Universidad Córdoba

Ingeniería Informática
Mención en Computación
3º Curso

Introducción al Aprendizaje Automático

Clasificación

Valentín Gabriel Avram Aenachioei
Víctor Rojas Muñoz
Damián González Carrasco
Ángel Simón Mesa
Pablo Fernández Arenas

Año académico 2022-2023
Córdoba, 25 de Marzo, 2023

Índice

Índice de tablas II

Índice de figuras III

1. Clasificación usando Perceptrones 1

1.1. Codifica una función que aplique el algoritmo de aprendizaje del perceptrón, siguiendo las directrices que se explicaron en clase . . . 1

1.2. Desarrolla un programa principal que haga uso de estas dos funciones, y que las aplique sobre el conjunto de datos . . . 1

1.3. Codifica ahora una función que aplique el algoritmo de clasificación de regresión logística, siguiendo también las directrices vistas en clase, y aplícalo al mismo conjunto de datos (umbral 0.5 para clasificación) . . . 2

1.4. Construye un conjunto de funciones que reciban dos vectores (valores reales y predicciones) y que construyan la matriz de confusión para este conjunto de resultados. Construye ahora un conjunto de funciones que reciben esta matriz de confusión (en el formato que hayas definido) y calculen las métricas de calidad que hemos estudiado en clase: exactitud, recall, precisión, F1 y Fb , selectividad y especificidad . . . 3

1.5. Con las funciones desarrolladas en el apartado 4, obtén los resultados de calidad de los clasificadores desarrollados (perceptrón y logística) . . . 3

1.6. Implementación del clasificador usando la libreria TuriCreate . 5

1.7. Evaluación del modelo creado usando TuriCreate . . . 6

1.8. Curva ROC del modelo creado con TuriCreate . . . 6

1.9. Red Neuronal para el reconocimiento de dígitos a partir de imágenes caligráficas . . . 8

1.9.1. Modelo 1 . . . 8

1.9.2. Modelo 2 . . . 9

1.9.3. Modelo 3 . . . 10

1.10. Red Neuronal para regresión . . . 12

1.10.1. Modelo 1 . . . 12

1.10.2. Modelo 2 . . . 12

1.10.3. Modelo 3 . . . 12

1.10.4. Conclusiones . . . 13

Índice de tablas

1. Perceptrón simple sin sobreajuste 2
2. Perceptrón logístico sobreajustado 2
3. Perceptrón logístico sin sobreajuste 3
4. Matriz de confusión para ambos clasificadores 4
5. Matriz de confusión para datos de ejemplo 5
6. Matriz de confusión del modelo de Turicreate 6
7. Rendimiento de los modelos 13

Índice de figuras

1. Curva ROC del modelo 7
2. Precisión del modelo 1 entrenamiento 9
3. Log Loss del modelo 1 entrenamiento 9
4. Precisión del modelo 2 entrenamiento 10
5. Log Loss del modelo 2 entrenamiento 10
6. Precisión del modelo 3 entrenamiento 11
7. Log Loss del modelo 3 entrenamiento 11

En este documento, vamos a realizar las explicaciones y análisis de resultados pertinentes a la tercera práctica de la asignatura, clasificación usando perceptrones.
No se va a especificar nada sobre el desarrollo del código. Este se ha realizado usando Google Colab. Usando **este enlance** se puede visualizar y ejecutar el código usado para la implementación de los algoritmos y pruebas realizadas para esta práctica.

1. Clasificación usando Perceptrones

1.1. Codifica una función que aplique el algoritmo de aprendizaje del perceptrón, siguiendo las directrices que se explicaron en clase

Este ejercicio se limita a la codificación del propio perceptrón, realizado en el **Google Colab**. adjunto.

1.2. Desarrolla un programa principal que haga uso de estas dos funciones, y que las aplique sobre el conjunto de datos

Para el conjunto de datos dado, podría haberse creado de primera mano un perceptrón cuya predicción se adapte a la perfección al conjunto de datos, sin necesidad de adaptar pesos. En cambio, se ha decidido inicializar tanto los pesos para ambas variables como el sesgo a un valor inicial igual a 1, e ir ajustando los pesos durante 2000 iteraciones, para poder así comprobar como los pesos se adaptan a los patrones del conjunto de datos.

Después de 2000 iteraciones, el clasificador predice los 8 patrones a la perfección, como es de esperar. Consideramos que estamos sufriendo de sobreajuste en el modelo.
Para prevenir este sobreajuste, decidimos repetir la prueba, con los mismo parametros iniciales, pero reduciendo el número de iteraciones, de 2000 a 6. Evitamos el sobreajuste a coste de errores en la predicción. Los datos de esta prueba sin sobreajuste puede verse en la tabla 1.

Predicción	Valor real
1	0
0	0
1	0
1	0
1	1
1	1
1	1
1	1

Tabla 1: Perceptrón simple sin sobreajuste

1.3. Codifica ahora una función que aplique el algoritmo de clasificación de regresión logística, siguiendo también las directrices vistas en clase, y aplícalo al mismo conjunto de datos (umbral 0.5 para clasificación)

Se ha adaptado el código del ejercicio 2, usando los mismos valores iniciales para pesos y sesgo. Se ha cambiado la función de activación por una función sigmoide, y se ha ajustado los pesos durante 2000 iteraciones.

Con un conjunto de datos de tan poco tamaño, el clasificador logístico vuelve a obtener predicciones perfectas en este caso.
Los resultados exactos obtenidos del clasificador logístico se pueden ver en detalle en la tabla 2, donde los Valores Sigmoides son los valores devueltos por la función de activación del clasificador, la función sigmoide, que define la probabilidad de pertenencia a la clase positiva.

Valor Sigmoide	Predicción	Valor real
0,000156	0	0
0,003331	0	0
0,006126	0	0
0,196063	0	0
0,906097	1	1
0,946793	1	1
0,998582	1	1
0,999230	1	1

Tabla 2: Perceptrón logístico sobreajustado

De nuevo, estamos ante un caso de sobreajuste. Para evitarlo, hemos vuelto a cambiar el numero de iteraciones, esta vez a 25, consiguiendo solucionar el problema del sobreajuste a costa de un peor rendimiento en la clasificación de patrones. Los resultados se puede visualizar en la tabla 3.

Valor Sigmoide	**Predicción**	**Valor real**
0.4671	0	0
0.4409	0	0
0.5460	1	0
0.6226	1	0
0.6936	1	1
0.7754	1	1
0.8257	1	1
0.8784	1	1

Tabla 3: Perceptrón logístico sin sobreajuste

1.4. Construye un conjunto de funciones que reciban dos vectores (valores reales y predicciones) y que construyan la matriz de confusión para este conjunto de resultados. Construye ahora un conjunto de funciones que reciben esta matriz de confusión (en el formato que hayas definido) y calculen las métricas de calidad que hemos estudiado en clase: exactitud, recall, precisión, F1 y Fb , selectividad y especificidad

Este ejercicio se limita al desarrollo del código, siguiendo las fórmulas para las métricas vistas en clase. El desarrollo se realiza en el **Google Colab**. adjunto.

1.5. Con las funciones desarrolladas en el apartado 4, obtén los resultados de calidad de los clasificadores desarrollados (perceptrón y logística)

Para ambos clasificadores, tanto el perceptrón como el clasificador logístico obtenemos los mismo resultados, pues ambos se trabaja sobre el mismo

conjunto de datos de tan solo ocho patrones, obteniendo unas predicciones perfectas.La matriz de confusión para ambos clasificadores se puede ver en la tabla 4 .

	Positivos predichos	**Negativos Predichos**
Positivos	4	0
Negativos	0	4

Tabla 4: Matriz de confusión para ambos clasificadores

Para ambos clasificadores, las medidas de calidad son las mismas, pues sobre los mismos datos realizan las mismas predicciones. Las medidas obtenidas son:

- Recal = 1
- Precision = 1
- Sensibilidad = 1
- Especificidad = 1
- F-score (con cualquier valor de β) = 1

Estos valores se pueden entender correctos por la naturaleza de conjunto de datos. En un conjunto de datos tan pequeño y uniforme, el modelo tiende rapidamente al sobreajuste.
Para comprobar el funcionamiento de las funciones, se han decidido probar estas con dos vectores binarios de 16 bits, que simulan los valores reales de un conjunto de datos, y los datos predichos usando cierto modelo.

$$v_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

$$v_2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{2}$$

El vector v_1 representa los valores reales de las clase, y el vector v_2 son las predicciones realizadas por el modelo. Usando estos datos de ejemplo, la matriz de confusión obtenida se puede ver en la tabla 6.

Así, las métricas de calidad para este ejemplo son:

	Positivos predichos	**Negativos Predichos**
Positivos	5	4
Negativos	3	4

Tabla 5: Matriz de confusión para datos de ejemplo

- Recal = 0.5555
- Precision = 0.625
- Sensibilidad = 0.5556
- Especificidad = 0.5714
- F-score ($\beta = 1$) = 0.5882

1.6. Implementación del clasificador usando la libreria TuriCreate

Para la implementación del clasificador logístico, se han seguido los consejos del guión de prácticas y la propia documentación de Turicreate. Se ha usado un SFrame para cargar el fichero de datos, y se ha hecho un recuento de aparición de cada palabra en las críticas.

Una vez tenidos los datos de entrada, se ha dividido el conjunto de datos en entrenamiento, un 80 %, y evaluación el 20 % restante. Se ha creado un modelo, entrenado con los datos de entrenamiento y evaluado su rendimiento con las propias métricas que ofrece Turicreate. Como una primera medida de evaluación, vemos que la precisión del modelo es del 86.878 %.

1.7. Evaluación del modelo creado usando TuriCreate

Un apartado positivo de las librerías como Turicreate es que ya tienen integradas un sistema de métricas y evaluación para los modelos desarrollados, que nos llega a ofrececer aún mas métricas de las que nosotros mediamos en un principio.

Una vez evaluado el modelo, obtenemos las siguientes métricas. La matriz de confusión se representa en la tabla 6.

	Positivos predichos	**Negativos Predichos**
Positivos	4917	36
Negativos	31	4987

Tabla 6: Matriz de confusión del modelo de Turicreate

Así, las métricas de calidad para este modelo son:

- Recall = 0.9937
- Precision = 0.9927
- Sensibilidad = 0.9917
- Especificidad = 0.9938
- F-score ($\beta = 1$) = 0.9932

Además, la evaluación de TuriCreate nos da interesantes métricas de evaluación que nosotros no estábamos teniendo en cuenta, como puede ser el propio error logístico, *Log Loss*, que tiene un valor de 0.03435.

1.8. Curva ROC del modelo creado con TuriCreate

Como se pide realizar la Curva ROC para el modelo creado con Turicreate, se ha utilizado el módulo propio de curva ROC de la librería scikit-learn, a partir de los datos obtenidos del modelo.

Como estamos usando un procedimiento propio de una librería, no necesitamos seleccionar manualmente los umbrales ni calcular las medidas de sensibilidad ni especificidad. En lugar de seleccionar los umbrales a mano,

y hacer nosotros los cálculos de las medidas de calidad de cada umbral, el propio método de sklearn divide el conjunto de predicciones en mil umbrales equivalentes. Para cada umbral, calcula sensibilidad y especificidad, y con los datos de todos los umbrales, grafica la curva ROC. Además, calcula el área bajo la curva ROC, como medida de calidad de esta.

La curva ROC se puede visualizar en la gráfica 1. Como medida de calidad de esta, podemos ver que su área bajo la curva es de 0.93 u^2.

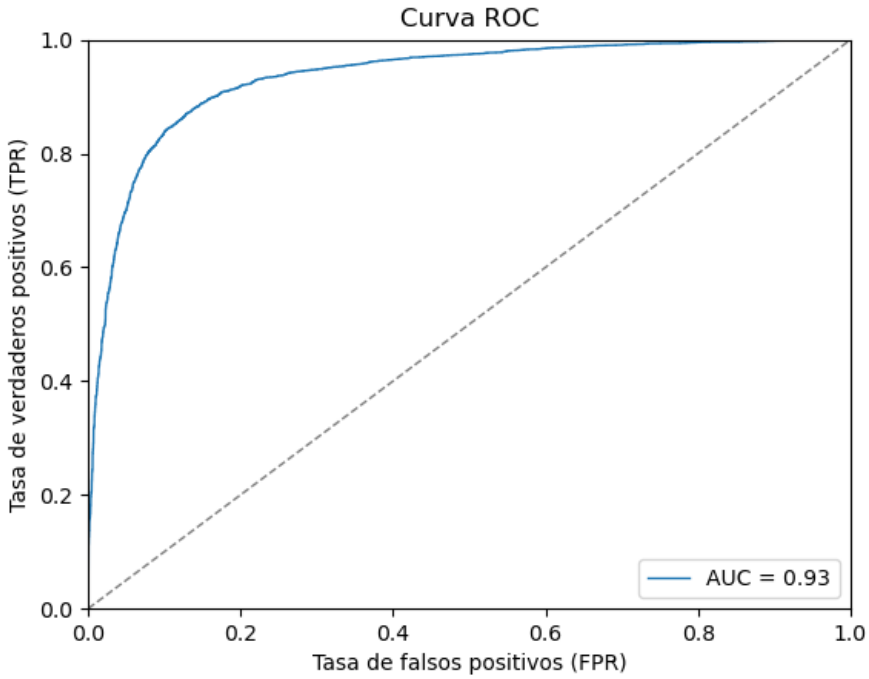

Figura 1: Curva ROC del modelo

1.9. Red Neuronal para el reconocimiento de dígitos a partir de imágenes caligráficas

Para este ejercicio, se ha hecho uso de las funciones propias de las librerías TensorFlow y Keras, entre otras, para la creación del modelo de la Red Neuronal. Se ha usado como conjunto de datos el dataset MNIST, redimensionado, particionado en entrenamiento y evaluación y preparado para que su salida sea categórica.

Para la creación del modelo, se han probado con varias arquitecturas diferentes, para comprobar la influencia de esta sobre el rendimiento. En cuanto a hiperparámetros, se han mantenido los mismos, pues aunque importantes, en redes neuronales con un gran número de nodos y capas la influencia de la arquitectura puede ser aún más significativa.
Como hiperparámetros, se ha mantenido en todas las pruebas un valor de Dropout igual a 0.2 para las capas ocultas, considerando 10 épocas con un tamaño de lote de 10.

Además, se ha usado como métrica a evaluar la precisión en la clasificación, usando el optimizador Adam y función de error o pérdida el *Categorical Cross Entropy*.

1.9.1. Modelo 1

El modelo considerado como 1 es aquel que se da como ejemplo en el guión de la práctica. Consta de la capa de entrada, que es invariable, dos capas ocultas con 120 y 64 nodos respectivamente, usando la función ReLU como función de activación, y como capa de salida, 10 nodos usando la función Softmax como función de activación.

El entrenamiento nos da los siguientes siguientes valores de Precisión y Perdida Logarítmica, como se puede ver en la figura 2 y 3, respectivamente.

Una vez entrenada la red neuronal, obtenemos los siguientes valores para las métricas de calidad que estamos midiendo:

- Precisión = 0.96499
- Log Loss = 0.24655

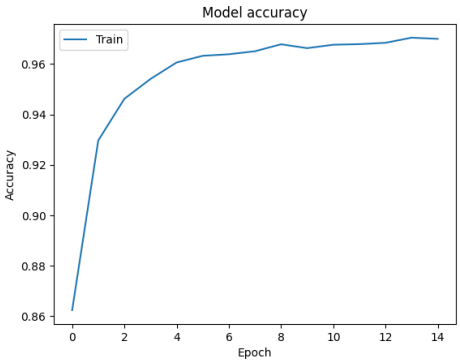

Figura 2: Precisión del modelo 1 entrenamiento

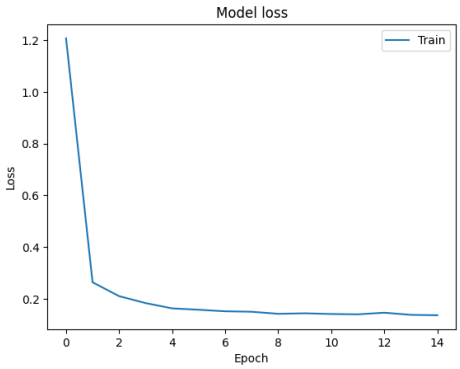

Figura 3: Log Loss del modelo 1 entrenamiento

Para este primer modelo propuesto, conseguimos unos muy buenos valores tanto para la precisión como para el Log Loss. Además, no podemos considerar que el modelo este sobreajustado, pues es capaz de obtener las mismas buenas predicciones del entrenamiento en la fase de evaluación.

1.9.2. Modelo 2

Para el modelo 2, hemos intentado crear un modelo de arquitectura mas compleja, intentando comprobar si el diseño de la arquitectura puede causar sobreajuste y llevar a un peor rendimiento.

La arquitectura que hemos probado consta de la capa de entrada, y cuatro capas ocultas, con 256, 128, 64 y 32 nodos respectivamente, usando la función ReLU como función de activación. Como capa de salida se usan 10 nodos, usando la función Softmax como función de activación. Entrenando, obtenemos los siguientes valores a medida que las épocas avanzan, visualizados en las figuras 4 para la Precisión y 5 para la Pérdida Logarítmica.

Una vez entrenada la red neuronal, obtenemos los siguientes valores para las métricas de calidad que estamos midiendo:

- Precisión = 0.97009
- Log Loss = 0.15021

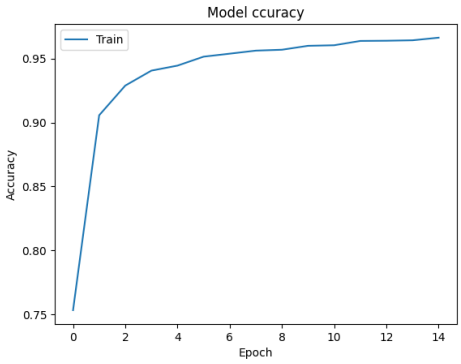

Figura 4: Precisión del modelo 2 entrenamiento

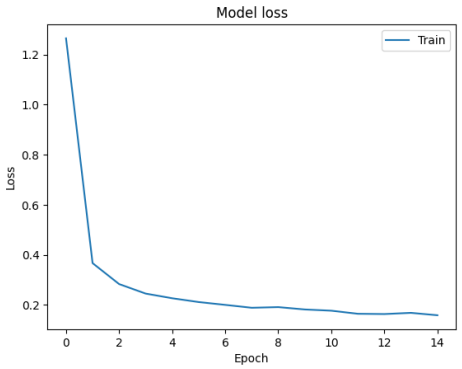

Figura 5: Log Loss del modelo 2 entrenamiento

En cuanto al entrenamiento de este modelo, conseguimos prácticamente los mismos valores de pérdida logística y valores ligeramente peores de precisión, comparados con el modelo anterior. Estas métricas se mejoran al evaluar el modelo, pero la mejora, aunque notable en la perdida logística, es apenas perceptible en cuanto a la precisión, que es la métrica que queremos optimizar en una clasificación.
Al evaluar el modelo, no podemos afirmar que suframos de sobreajuste, pues es capaz de mantener las buenas predicciones obtenidas al entrenar en la evaluación.

Aunque se puede afirmar que este modelo es mejor, quizás no sea la mejor opción, pues la mejora no merece el aumento de complejidad del modelo, complejidad que se traduce como un aumento en el tiempo de computo y gasto de memoria.

1.9.3. Modelo 3

Como tercer modelo, hemos probado con un modelo mucho mas simplificado, para ver si podemos llegar a obtener resultados parecidos a los modelos anteriores con un gasto de memoria y tiempo de computación mucho menor.

Para esta prueba, hemos usado el modelo mas sencillo posible, con una capa de entrada, una sola capa oculta de 64 nodos usando la función ReLU como función de activación, y la capa de salida con 10 nodos usando la función Softmax como función de activación. Al entrenar la red, obtenemos los valores de Precisión y Pérdida Logística visualizados en las figuras 6 y 7 respectivamente.

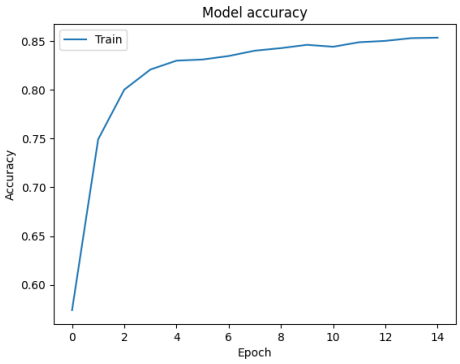

Figura 6: Precisión del modelo 3 entrenamiento

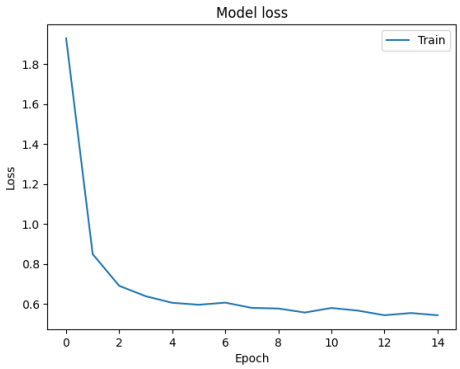

Figura 7: Log Loss del modelo 3 entrenamiento

Una vez entrenada la red neuronal, obtenemos los siguientes valores para las métricas de calidad que estamos midiendo:

- Precisión = 0.90899
- Log Loss = 0.49075

Podemos ver que en este modelo obtenemos peores valores tanto para Precisión como para Perdida Logarítmica, como es esperable por la propia estructura de la red. Aún así, un modelo mucho más sencillo comparado a los dos anteriores, presenta un rendimiento ligeramente peor, es decir, sigue siendo una opción mucho mas viable.

Aunque la pérdida logarítmica es mucho peor que en ambos modelos, la precisión no llega a ser menor del 90 %, es decir, sigue siendo un modelo muy preciso. Según el caso de aplicación del problema, este modelo podría llegar a ser la mejor opción por su poco gasto de tiempo y memoria.

1.10. Red Neuronal para regresión

Para este ejercicio, se ha hecho uso de las funciones propias de las librerías TensorFlow y Keras, entre otras, para la creación del modelo de la Red Neuronal.

Para las pruebas, se ha usado el dataset Hyderabad, usado en la práctica anterior, particionado en entrenamiento y evaluación. Se debe tener en cuenta que el conjunto de datos cuenta con valores perdidos y variables que distan en varias escalas de magnitud, lo que hace que el rendimiento del modelo sea mucho peor. De todas formas, se ha usado el conjunto de datos original, donde la única modificación hecha es la propia partición de los datos. Al igual que en el ejercicio anterior, se han mantenido ciertos hiperparámetros iguales, en este caso, siempre se han usado 10 épocas con un tamaño de lote 10, usando el optimizador Adam.

1.10.1. Modelo 1

La primera aproximación al problema es la recomendada en el propio guión de prácticas. Esta arquitectura consta de la capa de entrada, de 38 nodos y 2 capas ocultas, de 128 y 64 nodos respectivamente, ambas usando la función ReLU como función de activación, y un valor de Dropout de 0,2. Por último, hay una capa de salida de un solo nodo, sin función de activación.

1.10.2. Modelo 2

Como segundo modelo probado, se ha intentado un modelo mas complejo, usando 3 capas ocultas en lugar de 2, con 256, 128 y 64 nodos respectivamente, usando como función de activación una función sigmoide en lugar de ReLU. Se sigue manteniendo el valor de Dropout de 0,2.

1.10.3. Modelo 3

El tercer modelo probado ha sido similar al segundo, pero simplificándolo. En lugar de usar 3 capas con 256, 128 y 64 nodos respectivamente, se han mantenido las 3 capas, pero usando menos nodos, en este caso, 128, 64 y 32 nodos. Además, en lugar de usar la función de activación sigmoide usada en el modelo dos, se ha optado por usar la función de activación ReLU para las capas ocultas del modelo.

1.10.4. Conclusiones

Los resultados obtenidos por los tres modelos, tanto en entrenamiento como en evaluación, se puede visualizar en la tabla 7.

Model	Training MSE	Training Loss	Testing MSE	Testing Loss
1	6.4768e+13	6.4768e+13	2.7941e+13	2.7941e+13
2	1.6862e+14	1.6862e+14	1.8774e+14	1.8774e+14
3	4.6840e+13	4.6840e+13	2.7666e+13	2.7666e+13

Tabla 7: Rendimiento de los modelos

Podemos ver que el tercer modelo propuesto parece ser el mejor de los probados, o al menos, muy similar al modelo 1. Aunque ambos modelos obtienen prácticamente los mismos resultados en la evaluación, tanto en el Error Cuadrático Medio como en la función de perdida, el modelo 3 obtiene menores valores en la fase de entrenamiento. Aún así, hay que destacar que el primer modelo es algo mas sencillo, al usar muchos menos nodos en total.

El segundo modelo, al ser el mas complejo, resulta ser el peor. Esto puede ser por un posible sobreajuste en el modelo, pues llegamos a obtener peores valores en la evaluación, comparado con el entrenamiento. De todos modos, podemos afirmar que, por los valores de las métricas obtenidos, ningún modelo es útil.