UNIVERSIDAD FRANCISCO DE VITORIA

ESCUELA POLITÉCNICA SUPERIOR



Práctica II

Inteligencia artificial II

Grupo B02

Realizada por:

Javier Garzón

Diego Guadalupe

Alejandro Monteliú

Carlos Pujols

ÍNDICE

[1. Resultados de la práctica 1](#_Toc129778574)

[1.1 Puerta AND 4](#_Toc129778575)

[1.2 Puerta XOR 7](#_Toc129778576)

[2. Resultados de la práctica 2](#_Toc129778577)

[3. Resultados de la práctica 3](#_Toc129778578)

[3.1 Explica como has llevado a cabo la normalización y categorización 11](#_Toc129778579)

[3.2 Prueba distintas arquitecturas 13](#_Toc129778580)

[3.3 Matriz de confusión 17](#_Toc129778581)

[3.4 ¿Detectas algún problema en tus datos? 20](#_Toc129778582)

[3.5 ¿Qué variables influyen más y menos a la hora de contratar? 21](#_Toc129778583)

ÍNDICE DE FIGURAS

[Ilustración 1 CSV del perceptrón con lr 0.1 5](#_Toc129778612)

[Ilustración 2 CSV del Perceptrón con lr 0.01 6](#_Toc129778613)

[Ilustración 3 CSV del Perceptrón con lr de 0.5 7](#_Toc129778614)

[Ilustración 4 XOR 8](#_Toc129778615)

[Ilustración 5 CSV del Perceptrón con XOR 9](#_Toc129778616)

[Ilustración 6 Gráficas de una capa con 20 neuronas 14](#_Toc129778617)

[Ilustración 7 Gráfica de una capa con 40 neuronas 15](#_Toc129778618)

[Ilustración 8 Gráfica con 4 capas de neuronas 16](#_Toc129778619)

[Ilustración 9 Matriz de confusión 18](#_Toc129778620)

[Ilustración 10 Métricas del MLP 19](#_Toc129778621)

[Ilustración 11 Influencia de variables 22](#_Toc129778622)

ÍNDICE DE TABLAS

[Tabla 1 Tabla con funciones ReLU 10](#_Toc129778641)

[Tabla 2 Tabla con funciones sigmoide 11](#_Toc129778642)

[Tabla 3 Tabla con funciones ReLU y sigmoide 11](#_Toc129778643)

[Tabla 4 Tabla de las distintas arquitecturas 17](#_Toc129778644)

# Resultados de la práctica 1

## 1.1 Puerta AND

Hemos probado con distintos learning rates y umbrales para la realización del Perceptrón.

A continuación, se adjuntan las distintas tablas obtenidas del .csv:





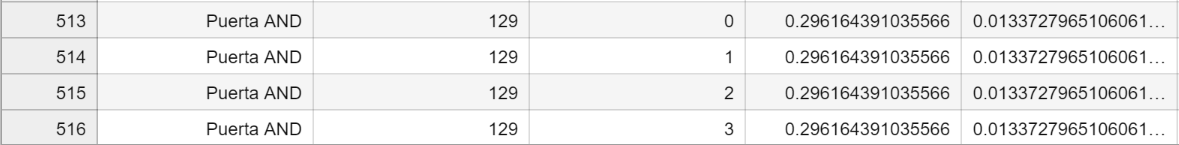
Ilustración 1 CSV del perceptrón con lr 0.1

Como podemos ver, en este caso hace 3 etapas con 4 iteraciones dentro de cada etapa puesto que tiene 4 casos la puerta AND, y hasta que los 4 casos no tengan un error cero en el perceptrón, la ejecución del programa no parará.

En la ejecución anterior, el learning rate está en 0.1 y el umbral de la función de salida en 0.2.

Adjuntamos foto del csv cuando cambiamos estos valores:







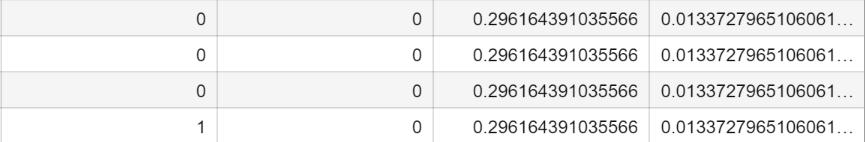


Ilustración 2 CSV del Perceptrón con lr 0.01

En está nueva captura de la tabla del csv, hemos obtenido los últimos registros del mismo.

Cambiamos el learning rate de 0.1 a 0.01 y el umbral de 0.2 a 0.5.

Los resultados tienen un gran cambio, ya que antes llegaba a la etapa 3 y conseguía estabilizar el error y los pesos, pero en este caso no es hasta la etapa 129 que consigue unos pesos para resolver el problema de la puerta AND.

Por último, como hemos comprobado que bajar el learning rate por debajo del 0.1 no es efectivo, probaremos a subirlo:



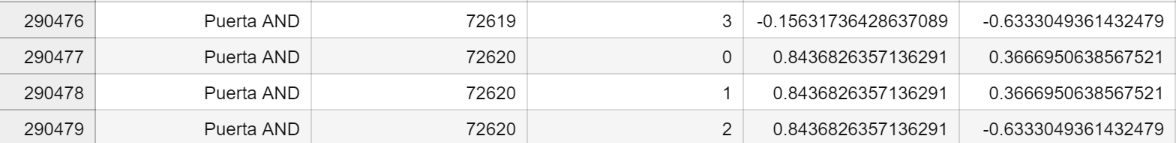






Ilustración 3 CSV del Perceptrón con lr de 0.5

En este último caso hemos probado con learning rate de 0.5 y umbral de 0.3.

Como podemos observar, el perceptrón ni siquiera ha aprendido en las 72620 etapas que ha estado funcionando, ya que paramos su ejecución antes de que aprendiese.

Hicimos esto porque es obvio que este learning rate no es para nada efectivo comparado al primero de todos.

Es por eso que el mejor learning rate para el perceptrón con la puerta AND es 0.1 y el mejor umbral es 0.2.

## Puerta XOR

Con esta puerta no es posible modelizar su comportamiento con un perceptrón.

La razón es que no es un problema linealmente separable.

Atendiendo a la siguiente imagen:

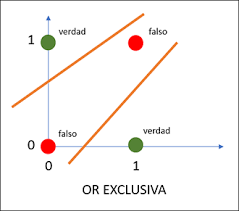


Ilustración 4 XOR

Podemos ver que no es posible separar este problema con una sola línea, es decir, no podemos resolver este problema con un perceptrón.

De igual manera, hicimos la ejecución del perceptrón intentando modelizar la puerta XOR y esto fue lo que obtuvimos:

Cabe destacar que cambiamos unas líneas de código respecto a la puerta AND. Con la puerta AND hasta que no estuviera estable no pararía su ejecución, pero con la puerta XOR esto es imposible puesto que estaría ejecutándose sin parar porque nunca sería estable ya que no puede separar un problema no lineal.

Es por ello que cambiamos su ejecución a 100 etapas y obtuvimos lo siguiente:



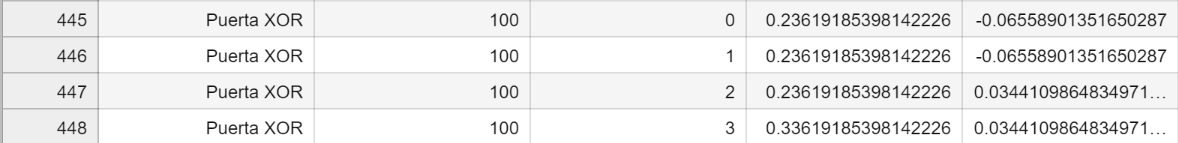




Ilustración 5 CSV del Perceptrón con XOR

Como podemos ver, agota todas las etapas y agotaría todas las que pusiéramos puesto que es imposible que modelice esta puerta.

El learning rate es el mismo que en la puerta anterior al igual que el umbral.

No es necesario probar otros porque independientemente del learning rate y del umbral, el perceptrón nunca podrá modelizar la puerta XOR por ser linealmente inseparable.

# Resultados de la práctica 2

En este ejercicio de la práctica, debemos modelar la puerta XOR, pero esta vez con un MLP (Perceptrón multicapa).

Gracias a usar un perceptrón multicapa, podemos modelar la puerta XOR de manera correcta, gracias a poder usar varias neuronas y funciones de activación no lineales.

Los resultados son los siguientes:

Tabla con funciones de activación ReLU:

|  |  |  |  |
| --- | --- | --- | --- |
| **Número de neuronas de la HL** | **Épocas** | **Salida** | **Error** |
| 2 | 10000 | 0.5 | 0.5 |
| 5 | 10000 | 0.5 | 0.5 |
| 10 | 10000 | 0.5 | 0.5 |

Tabla 1 Tabla con funciones ReLU

Como podemos ver, la salida es 0.5, es decir, nos otorga la salida correcta la mitad de las veces. El error también es 0.5, es decir, proporciona una salida errónea el 50% de las veces.

Esto ocurre porque la función ReLU para la neurona de salida del MLP no tiene un buen funcionamiento, y es por ello que hay que usar la función sigmoide.

Probemos ahora con la función sigmoide:

|  |  |  |  |
| --- | --- | --- | --- |
| **Número de neuronas de la HL** | **Épocas** | **Salida** | **Error** |
| 2 | 10000 | 1 | 0.001 |
| 5 | 10000 | 1 | 0.0001 |
| 10 | 10000 | 1 | 2.9\*10^-5 |
| 10 | 40000 | 1 | 4.9\*10^-9 |

Tabla 2 Tabla con funciones sigmoide

Como podemos ver, a medida que aumentamos el número de neuronas en la capa oculta, el error disminuye. Con 2 o 5 neuronas es suficiente, pero probamos con 10 y más épocas para ver hasta donde podía llegar.

A pesar de que la salida ya es perfecta desde el primer intento, el error bajó hasta 4.9\*10^-9, lo cual es un número ínfimo.

Por último, probemos con ambas funciones combinadas, ReLU para la capa oculta y sigmoide para la neurona de salida:

|  |  |  |  |
| --- | --- | --- | --- |
| **Número de neuronas de la HL** | **Épocas** | **Salida** | **Error** |
| 2 | 10000 | 0.5 | 0.25 |
| 5 | 10000 | 1 | 2.4\*10^-5 |
| 10 | 10000 | 1 | 1.3\*10^-6 |
| 10 | 40000 | 1 | 3.8\*10^-10 |

Tabla 3 Tabla con funciones ReLU y sigmoide

Como podemos ver, con dos neuronas el modelo no es bueno puesto que la salida no es siempre correcta y el error no es lo suficientemente bajo para este modelo.

Pero al subir el modelo a 5 neuronas, podemos ver el gran cambio, y es que combinar la función ReLU y sigmoide hace que el modelo sea lo más eficiente posible, y es por ello por lo que nos decantamos, el uso de la función ReLU para la capa oculta con 5 neuronas y la función sigmoide para la neurona de salida.

# Resultados de la práctica 3

## 3.1 Explica como has llevado a cabo la normalización y categorización

Hemos cargado el csv a una variable de Python por medio de la librería pandas.

Luego hemos utilizado la librería sklearn para poder transformar todos los datos categóricos a numéricos para poder introducirlos en la red.

Asimismo, hemos hecho una discretización en la edad de la persona. En vez de introducir la edad a la red, lo que hemos hecho es transformar las edades en 3 intervalos, uno para menores de edad, otro para adultos, y el último para jubilados.

De esta manera es más simple para la red entender la edad como un intervalo porque es indiferente si el usuario tiene 40 o 50 años, pero si es relevante si tiene 16 o 68.

Después, hemos hecho un conteo de la cantidad de registros que tenían la clase 0 y cuantos tenían la clase 1, siendo 0 que no contratan el servicio y 1 los que si lo hacen.

Hay un total de 36000 registros que tienen clase 0 y tan solo 4600 que tienen clase 1.

Es necesario que el dataset esté balanceado con el fin de que la red generalice bien. Si no lo balanceásemos, la red aprendería muy bien los de clase 0 pero no los de clase 1.

Para ello, quitamos 31000 registros de clase 0, y con ello la clase 0 tiene 5000 registros y la clase 1 tiene 4600. Con ello nuestro dataset está balanceado.

Después, dividimos el dataset completo en 3 partes, una para entrenar, otra para realizar la validación cruzada y la última para los test.

Una vez tenemos las 3 partes, normalizamos los datos para que estén en el rango de 0-1 y está listo el preprocesamiento para que la red pueda entrenar y generalizar.

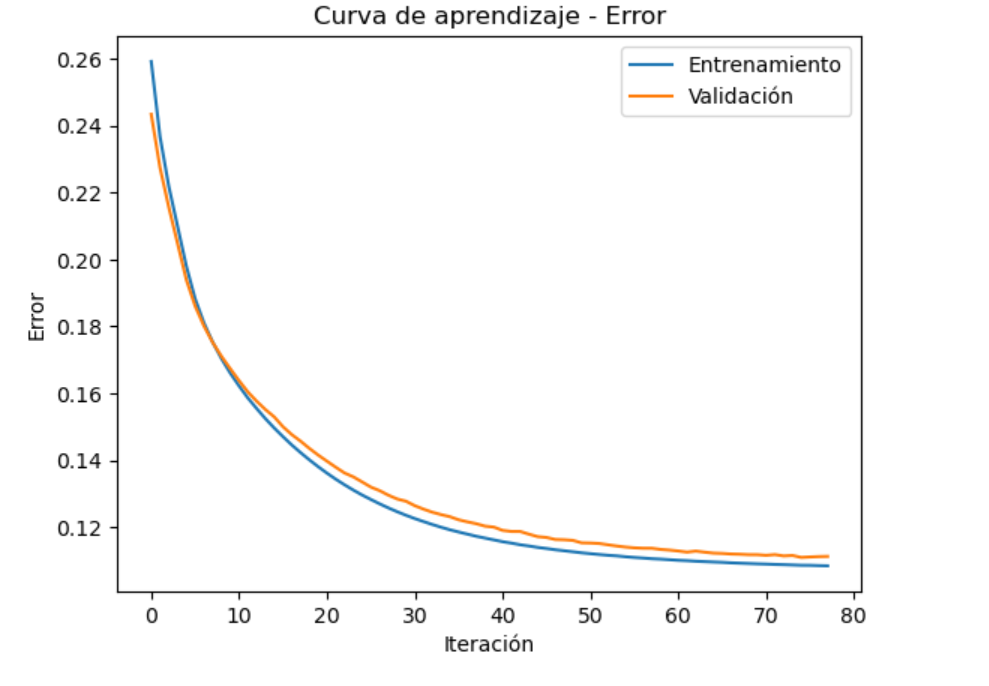
Por último, hemos quitado las siguientes columnas:

-'marital', 'education', 'contact', 'poutcome', 'day\_of\_week', 'month'

Esto se debe a que después de estudiar el dataset y ver que significa cada columna, decidimos quitar esas columnas para bajar la dimensionalidad del problema y quitar atributos que no son relevantes para el problema de saber si los usuarios deciden contratar el servicio del banco o no.

## Prueba distintas arquitecturas

**-Arquitectura de una capa con 20 neuronas:**



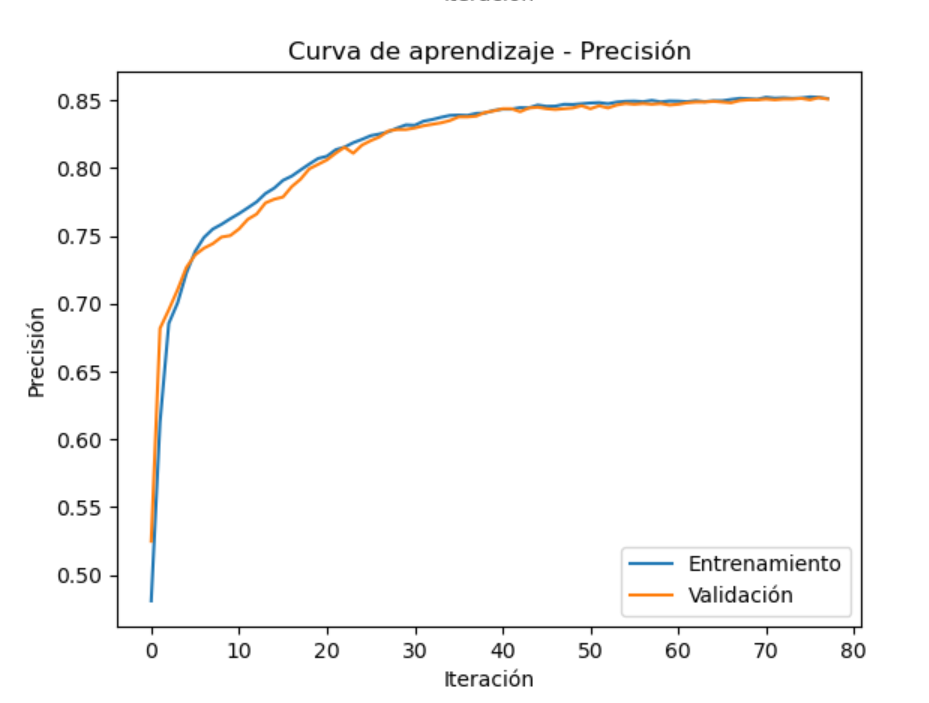
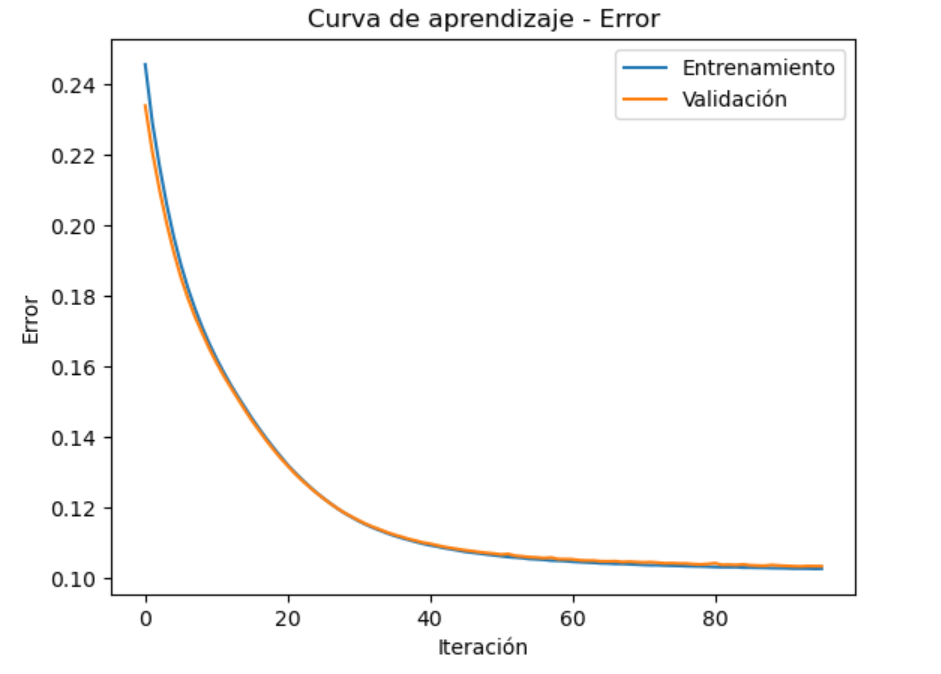


Ilustración 6 Gráficas de una capa con 20 neuronas

El loss de esta red es de 0.1 y el accuracy es del 85%

**-Arquitectura de una capa con 40 neuronas:**



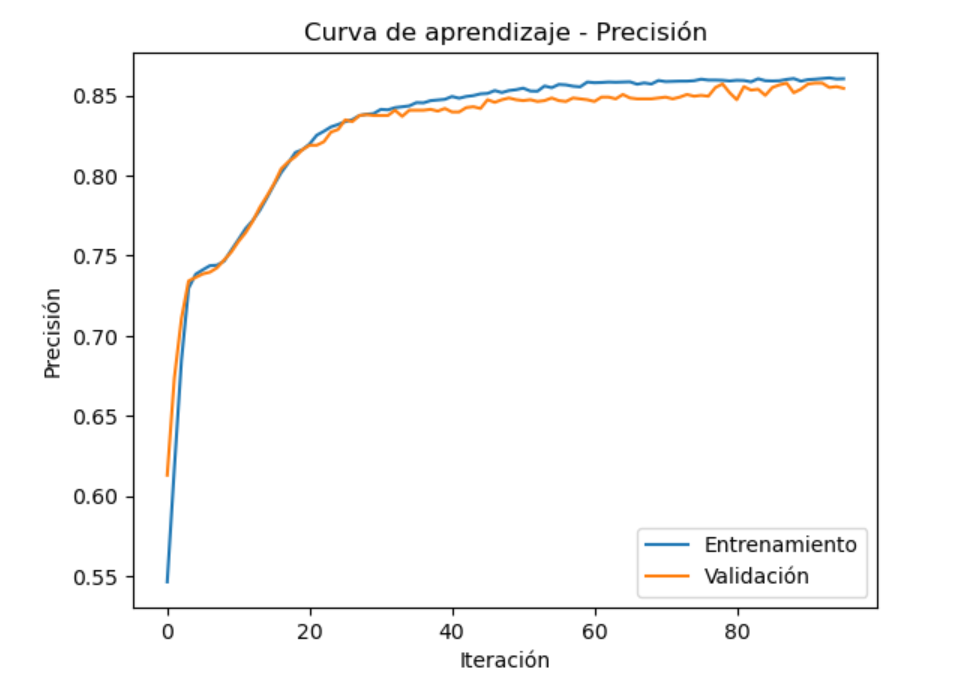
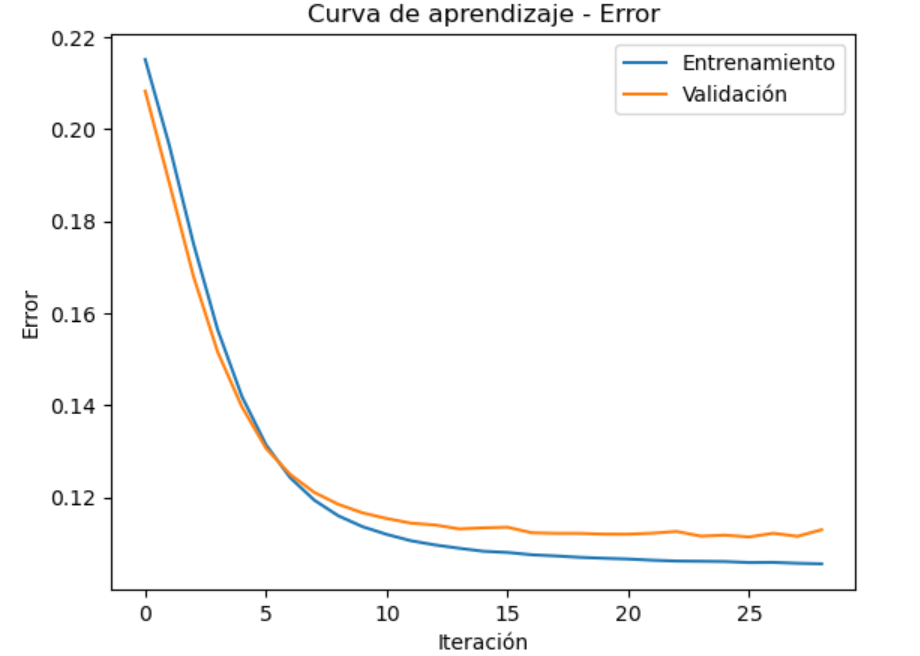


Ilustración 7 Gráfica de una capa con 40 neuronas

En las anteriores gráficas, vemos que el MLP tiene una capa con 40 neuronas. El loss es 0.1 y el accuracy es del 85%. Por ello, tiene el mismo valor en las métricas que la capa con 20 neuronas.

-Arquitectura de 4 capas:



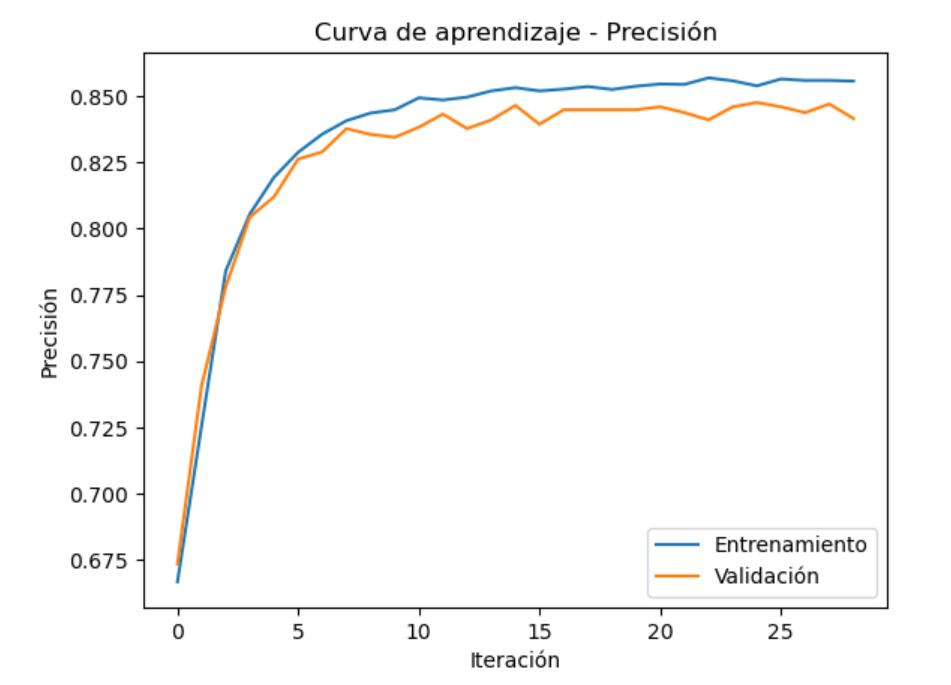


Ilustración 8 Gráfica con 4 capas de neuronas

Con esta arquitectura, tenemos 4 capas ocultas. La primera tiene 20 neuronas, la segunda 15, la tercera 10 y la última oculta tiene 5.

El loss de esta arquitectura es 0.1 y el accuracy es del 85%.

Podemos ver que tiene exactamente los mismos valores que los anteriores modelos.

Esto se debe a que las mejores métricas se consiguen con solo una capa con 20 neuronas, ya que para un dataset y problema sencillo como es este, no son necesarias más de 20 neuronas y una sola capa oculta.

La tabla quedaría así:

|  |  |  |  |
| --- | --- | --- | --- |
| **Número de capas** | **Número de neuronas** | **Loss** | **Accuracy** |
| 1 | 8 | 0.17 | 83% |
| 1 | 20 | 0.1 | 85% |
| 1 | 40 | 0.1 | 85% |
| 4 | 20, 15, 10, 5 | 0.1 | 85% |

Tabla 4 Tabla de las distintas arquitecturas

Atendiendo a la tabla anterior, podemos ver que una sola capa con 20 neuronas es suficiente y eficiente para este problema, ya que a pesar de poner más capas y neuronas el error no desciende y el accuracy no aumenta.

Si bajamos el número de neuronas por debajo del número de entradas, el loss aumenta y el accuracy disminuye.

## Matriz de confusión

La arquitectura que usamos es la de una capa y 20 neuronas ya que alcanza el error mínimo con y no es necesario cargar a la red con más neuronas ni capas.

Después del entrenamiento, pasamos el set de test extraído del dataset original.

La matriz de confusión queda así:

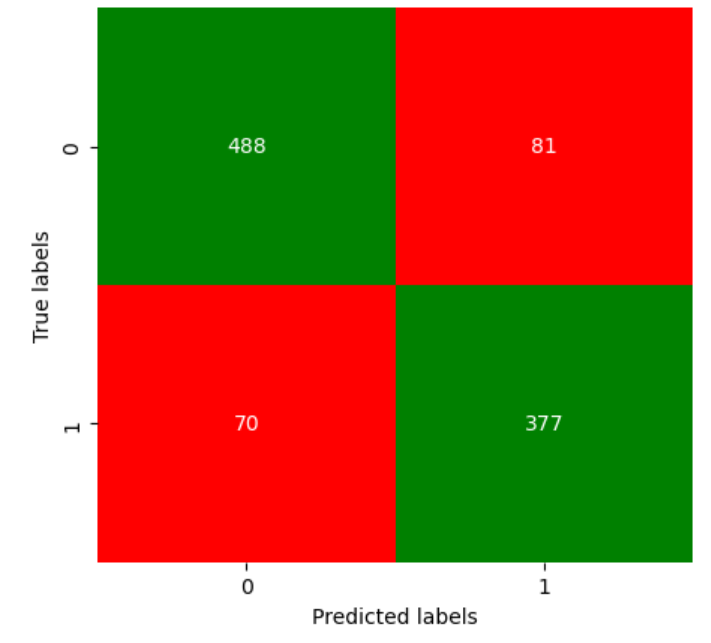


Ilustración 9 Matriz de confusión

Y las métricas quedan así:

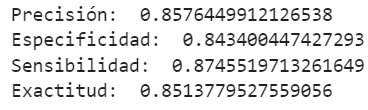


Ilustración 10 Métricas del MLP

Esto nos da mucha información relevante.

Para empezar, la matriz de confusión nos representa:

-Cuadrante arriba a la izquierda: True positives, es decir, las veces que la respuesta es 0 y la red ha dado 0.

-Cuadrante de abajo a la izquierda: False positives, es decir, las veces que la respuesta es 1 y la red ha dado 0.

-Cuadrante de abajo a la derecha: True negatives, es decir, las veces que la respuesta es 1 y la red ha dado 1.

-Cuadrante de arriba a la derecha: False negatives, es decir, las veces que la respuesta es 0 y la red ha dado 1.

Podemos observar que tanto el valor de los true positives y true negatives es bastante superior al número de false correspondientes.

Esto nos da un indicativo de que la red está bien entrenada, pero no es suficiente con ello.

Aquí entran las métricas. Son parámetros calculados a partir de la matriz de confusión que nos indican aspectos importantes.

-La precisión mide la capacidad de la red para dar con la respuesta correcta en cada caso. Tiene un 85%, que es un buen valor.

-La especificidad nos da el porcentaje de casos false, en este caso el porcentaje de casos 1 que ha dado la red. Tiene un 84%, que es un buen valor.

-La sensibilidad nos da el porcentaje de casos true, en este caso el porcentaje de casos 0 que ha dado la red. Tiene un 87%, que es un buen valor.

-La exactitud mide las predicciones correctas. En este caso tiene un 85%, que es un buen valor.

La primera vez que entrenamos la red sin el preprocesamiento correcto, la especificidad nos daba un 30%. Esto se debe a que había 30000 casos más de 0 que de 1. Esto hacía que la red predijese muchos más 0 de los que debería, y por ello el número de casos que la red daba 0 era mucho menor y la especificidad bajaba.

Para arreglarlo, balanceamos la carga de ambas clases para que estuvieran entorno al 50% y poder obtener las métricas con buenos resultados.

## ¿Detectas algún problema en tus datos?

El mayor problema que detectamos fue la carga de balanceo que tenía el dataset.

Como hemos dicho anteriormente, el dataset tenía muchísimos casos más de clase 0 que de 1, 31000 para ser exactos.

Para solucionar el problema, simplemente eliminamos registros con clase 0 para balancearlo.

Otra solución sería ponernos en contacto con la empresa que nos facilitó el dataset y pedir otro dataset distinto, pero esto no es posible en este caso.

Una vez se solucionó el problema, las métricas y la matriz de confusión obtuvieron buenos resultados, dándonos así una red neuronal que generaliza para este caso.

## ¿Qué variables influyen más y menos a la hora de contratar?

Atendiendo a la siguiente captura:

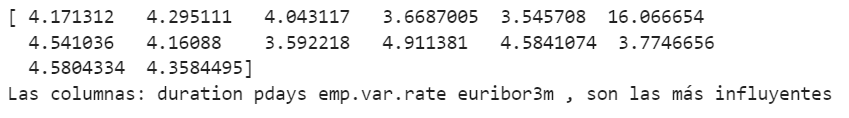


Ilustración 11 Influencia de variables

La tabla corresponde a la suma en valor absoluto de las matrices de pesos, para así ver la importancia de las conexiones y de la variable.

Podemos ver que hay una variable que despunta mucho de las demás.

Esta es la variable “duration”, que posee mucha fuerza en las conexiones de las neuronas.

Otras variables con una fuerte conexión son emp.var.rate y euribor3m.

Las variables que menos influyen son las columnas que hemos eliminado del dataset anteriormente que son marital, education, contact, poutcome, day\_of\_week, month.