Memoria

Laboratorio 2

**Grado:** Ingeniería Informática

**Asignatura:** Inteligencia Artificial 2

**Integrantes:** Diego Valencia, Sara Peirotén, Matías Sanz, Ana Scheleissner

**Clase y grupo:** Clase de 3ºC, grupo de trabajo C07

Contenido

[Índice de figuras 3](#_Toc130248699)

[1. Práctica 1: Perceptrón para funciones lógicas 5](#_Toc130248700)

[1.1 Explicación del Código 5](#_Toc130248701)

[1.2 Cuestiones 11](#_Toc130248702)

[1.2.1 Learning Rate más adecuado 11](#_Toc130248703)

[1.2.2 Umbral más adecuado 12](#_Toc130248704)

[1.2.3 Tabla AND 13](#_Toc130248705)

[1.2.4 Representación de la puerta lógica AND 14](#_Toc130248706)

[1.2.4 Representación de la puerta lógica XOR 14](#_Toc130248707)

[2 Práctica 2: MLP con Keras para funciones lógicas 15](#_Toc130248708)

[2.1 Introducción. 15](#_Toc130248709)

[2.2 Desarrollo. 16](#_Toc130248710)

[2.3 Cuestiones 17](#_Toc130248711)

[3 Práctica 3: MLP con Keras para funciones lógicas 20](#_Toc130248712)

[3.1 Introducción. 20](#_Toc130248713)

[3.2 Desarrollo. 20](#_Toc130248714)

[3.3 Cuestiones. 21](#_Toc130248715)

[3.3.1 Categorización y normalización. 21](#_Toc130248716)

[3.3.2 Prueba de arquitecturas 22](#_Toc130248717)

[3.3.3 Evaluación de datos de test y validación 23](#_Toc130248718)

[3.3.4 Problemas en los datos 24](#_Toc130248719)

[3.3.5 Variables más influyentes 25](#_Toc130248720)

[Bibliografía 28](#_Toc130248721)

# Índice de figuras

[Ilustración 1: definiciones 5](#_Toc130248650)

[Ilustración 2: matriz de pesos 5](#_Toc130248651)

[Ilustración 3: definiciones 2 6](#_Toc130248652)

[Ilustración 4: función de activación 6](#_Toc130248653)

[Ilustración 5: Fórmula de modificación de pesos 6](#_Toc130248654)

[Ilustración 6: Función aprendizaje 6](#_Toc130248655)

[Ilustración 7: Funciones CSV 7](#_Toc130248656)

[Ilustración 8: Función Entrenamiento completa 7](#_Toc130248657)

[Ilustración 9: Llamada a función 8](#_Toc130248658)

[Ilustración 10: Entrenamiento XOR 9](#_Toc130248659)

[Ilustración 11: Fórmula ecuación hiperplano 9](#_Toc130248660)

[Ilustración 12: Plot Gráfica Hiperplano 9](#_Toc130248661)

[Ilustración 13: Plot Hiperplano 2 10](#_Toc130248662)

[Ilustración 14: Rangos learnnig\_rate y umbral 10](#_Toc130248663)

[Ilustración 15: Código pruebas 10](#_Toc130248664)

[Ilustración 16: Muestra de resultados 10](#_Toc130248665)

[Ilustración 17: Tabla Learning rate 11](#_Toc130248666)

[Ilustración 18: Gráfico Learning rate adecuado 11](#_Toc130248667)

[Ilustración 19: Gráfica learning rate 2 12](#_Toc130248668)

[Ilustración 20: Gráfica umbral adecuado 12](file:///C:\Users\dvs26\Desktop\IA%20TODO%20PR\LAB02-GRUPOC07\Memoria_LAB02_GRUPOC07.docx#_Toc130248669)

[Ilustración 21:: Gráfica umbral adecuado 2 13](file:///C:\Users\dvs26\Desktop\IA%20TODO%20PR\LAB02-GRUPOC07\Memoria_LAB02_GRUPOC07.docx#_Toc130248670)

[Ilustración 22: tabla de datos 14](#_Toc130248671)

[Ilustración 23: Gráfica AND 14](#_Toc130248672)

[Ilustración 24: Gráfica XOR 14](#_Toc130248673)

[Ilustración 25: Número de neuronas adecuado 15](#_Toc130248674)

[Ilustración 26: Función de entrenamiento 16](#_Toc130248675)

[Ilustración 27: Creación del dataframe 16](#_Toc130248676)

[Ilustración 28: Creación del dataframe y diccionario 16](#_Toc130248677)

[Ilustración 29: Tabla Sigmoide 17](#_Toc130248678)

[Ilustración 30: Tabla ReLU 18](#_Toc130248679)

[Ilustración 31Función para crear red MLP 20](#_Toc130248680)

[Ilustración 32 Función de entrenamiento del MLP con callback 20](#_Toc130248681)

[Ilustración 33 Función que evalua el MLP 21](#_Toc130248682)

[Ilustración 34 Transformación mediante LabelEncoder y to\_categorical de la variable objetivo 21](#_Toc130248683)

[Ilustración 35 Transformación de las columnas no numéricas mediante ft\_transform 21](#_Toc130248684)

[Ilustración 36 Normalización de datos mediante la función MinMax 21](#_Toc130248685)

[Ilustración 37 Separación del dataset en conjunto de entrenamiento, validación y test junto con sus salidas 22](#_Toc130248686)

[Ilustración 38 Función para evaluar las neuronas válidas de las capas ocultas 22](#_Toc130248687)

[Ilustración 39 llamada a la función validar neuronas que devuelve el rango adecuado 22](#_Toc130248688)

[Ilustración 40 Automatizador para encontrar mejor modelo en base a neuronas, capas ocultas e iteraciones 23](#_Toc130248689)

[Ilustración 41 Valores que dan los mejores resultados 23](#_Toc130248690)

[Ilustración 42 Crear la mejor red y entrenarla 23](#_Toc130248691)

[Ilustración 43 Evaluar los datos de validación y de test 23](#_Toc130248692)

[Ilustración 44 Evaluación del mejor resultado con los datos de validación 24](#_Toc130248693)

[Ilustración 45 Histograma de datos original 25](#_Toc130248694)

[Ilustración 46 Histograma con datos reestructurados 25](#_Toc130248695)

[Ilustración 47 Sacar correlacion de matrices 26](#_Toc130248696)

[Ilustración 48 Código de creación de mapa de correlaciones 26](#_Toc130248697)

[Ilustración 49 Mapa de correlación 26](#_Toc130248698)

# Práctica 1: Perceptrón para funciones lógicas

La primera práctica consiste en crear un modelo de perceptrón para solucionar dos problemas de puertas lógicas: la puerta AND y la puerta XOR.

Se responderán las cuestiones planteadas en el enunciado y se presentarán los datos obtenidos durante las ejecuciones del código en tablas.

Durante esta práctica se hará alusión a una serie de términos:

* Yr: salida real de la neurona.
* Yd: salida deseada de la neurona.
* Etha: sinónimo de learning rate.
* Umbral: Threshold (valor a superar en la función de activación).
* Epoch: épocas (número de veces que recorre todos los datos de entrada).

## Explicación del Código

Definiremos el periodo, etha y el umbral con los cuales entrenaremos y trabajaremos en el perceptrón.

Texto

Descripción generada automáticamente con confianza baja

Ilustración : definiciones

Tanto para la puerta lógica AND, como para la puerta lógica XOR, vamos a generar unos pesos aleatorios entre -1 y 1. Además se definirán las entradas y salidas de cada una de las puertas lógicas.

Datos  AND


Ilustración : matriz de pesos

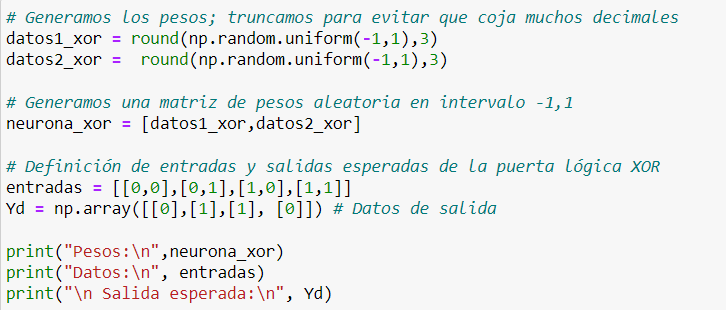


Ilustración : definiciones 2

Como bien se explica en la teoría dada en el tema 3: si la suma ponderada es menor que el umbral entonces se nos devolverá un cero. En caso contrario se nos devolverá como salida un 1.

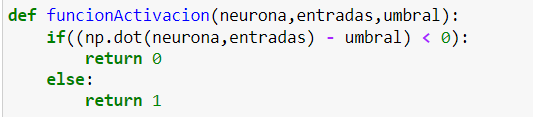


Ilustración : función de activación

En nuestra función aprendizaje actualizaremos los pesos; sumando nuestros antiguos pesos a la multiplicación de nuestro etha, errores y las entradas. Para ello seguiremos la siguiente fórmula de la teoría para la modificación de los pesos.



Ilustración : Fórmula de modificación de pesos

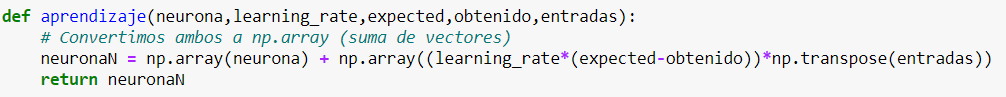


Ilustración : Función aprendizaje

Para generar el archivo .csv, se le pasa una fila que contiene las iteraciones, las entradas, los pesos aleatorios generados, la salida de la red, la salida esperada y los pesos actualizados.

Texto

Descripción generada automáticamente

Ilustración : Funciones CSV

Para entrenar nuestro perceptrón para la puerta lógica AND, hemos creado un while, el cual no parará hasta que nuestra variable “Fin” no sea True y la iteración en la que nos encontremos sea mayor a nuestro periodo (100), puesto que significará que hemos recorrido todas nuestras epochs.

Nos guardamos la salida real y la neurona obtenida para luego poder utilizarlas a la hora de calcular el error y actualizar los pesos.

Hemos creado un vector de ceros, con el cual vamos a ir monitorizando cuantos ceros encontramos por cada recorrido de la tabla lógica, es decir, si encontramos cuatro ceros seguidos en la columna del error consideramos que se ha encontrado una solución y paramos la ejecución.

Texto

Descripción generada automáticamente con confianza media

Ilustración : Función Entrenamiento completa

Llamada a la función del entrenamiento del perceptrón de la puerta lógica AND.



Ilustración : Llamada a función

En el entrenamiento del perceptrón de la puerta lógica XOR, hemos realizado prácticamente los mismos pasos que en el perceptrón de AND. En lo que difieren ambos entrenamientos es en que para este caso no hemos monitoreado el número de errores seguidos que obtenemos, puesto que, como bien sabemos no podemos resolver la puerta XOR con un solo perceptrón, por lo que nunca encontrará una solución. En este caso el modelo parará cuando se hayan realizado todas las iteraciones.

Texto

Descripción generada automáticamente con confianza media

Ilustración : Entrenamiento XOR

Para la representación tanto de AND como de XOR hemos seguido la fórmula de la ecuación del hiperplano de los apuntes:

Diagram

Description automatically generated

Ilustración : Fórmula ecuación hiperplano

Texto

Descripción generada automáticamente

Ilustración : Plot Gráfica Hiperplano

Texto

Descripción generada automáticamente

Ilustración : Plot Hiperplano 2

Por último, para encontrar el learning rate y umbrales óptimos hemos automatizado las pruebas con dos for:

* El bucle del umbral va desde 0.1 hasta 0.9.
* El bucle del learning rate va 0.1 a 0.55.

Texto

Descripción generada automáticamente

Ilustración : Rangos learnnig\_rate y umbral

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración : Código pruebas



Ilustración : Muestra de resultados

## Cuestiones

A continuación, se procederá a resolver las preguntas correspondientes a la primera práctica del segundo laboratorio.

### Learning Rate más adecuado

Tras la ejecución del código de las pruebas se visualiza la siguiente tabla, en la que por cada learning rate se ejecuta 9 veces para poder obtener diferentes soluciones, es decir, si se encuentra solución o no y en cuantas iteraciones se encuentra dicha solución.

Tabla

Descripción generada automáticamente

Ilustración : Tabla Learning rate

Se ha representado en una gráfica la relación entre el learning rate y el número de iteraciones. Como se puede observar con un learning 0.1 obtenemos una media de 10 iteraciones más o menos.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ilustración : Gráfico Learning rate adecuado

En esta gráfica relacionamos el etha con la media de aciertos y como podemos observar con un learning rate menor a 0.1, obtenemos una tasa de acierto del 100%. Pero como anteriormente hemos visto que con etha=0.1 obtenemos la solución alrededor de las 10 iteraciones concluimos que etha va a ser 0.1.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Ilustración : Gráfica learning rate 2

### Umbral más adecuado

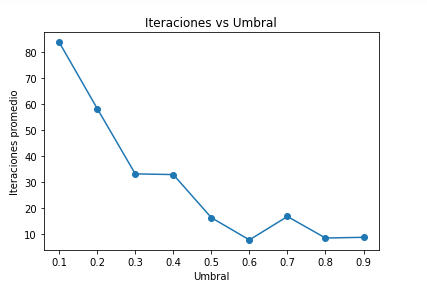
En las siguientes gráficas hemos representado la relación entre el Umbral y el número de iteraciones para calcular el mejor umbral

Ilustración : Gráfica umbral adecuado

Como podemos observar en la gráfica, cuando ponemos un umbral de 0.6, solo se necesitan un promedio de 10 iteraciones.

En la siguiente tabla vamos a comparar los aciertos con el umbral.

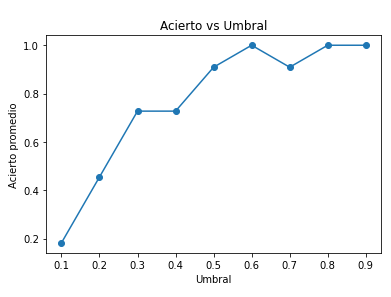


Ilustración :: Gráfica umbral adecuado 2

En esta gráfica podemos observar que con un learning rate de 0.6, vanos a conseguir un 100% de aciertos. Tras ver estas dos tablas hemos elegido usar un umbral de 0.6.

Podemos ver que, en las dos gráficas, poniendo el umbral a 0.8 nos da el mismo resultado, pero hemos escogido el 0.6 ya que si la neurona tiene el umbral muy alto, puede que nunca se active.

### Tabla AND

A continuación, se presenta una tabla como la explicada en la teoría, que recoge los datos de las entradas, los pesos iniciales, la salida real de la neurona y la salida deseada de la neurona, el error y los pesos finales.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iteración | Entradas | | Pesos iniciales | | Yr | Yd | Error | Pesos finales | |
| X1 | X2 | W1 | W2 | W1 | W2 |
| 1 | 0 | 0 | 0.753 | 0.375 | 0 | 0 | 0 | 0.753 | 0.375 |
| 0 | 1 | 0.753 | 0.375 | 0 | 0 | 0 | 0.753 | 0.375 |
| 1 | 0 | 0.753 | 0.375 | 0 | 1 | 1 | 0.653 | 0.375 |
| 1 | 1 | 0.653 | 0.375 | 1 | 1 | 0 | 0.653 | 0.375 |
| 2 | 0 | 0 | 0.653 | 0. 375 | 0 | 0 | 0 | 0.653 | 0.375 |
| 0 | 1 | 0.653 | 0. 375 | 0 | 0 | 0 | 0.653 | 0.375 |
| 1 | 0 | 0.653 | 0. 375 | 0 | 1 | 1 | 0.553 | 0.375 |
| 1 | 1 | 0.553 | 0. 375 | 1 | 1 | 0 | 0.553 | 0.375 |
| 3 | 0 | 0 | 0.553 | 0.375 | 0 | 0 | 0 | 0.553 | 0.375 |
| 0 | 1 | 0.553 | 0.375 | 0 | 0 | 0 | 0.553 | 0.375 |
| 1 | 0 | 0. 553 | 0.375 | 0 | 1 | 1 | 0.453 | 0.375 |
| 1 | 1 | 0.453 | 0.375 | 1 | 1 | 0 | 0.453 | 0.375 |
| 4 | 0 | 0 | 0.453 | 0. 375 | 0 | 0 | 0 | 0.453 | 0.375 |
| 0 | 1 | 0.453 | 0.375 | 0 | 0 | 0 | 0.453 | 0.375 |
| 1 | 0 | 0.453 | 0. 375 | 0 | 1 | 1 | 0.353 | 0.375 |
| 1 | 1 | 0. 353 | 0. 375 | 1 | 1 | 0 | 0.353 | 0.375 |
| 5 | 0 | 0 | 0. 353 | 0. 375 | 0 | 0 | 0 | 0.353 | 0.375 |
| 0 | 1 | 0. 353 | 0. 375 | 0 | 0 | 0 | 0.353 | 0.375 |
| 1 | 0 | 0. 353 | 0. 375 | 0 | 0 | 0 | 0.353 | 0.375 |
| 1 | 1 | 0. 353 | 0. 375 | 1 | 1 | 0 | 0.353 | 0.375 |

Ilustración : tabla de datos

### 1.2.4 Representación de la puerta lógica AND

En el siguiente gráfico, el punto azul corresponde a Yd = 1 y los puntos rojos a Yd=0. Se puede comprobar que la recta separa correctamente los puntos azules de los rojos:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Ilustración : Gráfica AND

### Representación de la puerta lógica XOR

En el siguiente gráfico, el punto azul corresponde a Yd = 1 y los puntos rojos a Yd=0. Se puede comprobar que la recta no separa correctamente los puntos azules de los rojos:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Ilustración : Gráfica XOR

# Práctica 2: MLP con Keras para funciones lógicas

## 2.1 Introducción.

El perceptrón multicapa es un modelo de red neuronal feedforward no lineal, que se entrena mediante aprendizaje supervisado.

Este modelo es utiliza el algoritmo de aprendizaje por minimización del error cuadrático medio mediante la regla Delta. Sus aplicaciones principales incluyen la clasificación y el reconocimiento de señales, cuenta, además, con ventajas como la capacidad de generalización y entrenamiento rápido en tiempo de ejecución.

Este modelo consta de al menos tres capas interconectadas (multicapa). Sin embargo, una de las principales dificultades del modelo es determinar el número óptimo de capas ocultas y el número de neuronas en cada capa. Si no se establecemos adecuadamente estos valores o variables el modelo puede sufrir de sobreajuste o quedarse por debajo.

Para verificar empíricamente el número de neuronas adecuado en una capa oculta, se puede utilizar la siguiente fórmula:

Texto

Descripción generada automáticamente

Ilustración : Número de neuronas adecuado

Siendo ‘M’ el número de ejemplos del número de entrenamiento, ‘N’ el número de neuronas de la capa de entrada y ‘n’ el número de neuronas de la capa intermedia.

En nuestro caso, usaremos, sin codificarlo, tanto 2 y 3 neuronas pues los valores cumplen los parámetros que le pedimos.

Hemos de hacer uso de dos funciones de activación, la ReLU y la Sigmoidea.

La función de activación RELU nos aporta al modelo una funcionalidad, en la que, los valores negativos que entran pasan a 0 y los positivos se mantienen igual.

El problema de la Sigmoidea es que satura el gradiente, siendo esto lo que queremos para calcular el error.

## 2.2 Desarrollo.

A la hora de codificar el desarrollo de la práctica, primero instanciamos el MLP, luego definimos la capa oculta y la capa de salida, a continuación, configuramos el método de entrenamiento, agregando como métrica de error el *error cuadrático medio* y como optimizador *Adam*.

Todo esto lo llevamos a cabo en una función, la cual llamaremos dos veces, especificando en cada una de las llamadas el tipo de función de activación que queremos usar.

Texto

Descripción generada automáticamente

Ilustración : Función de entrenamiento

Por último, entrenamos la red, obtenemos las predicciones y obtenemos el error.

A la hora de obtener las tablas pedidas en el apartado de cuestiones, hemos creado un diccionario que recoja los datos de las iteraciones de las ejecuciones, este es posteriormente transformado a un dataframe para poder verlo mucho mejor.



Ilustración : Creación del dataframe

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ilustración : Creación del dataframe y diccionario

El procedimiento ‘Adam’ funciona bastante bien en conjuntos de datos relativamente grandes, en muestras de datos pequeñas puede no ser el más adecuado teniendo alternativas como el de descenso de gradiente o el ‘**lbfgs’**, en nuestro caso usaremos el ‘**Adam’** no solo por su uso extendido sino porque lo especifica el enunciado.

El ajuste insuficiente o **underfitting** sucede cuando hay muy pocas neuronas en las capas ocultas para detectar adecuadamente las señales en un conjunto de datos complicado. Del mismo modo, del exceso de número de neuronas derivan otros problemas como el sobreajuste u **overfitting** entro otros.

## 2.3 Cuestiones

A continuación, muestro los resultados de la ejecución para 2 y 3 neuronas en la capa oculta, con unos Epochs que varían entre 2000 y 5000 en intervalos de 200, usando como función de activación la función Sigmoidea.

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente con confianza media

Ilustración : Tabla Sigmoide

Y adjunto la tabla con descripción idéntica a la anterior cambiando, esta vez, la función de activación Sigmoidea por la ReLU. E iterando entre 250 Epochs en lugar de 200.

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

Ilustración : Tabla ReLU

Las tablas responden a lo pedido en el enunciado mostrando, por columnas, las neuronas de capa oculta, los Epochs, la Salida y el Error.

En las posteriores gráficas se observan el Error en función de los Epochs, pero no debemos de dejar de mirar la columna de salida donde de manera visual confirmamos si ha acertado, o no.

Se observa, en el caso de la función Sigmoidea, como para 2 neuronas devuelve una salida correcta en grupos aislados, sin embargo, en el caso de 3 neuronas en la capa oculta obtenemos un mayor número de aciertos.

En el caso de la función ReLU, observando los resultados para 2 y 3 neuronas, ambos poseen valores de salida acertados asilados, de igual manera existe un mayor número de aciertos para el caso de 3 neuronas.

# Práctica 3: MLP con Keras para funciones lógicas

## 3.1 Introducción.

En esta tercera practica se nos da un dataset con diferentes datos de clientes de un banco; proporcionando en la última columna la variable objetivo; que nos dice si un cliente se suscribe o no a un depósito a plazos.

Debemos realizar la práctica utilizando la librería de Keras/TensorFlow como en el ejercicio anterior, pero demostrando en este caso que se puede aplicar a un caso real con muchas más entradas que la resolución con operaciones lógicas.

## 3.2 Desarrollo.

Hemos creado una función que al ser llamada crea una red para un numero de neuronas para la capa oculta, numero de capas ocultas y un tamaño de entrada que correspondería a las columnas o características de nuestro dataset.

Hemos utilizado la función RELU para las capas ocultas, la sigmoidea para la de salida junto con la función de error cuadrático medio. Esta función devolverá el modelo creado que será utilizado para entrenar el Perceptrón Multicapa:

Texto

Descripción generada automáticamente

Ilustración Función para crear red MLP

Para entrenar el MLP hemos creado esta función a la que pasamos el modelo, las entradas de entrenamiento y las salidas esperadas sobre la concesión del deposito a plazos. Hemos utilizado una función callback de Keras llamada EarlyStopping la cual puede ser configurada para ser parada cuando la precisión o el error no mejoren en las ultimas x iteraciones. Para nuestro entrenamiento hemos decido parar cuando el error no mejore en las últimas 5 iteraciones.

Entrenamos el modelo mediante la funcion fit y le insertamos las entradas de entrenamiento, sus salidas esperadas, los datos de validación y el callback antes definido.

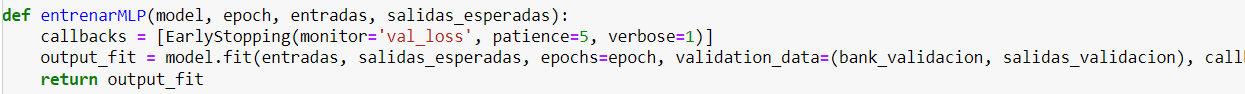


Ilustración Función de entrenamiento del MLP con callback

Una vez tenemos el modelo entrenado hemos creado esta función que nos devuelve la precisión, la matriz de confusión y un reporte de clasificación que nos proporciona mucha información y porcentajes de nuestra solución.

Imagen que contiene Texto

Descripción generada automáticamente

Ilustración Función que evalua el MLP

## 3.3 Cuestiones.

### 3.3.1 Categorización y normalización.

Para realizar el estudio del dataset primero debemos realizar una limpieza de los datos, categorizarlos y normalizarlos. De esta forma podremos entrenar mejor el modelo de Keras y operará más fácilmente.

Para categorizar los datos hemos transformado las salidas de la columna que nos da la información sobre si se ha aceptado o no el depósito; la columna tiene los parámetros ‘yes’ no’ y debemos transformarlos en 0 y 1; para ello utilizamos LabelEncoder y to\_categorical de keras.

Texto, Logotipo, nombre de la empresa

Descripción generada automáticamente con confianza media

Ilustración Transformación mediante LabelEncoder y to\_categorical de la variable objetivo

Después modificamos las demás columnas que tienen variables con caracteres para transformarlas en números con fit\_transform; de esta forma todos los datos quedarán transformados en números.

Texto

Descripción generada automáticamente

Ilustración Transformación de las columnas no numéricas mediante ft\_transform

Después de tener todos los datos en formato numérico debemos normalizarlos para que nuestro modelo entienda mejor los datos y le sea mas sencillo realizar los cálculos. Para realizar la normalización hemos utilizado MinMax donde restas el mínimo al conjunto completo de datos y lo divides por máximo restado al mínimo; de esta forma nuestros datos están en un rango entre 0 y 1 y normalizados:



Ilustración Normalización de datos mediante la función MinMax

Debemos separar nuestros datos en tres conjuntos: uno para entrenar el modelo y dos conjuntos de pruebas; uno de test y otro de validación; los cuales nos sirven para evaluar el modelo. Para estos tres conjuntos hemos usado un 10% del total de los datos para test. Y del 90% restante el 80% es usado para el entrenamiento y el 20% para la validación. Tras realizar esto tendremos los dataset separados:

Texto

Descripción generada automáticamente

Ilustración Separación del dataset en conjunto de entrenamiento, validación y test junto con sus salidas

Además, hemos sacado la columna con la variable objetivo e insertado en las variables correspondientes a las salidas esperadas.

### 3.3.2 Prueba de arquitecturas

Para realizar nuestras pruebas en primer lugar hemos visto el rango de neuronas que cumple con la regla de numero de neuronas adecuado.

Texto

Descripción generada automáticamente

Ilustración Función para evaluar las neuronas válidas de las capas ocultas



Ilustración llamada a la función validar neuronas que devuelve el rango adecuado

Una vez sabemos con cuantas neuronas es válido entrenar; hemos creado un automatizador de pruebas el cual itera entre una lista de neuronas, de capas ocultas y de iteraciones; de esta forma conoceremos la mejor estructura para nuestro modelo.

Insertamos una variable de precisión sobre el modelo puesto que si ejecutas varias veces el modelo pueden salir resultados diferentes; por ello realizamos la media entre todas las pruebas de precisión. No hemos podido realizar pruebas de precisión por tiempo.

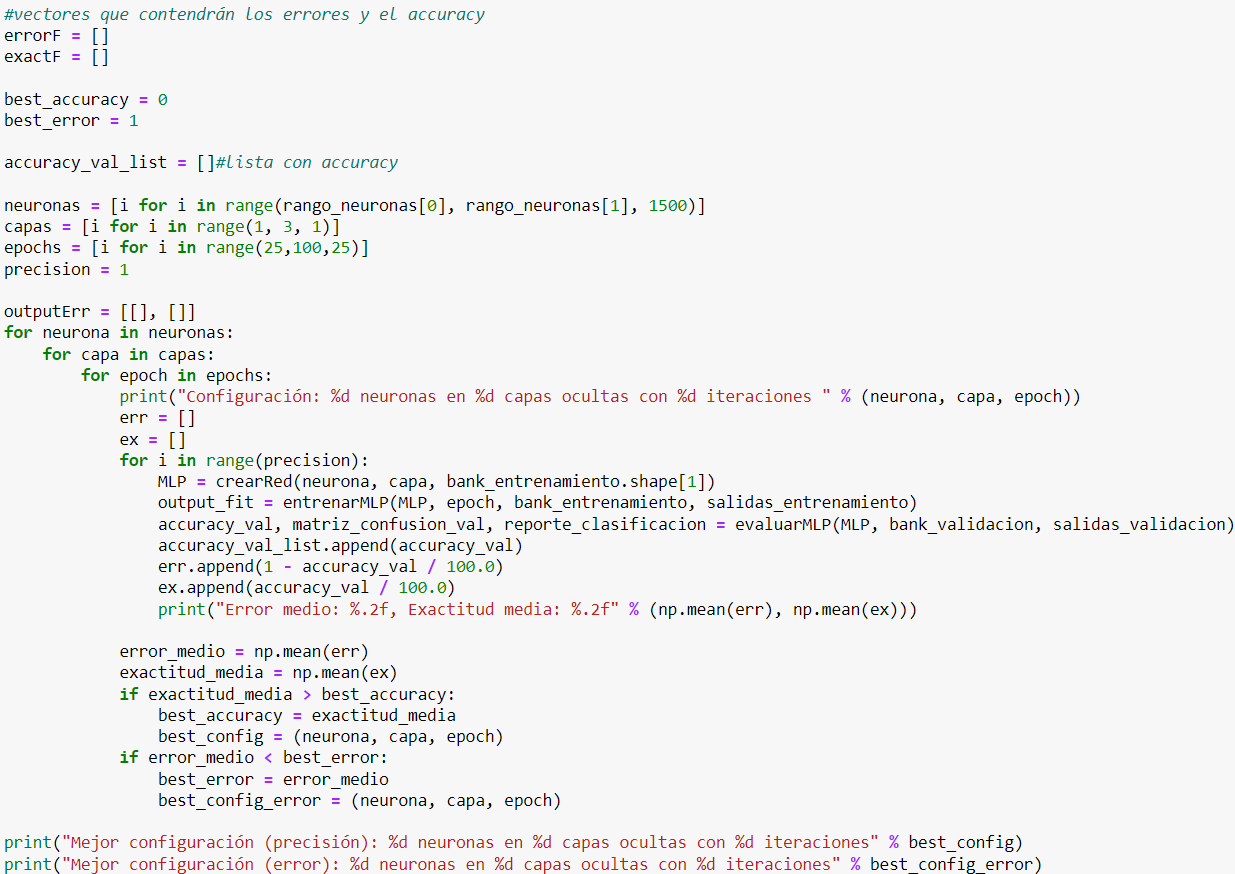


Ilustración Automatizador para encontrar mejor modelo en base a neuronas, capas ocultas e iteraciones

Nuestro automatizador nos pinta la mejor configuración del rango de neuronas valido con variaciones de 1500, capas entre 1 y 2 e iteraciones entre 25 y 100 con saltos de 75:

### 3.3.3 Evaluación de datos de test y validación

Después de realizar nuestras pruebas concluimos que la mejor configuración para nuestro modelo es:

Ilustración Valores que dan los mejores resultados

Creamos, entrenamos el modelo y evaluamos el modelo para los datos de validación y de test:

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

Ilustración Crear la mejor red y entrenarla

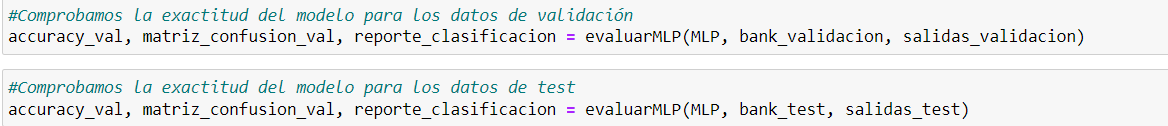


Ilustración Evaluar los datos de validación y de test

Para los datos de validación los resultados son:

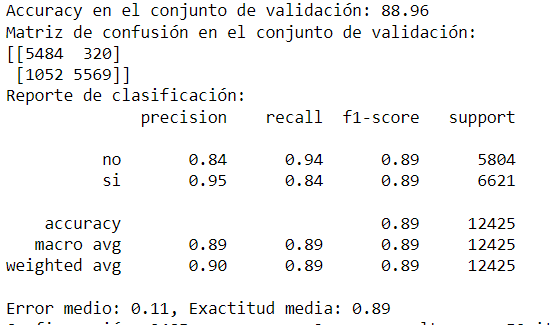


Ilustración Evaluación del mejor resultado con los datos de validación

La matriz de confusión para los datos de validación nos dice que:

Verdaderos negativos: 5311

Falsos pósitos: 493

Falsos negativos: 950

Verdaderos positivos 5671

Para los datos de test los resultados son:

No hemos podido realizar las pruebas puesto que fuimos muy ambiciosos a ultima hora y hemos querido hacer un automatizador muy grande que no ha acabado a la hora prevista estando más de 5 horas ejecutando.

### 3.3.4 Problemas en los datos

El dataset proporcionado en el aula virtual está muy desproporcionado. Tenemos muchos casos donde no se concede el depósito y muy pocos donde sí. Es necesario realizar una redimensión del dataset para que entrenemos con un conjunto de datos que tenga las mismas entradas de concesión de depósito como de no concesión para la variable objetivo:

Gráfico, Gráfico de barras

Descripción generada automáticamente

Ilustración Histograma de datos original

Para solucionar esto hemos pensado varias soluciones; como usar resample de sklearn donde crear nuevos valores aleatorios hasta compensar el dataset o la que hemos acabado utilizando que ha sido igualar los datos repitiéndolos hasta haya el mismo numero de concesión de depósitos y de no concesión. De esta forma es la única manera en la que no nos inventamos los datos; aunque los tres conjuntos de datos que hemos realizado tengan datos repetidos.

Gráfico

Descripción generada automáticamente

Ilustración Histograma con datos reestructurados

### 3.3.5 Variables más influyentes

Para realizar un estudio de las columnas más relevantes del dataset en primer lugar propusimos un análisis de correlación de columnas junto con un mapa de calor:

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja

Ilustración Sacar correlacion de matrices



Ilustración Código de creación de mapa de correlaciones

Gráfico

Descripción generada automáticamente

Ilustración Mapa de correlación

En este mapa los valores más altos de correlación aparecen en rojo y los altos en azul.

Este método pensándolo entre todos posteriormente no es valido puesto que podría haber una columna fundamental que si quitamos nuestro modelo evaluara los datos de validación y test de peor manera. Por tanto, nos hemos basado en el enunciado que nos proporciona una lista de las columnas menos relevantes: 'job', 'marital', 'education', 'housing', 'loan', 'contact', 'poutcome', 'day\_of\_week', 'month' y 'default' [1] .Hemos decidido eliminarlas y realizar las pruebas de toda la practica sin estas.

No hemos podido saber bien que columnas son las mas relevantes; se nos ocurrió un método mejor que por falta de tiempo no hemos podido finalizar que era realizar las pruebas del modelo con todas las columnas e ir eliminando una y observando el resultado y mediante diferentes pruebas de combinaciones de columnas guarda el nombre de las columnas utilizadas que mejores resultados ha dado.

# Bibliografía

|  |  |
| --- | --- |
| [1] | A. Mohanty, «medium.com,» 15 Mayo 2019. [En línea]. Available: https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f. [Último acceso: 2023]. |