Predicción de reservas de hotel mediante DNN

Ciencia de Datos e IA - 3º
Gonzalo Lope Carrasco - g.lope@alumnos.upm.es
Javier Pérez Vargas - javier.perez.vargas@alumnos.upm.es
Mario Ruiz Vaquett - mario.ruiz@alumnos.upm.es

Abril 2024

Índice

1. Introducción

El propósito de esta actividad es desarrollar un modelo de clasificación utilizando un conjunto de datos sobre reservas de hoteles. El objetivo es predecir si el usuario finalmente irá al hotel o si cancelará la reserva, basándonos en su perfil y en las características de la reserva.

El conjunto de datos, tras ser preprocesado y limpiado, contiene 36270 instancias y 16 atributos (Nº de adultos, Nº de niños, Tipo de habitación, Mes de reserva...). Cada una de las instancias guarda información sobre una reserva de hotel hecha entre 2017 y 2018.

Exploraremos el rendimiento de diversas arquitecturas de redes neuronales en la tarea de clasificación de reservas de hoteles. El objetivo principal es determinar qué configuración de red neuronal es más efectiva para predecir si un cliente finalmente asistirá al hotel o cancelará la reserva. Evaluaremos múltiples arquitecturas, variando el número de capas ocultas, el tamaño de las capas, los optimizadores y otros parámetros clave.

2. Proceso de diseño

Inicialmente determinaremos qué arquitecturas funcionan mejor para nuestro conjunto de datos. Posteriormente haremos modificaciones, aplicaremos técnicas de regularización, variaremos los optimizadores, etc... con el objetivo de obtener los parámetros adecuados para tener el menor error posible.

2.1. Tamaño de la red neuronal

En primer lugar tenemos que dar con el tamaño idóneo de la red, equilibrando el tiempo de entrenamiento con la capacidad de aprendizaje (pero sin llegar a tener overfitting). Para ello probamos con 4 arquitecturas:

- **A1**: Una arquitectura pequeña con pocas capas pequeñas → **[20, 20]**
- **A2**: Una arquitectura pequeña con muchas capas pequeñas → **[20, 20, 20, 20, 20, 20]**
- **A3**: Una arquitectura grande con pocas capas grandes → **[1000, 1000]**
- **A4**: Una arquitectura grande con muchas capas grandes → **[1000, 1000, 1000, 1000, 1000]**

En todos los casos se ha usado el optimizador Adam, la función de activación relu, un learning rate de 0.001 aunque en caso de no aprender en los 200 epochs lo hemos cambiado acordemente. Obtuvimos los siguiente resultados.

Arquitectura	Error Entr.	Error Val.	Bias	Variance	Tiempo
A1	14.78	14.75	4.78	-0.03	61.64s
A2	13.81	15.25	3.81	1.44	82.97s
A3	4.90	12.46	-5.10	7.56	69.25s
A4	1.96	12.85	-8.04	10.89	143.11s

Cuadro 1: Resultados con las 4 distintas arquitecturas

Como podemos observar los modelos pequeños ya aprenden bastante bien, pero todavía falta algo de información por aprender, por lo que hay un poco de underfitting. Al contrario, en los modelos grandes el overfitting es evidente, en especial en el segundo modelo que predice de manera casi perfecta todos los datos de entrenamiento. De ahí que el bias sea negativo, pues tiene menos error que el humano. No obstante, la varianza es alta porque no generaliza bien a otros datos.

Así, parece que los modelos A3 y A4 tienen potencial siempre y cuando podamos reducir el overfitting, obteniendo así un modelo con mejor poder de predicción. A continuación probamos con diferentes métodos de reducción del overfitting y otras técnicas.

2.2. Técnica *dropout*

Una vez tenemos las arquitecturas más adecuadas para nuestro conjunto de datos, vamos a probar la técnica de *dropout*, que consiste en desactivar aleatoriamente una fracción de las neuronas durante el entrenamiento, lo que significa que temporalmente se eliminan de la red junto con todas sus conexiones de entrada y salida. Vamos a probar esta técnica en nuestra red, de forma que el modelo tendrá más capacidad de generalización. Probaremos con las dos arquitecturas de gran tamaño y con diferentes porcentajes de *dropout*.

Arquitectura	D. Rate	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	0.1	5.17	12.41	-4.83	7.23	82.64s
A3	0.2	5.94	12.32	-4.06	6.38	71.38s
A3	0.3	6.87	13.18	-3.13	6.31	82.64s
A4	0.1	2.25	12.66	-7.75	10.41	143.09s
A4	0.2	3.02	13.18	-6.98	10.16	105.21s
A4	0.3	5.04	12.77	-4.96	7.73	143.06s
A4	0.4	7.46	12.27	-2.54	4.81	105.00s
A4	0.5	10.00	12.02	-0.00	2.02	104.63s

Cuadro 2: Resultados con la técnica *dropout*

Los porcentajes de *dropout* se han elegido en orden incremental desde 0.1 hasta que no hay más mejora. Como es lógico, los modelos son capaces de generalizar más (y por tanto, aumenta el bias y disminuye la varianza) cuando el dropout rate es más alto, ya que el modelo es menos independiente de conexiones específicas y es más robusto. No obstante, cuando aumentamos en exceso el *dropout rate* el modelo puede olvidar demasiada información útil, perdiendo precisión.

Podemos observar que se obtienen buenos resultados con esta técnica, especialmente con la segunda arquitectura y con un *dropout rate* de 0.5, donde se consigue la misma precisión que la humana, y el error en el conjunto de validación es bastante bajo.

2.3. Regularización

La regularización es capaz de penalizar la complejidad de un modelo, previniendo en sobreajuste. Vamos a probar los diferentes tipos de regularización para determinar, finalmente, cuál es el idóneo en nuestra red neuronal.

2.3.1. Sin regularización

En primer lugar, vamos a ver los resultados sin aplicar regularización.

Arquitectura	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	4.55	13.73	-5.45	9.18	82.95s
A4	1.41	13.48	-8.59	12.07	143.23s

Cuadro 3: Resultados sin aplicar regularización

2.3.2. Regularización L1

La regularización L1 es una técnica de regularización que penaliza los coeficientes de los predictores usando la norma L1. En este primer caso, los resultados son los siguientes:

Arquitectura	λ_{L1}	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	0.001	19.12	20.60	9.12	1.48	82.94s
A3	0.0001	15.49	17.07	5.49	1.57	48.67s
A4	0.001	32.68	33.50	22.68	0.82	143.15s
A4	0.0001	11.51	14.31	1.51	2.80	143.59s

Cuadro 4: Resultados aplicando regularización L1

2.3.3. Regularización L2

La regularización L2 usa la norma L2 en vez de la L1, es decir, penaliza los coeficientes de los predictores proporcionalmente al cuadrado de su magnitud. Los resultados si aplicamos la regularización L2 son los siguientes:

Arquitectura	λ_{L2}	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	0.001	13.68	15.41	3.68	1.74	82.61s
A3	0.0001	10.55	14.64	0.55	4.09	82.68s
A4	0.001	9.80	13.40	-0.20	3.60	142.96s
A4	0.0001	3.86	15.16	-6.14	11.30	85.02s

Cuadro 5: Regularización L2

2.3.4. Regularización L1-L2

Este tipo de regularización combina los efectos de L1 y L2 en un solo modelo, permitiendo la selección de características y la suavización de coeficientes simultáneamente. En este caso, podemos ver que la salida es la siguiente:

Arquitectura	λ_{L1-L2}	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	0.001	19.57	21.37	9.57	1.80	82.65s
A3	0.0001	15.17	16.65	5.17	1.49	49.04s
A4	0.001	32.68	33.50	22.68	0.82	143.09s
A4	0.0001	12.24	14.64	2.24	2.40	92.33s

Cuadro 6: Regularización L1-L2

Podemos observar los efectos del valor de λ en los 3 casos. Cuando es grande, se restringe demasiado el aprendizaje al modelo, viendo claramente un mayor bias. Cuando es pequeño, se le 'permite' aprender más al modelo, obteniendo así menos error, tanto en validación como en entrenamiento. No obstante, valores aún más pequeños de λ darían lugar a resultados similares a los obtenidos sin regularización, y por tanto tendríamos modelos sobreajustados.

Es interesante ver que en los casos de L1 y L1-L2 con un $\lambda = 0{,}001$, la arquitectura A4 (muchas capas grandes), obtiene resultados claramente peores que la arquitectura A3; en cambio, usando un $\lambda = 0{,}0001$ la arquitectura A4 mejora a la A3. Esto se podría explicar porque la regularización L1 tiende a hacer muchos pesos igual a 0 (eliminar características irrelevantes), y dado que en un modelo muy complejo como es A4 existen muchas conexiones que pueden ser poco importantes, entonces esta regularización los penaliza más.

En el caso de regularización L2, la arquitectura A4 es mejor con ambos valores de λ, dando además los mejores resultados. Por tanto, comparando con el resto de tablas, podemos decir que el modelo que usa regularización L2 con muchas capas grandes es el más adecuado.

2.4. Funciones de activación

Las funciones de activación en redes neuronales, como ReLU o sigmoide, introducen no linealidades en las salidas de las neuronas, permitiendo a la red aprender y modelar relaciones complejas entre las

características de entrada y las salidas. Esto facilita la capacidad de la red para aprender y generalizar patrones en datos no lineales. Vamos a ver cómo funcionan las distintas funciones de activación con nuestros datos:

Arquitectura	Función	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	Linear	20.61	21.31	10.61	0.7	34.52
A4	Linear	20.79	21.95	10.79	1.16	72.84
A3	Relu	5.42	13.61	-4.58	8.19	38.78
A4	Relu	1.87	13.76	-8.13	11.89	52.32
A3	Elu	12.86	14.35	2.86	1.49	42.15
A4	Elu	2.39	13.31	-7.60	10.92	53.11
A3	TanH	9.74	14.15	-0.26	4.41	40.58
A4	TanH	1.72	14.20	-8.28	12.48	54.15

Cuadro 7: Resultados con diferentes funciones de activación

Podemos observar que la función lineal no es útil para este conjunto de datos, ya que no permite a la red aprender relaciones complejas entre los datos, dando lugar a errores muy altos. En cambio, las otras tres funciones parecen funcionar mejor, especialmente **relu** y **elu**, aunque en el caso de la arquitectura A4 se sobreajustan, por lo que sería necesario aplicar regularización.

2.5. Inicialización de pesos

Cuando trabajamos con una red neuronal, el objetivo es que los pesos se vayan ajustando para conseguir un mayor poder predictivo. El principal factor que interviene en los pesos es el algoritmo de optimización. No obstante, existen métodos de inicialización de los pesos (normalmente se inician con pesos aleatorios) que pueden dar mejores resultados:

Arquitectura	Inicialización	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	Sin inicialización	4.77	13.62	-5.23	8.85	82.54s
A3	Uniform	5.37	13.48	-4.63	8.11	82.55s
A3	Normal	4.30	13.32	-5.70	9.02	48.37s
A3	He (Normal)	5.55	13.76	-4.45	8.21	46.48s
A3	He (Uniform)	5.34	14.50	-4.66	9.16	82.81s
A4	Sin inicialización	1.38	13.56	-8.62	12.19	73.78s
A4	Uniform	1.43	13.29	-8.57	11.86	74.40s
A4	Normal	1.13	13.54	-8.87	12.41	142.87s
A4	He (Normal)	1.16	13.73	-8.84	12.57	74.22s
A4	He (Uniform)	1.13	12.88	-8.87	11.75	74.71s

Cuadro 8: Resultados con diferentes métodos de inicialización

Después de evaluar los diferentes métodos de inicialización para las arquitecturas A3 y A4, concluimos que los enfoques He (tanto Normal como Uniforme) ofrecen el mejor equilibrio entre rendimiento y eficiencia. Estos métodos demuestran errores de validación más bajos y sesgos reducidos en comparación con las inicializaciones Uniforme y Normal estándar. Aunque los modelos inicializados con He muestran una varianza ligeramente más alta, esta mayor variabilidad es compensada por una mejora significativa en el rendimiento general del modelo.

En este caso, el mejor método de inicialización sería el He Uniforme con una arquitectura A4, ya que tiene el menor sesgo con el menor error de validación en un tiempo acorde.

2.6. Algoritmos de optimización

Los algoritmos de optimización en un modelo tienen la tarea de ajustar iterativamente los pesos de un modelo para minimizar una función de pérdida. Es importante utilizar diferentes algoritmos de optimización, ya que no sabemos cuál es el que mejor funciona con nuestro conjunto de datos.

2.6.1. Descenso de Gradiente

El descenso de gradiente es un algoritmo que va actualizando los parámetros utilizando la dirección del gradiente de la función de pérdida. En este caso, los resultados son...

Arquitectura	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	11.6	13.9	1.6	2.3	120.24s
A4	7.29	12.65	-2.71	5.36	142.43s

Cuadro 9: Resultados con el descenso de gradiente

2.6.2. Disminución de la tasa de aprendizaje

Este método permite que, como el mismo nombre indica, en vez de tener una tasa de aprendizaje constante, la tasa de aprendizaje decrezca durante el entrenamiento.

Arquitectura	Initial LR	Decay Rate	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	0.1	0.96	11.81	14.39	1.81	2.58	67.77s
A3	0.1	0.98	14.95	15.33	4.95	0.38	59.14
A4	0.1	0.96	10.3	13.7	0.3	3.4	88.45s
A4	0.1	0.98	9.87	15.49	-0.13	5.52	92.60

Cuadro 10: Resultados con disminución de la tasa de aprendizaje

2.6.3. Momentum

En este caso, el algoritmo nos ayuda a acelerar la convergencia y evitar quedar atrapado en mínimos locales utilizando utilizando la inercia del paso anterior. Si aplicamos este algoritmo, los resultados que nos salen son los siguientes:

Arquitectura	Tasa Momentum	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	0.9	6.86	13.31	-3.14	6.45	60.17s
A4	0.9	3.04	13.15	-6.96	10.11	92.23s

Cuadro 11: Resultados con momentum

2.6.4. Gradiente Acelerado de Nesterov (NAG)

Este algoritmo es una variante del algoritmo de optimización de descenso de gradiente, que mejora en términos de convergencia más rápida haciendo una estimación más precisa del gradiente. Aplicado a nuestro modelo:

Arquitectura	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	6.38	14.41	-3.62	8.03	61.91s
A4	1.87	13.48	-8.13	11.61	82.77s

Cuadro 12: Resultados con NAG

2.6.5. RMSprop

Este algoritmo adapta la tasa de aprendizaje de manera automática e individualizada para cada parámetro del modelo, basándose en la magnitud de los gradientes recientes asociados con ese parámetro. En este caso los resultados que nos dan son:

Arquitectura	Error Entr.	Error Val.	Bias	Variance	Tiempo
A3	3.45	13.97	-6.55	10.52	81.03s
A4	2.36	13.28	-7.64	10.92	142.53s

Cuadro 13: Resultados con RMSprop

La mayoría de algoritmos utilizados proporcionan resultados similares: un porcentaje de error cercano a 13 en el conjunto de validación, al igual que con Adam, que es el que usamos por defecto en los demás apartados.

2.7. Otras combinaciones

Ya visto el funcionamiento de los distintos algoritmos y técnicas, vamos a probar diferentes combinaciones para determinar qué parámetros son los mejores:

Arq.	F. Act.	L. Rate	F. Opt	Init.	Dropout	L2 Reg.	E. Entr.	E. Val.	Bias	Var.
A3	Relu	0.001	Adam	He(N)	0.4	X	8.6	12.88	-1.40	4.28
A3	Relu	0.1	SGD	He(N)	0.01	X	10.22	13.90	0.22	3.68
A3	Relu	0.001	RMSprop	He(N)	0.4	X	10.37	13.04	0.37	2.67
A3	Elu	0.001	Adam	He(N)	0.01	X	8.93	12.96	-1.07	4.03
A3	Relu	0.005	Adam	He(N)	X	0.0001	11.81	13.73	1.81	1.92
A3	Relu	0.1	SGD	He(N)	X	0.00001	11.83	14.01	1.83	2.18
A4	Relu	0.001	Adam	He(N)	0.5	X	10.22	12.66	0.22	2.44
A4	Relu	0.001	Adam	He(N)	0.4	X	8.64	12.73	-1.36	4.09
A4	Relu	0.1	SGD	He(N)	0.1	X	11.78	13.73	1.78	1.95
A4	Relu	0.001	Adam	He(U)	X	0.001	11.13	13.62	1.13	2.49
A4	Relu	0.001	Adam	He(U)	X	0.0005	8.90	12.54	-1.10	3.64
A4	Relu	0.001	Adam	He(U)	X	0.0001	2.46	13.01	-7.54	10.55
A4	Relu	0.001	RMSprop	He(N)	X	0.0005	9.82	12.77	-0.18	2.95
A4	Elu	0.001	Adam	He(N)	X	0.0002	10.92	12.83	0.92	1.85

Cuadro 14: Resultados con otras combinaciones

3. Resultado final

A lo largo de todo el proceso de pruebas llegamos a la conclusión de que los resultados dependían también mucho de la partición aleatoria de los datos a la hora de limpiarlos, por lo que es muy difícil asegurar qué modificaciones tienen un impacto real en la mejora del error de validación. Lo que si podemos observar es la gran mejora en la reducción del bias con el método de Dropout, por lo que en el modelo final la usaremos para asegurarnos una mayor capacidad de extrapolación de las predicciones fuera de los datos de entrenamiento.

Por ello hemos decidido usar para el modelo final los siguientes parámetros:

Arq.	F. Act.	L. Rate	F. Opt	Init.	Dropout
A4	Relu	0.001	Adam	He(N)	0.5

Cuadro 15: Parámetros del modelo final

Con estos parámetros hemos formado el modelo y procedemos a entrenarlo. El proceso de entrenamiento se vería así gráficamente:

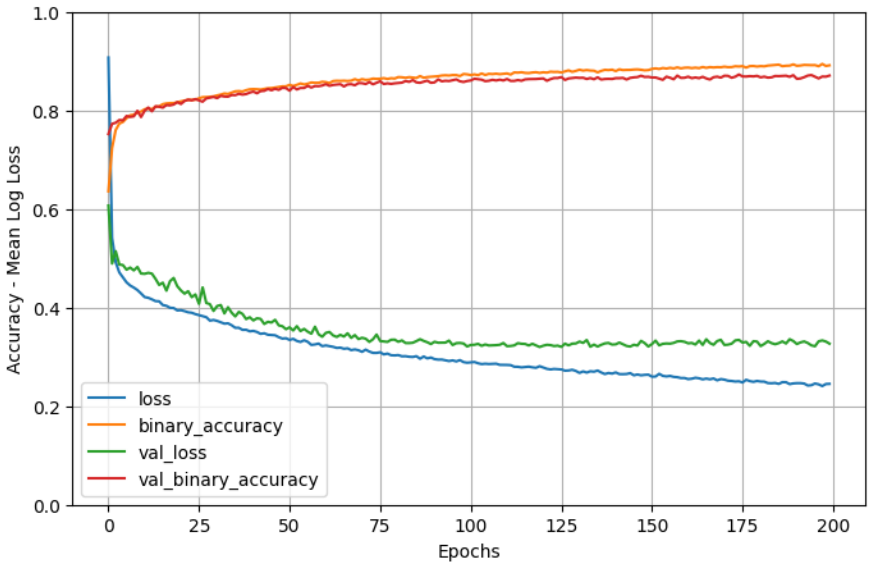

Figura 1: Gráfica del entrenamiento del modelo final

Una vez elegidos los parámetros, vamos a evaluar finalmente el conjunto de test:

Test accuracy
87.896

Cuadro 16: Pérdida y accuracy en el conjunto de test

La matriz de confusión resultante es la siguiente:

	True	False
True	929	181
False	258	2259

Cuadro 17: Matriz de confusión en el conjunto de test

Otras métricas:

	Precision	Recall	F1-Score
True	0.84	0.78	0.81
False	0.90	0.93	0.91

Cuadro 18: Otras métricas para el conjunto de test

4. Conclusiones

En conclusión, hemos aprendido que la arquitectura de la red no tiene una relevancia esencial en el desempeño del modelo, variando a lo sumo en un 3 % entre dos arquitecturas altamente dispares. A su vez, cuánto más grande sean los modelos más datos se necesitan para prevenir el sobre entrenamiento, o se necesitan aplicar medidas como el dropout en gran porcentaje para evitar que el modelo se sobre especialice en los datos de entrenamiento.

Estos resultados pueden indicar que debido al limitado tamaño de los datos de entrenamiento, algunos modelos de Aprendizaje Automático tradicionales como árboles de decisión podrían haber hecho un mejor trabajo a menor coste y mayor interpretabilidad. De todas formas, asumiendo un error del 10 % intrínseco en los datos, una precisión del 87,89 % es un resultado muy bueno y que podría usarse de manera satisfactoria en el mundo real.