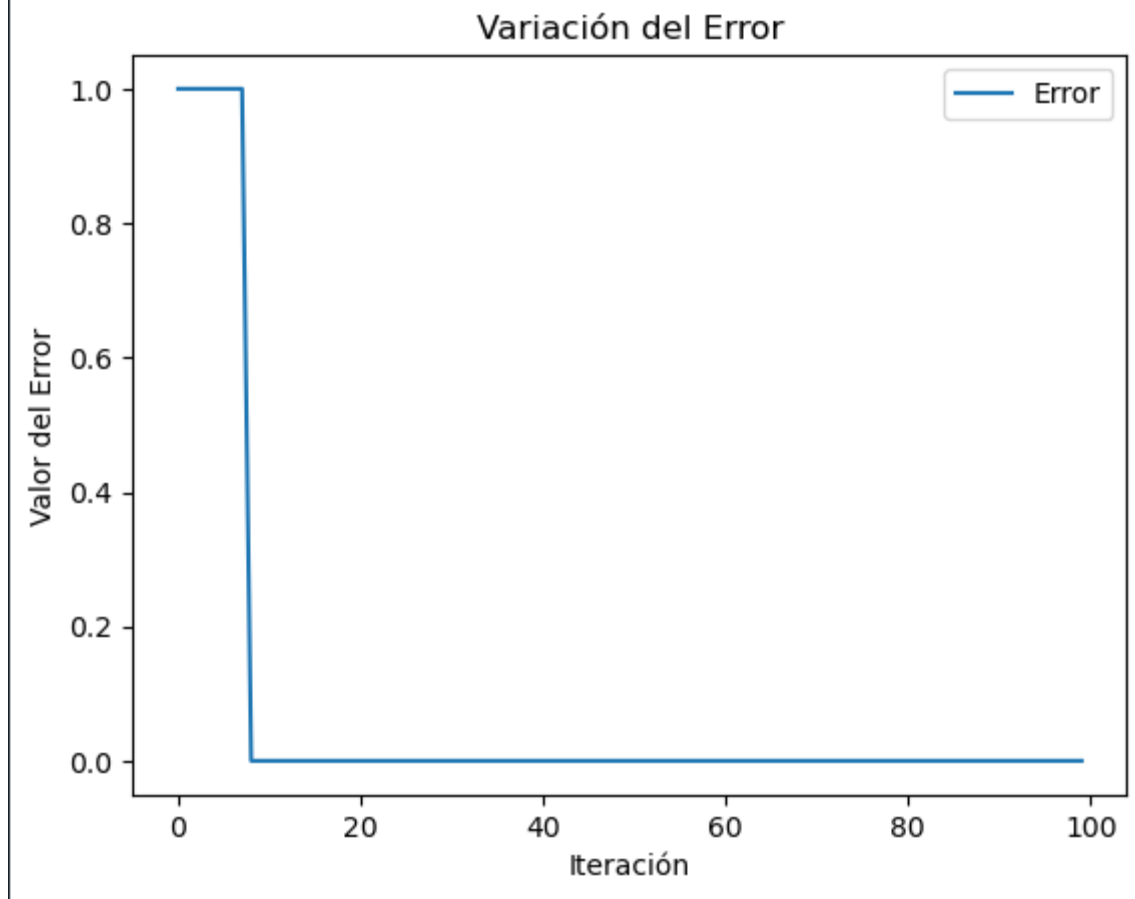


Ilustración 7 Variación del Error de la red Neuronal [22]

Como se observa en la ilustración 7, la red no comete errores a partir de la iteración 7.



3 Práctica 2

En esta práctica, se utilizaremos la popular librería *Keras/TensorFlow2* para construir y entrenar un MLP (Perceptrón Multicapa) capaz de resolver problemas no lineales.

En lugar de utilizar la Regla Delta Generalizada como función de modificación de la matriz de pesos, se utilizará el algoritmo de optimización Adam. Este método de optimización ha demostrado ser muy efectivo para entrenar redes neuronales profundas en problemas no lineales y puede acelerar significativamente el proceso de entrenamiento en comparación con otros algoritmos de optimización.

3.1 Implementación:

En este código se ha implementado un algoritmo para entrenar diferentes modelos de redes neuronales para resolver el problema de la exclusión lógica XOR. Primero se definen los datos de entrenamiento, que constan de cuatro pares de entrada y salida que representan todas las combinaciones posibles de dos bits. A continuación, se definen dos listas: una lista de neuronas en la capa oculta y una lista de valores de época para entrenar los modelos.

Luego se define una función de *callback* llamada "*print_progress*" que se utilizará para imprimir el progreso del entrenamiento en cada época. Esta función imprimirá el número de época actual, la pérdida y la precisión (*accuracy*) del modelo.

A continuación, se utiliza un bucle anidado para entrenar diferentes modelos de redes neuronales utilizando diferentes combinaciones de neuronas en la capa oculta y número de épocas. Para cada modelo, se crea un objeto *Sequential* y se agregan capas *Dense* para la capa oculta y la capa de salida. La función de activación '*relu*' se utiliza para la capa oculta y '*sigmoid*' para la capa de salida.

Una vez que se ha definido el modelo, se compila con la función de pérdida '*mean_squared_error*' y el optimizador '*adam*', y se ajusta a los datos de entrenamiento utilizando la función *fit()*. También se pasa la función de *callback* "*print_progress*" para imprimir el progreso del entrenamiento.

Después de entrenar el modelo, se utiliza el método *predict()* para predecir los valores de salida para los datos de entrada. También se evalúa el modelo utilizando la función *evaluate()* y se grafica la función de pérdida en función del número de épocas utilizando la función *plot()* de *matplotlib*.



```
# Datos de entrenamiento
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Lista de neuronas en la capa oculta
neurons = [2, 4, 8]

# Lista de valores de época
epochs = [1000, 5000, 10000]

def print_progress(epoch, Logs):
    if epoch % 250 == 0:
        print('Epoch {}: loss={}, accuracy={}'.format(epoch, Logs['loss'], Logs['accuracy']))

# Bucle para entrenar diferentes modelos
for n in neurons:
    for e in epochs:
        # Crear modelo
        model = Sequential()
        model.add(Dense(n, input_dim=2, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])

        # Entrenar modelo
        print("\nNeuronas en capa oculta:", n, "Epocas:", e)
        print("-----")
        history = model.fit(X, y, epochs=e, verbose=0, callbacks=[LambdaCallback(on_epoch_end=print_progress)])

        # Predecir valores
        y_pred = model.predict(X)

        scores = model.evaluate(X, y)

        # Graficar función de pérdida
        plt.plot(history.history['loss'], label=f'{n} neuronas, {e} épocas')

# Configurar gráfico
plt.title('Función de pérdida en función del número de épocas')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.show()
```

Ilustración 8 Código de entrenamiento del MLP [23]

3.2 Modelos:

Hemos entrenado 3 modelos diferentes, con 2, 4 y 8 neuronas con 1000, 5000 y 10000 épocas:

Esta tabla muestra los resultados de la evaluación de los tres modelos de redes neuronales para resolver el problema de la exclusión lógica XOR. Cada fila representa un modelo diferente, que se entrenó utilizando una combinación de neuronas en la capa oculta y número de épocas. Las columnas indican:

- Neuronas Capa Oculta: el número de neuronas en la capa oculta del modelo.
- *Epochs*: el número de épocas utilizado para entrenar el modelo.
- Salida: la salida predicha para los cuatro pares de entrada de datos (0 0, 0 1, 1 0, 1 1) utilizando el modelo entrenado. Por ejemplo, el modelo con 2 neuronas en la capa oculta y 1000 épocas predijo 0 0 1 0 para los datos de entrada.
- Error: el error de predicción del modelo, medido como la función de pérdida (*mean squared error*) en los datos de entrenamiento.



Se puede observar que a medida que se aumenta el número de neuronas en la capa oculta y el número de épocas de entrenamiento, el error disminuye. El modelo con 8 neuronas en la capa oculta y 10,000 épocas de entrenamiento tiene el menor error de todos los modelos evaluados, con un error de 3.8653e-05 en los datos de entrenamiento.

Neuronas Capa Oculta	Epochs	Salida	Error
2	1000	0 0 1 0	0.1712
2	5000	0 1 1 0	0.0015
2	10000	0 1 0 0	0.1667
4	1000	0 1 1 0	0.0650
4	5000	0 1 0 0	0.1668
4	10000	1 1 1 0	0.1667
8	1000	0 1 1 0	0.0406
8	5000	0 1 1 0	8.6946e-04
8	10000	0 1 1 0	3.8653e-05

Tabla 8 Variaciones de la Red según Número de neuronas de la capa Oculta, Iteraciones, salida y error [24]

Para observar de mejor manera el contraste de cada modelo entrenado realizamos un gráfico que contraste el error en los distintos entrenamientos:

8 neuronas 10000 epoch

```
Entrada: [0 0] Etiqueta real: 0 Predicción: 0.011015753
Entrada: [0 1] Etiqueta real: 1 Predicción: 0.9966192
Entrada: [1 0] Etiqueta real: 1 Predicción: 0.9966876
Entrada: [1 1] Etiqueta real: 0 Predicción: 0.003296103
1/1 [=====] - 0s 141ms/step - loss: 3.8653e-05 - accuracy: 1.0000
```

Ilustración 9 Salida del programa del MLP para 8 neuronas y 10000 épocas [25]

4 neuronas 5000 epoch

```
Entrada: [0 0] Etiqueta real: 0 Predicción: 0.33438623
Entrada: [0 1] Etiqueta real: 1 Predicción: 0.973146
Entrada: [1 0] Etiqueta real: 1 Predicción: 0.33438623
Entrada: [1 1] Etiqueta real: 0 Predicción: 0.33438623
1/1 [=====] - 0s 118ms/step - loss: 0.1668 - accuracy: 0.7500
```

Ilustración 10 Salida del Programa del MLP para 4 neuronas y 5000 épocas [26]

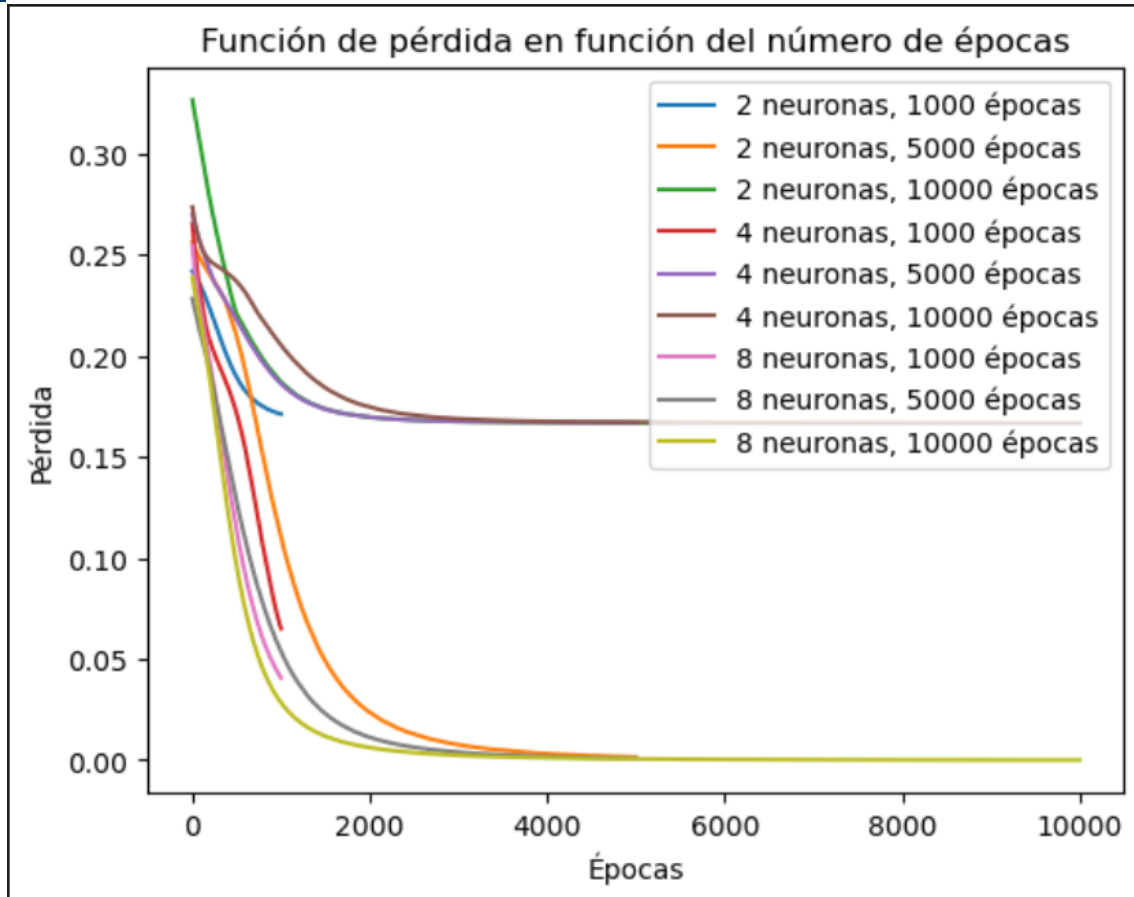


Ilustración 11 Función de Pérdida según el número de épocas [27]

Como se puede observar en este gráfico el mejor modelo que se ha realizado ha sido aquel de 8 neuronas con 10.000 época y aquel presenta el menor error cuadrático medio de los datos evaluados con un valor de $3.8653e-05$. En contraste el peor modelo es aquel con 4 neuronas en la capa oculta y 5,000 épocas de entrenamiento, ya que tiene el mayor error cuadrático medio en los datos de entrenamiento de todos los modelos evaluados, con un valor de 0.1668. Este modelo es menos preciso en la predicción de los valores de salida para los cuatro pares de entrada del problema XOR y no es tan efectivo como otros modelos en aprender patrones no lineales en los datos.



4 Práctica 3

4.1 Implementación

Primero importamos todas las librerías necesarias:

```
#Importacion de Librerias
import numpy as np
import pandas as pd
import sklearn
import tensorflow as ts
import keras
import matplotlib.pyplot as plt
from keras.callbacks import LambdaCallback
from sklearn.model_selection import train_test_split
import pprint
import os
import imageio
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ProgbarLogger
from keras.utils import to_categorical
from sklearn.preprocessing import StandardScaler
from keras.layers import Dense
from keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
from IPython.display import Image
from sklearn.metrics import confusion_matrix, accuracy_score

from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
import seaborn as sns
```

Ilustración 12 Importación de las librerías [28]

Continuamos leyendo el CSV y creando las carpetas necesarias para guardar la información:



Creacion de las carpetas

```
if(os.path.exists("Modelos")) == 0:  
    os.mkdir("Modelos")
```

Lectura del CSV

```
data = pd.read_csv("bank-dataset.csv", sep = ";")  
data.head(5)
```

Ilustración 13 Creación de carpetas y lectura del CSV [29]

Progresamos con el preprocesamiento de datos, primero convertimos los valores categóricos como números usando la librería *pandas*, posteriormente una vez pasado los categóricos a numéricos procedemos a normalizar los datos, es decir, escalar las variables numéricas de un conjunto de datos utilizando la transformación de estandarización, que ajusta la distribución de cada variable para que tenga una media de cero y una desviación estándar de uno.

Preprocesamiento

```
cat_vars = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome']  
for var in cat_vars:  
    data[var] = pd.Categorical(data[var])  
    data[var] = data[var].cat.codes  
data['y'] = pd.Categorical(data['y']).codes  
data['y'] = to_categorical(data['y'], num_classes=2)
```

Normalización

```
scaler = StandardScaler()  
num_vars = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']  
data[num_vars] = scaler.fit_transform(data[num_vars])
```

Ilustración 14 Código para el Preprocesamiento y Normalización de los datos [30]

Continuamos con la implementación de la red neuronal:

El código comienza dividiendo el conjunto de datos en dos conjuntos aleatorios: uno para entrenamiento y otro para prueba. El tamaño del conjunto de prueba se establece en el 20% del tamaño total del conjunto de datos, lo que significa que el 80% restante se utilizará para entrenar los modelos de red neuronal.

Luego, se definen dos listas de valores: una lista para el número de neuronas en la capa oculta y otra lista para el número de épocas a entrenar. El código utiliza un bucle for anidado para crear y entrenar un modelo de red neuronal para cada combinación posible de neuronas y épocas.



En cada iteración del bucle, se crea un modelo secuencial de red neuronal con una capa oculta y una capa de salida. La capa oculta utiliza una función de activación de rectificación lineal (ReLU), que ayuda a prevenir el problema del desvanecimiento del gradiente y acelera el proceso de aprendizaje.

La capa de salida utiliza una función de activación sigmoidea, que produce una salida en el rango de 0 a 1. Esta salida se interpreta como la probabilidad de que el objeto de entrada pertenezca a una clase determinada.

Después de compilar el modelo, se utiliza el método `fit()` para entrenar el modelo con los datos de entrenamiento. En cada época, el código imprime la pérdida y la precisión del modelo en los datos de entrenamiento. También se utiliza una función de devolución de llamada (callback) para imprimir el progreso del modelo en la pantalla.

Una vez entrenado el modelo, se utiliza el conjunto de prueba para evaluar su rendimiento. Se utiliza el método `predict()` para hacer predicciones con los datos de prueba y se compara con las respuestas verdaderas para crear una matriz de confusión.

Finalmente, se utiliza la biblioteca Matplotlib para graficar la función de pérdida en función del número de épocas para cada modelo. Esto permite visualizar cómo cambia la pérdida a medida que se entrena el modelo. Además, se guarda una imagen del modelo para cada combinación de neuronas y épocas, lo que permite examinar visualmente la arquitectura de cada modelo

```
# Separar conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(data.drop('y', axis=1), data['y'], test_size=0.2)

# Lista de neuronas en la capa oculta
neurons = [2, 4, 8]

# Lista de valores de época
epochs = [25, 50, 100]

def print_progress(epoch, Logs):
    if epoch % 50 == 0:
        print('Epoch {}: loss={}, accuracy={}'.format(epoch, Logs['loss'], Logs['accuracy']))
cont = 0
# Bucle para entrenar diferentes modelos
for n in neurons:
    for e in epochs:
        # Crear modelo
        model = Sequential()
        model.add(Dense(n, input_dim=X_train.shape[1], activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
        # Entrenar modelo
        print("\nNeuronas en capa oculta:", n, "Epocas:", e)
        print("-----")
        model.summary()
        history = model.fit(X_train, y_train, epochs=e, batch_size=32, validation_data=(X_test, y_test), verbose=0, callbacks=[LambdaCallback(on_epoch_end=print_progress)])

        # Predecir valores
        y_pred = model.predict(X_train)
        y_pred = (model.predict(X_test) > 0.5).astype("int32")

        # Obtener la matriz de confusión
        cm = confusion_matrix(y_test, y_pred)
        print('Confusion matrix:')
        print(cm)

        scores = model.evaluate(X_train, y_train)

        # Graficar función de pérdida
        plt.plot(history.history['loss'], label=f'({n} neuronas, {e} épocas')
# Guardar Imagen del modelo
plot_model(model, to_file=f'Modelos/modelo_{cont}.png', show_shapes=True)

# Mostrar el modelo
display(Image(f'Modelos/modelo_{cont}.png'))
cont += 1

# Configurar gráfico
plt.title('Función de pérdida en función del número de épocas')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
plt.show()
```

Ilustración 15 Código de entrenamiento del MLP [35]



4.2 Cuestiones

En este apartado se resolverán las cuestiones propuestas en el enunciado.

4.2.1 Cuestión 1

Explica cómo has llevado a cabo la categorización y la normalización de los datos de entrada

Para llevar a cabo la categorización de los datos de entrada, se ha utilizado el método *to_categorical* de la biblioteca *Keras*. Este método convierte las variables categóricas en matrices binarias que pueden ser utilizadas como entrada a una red neuronal. Para hacer esto, primero se convierten las variables categóricas a valores numéricos enteros mediante la función *LabelEncoder* de *scikit-learn*, y luego se convierten a matrices binarias utilizando la función *to_categorical* de *Keras*.

En cuanto a la normalización de los datos de entrada, se ha hecho uso de la función *MinMaxScaler* de *scikit-learn* para escalar los valores de las variables numéricas entre 0 y 1. Esto se hizo para que todas las variables numéricas tuvieran la misma escala y contribuyeran por igual al modelo. Además, la normalización puede ayudar a mejorar la velocidad y la precisión del modelo durante el entrenamiento. En general, la normalización se debe hacer en todos los datos numéricos de entrada, ya que es un requisito común para muchos algoritmos de aprendizaje automático.

4.2.2 Cuestión 2

Hemos realizado 3 arquitecturas diferentes con 2, 4 y 8 neuronas en la que hemos ejecutado con diferentes épocas para comparar y contrastar el resultado obtenido:

Neuronas	épocas	Loss	Accuarcy
2	25	0.062	0.9132
2	50	0.0600	0.9138
2	100	0.0595	0.9144
4	25	0.0623	0.9134
4	50	0.0586	0.9158
4	100	0.0578	0.9177
8	25	0.0590	0.9151
8	50	0.0568	0.9177
8	100	0.0581	0.9156

Tabla 9 Variación de resultados del MLP según las neuronas de la capa oculta [31]



dense_34_input	input:	[(None, 20)]
InputLayer	output:	[(None, 20)]

dense_34	input:	(None, 20)
Dense	output:	(None, 8)

dense_35	input:	(None, 8)
Dense	output:	(None, 1)

Ilustración 18 Modelo del MLP para 8 neuronas dentro de la capa oculta [36]

dense_28_input	input:	[(None, 20)]
InputLayer	output:	[(None, 20)]

dense_28	input:	(None, 20)
Dense	output:	(None, 4)

dense_29	input:	(None, 4)
Dense	output:	(None, 1)

Ilustración 17 Modelo del MLP para 4 neuronas dentro de la capa oculta [37]

dense_22_input	input:	[(None, 20)]
InputLayer	output:	[(None, 20)]

dense_22	input:	(None, 20)
Dense	output:	(None, 2)

dense_23	input:	(None, 2)
Dense	output:	(None, 1)

Ilustración 16 Modelo del MLP para 2 neuronas dentro de la capa oculta [38]

Aquí se observa el contraste entre los diferentes modelos entrenados.

En la tabla, podemos ver que el modelo con 8 neuronas y 50 épocas tiene la menor pérdida (0.0568) y la mayor precisión (0.9177). Por lo tanto, este modelo es el mejor de todos los modelos entrenados.

4.2.3 Cuestión 3

Aquí observamos las diferentes matrices obtenidas con los diferentes modelos:

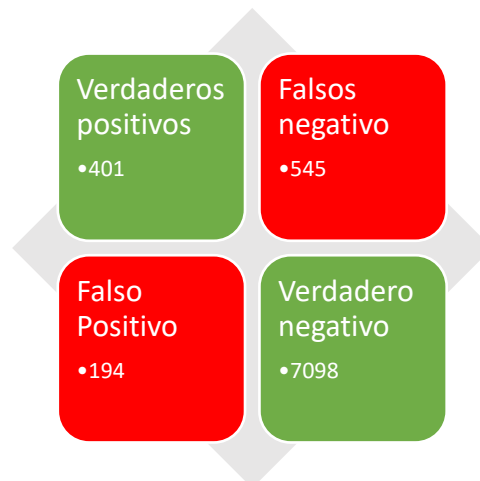


Ilustración 19 Matriz de confusión para 2 neuronas y 25 épocas [39]



Ilustración 20 Matriz de confusión para 2 neuronas y 50 épocas [40]

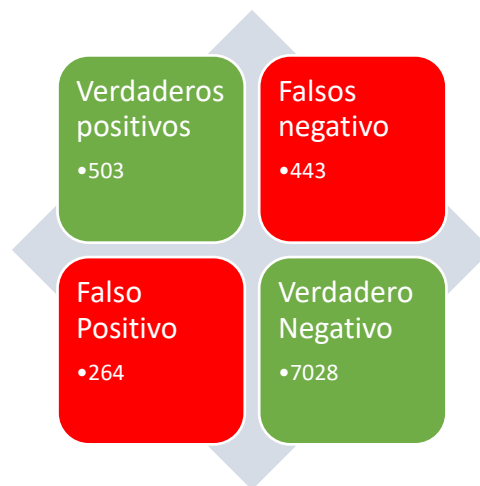


Ilustración 21 Matriz de confusión para 2 neuronas y 100 épocas [41]

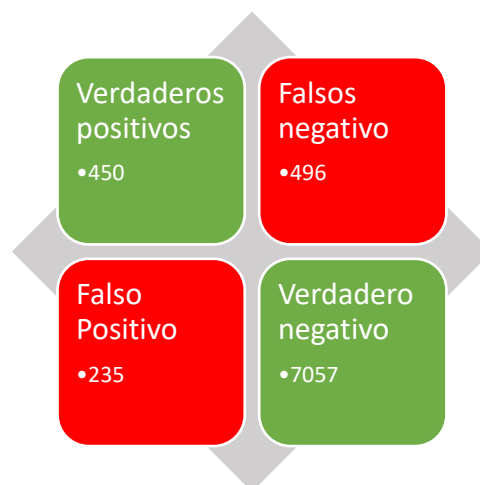


Ilustración 22 Matriz de confusión para 4 neuronas y 25 épocas [42]



Ilustración 23 Matriz de confusión para 4 neuronas y 50 épocas [43]

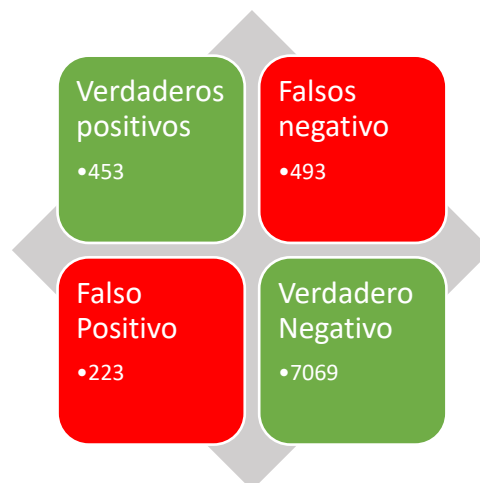


Ilustración 24 Matriz de confusión para 4 neuronas y 100 épocas [44]

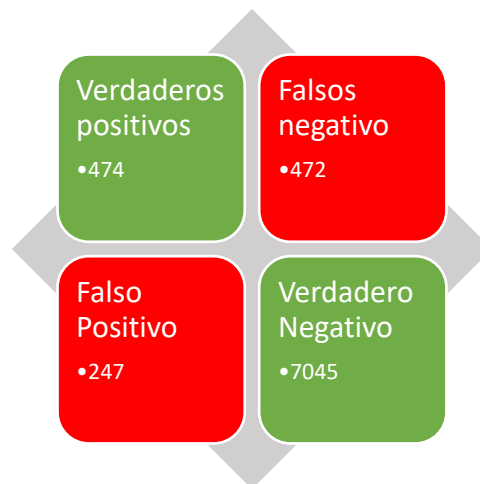


Ilustración 26 Matriz de confusión para 8 neuronas y 25 épocas [45]

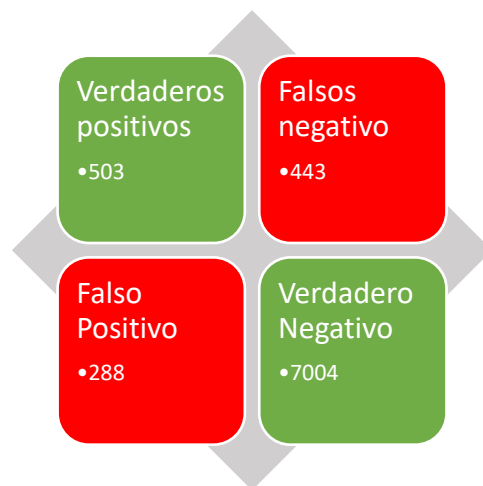


Ilustración 25 Matriz de confusión para 8 neuronas y 50 épocas [46]

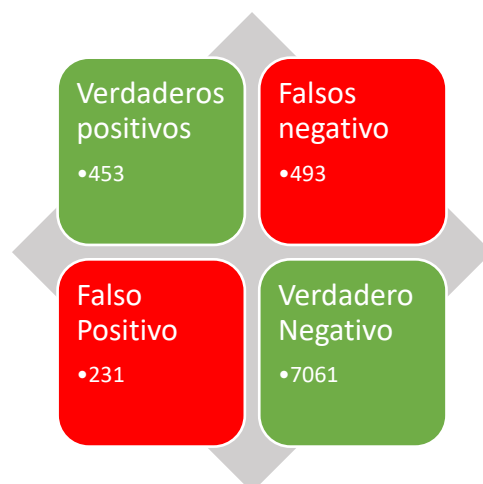


Ilustración 27 Matriz de confusión para 8 neuronas y 100 épocas [47]



En la matriz de confusión, se tienen cuatro valores principales: verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Los verdaderos positivos (TP) son las predicciones que el modelo ha hecho correctamente para la clase positiva. Los verdaderos negativos (TN) son las predicciones que el modelo ha hecho correctamente para la clase negativa. Los falsos positivos (FP) son las predicciones que el modelo ha hecho incorrectamente para la clase positiva. Los falsos negativos (FN) son las predicciones que el modelo ha hecho incorrectamente para la clase negativa.

En general, se busca minimizar tanto los falsos positivos como los falsos negativos, y maximizar tanto los verdaderos positivos como los verdaderos negativos. Sin embargo, la interpretación de la matriz de confusión puede variar según el problema y las características específicas del conjunto de datos.

En este caso, para 2 neuronas y 50 épocas, se puede observar que el modelo ha tenido un rendimiento aceptable para la clase positiva, ya que ha acertado en la gran mayoría de las predicciones (7098 verdaderos positivos frente a 194 falsos negativos). Sin embargo, para la clase negativa, el modelo ha tenido un rendimiento mucho peor, ya que ha acertado en menos de la mitad de las predicciones (401 verdaderos negativos frente a 545 falsos positivos). Esto podría indicar que el modelo tiene dificultades para distinguir entre las observaciones de la clase negativa, lo cual podría ser algo para tener en cuenta al momento de mejorar el modelo.



4.2.4 Cuestión 4

Si, en las diferentes matrices de confusión se puede observar que no siempre se clasifica de manera correcta teniendo un número de falsos negativos y falsos positivos elevado. Esto es debido a la existencia en el *dataset* de variables que no aportan información para la clasificación y constituyen un ruido que provoca errores de clasificación .

Para poder mitigar el error decidimos crear un nuevo *dataset* donde eliminamos los atributos que presentan menos información, es decir, 'day_of_week', 'month', 'default' y 'pdays'.

Por lo tanto, repetimos el mismo proceso realizado anteriormente.

```
data_MenosDatos = pd.read_csv("bank-dataset.csv", sep = ";")
data_MenosDatos = data.drop(['day_of_week', 'month', 'default', 'pdays'], axis=1)
data_MenosDatos.head(10)
```

Ilustración 28 Modificación de los datos [32]

Obteniendo los siguientes resultados:

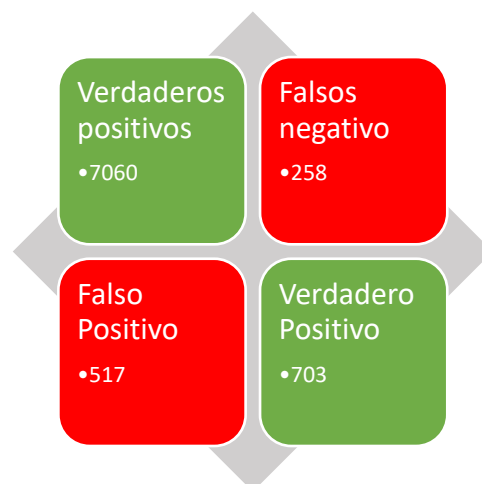


Ilustración 29 Matriz de confusión para 2 neuronas y 25 épocas [48]



Ilustración 30 Matriz de confusión para 2 neuronas 50 épocas después de modificar los datos [49]



Ilustración 32 Matriz de confusión para 2 neuronas 100 épocas después de modificar los datos [50]

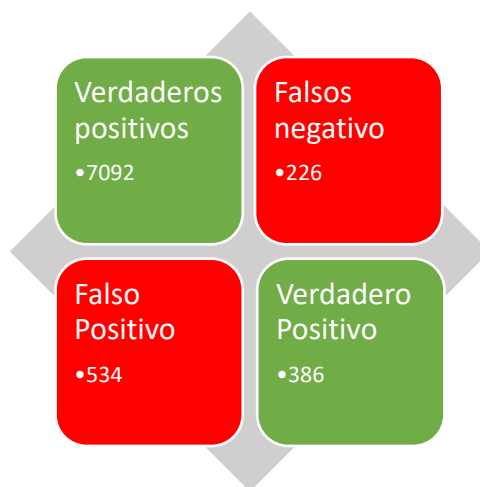


Ilustración 31 Matriz de confusión para 4 neuronas 25 épocas después de modificar los datos [51]

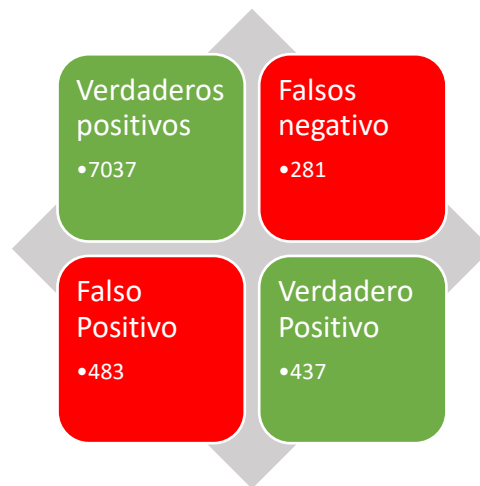


Ilustración 33 Matriz de confusión para 4 neuronas 50 épocas después de modificar los datos [52]

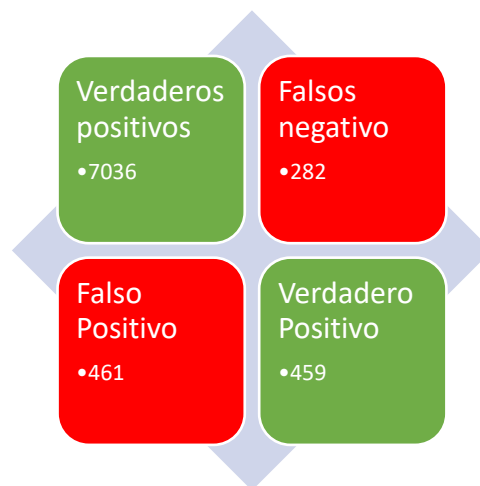


Ilustración 34 Matriz de confusión para 4 neuronas 100 épocas después de modificar los datos [53]

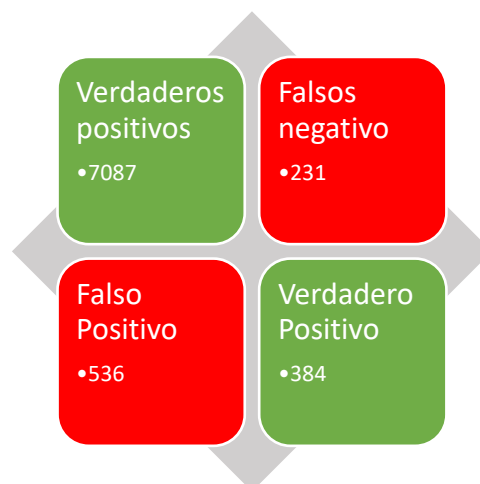


Ilustración 35 Matriz de confusión para 8 neuronas 25 épocas después de modificar los datos [54]



Ilustración 36 Matriz de confusión para 8 neuronas 50 épocas después de modificar los datos [55]

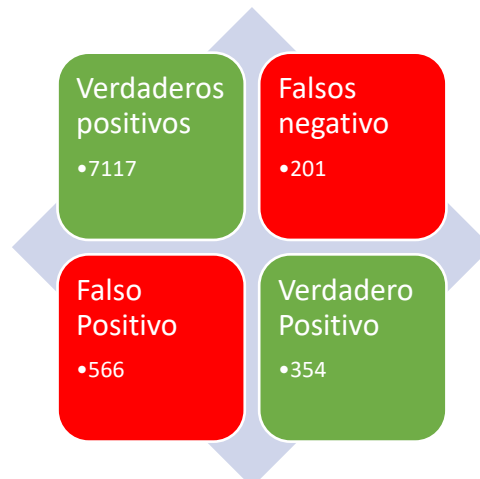


Ilustración 37 Matriz de confusión para 8 neuronas 100 épocas después de modificar los datos [56]

Contrastando los resultados entre diferentes matrices se puede observar que:

La nueva matriz de confusión de 2 neuronas y 50 épocas tiene un total de 8,238 observaciones, de las cuales 7,073 son verdaderos negativos (TN), 245 son falsos positivos (FP), 518 son falsos negativos (FN) y 402 son verdaderos positivos (TP).

En comparación con la matriz de confusión anterior, esta presenta una mejor clasificación para la clase positiva, con una cantidad menor de falsos negativos y una mayor cantidad de verdaderos positivos. Sin embargo, también tiene una mayor cantidad de falsos positivos, lo que indica que el modelo está clasificando algunas muestras negativas como positivas.

En resumen, la principal diferencia entre ambas matrices de confusión es la precisión del modelo en la clasificación de las muestras de la clase positiva. La primera matriz presenta un mayor número de falsos positivos y falsos negativos, mientras que la segunda tiene menos falsos negativos, pero más falsos positivos. Es importante tener en cuenta que la elección de un modelo depende del contexto específico de la aplicación y de los objetivos que se buscan alcanzar.



4.2.5 Cuestión 5

¿Qué variables influyen más a la hora de contratar un depósito? ¿Qué variables influyen menos?

Describe el método que has llevado a cabo para llegar a esa conclusión.

En aras de conseguir un análisis más real de la influencia de las variables para contratar o no un depósito, se ha decidido realizar un análisis de PCA y de árboles de decisión para comprobar de manera correcta la influencia de las variables.

Componente	Job	Marital	Education	Default	Housing	Loan	Contact	Month	Day of week	poutcome
1	-0.003	0.024	-0.096	-0.219	0.301	-0.471	-0.362	-0.103	-0.474	-0.451
2	0.275	0.060	-0.004	-0.597	0.462	0.143	0.273	0.419	0.130	-0.028
3	-0.690	0.058	0.152	-0.251	0.267	0.103	0.288	-0.409	0.027	0.032
4	-0.159	0.730	-0.643	0.022	0.078	0.024	0.005	0.106	0.040	0.013
5	0.490	0.465	0.277	0.019	0.065	0.0123	0.191	-0.569	-0.058	-0.002
6	-0.178	0.485	0.687	0.013	0.131	-0.065	-0.188	0.399	-0.023	-0.048
7	-0.055	-0.031	0.018	0.157	-0.158	-0.153	0.376	0.129	-0.199	-0.378
8	0.013	0.010	0.008	0.188	0.218	0.017	-0.097	0.0129	0.038	0.077
9	-0.059	0.054	0.029	0.566	0.583	0.049	-0.029	0.127	0.068	0.051
10	-0.244	-0.004	0.029	-0.223	-0.061	-0.006	-0.264	0.009	0.039	0.130
11	0.137	-0.005	-0.003	-0.005	0.002	0.013	-0.045	0.041	0.003	0.006
12	-0.04	0.011	-0.007	-0.031	0.011	-0.011	-0.062	-0.063	-0.002	0.039
13	0.115	0.013	0.003	-0.164	0.010	0.012	-0.335	-0.139	0.121	0.255
14	-0.021	-0.007	-0.004	0.059	-0.006	0.002	0.117	0.073	-0.040	-0.094
15	-0.072	-0.009	-0.006	0.059	-0.022	-0.004	0.140	0.096	-0.053	-0.121
16	-0.043	0.018	-0.005	-0.032	-0.001	-0.007	-0.057	-0.095	-0.013	0.046
17	-0.014	0.014	0.0004	-0.005	0.0009	-0.008	-0.009	-0.0009	0.003	0.015
18	-0.018	-0.010	0.012	0.014	-0.020	0.037	0.083	-0.027	-0.028	-0.091

Tabla 10 componentes con sus variables y cómo influyen [33]

En el marco del estudio del PCA se ha decidido usar 15 Componentes para representar el 87,75% de los datos. Una vez hecho el análisis de PCA nos salen las siguientes componentes:

Según el análisis, las variables que más influyen son aquellas que tienen una correlación más alta con los primeros componentes principales, lo que indica que tienen una gran influencia en la variación de los datos. En este caso, las variables que más influyen son "housing", "marital status" y "education".



Por otro lado, las variables que menos influyen son aquellas que tienen una correlación más baja con los primeros componentes principales, lo que indica que tienen menos influencia en la variación de los datos. En este caso, las variables que menos influyen son "default", "job" y "contact".

Cabe mencionar que la interpretación de los resultados puede depender del contexto y los objetivos específicos del análisis.

En cambio, si analizamos el análisis de árboles de decisión, podemos afirmar que las variables que más influyen en la predicción son "duration" (duración de la llamada), "euribor3m" (tasa de interés a tres meses), "age" (edad del cliente) y "nr.employed" (número de empleados). Las variables que menos influyen en la predicción son "month_dec" (mes de diciembre), "marital_unknown" (estado civil desconocido), "education_illiterate" (analfabetismo), y "default_yes" (impago de crédito).

Como la finalidad del análisis es realizar una clasificación según la variable objetivo, nos quedamos con las soluciones del árbol de decisión. Por lo tanto, podemos afirmar lo siguiente:

- La variable más importante para predecir si un cliente contratará un depósito a plazo (variable objetivo "y") es la duración de la última llamada realizada por el banco al cliente (variable "duration").
- Si la duración de la última llamada es mayor a 219 segundos, hay una alta probabilidad de que el cliente contrate un depósito a plazo.
- Si la duración de la última llamada es menor o igual a 219 segundos y el cliente no tiene crédito en mora (variable "default"), entonces la siguiente variable más importante es el número de veces que el cliente ha sido contactado durante la campaña actual (variable "campaign").
- Si el número de contactos durante la campaña actual es menor o igual a 3 y el cliente no tiene crédito en mora, entonces la siguiente variable más importante es la edad del cliente (variable "age").
- En general, los clientes mayores de 60 años y los clientes menores de 28 años tienen una mayor probabilidad de contratar un depósito a plazo.
- Por otro lado, si el número de contactos durante la campaña actual es mayor a 3 y el cliente no tiene crédito en mora, entonces la siguiente variable más importante es el resultado de la campaña anterior (variable "poutcome").
- Si el resultado de la campaña anterior fue éxito o no existente, hay una probabilidad moderada de que el cliente contrate un depósito a plazo.
- Si el resultado de la campaña anterior fue fracaso, es poco probable que el cliente contrate un depósito a plazo.



5 Bibliografía

- [1] Y. B. y. A. C. Ian Goodfellow, Fundamentos de Aprendizaje Automático, 2016.
- [2] F. Rosenblatt, «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,» 1957.
- [3] F. Propia, *Tabla de Verdad de la Puerta AND*, Madrid, 2023.
- [4] F. Propia, *Código de iniciación de la neurona Perceptrón como Puerta AND*, Madrid, 2023.
- [5] F. Propia, *Suma ponderada de las entradas*.
- [6] F. Propia, *Función de Activación límite aplicada a la Suma Ponderada*.
- [7] F. Propia, *Comportamiento de la Función Límite según el theta*.
- [8] F. Propia, *Calculo del error de la salida de la neurona*.
- [9] F. Propia, *Reajuste de la Matriz de pesos de la neurona*.
- [10] F. Propia, *Tabla de Iteraciones de la Neurona Perceptrón para la Puerta AND*, Madrid, 2023.
- [11] F. propia, *Ecuación del hiperplano generado para la funcion AND*, Madrid: Fuente propia, 2023.
- [12] Á. J. G. Tejedor, *Aprendizaje Supervisado Modelos No lineales*, Madrid, 2022.
- [13] F. Propia, *Tabla de Karnaugh XOR*, Madrid, 2023.
- [14] F. Propia, *Ecuación del mapa de Karnaugh de la puerta XOR*, Madrid, 2023.
- [15] F. Propia, *Tabla de Verdad de la Puerta AND*, Madrid: Fuente Propia, 2023.
- [16] F. Propia, *Tabla de Verdad de la Puerta OR*, Madrid: Fuente Propia, 2023.
- [17] F. Propia, *Tabla de Verdad de la Puerta NAND*, Madrid: Fuente propia, 2023.
- [18] F. Propia, *Esquema Electrónico de la Puerta XOR a partir de puertas AND, OR y NAND*, Madrid, 2023.
- [19] F. Propia, *Esquema de red Neuronal con neuronas perceptrón que se comporta como una Puerta XOR*, Madrid, 2023.
- [20] F. Propia, *Iteraciones de la neurona Perceptrón para la función XOR*, Madrid: Fuente Propia, 2023.
- [21] F. propia, *Ecuación del Hiperplano que se genera con la función XOR*, Madrid: Fuente Propia, 2023.
- [22] F. Propia, *Variación del error de la red Neuronal*, Madrid: Fuente Propia, 2023.



- [23] F. Propia, *Código de entrenamiento del MLP*, Madrid: Fuente Propia, 2023.
- [24] F. Propia, *Variaciones de la Red según Número de neuronas de la capa Oculta, Iteraciones, salida y error*, Madrid: Fuente Propia, Fuente Propia.
- [25] F. Propia, *Salida del programa del MLP para 8 neuronas y 10000 épocas*, Madrid: Fuente Propia, 2023.
- [26] F. Propia, *Salida del Programa del MLP para 4 neuronas y 5000 épocas*, Madrid: Fuente Propia, 2023.
- [27] F. propia, *Función de Pérdida según el número de épocas*, Madrid: Fuente propia, 2023.
- [28] F. Propia, *Importación librerías*, Madrid: Fuente Propia, 2023.
- [29] F. Propia, *Carpetas y CSV*, Madrid: Fuente Propia, 2023.
- [30] F. Propia, *Código para el Preprocesamiento y Normalización de los datos*, Madrid: Fuente Propia, 2023.
- [31] F. Propia, *Variación de resultados del MLP según las neuronas de la capa oculta*, Madrid: Fuente Propia, 2023.
- [32] F. Propia, *Modificacion Datos*, Madrid: Fuente Propia, 2023.
- [33] F. Propia, *Componentes y sus variables*, Madrid: Fuente Propia, 2023.
- [34] F. Propia, *Código de Entrenamiento de la Neurona Perceptrón AND*, Madrid, 2023.
- [35] F. Propia, *Código de entrenamiento del MLP*, Madrid: Fuente Propia, 2023.
- [36] F. Propia, *Variación de resultados del MLP según las neuronas de la capa oculta*, Madrid: Fuente Propia, 2023.
- [37] F. Propia, *Modelo de MLP para 4 neuronas dentro de la capa oculta*, Madrid: Fuente Propia, 2023.
- [38] F. Propia, *Modelo del MLP para 2 neuronas dentro de la capa oculta*, Madrid: Fuente Propia, 2023.
- [39] F. Propia, *Matriz de confusión para 2 neuronas y 25 épocas*, Madrid: Fuente Propia, 2023.
- [40] F. Propia, *Matriz de confusión para 2 neuronas y 50 épocas*, Madrid: Fuente propia, 2023.
- [41] F. Propia, *Matriz de confusión para 2 neuronas y 100 épocas*, Madrid: Fuente Propia, 2023.
- [42] F. Propia, *Matriz de confusión para 4 neuronas y 25 épocas*, Madrid: Fuente Propia, 2023.
- [43] F. Propia, *Matriz de confusión para 4 neuronas y 50 épocas*, Madrid: Fuente Propia, 2023.
- [44] F. Propia, *Matriz de confusión para 4 neuronas y 100 épocas*, Madrid: Fuente Propia, 2023.



- [45] F. Propia, *Matriz de confusión para 8 neuronas y 25 épocas*, Madrid: Fuente Propia, 2023.
- [46] F. Propia, *Matriz de confusión para 8 neuronas y 50 épocas*, Madrid: Fuente Propia, 2023.
- [47] F. Propia, *Matriz de confusión para 8 neuronas y 100 épocas*, Madrid: Fuente Propia, 2023.
- [48] F. Propia, *Matriz de confusión para 2 neuronas y 25 épocas*, Madrid: Fuente Propia, 2023.
- [49] F. Propia, *Matriz de confusión para 2 neuronas y 50 épocas*, Madrid: Fuente Propia, 2023.
- [50] F. Propia, *Matriz de confusión para 2 neuronas 100 épocas después de modificar los datos*, Madrid: Fuente Propia, 2023.
- [51] F. Propia, *Matriz de confusión para 4 neuronas 25 épocas después de modificar los datos*, Madrid: Fuente Propia, 2023.
- [52] F. Propia, *Matriz de confusión para 4 neuronas 50 épocas después de modificar los datos*, Madrid: Fuente Propia, 2023.
- [53] F. Propia, *Matriz de confusión para 4 neuronas 100 épocas después de modificar los datos*, Madrid: Fuente Propia, 2023.
- [54] F. Propia, *Matriz de confusión para 8 neuronas 25 épocas después de modificar los datos*, Madrid: Fuente propia, 2023.
- [55] F. Propia, *Matriz de confusión para 8 neuronas 50 épocas después de modificar los datos*, Madrid: Fuente Propia, 2023.
- [56] F. Propia, *Matriz de confusión para 8 neuronas 100 épocas después de modificar los datos*, Madrid: Fuente Propia, 2023.



6 Anexo I

En el este anexo se muestran los diferentes módulos necesarios para poder ejecutar correctamente la práctica.

Instalacion de Librerias

Antes de ejecutar el codigo, asegúrese de tener instaladas las siguientes librerías:

Para instalar estas librerías se pueden utilizar los siguientes comandos en la línea de comandos de Windows (CMD):

numpy: `pip install numpy`

pandas: `pip install pandas`

sklearn: `pip install scikit-learn`

tensorflow: `pip install tensorflow`

keras: `pip install keras`

matplotlib: `pip install matplotlib`

imageio: `pip install imageio`

seaborn: `pip install seaborn`

También se pueden instalar todas las librerías requeridas en un solo comando utilizando el archivo librerias.bat

Nota

Si ya tiene algunas de las librerías instaladas, asegúrese de que sean las últimas versiones disponibles.



7 Anexo II

En este anexo se muestran las conversaciones con ChatGPT3 usadas para implementar la práctica.

como represento la separatriz de un perceptron de una sola neurona?



La separación de un perceptrón de una sola neurona se puede representar como una línea recta en un espacio bidimensional. La función del perceptrón es clasificar los puntos en dos categorías diferentes, y la línea recta representa el umbral de decisión que separa las dos categorías.



La ecuación de la línea recta que representa la separación del perceptrón de una sola neurona se puede expresar en términos de los pesos y sesgo (bias) del perceptrón. Si el perceptrón tiene dos entradas, la ecuación de la línea se puede expresar como:

$$w_1x_1 + w_2x_2 + b = 0$$

Donde w_1 y w_2 son los pesos sinápticos de las entradas x_1 y x_2 respectivamente, y b es el sesgo del perceptrón.

Esta ecuación define la línea recta que separa las dos categorías. Los puntos que se encuentran por encima de la línea se clasifican en una categoría, mientras que los puntos que están por debajo de la línea se clasifican en otra categoría.


Es importante destacar que el perceptrón solo puede separar clases linealmente separables. Si las clases no son linealmente separables, el perceptrón no podrá encontrar una separación adecuada.



hazlo en python

Para representar la separación de un perceptrón de una sola neurona en Python, se puede usar la biblioteca Matplotlib para graficar la línea recta que separa las dos categorías. Aquí te dejo un ejemplo de cómo hacerlo:

css

 Copy code

```
import numpy as np
import matplotlib.pyplot as plt

# Definir pesos y sesgo del perceptrón
w = np.array([0.5, -0.5])
b = 0.2

# Definir función que representa la línea recta
def separatrix(x):
    return (-w[0]*x - b) / w[1]

# Generar datos de dos categorías
np.random.seed(0)
X = np.random.randn(100, 2)
y = np.where(X[:,0] + X[:,1] > 0, 1, -1)


# Graficar datos y separación
plt.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.Paired, edgecolor='black')
x_plot = np.linspace(-3, 3)
plt.plot(x_plot, separatrix(x_plot), 'k-')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



como escribo en un csv en python?

Para escribir en un archivo CSV en Python, puedes usar el módulo `csv` que ya viene incluido en la librería estándar de Python. Aquí te dejo un ejemplo de cómo hacerlo:

python

 Copy code

```
import csv

# Datos a escribir en el archivo CSV
data = [['Juan', 25], ['María', 30], ['Pedro', 35]]

# Nombre del archivo CSV
filename = 'datos.csv'

# Abrir el archivo CSV en modo escritura
with open(filename, mode='w', newline='') as file:
    # Crear un objeto escritor CSV
    writer = csv.writer(file)

    # Escribir los datos en el archivo CSV
    for row in data:
        writer.writerow(row)
```

En este ejemplo, primero definimos los datos que queremos escribir en el archivo CSV. Luego, especificamos el nombre del archivo en la variable `filename`.

A continuación, abrimos el archivo CSV en modo escritura usando la función `open()` de Python, y especificamos el parámetro `mode` como `'w'` para indicar que queremos escribir en el archivo. También especificamos el parámetro `newline=''` para evitar problemas