The structure of the report to write is the following:

1. Cover page. Include a cover page with title, authors, email, course, and date.

2. Introduction. Explain the problem to solve and the datasets.

3. Design process. Describe the process you followed to get the final results, showing the

intermediate network architectures and the rest of the hyperparameters used. Explain your

design decisions, justifying why you tested each neural model. Show the performance of each

intermediate model.

4. Final results. Describe the ultimate neural network, clearly showing all the hyperparameters

used. Display how the accuracy changes during the training process of this model. Calculate and

analyze the confusion matrix for the final test set.

5. Conclusions. Summarize your work and the most relevant results.

Introduction

En este apartado se definirá el problema propuesto por los profesores para su resolución por parte de los alumnos matriculados en la asignatura de *Artificial neural networks and deep learning* del máster oficial de Inteligencia Artificial impartido en la Universidad Politécnica de Madrid.

Además, se realizará una breve explicación de los datos proporcionados para la resolución de la práctica.

## El problema

Esta actividad tiene como objetivo seguir el proceso de construcción de una red neuronal artificial profunda para un problema de clasificación utilizando un conjunto de datos de reservas de hotel. El objetivo es predecir si un cliente cumplirá con la reserva del hotel o la cancelará en función de su perfil teniendo en cuenta atributos como si es huésped habitual, número de cancelaciones anteriores, etc. y las características de la reserva como podrían ser número de niños, si se solicita espacio de estacionamiento para vehículos, etc.

Para la realización del problema, se pide preparar el entorno de trabajo para Python 3 con Tensorflow 2 y Keras, además, se permite la elección de cualquier entorno de desarrollo, aunque se recomienda Google Colab®.

Para ayudar a los alumnos, se ha proporcionado por parte de los profesores el notebook de los ejercicios de clase *(1. One hidden layer.ipynb)*. Este notebook cuenta con las características principales del código necesario para la creación, entrenamiento y evaluación de una red neuronal. Sin embargo, esto no será suficiente para poder trabajar con ello puesto que se deberán realizar los cambios necesarios para poder ajustar el código al problema propuesto finalizando así la fase de preparación del entorno de trabajo.

Por otra parte, se les han proporcionado a los alumnos un archivo *.csv* con los datos en bruto (*HotelReservationsRawDataset.csv*) además de un notebook para la limpieza y la preparación de los mismos (*PreparingHotelReservationsDataset.ipynb*). Se deberá abrir dicho notebook y modificar las rutas correspondientes para que tome el archivo de los datos en bruto, posteriormente, ejecutar el notebook para limpiar y preparar los datos. De esta forma se obtendrán los archivos resultantes para alimentar los modelos neuronales.

Se espera que el resultado de este proceso sean dos archivos CSV:

* *HotelReservationsPreparedCleanAttributes.csv*

Contiene las instancias preparadas y limpias con los atributos (predictores).

* *HotelReservationsOutput.csv.*

Incluye la clase objetivo, etiquetada como 0 (cancelada) y 1 (no cancelada).

Posteriormente, se deberá construir una red neuronal profunda. Como se ha indicado anteriormente, será necesario modificar el notebook proporcionado *(1. One hidden layer.ipynb)* de forma que:

1. Implemente el proceso de carga de datos de los dos archivos .csv: atributos y clases.
2. Divida el conjunto de datos en tres particiones: 80% para entrenamiento, 10% para pruebas de desarrollo y el 10% restante para pruebas finales.
3. Modificar el código de forma que se adapte al problema planteado.
4. Seguir el proceso de construcción de redes neuronales profundas para encontrar la arquitectura neuronal y otros hiperparámetros que consigan los mejores resultados tanto en rendimiento como en precisión de clasificación.

## Los datos

El conjunto de datos proporcionado se ha sacado del artículo: *Antonio, Nuno, et al. "Hotel booking demand datasets.”* Dicho artículo describe un conjunto de datos de demanda hotelera. Este conjunto de datos cuenta con 31 variables que describen las 119,390 instancias. Cada observación representa una reserva de hotel incluyendo reservas que llegaron a efectuarse y reservas que fueron canceladas. Dado que estos son datos reales de hoteles, todos los elementos de datos relacionados con la identificación del hotel o del cliente fueron eliminados.

Para esta práctica se ha proporcionado un subconjunto del dataset previamente descrito, dicho subconjunto de datos sin procesar cuenta con 36,275 instancias y 19 variables. Cada observación se refiere a una reserva de hotel programada para llegar entre 2017 y 2018. Hay reservas que llegaron a realizarse y reservas que fueron canceladas.

A continuación, se muestra la naturaleza de las variables:

| Variable | Tipo | Rango |
| --- | --- | --- |
| Booking\_ID | Categórica | [INN00001 - INN36275] |
| no\_of\_adults | Numérica | [0 - 4] |
| no\_of\_children | Numérica | [0 - 10] |
| no\_of\_weekend\_nights | Numérica | [0 - 7] |
| no\_of\_week\_nights | Numérica | [0 - 17] |
| type\_of\_meal\_plan | Categórica | [Not Selected – Meal Plan 3] |
| required\_car\_parking\_space | Numérica | [0 - 1] |
| room\_type\_reserved | Categórica | [Room Type 1 – Room Type 7] |
| lead\_time | Numérica | [0 - 443] |
| arrival\_year | Numérica | [2017 -2018] |
| arrival\_month | Numérica | [1 - 12] |
| arrival\_date | Numérica | [1 – 31] |
| market\_segment\_type | Categórica | [Online – Offline – Corporate – Complementary - Aviation] |
| repeated\_guest | Numérica | [1 - 0] |
| no\_of\_previous\_cancellations | Numérica | [0 - 13] |
| no\_of\_previous\_bookings\_not\_canceled | Numérica | [0 - 58] |
| avg\_price\_per\_room | Numérica | [0 - 540] |
| no\_of\_special\_requests | Numérica | [0 - 5] |
| booking\_status | Categórica | [Canceled – Not Canceled] |

Para la limpieza y preparación de los datos se realizan los siguientes pasos:

1. **Comprobación de ausencia de valores**

Comprobar si a alguna instancia de datos le faltan valores o no. En caso de que las haya, se eliminarían puesto que, por lo general, los algoritmos de aprendizaje automático, incluidas las redes neuronales, no pueden manejar datos que contienen valores faltantes. La mayoría de los algoritmos requieren datos completos para poder realizar operaciones matemáticas y aprender patrones de manera efectiva.

1. **Revisión de variables categóricas**

Para cada atributo categórico, comprobamos cuantas instancias por etiqueta hay. Esto puede ser beneficioso dado que dependiendo de la situación se podrán hacer más manejables los datos.

* 1. **Booking\_ID:** Esta variable se utiliza para identificar o categorizar los distintos elementos, podrían considerarse códigos alfanuméricos o identificadores únicos. De momento no tendremos en cuenta esta variable.
  2. **type\_of\_meal\_plan:** De esta variable se muestran los siguientes resultados:

| Etiqueta | Nº de instancias |
| --- | --- |
| Not Selected | 5130 |
| Meal Plan 1 | 27835 |
| Meal Plan 2 | 3305 |
| Meal Plan 3 | 5 |

Debido a que solo figuran 5 instancias en todo el conjunto con la etiqueta de ***Meal Plan 3,*** se procede a eliminar dichas tuplas dado que, si un valor específico en una variable categórica tiene muy pocas instancias, el modelo puede tener dificultades para generalizar y aprender patrones asociados con ese valor. La falta de representación puede llevar a que el modelo no sea capaz de hacer predicciones precisas para ese valor en situaciones futuras.

* 1. **room\_type\_reserved:** De esta variable se muestran los siguientes resultados:

| Etiqueta | Nº de instancias |
| --- | --- |
| Room Type 1 | 28129 |
| Room Type 2 | 692 |
| Room Type 3 | 7 |
| Room Type 4 | 6056 |
| Room Type 5 | 265 |
| Room Type 6 | 966 |
| Room Type 7 | 155 |

Debido a que los tipos 2, 3, 5, 6 y 7 no se repiten tanto en comparación con las otras etiquetas, se decide agruparlas en una etiqueta aparte denominada *Others.* Esto se debe a que, si hay muchas categorías en una variable categórica y algunas de ellas tienen muy pocas instancias, puede haber demasiada complejidad en el modelo. La reducción de la cantidad de categorías ayuda a simplificar el modelo y mejora la capacidad de generalización al proporcionar patrones más robustos. Esto puede ayudar a prevenir sobreajuste a categorías específicas con muy pocas instancias.

El resultado sería el siguiente:

| Etiqueta | Nº de instancias |
| --- | --- |
| Room Type 1 | 28129 |
| Room Type 4 | 6056 |
| Other | 2085 |

Todas aquellas instancias que tuvieran cualquiera de los valores eliminados en la variable **room\_type\_reserved,** se sustituye por el valor *“Other”.* De esta forma también se nivela el número de instancias de cada etiqueta.

* 1. **market\_segment\_type:** De esta variable se muestran los siguientes resultados:

| Etiqueta | Nº de instancias |
| --- | --- |
| Online | 23214 |
| Offline | 10527 |
| Corporate | 2017 |
| Complementary | 387 |
| Aviation | 125 |

Siguiendo el mismo razonamiento en con la variable anterior, se hace una agrupación de las etiquetas minoritarias, dado como resultado lo siguiente:

| Etiqueta | Nº de instancias |
| --- | --- |
| Online | 23214 |
| Offline | 10527 |
| Other | 2529 |

1. **Eliminación de variables**

En ocasiones, algunas variables del problema no aportan información relevante para su resolución, es el caso de las siguientes variables:

* 1. **Booking\_ID:** Dado que únicamente se utiliza para identificar o categorizar las distintas instancias, no presenta ninguna información relevante para la categorización del problema, por tanto, se elimina dicha variable del conjunto de datos.
  2. **arrival\_year:** De forma similar a lo que pasaba con el atributo *Booking\_ID*, esta no presenta ninguna información relevante ya que solo toma dos posibles valores y resulta prescindible al contar ya con las variables *arrival\_month* y *arrival\_date.*

1. **Codificación de atributos discretos**

Este proceso consiste en cambiar los valores de una variable categórica a valores numéricos. El resultado queda de la siguiente manera:

| Valor Codificado | type\_of\_meal\_plan | room\_type\_reserved | market\_segment\_type |
| --- | --- | --- | --- |
| 0 | Not Selected | Room Type 1 | Online |
| 1 | Meal Plan 1 | Room Type 4 | Offline |
| 2 | Meal Plan 2 | Other | Other |

Esto es necesario porque los algoritmos de redes de neuronas requieren que las entradas sean numéricas ya que utilizan operaciones matemáticas y estadísticas en sus cálculos, y trabajar con variables numéricas facilita estos procesos.

1. **Extracción de la variable objetivo**

La variable objetivo debe extraerse del conjunto de datos dado que sus valores se usarán como los valores esperados en la salida de la red permitiendo luego el proceso de aprendizaje.

Además, debido a que se trata de una variable categórica, se le debará aplicar el proceso de codificación, quedando de la siguiente manera:

| Valor Codificado | booking\_status |
| --- | --- |
| 0 | Canceled |
| 1 | Not Canceled |

1. **Normalización del conjunto de datos de entrada.**

Se usa la normalización *Max-Min Scaling*, este es el proceso de reescalar los valores de las variables para que estén en un rango específico, como -1 a 1 o 0 a 1. En el caso de *Max-Min Scaling,* la fórmula para normalizar una variable es:

Esto facilita el entrenamiento de las redes neuronales.

1. **Visualización de datos**

Tras haber realizado todas estas operaciones, se pueden sacar ciertas métricas para entender la naturaleza de los datos.

* 1. **Matriz de correlación**

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

La matriz de correlación permite observar las relaciones entre pares de atributos: valores cercanos a -1 o +1 indican una fuerte correlación. Una tasa de correlación negativa indica que cuando el valor de un atributo aumenta, el valor del otro atributo disminuye, y viceversa. Por otro lado, valores de correlación positiva sugieren que ambas características aumentan o disminuyen simultáneamente.

Todos los atributos muestran una correlación bastante baja. Lo que más nos interesaría de esta información es la correlación entre los atributos y la variable a predecir *booking\_status* dado que una correlación muy baja nos indicaría que esa variable no es significativa para la predicción de *booking\_status.*

Se puede ver como la variable *room\_type\_reserved* guarda la menor correlación con nuestra variable a predecir, siendo esta del orden de 0.003 cuando el resto de las variables guardan una correlación con *booking\_status* de en torno a 10 veces mayor. Por contraparte, la variable con la que guarda mayor correlación es *lead\_time* siendo esta del orden de 0.438.

Esta información nos permite saber qué atributos son más o menos aptos para la predicción de la variable *booking\_status.*

* 1. **Estadísticas descriptivas de los atributos**

Tabla

Descripción generada automáticamente

Aquí se muestra información valiosa sobre la distribución y la variabilidad de los datos. Para empezar, se puede advertir que los valores de todas las variables están comprendidos entre -1 y 1 debido a la normalización descrita anteriormente.

Por otra parte, la media de los atributos indica el punto central de la distribución de los datos. Si la distribución es simétrica, la media es un indicador robusto de la tendencia central. Sin embargo, la presencia de valores extremos (outliers) puede afectar la interpretación de la media.

Valores altos o bajos en la media indica que los valores tienden a estar en la parte superior o en la parte inferior, respectivamente, del rango posible. Esto puede sugerir la presencia de valores extremos o sesgos hacia valores más grandes o hacia valores más pequeños respectivamente.

De igual manera, conocer la desviación estándar permite saber la dispersión o variabilidad de los datos con respecto a la media. Valores más altos indican mayor variabilidad, mientras que valores más bajos indican menor variabilidad.

Design process

Primeros pasos

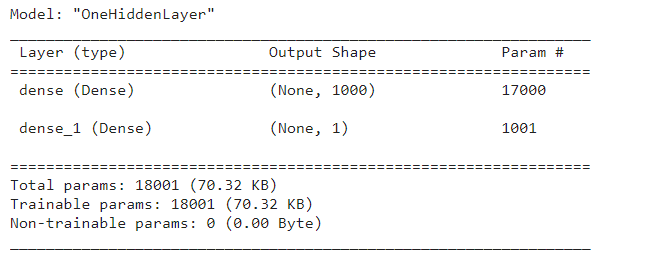
Se ha obtenido el notebook de los ejercicios de clase *(1. One hidden layer.ipynb)* para comenzar a trabajar sobre él y ajustarlo a nuestro problema.

Los cambios a realizar son los siguientes:

* Modificar las rutas en consecuencia para apuntar los archivos csv obtenidos tras ejecutar el notebook
  + *PreparingHotelReservationsDataset.ipynb*
  + *HotelReservationsPreparedCleanAttributes*
  + *HotelReservationsOutput*

***METER AQUÍ LO DE DIVISIÓN TRAIN-TEST ETC.***

* Cambiar la función de activación. Se tiene en cuenta que la función de activación *tanh* se elige para la capa oculta debido a que los valores de nuestros datos están comprendidos en el rango (-1,1). Por otra parte, se usa *sigmoid* para la capa de salida porque esta función alcanza sus límites en 0 y 1, esto nos conviene porque la salida esperada de la red debe ser 0 o 1 indicando si se cancela la reserva del hotel o no se cancela respectivamente.
* Cambiar la función de pérdida. Se usará la entropía cruzada ya que nuestro problema es de clasificación y más específicamente de clasificación binaria. Esta función mide la discrepancia entre las predicciones y las etiquetas reales en un problema de clasificación.
* Cambiar la métrica. Se usará la precisión binaria dado que calcula la frecuencia con la que las predicciones coinciden con las etiquetas binarias.
* Modificar el DataFrame para visualizar la matriz de confusión dado que en los ejercicios de clase se tenían tres etiquetas y en nuestro problema solo se tiene una con dos posibles valores.



Una vez realizados estos cambios, se puede proceder con las pruebas.

Como punto de partida, hemos ejecutado el notebook sin hacer ningún cambio en la arquitectura de la red ni en los hiperparámetros, quedando de la siguiente forma:

**Hiperparámetros:**

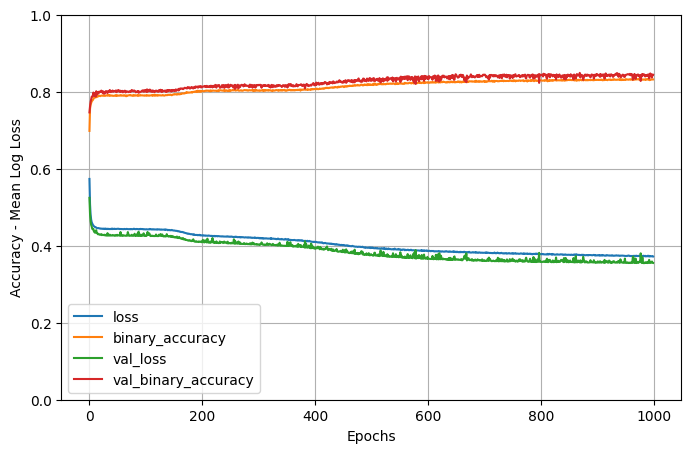
* n\_epochs = 1000
* lr = 0.1
* batch\_size = 512
* n\_hidden = 1000

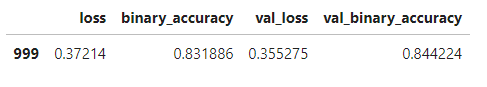
**Optimizador**

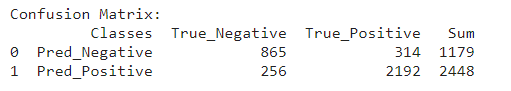
* optimizer=tf.keras.optimizers.SGD(learning\_rate=lr)

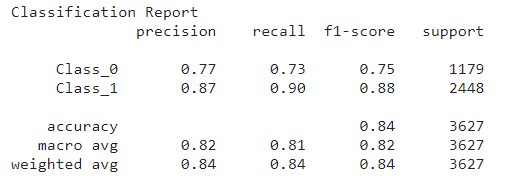
**Resultados:**

****





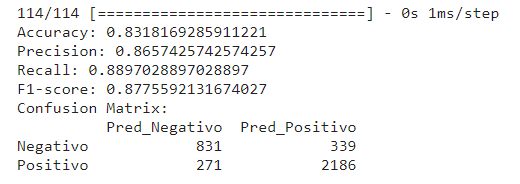




**CON EL TEST**

Esto no debería hacerse hasta no tener el modelo definitivo para no “sesgarnos” pero siempre es bueno mirar como lo vamos haciendo de bien o mal en el test.

Tenemos que mantener en mente no fijarnos en estos resultados a la hora de modificar el modelo, sino fijarnos en los resultados de validación.



**Observaciones:**

* Para la complejidad del problema (clasificación binaria) se tiene un gran número de parámetros entrenables (18001), esto se debe a la cantidad de neuronas de la capa oculta.
* La gráfica muestra cómo tanto la pérdida como la precisión baja o sube (respectivamente) y se estabilizan rápidamente sin que se produzca *overfitting.*
* Mirando la matriz de confusión y apoyándonos en el reporte de clasificación, podemos observar como el modelo clasifica correctamente la gran mayoría de ejemplos de la clase 1, llegando a un desempeño del 90% mientras que de la clase 0 clasifica correctamente el 73% de los ejemplos de esta clase. Teniendo en cuenta que no se ha producido *overfitting* podemos deducir que nuestro modelo tiene un valor más elevado de *Bias* que de *Variance*

**Propuestas de mejora:**

* Reducir *Bias* incrementando la complejidad del modelo y modificando la arquitectura. Teniendo en mente la rápida estabilización de pérdida y precisión que vimos en el gráfico, se podría abordar este problema incrementando el número de capas ocultas y disminuyendo la cantidad de neuronas por capas, de esta forma se tendrán menos parámetros entrenables. De igual manera, se puede disminuir el número de épocas de entrenamiento. Con esto se pretende disminuir el costo computacional y el tiempo de entrenamiento intentando mejorar o en el peor de los casos mantener el desempeño del modelo con respecto a la clase peor clasificada.
* El conjunto de datos que se tiene se compone de 24386 ejemplos de la clase 1 y 11884 ejemplos de la clase 0 esto podría ser una de las razones por las que el modelo tienen un mejor ratio de acierto clasificando la clase 1 dado que dispone de más ejemplos de este tipo para aprender. Para intentar mejorar el *recall* de la clase 0 se plantean los siguientes métodos:
  + Usar un ajuste de pesos asignando un peso más alto a la clase 0 durante el entrenamiento para que el modelo se enfoque más en clasificar correctamente esta clase.
  + Usar técnicas de oversampling (crear copias de instancias de la clase menos representada) o undersampling (eliminar instancias de la clase más representada) para equilibrar la proporción de ejemplos entre las clases. Esto puede ayudar al modelo a aprender patrones en la clase menos representada.

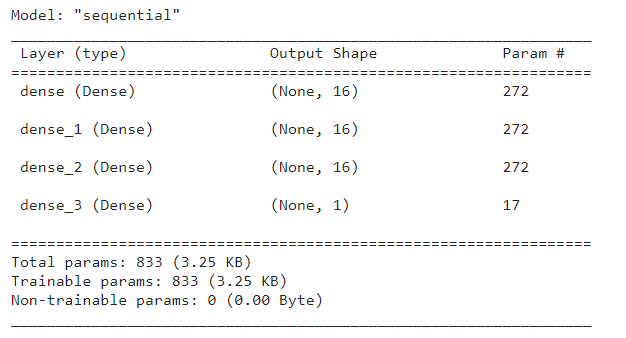
Mejorando el Modelo

A continuación, se comenzarán a abordar las propuestas obtenidas a partir de las conclusiones sacadas a partir de las pruebas realizadas.

Prueba 1

Se comenzará abordando la propuesta de reducir el *Bias* incrementando la complejidad del modelo y modificando la arquitectura del modelo aprovechando para tratar de disminuir la complejidad computacional y el tiempo de entrenamiento.

**Modelo usado**:



**Hiperparámetros**:

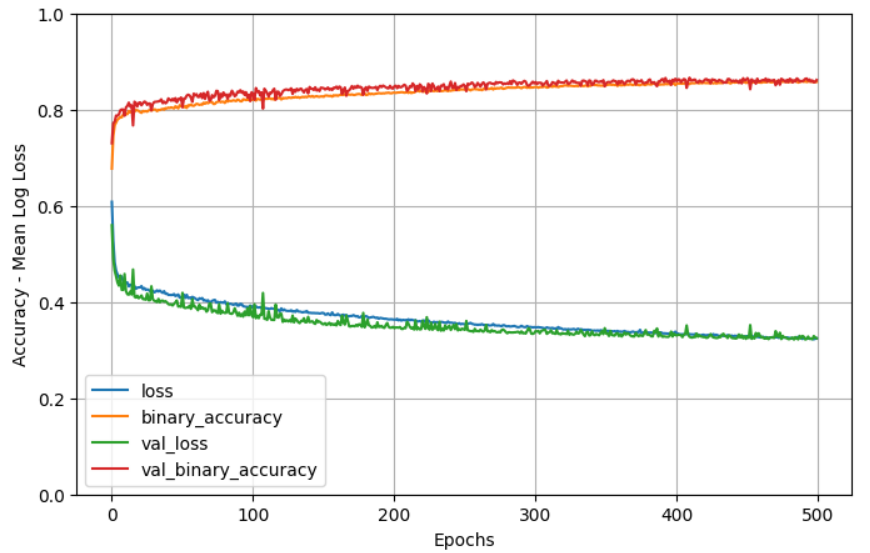
* n\_epochs = 500
* lr = 0.1
* batch\_size = 512
* n\_hidden = 16

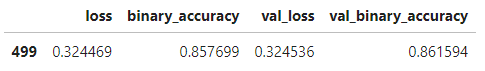
**Optimizador**

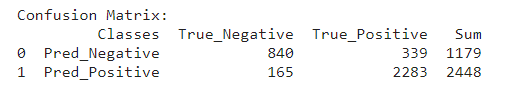
* optimizer=tf.keras.optimizers.SGD(learning\_rate=lr)

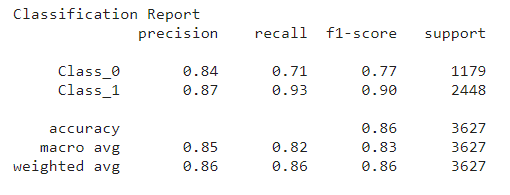
**Resultados:**











**Observaciones:**

* Mejora en el rendimiento del modelo dado por la disminución de parámetros entrenables de 18001 a 833 obteniendo:
  + Disminución del tiempo de entrenamiento de 5:22 min a 1:20 min
  + Un mayor porcentaje de precisión de 0.831 a 0.857
* Otro objetivo a conseguir era conseguir un mejor ratio de clasificación en la clase 0, sin embargo, observando la matriz de confusión y el reporte de clasificación, podemos advertir una disminución del 2% clasificando ejemplos de la clase 0 y un aumento del 3% clasificando ejemplos de la clase 1 lo que explica un mayor porcentaje de precisión en la totalidad del problema.

**Propuestas de mejora:**

* Dado que no se produjo ningún otro cambio significativo en los resultados más allá de los observados, se mantiene la propuesta de mejorar el *recall* de la clase 0 con métodos como el ajuste de pesos o el oversampling / undersampling.

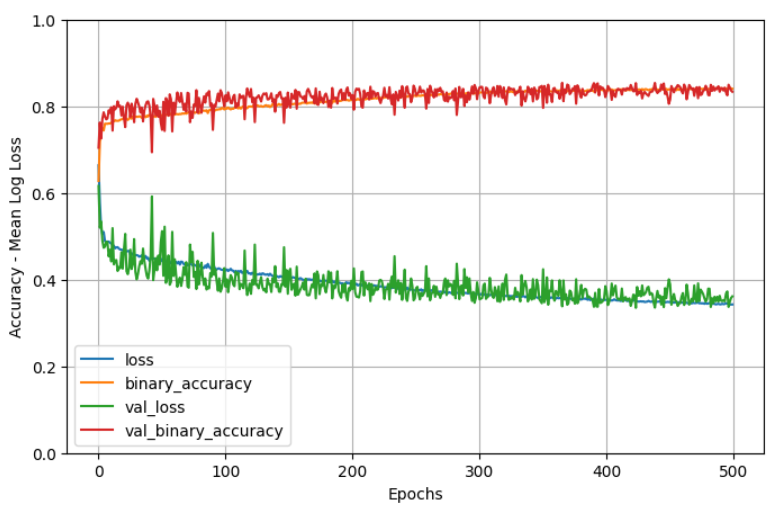
Prueba 2

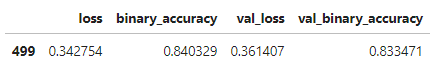
Con el fin de intentar evitar el posible sobreajuste que podría darse al generar copias adicionales de instancias de la clase menos representada y perder capacidad de generalización, se ha optado por utilizar la función *compute\_class\_weight* de scikit-learn para calcular automáticamente los pesos de clase inversamente proporcionales a las frecuencias de las clases en los datos de entrenamiento. Luego, convertir estos pesos de clase en un diccionario para que puedan ser utilizados en Keras. El objetivo de esto es aumentar el *binary\_accuracy* clasificando correctamente una mayor cantidad de instancias de la clase minoritaria.

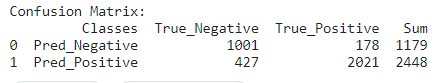
Se mantendrá la misma arquitectura, hiperparámetros y optimizador.

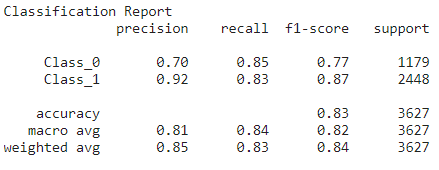
**Resultados:**











**Observaciones:**

Se ha conseguido aumentar considerablemente la cantidad de instancias de la clase 0 (clase minoritaria) correctamente clasificadas de 0.71 a 0.85, sin embargo, la cantidad de instancias correctamente clasificadas de la clase 1 ha disminuido bastante, de 0.93 a 0.83. Aunque la magnitud de decrecimiento en el porcentaje de instancias correctamente clasificados de la clase 1 es menor que la magnitud de aumento en el porcentaje de instancias correctamente clasificadas de la clase 0, el porcentaje de *binary accuracy* ha disminuido, esto se debe a la diferencia en la cantidad de instancias entre las dos clases. Esto nos lleva a pensar que si los datos ya están sesgados, quizá sea interesante no aplicar *class weight* de manera que el modelo aprenda ese sesgo.

**Propuestas de mejora:**

* Continuar abordando el problema con *class weight* teniendo en mente la posibilidad de descartar en futuras pruebas.
* Debido a que anteriormente se mejoraron los resultados añadiendo más complejidad a la red, se propone seguir en esta línea e intentar mejorar el algoritmo de optimización.

Prueba 3

Se mantiene *class weight*. Por otra parte, con el fin de incrementar el *binary accuracy* se incrementará la complejidad de la red y se tratará de mejorar el algoritmo de optimización.

A continuación, se realizarán varias pruebas ajustando la complejidad de la red y variando el algoritmo de optimización.

**Prueba 3.1**

Modificación del algoritmo de optimización: se le añade momento y la variación Nest

erov del SGD.

*optimizer=tf.keras.optimizers.SGD(learning\_rate=0.1,*

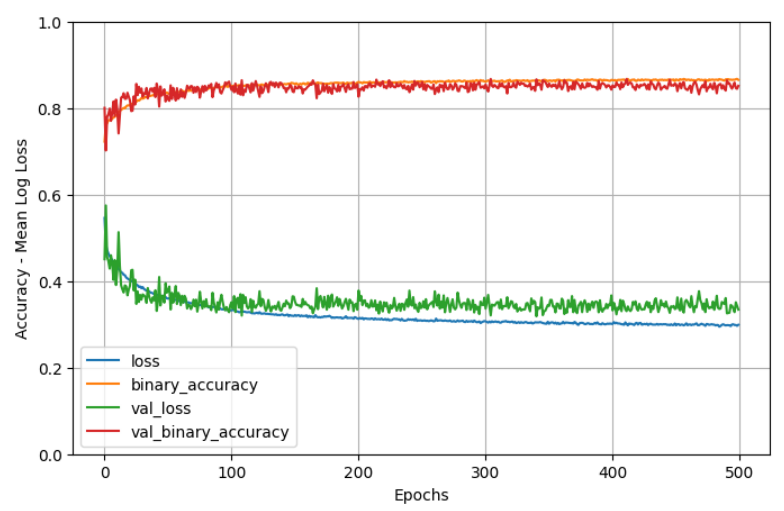
*momentum=0.9,*

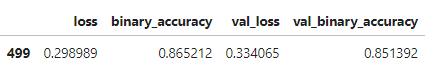
*nesterov=True,*

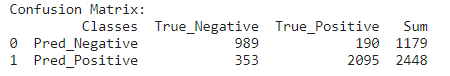
*weight\_decay=None)*

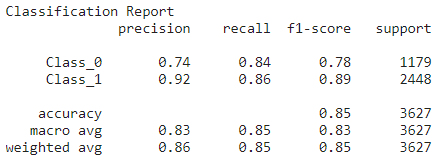
**Resultados:**

****

****

****

****

****

**Observaciones:**

Reducción del loss y aumento del binary accuracy, vamos a tratar de mejorarlo disminuyendo el learning rate, y para que pueda converger, aumentaremos el número de epochs.

**Prueba 3.2**

Epochs: 800

*optimizer=tf.keras.optimizers.SGD(learning\_rate=0.01,*

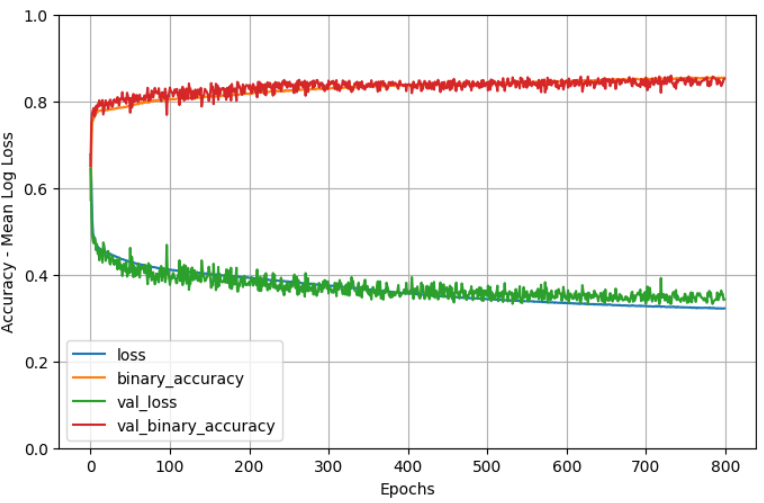
*momentum=0.9,*

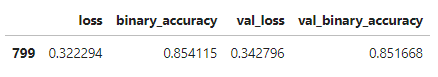
*nesterov=True,*

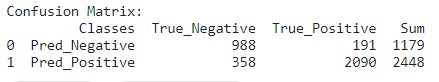
*weight\_decay=None)*

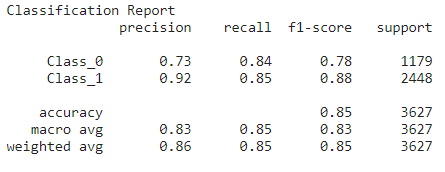
**Resultados:**

****

****

****

****

****

**Observaciones:**

Como era de esperar, el tiempo de ejecución ha sido un poco mayor dado que se incrementaron el número de épocas, sin embargo, no se ha llegado a alcanzar una pérdida menor ni un accuracy mayor con este learning rate sino que los resultados han sido ligeramente peores.

Para tratar de mejorar los resultados podríamos aumentar aún más el número de épocas para tratar de alcanzar mejores resultados con un learning rate menor o podemos tratar de aplicar decaimiento de la tasa de aprendizaje.

**Prueba 3.3.**

Para no seguir aumentando el tiempo de entrenamiento con un mayor número de épocas, se hará una mezcla de las anteriores dos pruebas. Se replicará la prueba 3.1 y a partir de la época 500, se aplicará el decaimiento de la tasa de aprendizaje hasta la época 800.

Añadimos el decaimiento de la tasa de aprendizaje.

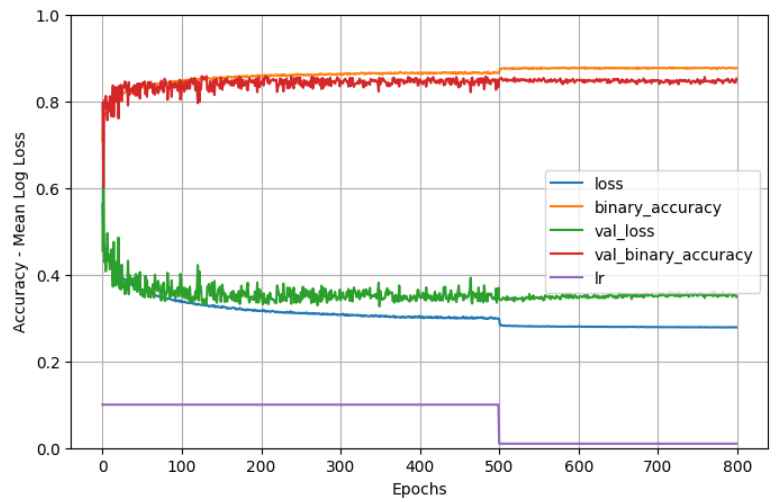
lr = 0.1

decay\_epochs = 500

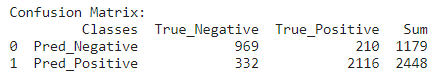
decay\_factor = 0.1

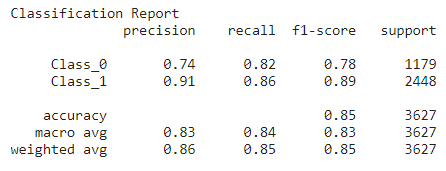
**Resultados:**

****

****

****

****

****

**Observaciones:**

En el gráfico se ve claramente el cambio de tasa de aprendizaje disminuyendo la pérdida y aumentando el binary accuracy, sin embargo, la pérdida en validación y el binary accuracy en validación no tienen grandes mejoras, de hecho, en el diagrama no puede apreciarse. Hay que referirse a las tablas para ver un cambio que, aunque pequeño, se puede apreciar una leve mejora en tanto en binary accuracy como en val binary accuracy.

**Prueba 3.4.**

Se tratará de ajustar más el decaimiento de la tasa de aprendizaje realizando mas disminuciones en ella buscando disminuir el val\_loss y aumentar el val\_binary\_accuracy. Por otro lado, dado que el tiempo de ejecución ha ido aumentando debido a que se ha aumentado el número de épocas, trataremos de disminuirlo modificando la arquitectura de la red aumentando las capas y disminuyendo el numero de neuronas por capas. Con esto se busca tener un numero de parámetros entrenables menor y darle más complejidad a la red para tratar de que se ajuste mejor a los datos.

Nuevo modelo:

Tabla

Descripción generada automáticamente con confianza baja

**Hiperparámetros:**

* n\_epochs = 800
* lr = 0.1
* batch\_size = 512
* n\_hidden = 8

La tasa de aprendizaje disminuye a 0.05 a partir de la epoca 400 y luego vuelve a disminuir a 0.01 a partir de la época 600.

**Resultados:**



Gráfico

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Tabla

Descripción generada automáticamente con confianza media

**Observaciones:**

Se puede observar como efectivamente se ha conseguido disminuir el tiempo de entrenamiento, tal y como se esperaba. Por otra parte, fijándose en el gráfico, se aprecia como los valores de validación se aproximan a los de entrenamiento los cuales siempre mejoran a diferencia de la prueba anterior en la que llegados a un punto los valores de validación se estabilizaban mientras los de entrenamiento seguían mejorando.

Además, fijándose en los valores, se puede ver que, Esaunque la precisión en entrenamiento (binary\_accuracy) haya empeorado un poco, lo que demuestra que puede mejorar, la precisión en entrenamiento (val\_binary\_accuracy) ha mejorado, el cual era nuestro objetivo.

Prueba 4

Esta prueba esta enfocada a comprobar si era acertada la suposición hecha anteriormente, en la cual se indicaba que, si los datos ya están sesgados, quizá sea interesante no aplicar class weight de manera que el modelo aprenda ese sesgo.

De manera que se replicará la variante de la prueba anterior con mejor resultado, esta vez sin class weight.

Esto significa que se repetirá la **prueba 3.4,** la cual obtuvo un **val\_binary\_accuracy de 0.854701** esta vez **sin class weight.**

**Resultados:**



Gráfico

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Tabla

Descripción generada automáticamente con confianza media

**Observaciones:**

Tanto *binary\_accuracy* como *val\_binary\_accuracy* han obtenido un mejor resultado que cualquiera de las variantes de la prueba 3 donde se usaba *class weight,* donde mejor se puede ver el cambio es en el *recall* de ambas clases dónde la diferencia entre ambas se ha hecho más que notable lo que demuestra que la suposición realizada era correcta. Debido a que la clase mayoritaria es *Not\_Canceled* (valor 1), conseguir un mayor número de aciertos clasificando correctamente esta clase a pesar de disminuir un poco el número de correctamente clasificados de la clase contraria, hace que la precisión global del sistema aumente.

Prueba 5

Tras haber obtenido el mejor resultado hasta el momento en la prueba anterior, se tratará de mantener el mismo resultado o mejorarlo variando el batch size y disminuyendo el número de épocas para mejorar el tiempo de ejecución, esto implica hacer modificaciones en el decaimiento de la tasa de aprendizaje.

**Prueba 5.1**

Disminución del número de épocas y, por tanto, ajuste del decaimiento de la tasa de aprendizaje. Además, se disminuirá el tamaño del batch.

**Hiperparámetros:**

* n\_epochs = 600
* lr = 0.1
* batch\_size = 256
* n\_hidden = 8

La tasa de aprendizaje disminuye a 0.05 a partir de la época 300 y luego vuelve a disminuir a 0.01 a partir de la época 500.

**Resultados:**



Gráfico

Descripción generada automáticamente

Tabla

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamente

Tabla

Descripción generada automáticamente con confianza media

**Observaciones:**

Uno de los objetivos buscados era disminuir el tiempo de ejecución, sin embargo, este ha aumentado debido a la disminución del tamaño del batch, incluso habiendo disminuido considerablemente el número de épocas. Esta observación cobra sentido ya que al disminuir el tamaño del batch se deben hacer más actualizaciones de los pesos durante el entrenamiento.

Por esta misma razón se podría llegar a pensar que la precisión del modelo mejora dado que se hace un mayor número de ajustes en la red, sin embargo, el valor de val\_binary\_accuracy es muy parecido al obtenido en la prueba anterior e incluso ha empeorado un poco.

**Prueba 5.2**

En esta ocasión se probará a aumentar el tamaño del batch manteniendo el número de épocas.

**Hiperparámetros**

* n\_epochs = 600
* lr = 0.1
* batch\_size = 1024
* n\_hidden = 8

**Resultados:**



Gráfico

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

**Observaciones:**

Como era de esperar el tiempo de ejecución ha disminuido de forma considerable debido al tamaño del batch, que, al haber aumentado, el modelo actualiza los pesos un menor número de veces.

Por otro lado, el valor de *val\_binary\_accuracy* ha mejorado alcanzando casi el 87%, siendo este el mejor resultado hasta el momento.

**Prueba 5.3**

Al haber mejorado el resultado aumentando el tamaño del batch, se va a probar a seguir aumentándolo, sin embargo, aumentarlo más sin aumentar el número de épocas podría empezar a empeorar el resultado esto se debe a que el modelo seguirá disminuyendo el número de veces que actualiza sus pesos lo que significa que no se ajustará a los datos empeorando así su precisión, por tanto, para compensar esto, se aumentará el número de épocas, lo que requerirá modificar el decaimiento de la tasa de aprendizaje para que mantenga un ratio de disminución parecido en relación al número de épocas.

**Hiperparámetros**

* n\_epochs = 900
* lr = 0.1
* batch\_size = 2048
* n\_hidden = 8

La tasa de aprendizaje disminuye a 0.05 a partir de la época 500 y luego vuelve a disminuir a 0.01 a partir de la época 700.

**Resultados:**



Gráfico

Descripción generada automáticamente

Imagen que contiene Tabla

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza media

Tabla

Descripción generada automáticamente con confianza media

**Observaciones:**

El tiempo de ejecución se ha mantenido bastante parecido en comparación a la prueba anterior lo que resulta lógico ya que aunque se ha incrementado el tamaño del batch (lo que bajaría el tiempo de ejecución), se ha aumentado el número de épocas (lo que aumenta el tiempo de ejecución).

Se puede observar en el gráfico como los valores de validación se mantienen junto con los de entrenamiento lo que significa que no se produce sobreajuste/sobre entrenamiento.

Por último, fijándose en el valor de *val\_binary\_accuracy* se puede ver como se ha conseguido mejorar el desempeño del modelo esta vez superando el 87% de precisión.

Final Results

Conclusions

Una buena forma de mejorar el modelo es incrementando el número de capas y disminuyendo el número de neuronas por capa. De esta forma se aumenta la complejidad del modelo, lo que le permite adaptarse mejor a los datos y disminuye el número de parámetros entrenables lo cual repercute en un menor tiempo de ejecución durante el entrenamiento.

Encontrar un equilibrio entre el tamaño del batch y el número de épocas resulta muy conveniente para la resolución del problema ya que uno aumenta el tiempo de ejecución y el otro lo disminuye, sin embargo, encontrar un equilibrio puede dar como resultado una mayor precisión del modelo y un tiempo de ejecución pequeño.

Debido al desbalanceo entre las clases predictoras, usar class weight podría ser un buen método para incrementar el desempeño del modelo, sin embargo, se ha comprobado que en este caso no es así dado que el conjunto de datos ya viene sesgado y por tanto conviene hacer que el modelo aprenda ese sesgo.

Una posible aproximación, podría ser el aumentar la cantidad de instancias de la clase minoritaria para, de esta forma, conseguir que el modelo aprenda a clasificar mejor este tipo de instancias, sin embargo, estaríamos perdiendo fiabilidad con respecto a la realidad dado que los datos están sacados de hoteles reales y estos ya presentan dicho sesgo.

Otra posible aproximación al problema que no interferiría con la realidad sería eliminar algunas de las variables. Se ha observado en la matriz de correlación como algunas de estas variables apenas guardan una relación lineal con la clase a predecir lo que dificulta el aprendizaje del modelo. Por ello, una buena solución sería quedarse con aquellas variables que guarden una mayor correlación con la clase a predecir.