Práctica evaluable Redes Neuronales Recurrentes (RNNs) y Series Temporales

---

Deep Learning y sus aplicaciones

Máster en Ingeniería Informática

Autor:

Rebé Martín, Jorge

Fecha: 17 de enero de 2024

1. Introducción

En este documento se describe un trabajo consistente en la resolución de un problema de predicción utilizando redes neuronales recurrentes [1], ya que es un problema de series temporales.

Se trata de predecir el número de bicicletas alquiladas por hora en la ciudad de Washington DC, en base a varios atributos como la estación del año, temperatura, etcétera.

2. Objetivos

El objetivo de este trabajo es la construcción y evaluación de varios modelos de redes neuronales que se utilizarán para realizar predicciones sobre el número de bicicletas alquiladas por hora en Washington.

Se utilizarán varias arquitecturas de referencia y se realizará una búsqueda de valores óptimos para los hiperparámetros de cada modelo a estudiar, de forma que se pueda tener el mejor modelo posible.

3. Estudio de los datos a predecir

De los dos conjuntos de datos disponibles [2], se ha escogido el que tiene los datos de alquiler de bicicletas acumulados por horas, debido principalmente a que contiene muchos más datos (número de días por número de horas).

3.1. Estudio de la correlación entre atributos numéricos

Para estudiar la correlación entre los atributos numéricos, se ha creado una matriz de correlación utilizando pandas (consultar el Notebook de Jupyter para ver los valores numéricos de la matriz).

A continuación, se ha representado con un mapa de calor dichos valores para visualizar mejor la correlación entre los distintos atributos, mostrando la matriz como una matriz diagonal. En la Figura 1 se muestra el mapa de calor de la matriz diagonal de correlación obtenida con seaborn [3].

3.2. Estudio de la variable a predecir

3.2.1. Medidas de centralización

En la Tabla 1 se muestran las medidas de centralización básicas para la variable a predecir.

Métrica	Valor
Media	1.27741
Mediana	1.41720
Moda	5

Tabla 1: Métricas de centralización y sus valores para la variable a predecir 'cnt'

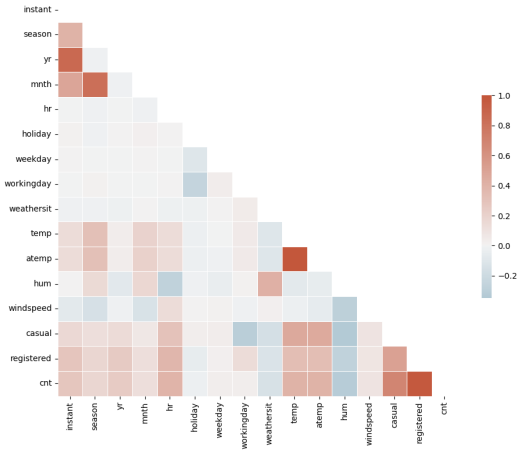

Figura 1: Matriz diagonal de correlación de los atributos numéricos

Un gráfico que puede ser interesante es el del número medio de bicicletas alquiladas por hora, que nos puede dar de un vistazo una idea de como se comporta la variable a predecir. En la Figura 2 se muestra un gráfico de barras generado con seaborn.

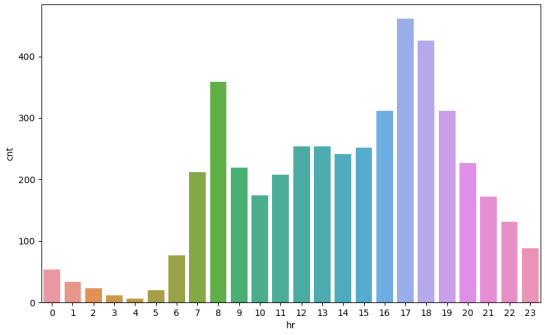

Figura 2: Gráfico de barras de la media de bicicletas alquilada por cada hora

3.2.2. Medidas de variabilidad

En la Tabla 2 se muestran algunas métricas de variabilidad para la variable a predecir.

Métrica	Valor
Desviación típica	181.38759
Varianza	32901.46110
Máximo	5
Mínimo	1
Rango	926

Tabla 2: Métricas de variabilidad y sus valores para la variable a predecir 'cnt'

En la Figura 3 se muestra un gráfico de caja y bigotes de la variable a predecir. Se puede ver que hay una gran cantidad de outliers (todos los valores que son mayores de 600), y que la mayoría de las horas se alquilan entre 20 y 300 bicicletas.

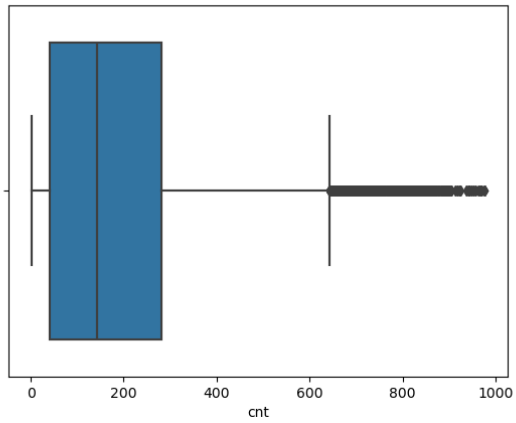

Figura 3: Gráfico de caja y bigotes del número de bicilcetas alquiladas por hora

3.2.3. Medidas de distribución

Se han calculado los índices de Skewness y Kurtosis [4] para medir la distribución de la variable a predecir. En cuanto al índice de Skewness, es una medida de la falta de simetria. Cuanto más lejano de uno, menos simétrica es la distribución de los datos.

Kurtosis es una medida que indica lo larga que es la 'cola' de una distribución de datos, es decir, indica los outliers. Como hemos visto antes, hay bastantes outliers a la derecha (outliers para alquileres de bicicletas por horas por exceso, pero no por defecto). El valor de 1.41 indica que hay algunos outliers, aunque no demasiados.

Métrica	Valor
Skewness	1.27741
Kurtosis	1.41720

Tabla 3: Métricas de la distribución y sus valores para la variable a predecir 'cnt'

En la Figura 4 vemos la distribución de los valores del número de bicicletas alquiladas por hora, y coincide con lo descrito anteriormente en esta sección. Hay algunos outliers para valores altos de la variable (densidad muy baja), coincidiendo con los índices calculados previamente. La forma de la distribución no es simétrica, pero tampoco es completamente asimétrica.

4. Preprocesamiento de los datos

Una vez estudiado el conjunto de datos, hay que realizar el preprocesamiento de los mismos. Hay que eliminar atributos irrelevantes y que no son útiles para predecir la variable del número de bicicletas, y hay atributos que tenemos que escalar o transformar para que puedan ser la entrada de la red neuronal.

En la Figura 5 se muestra la primera fila del dataset antes de realizar el procesamiento de los datos descrito en esta sección.

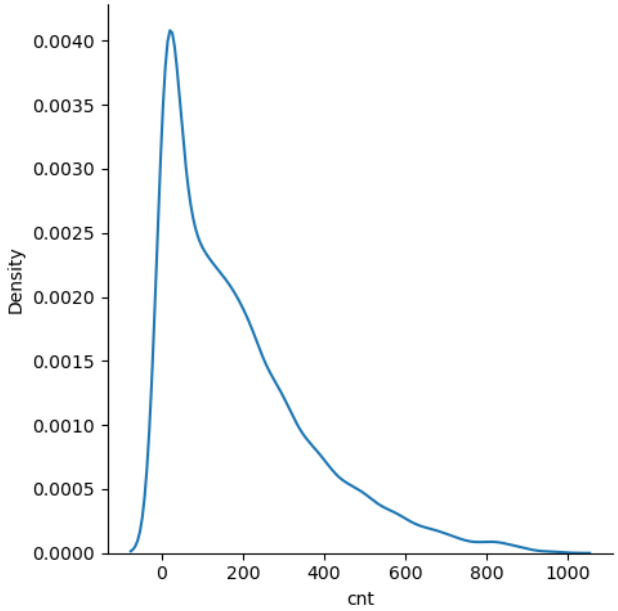

Figura 4: Distribución de los valores de la variable a predecir

Datos Sin Procesar

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16

Figura 5: Datos Sin Procesar

4.1. Selección de Atributos

Hay atributos del conjunto de datos que no son útiles para la predicción del número de bicicletas alquiladas por hora y, por tanto, conviene eliminarlos. En primer lugar, ‘instant’ no es más que un identificador, por lo que lo eliminaremos. ‘dteday’ es el día del que son los datos, que se podría utilizar para realizar búsquedas por si hubiera ocurrido algo que influyera en el uso de las bicicletas (huracanes, tormentas, etcétera), y no se usará para la predicción del número de bicicletas alquiladas por hora.

También se ha decidido eliminar los atributos ’yr’ y ’weekday’, que representan el año y el día de la semana, respectivamente, ya que tienen una baja correlación con la variable a predecir.

Como se ve en la Figura 1, las variables ‘temp’ y ‘atemp’ están altamente correladas, y es por ello que se ha decidido eliminar el atributo ‘atemp’, ya que representan lo mismo prácticamente (‘temp’ temperatura real escalada, y ‘atemp’ sensación térmica).

Hay otras que directamente son la variable a predecir (’cnt’, la variable a predecir, es la suma de ’casual’ y ’registered’, siendo ’casual’ alquileres de usuarios casuales y ’registered’ los alquileres de usuarios registrados). Por tanto, se eliminarán los atributos ’casual’ y ’registered’.

En resumen, se han eliminado los atributos ‘instant’, ‘dteday’, ‘casual’, ‘registered’, ‘atemp’, ‘yr’ y ‘weekday’.

4.2. Tratamiento de Atributos Nominales

Los atributos nominales restantes en el conjunto de datos tras la selección de atributos (estación del año 'season' y clima en el momento del registro 'weathersit') no pueden introducirse a la red neuronal tal como están porque solo se pueden introducir valores numéricos. Por tanto, se les discretizará utilizando el método One Hot Encoder, que crea un atributo booleano para cada valor posible del atributo nominal sobre el que se aplica.

Esto implica un incremento en la dimensionalidad, ahora se tienen 6 atributos más (4 valores posibles para cada atributo al que se le ha aplicado One Hot Encoder, pero se eliminan los originales). One Hot Encoder utilizando el método de pandas `get_dummies` [5].

4.3. Escalado de la salida

Además de las transformaciones anteriores, se va a escalar la variable a predecir entre 0.1 y 0.9, utilizando MinMaxScaler de sklearn [6]. Se escala entre 0.1 y 0.9 porque en la neurona de salida vamos utilizar una función de activación de tipo sigmoide, y los valores extremos 0 y 1 no se alcanzan.

En la Figura 6 se muestra el dataset tras el preprocesamiento descrito en esta sección.

Datos Procesados

mnth	hr	holiday	workingday	temp	hum	windspeed	cnt	season_1	season_2	season_3	season_4	weathersit_1	weathersit_2	weathersit_3	weathersit_4
1	0	0	0	0.24	0.81	0.0	0.112295	1	0	0	0	1	0	0	0

Figura 6: Datos Procesados

4.4. Preparación de los datos

Los datos de entrada a una capa LSTM tienen que ser un tensor con la forma [numero_total_ejemplos, historia, numero_atributos] [7], por lo que se utilizará una función para transformar el dataset en un tensor con esa forma, tanto para la entrada a la red como para la salida. Más información en el cuaderno de Jupyter.

4.5. División en Entrenamiento, Validación y Prueba

Una vez preprocesados los datos, se van a dividir en entrenamiento y prueba. Se utilizarán para entrenar $\frac{2}{3}$ de los datos, y el resto se utilizarán para evaluar cada uno de los modelos entrenados.

Cuando se entrene cada uno de los modelos, se utilizará un 20 % de los datos de entrenamiento para realizar validación de los valores de los hiperparámetros que se estén estudiando.

5. Metodología utilizada en la Creación, Entrenamiento y Evaluación de los Modelos

Una vez divididos los datos en entrenamiento y prueba, se va a explicar la metodología usada en la creación, entrenamiento y evaluación de los modelos. Se van a entrenar diversos modelos, y para cada uno de ellos se buscarán los mejores valores para algunos hiperparámetros del modelo. Para ver cual de los valores de esos parámetros es mejor, se validará el modelo sobre el conjunto de validación (20 % del conjunto de entrenamiento). Más sobre este proceso en la Sección 6.2

Una vez seleccionados los mejores valores para cada hiperparámetro, se entrenará un modelo desde cero con todo el conjunto de entrenamiento, y después ese modelo se evaluará sobre el conjunto de test. Este entrenamiento y evaluación se repetirá 2 veces para dar algo de validez a los resultados, pero habría que repetirlo más veces.

6. Entrenamiento de los Modelos

6.1. Selección de Hiperparámetros

Hay una gran cantidad de hiperparámetros cuyos valores se pueden optimizar para cada modelo. En el caso de este trabajo, se han buscado valores óptimos sobre los siguientes hiperparámetros

- Número de elementos LSTM
- Número de capas LSTM
- Tasa de aprendizaje (Learning Rate - LR)
- Tasa de Dropout

Se podría haber buscado valores para otros, como el tamaño de la historia, distintos optimizadores, etcétera.

6.2. Búsqueda automática de los mejores valores para los hiperparámetros

Para realizar automáticamente la búsqueda de los mejores valores para los hiperparámetros seleccionados para cada modelo, se ha utilizado Keras Tuner [8]. Esta API de Keras permite especificar los hiperparámetros de cada modelo y los valores sobre los que se quiere buscar, elegir un Tuner que realiza búsquedas sobre el espacio de posibles valores para cada hiprrparámetro y conseguir construir el mejor modelo posible.

Hay varias opciones disponibles para los Tuners: GridSearch, que realiza una búsqueda sobre todo el espacio de posibles valores para los hiperparámetros, lo que significa que es muy costoso computacionalmente. Hay otros Tuners, como el RandomSearch, que hace búsquedas aleatorias sobre el espacio de búsqueda, o BayesianOptimization o Hyperband Tuner, que utilizan algoritmos de optimización más complejos y, por tanto, con buenos resultados con menor coste computacional que GridSearch.

En este caso, se ha seleccionado como Tuner RandomSearch [9]. Se ha seleccionado como número de búsquedas máximo 10, con un total de 3 ejecuciones por búsqueda. Es decir, **se repetirán los experimentos 3 veces**.

6.3. Callbacks

Durante el entrenamiento, es posible que el valor de la función de pérdida sobre el conjunto de validación deje de mejorar, por lo que necesitamos formas de controlar durante el entrenamiento la tasa de aprendizaje o de parar el entrenamiento si suceden muchos *epochs* sin mejoras en dicho valor.

Estos métodos se conocen como Callbacks[10], y hay varias que se pueden utilizar en Keras (además de poder crear callbacks nuevas). En este caso, vamos a utilizar las dos siguientes:

- EarlyStopping [11]: Para el entrenamiento cuando una métrica ha dejado de mejorar durante un determinado número de epochs. En el caso del entrenamiento, se monitoriza el valor de la función de pérdida sobre el conjunto de validación, y una paciencia de 2 epochs. Esto quiere decir que si durante 2 epochs seguidos no mejora ese valor, se parará el entrenamiento.
- ReduceLROnPlateau [12]: Multiplica la tasa de aprendizaje por un factor cuando una métrica que se está monitorizando no ha mejorado una cantidad determinada durante un número de epochs.

6.4. Modelos Entrenados

En esta subsección se explicarán superficialmente cada uno de los modelos entrenados, así como los resultados obtenidos durante la búsqueda de mejores valores para los hiperparámetros seleccionados en cada modelo.

6.4.1. LSTM - 1 capa

Este es el modelo básico, una única capa de LSTM. Los hiperparámetros a considerar son el número de elementos LSTM, la tasa de dropout y la tasa de aprendizaje. En la Figura 7 se muestran los resultados de la búsqueda para tres ejecuciones.

6.4.2. LSTM con Ventana Temporal

Es el mismo modelo que el anterior, solo que ahora se utilizará una historia de 3 para la entrada. Los hiperparámetros considerados son los mismos que en el modelo anterior. En la Figura 8 se muestran los resultados de la búsqueda para tres ejecuciones.

6.4.3. LSTM - Varias Capas

En el caso de este modelo de LSTM apiladas [13], se van a buscar valores óptimos sobre los siguientes hiperparámetros: número de capas ocultas, tasa de dropout y tasa de aprendizaje. En la Figura 9 se muestran los resultados de la búsqueda para tres ejecuciones.

	units	dropout-rate	learning_rate	Val Loss
0	48	0.1	0.0010	0.006660
1	32	0.0	0.0010	0.006769
2	32	0.2	0.0010	0.007218
3	48	0.2	0.0001	0.007580
4	48	0.1	0.0001	0.007746
5	16	0.1	0.0010	0.007908
6	32	0.0	0.0001	0.008531
7	32	0.1	0.0100	0.011556
8	48	0.0	0.0100	0.012167
9	32	0.2	0.0100	0.012842

Figura 7: Valor de la función de pérdida para cada combinación hiperparámetro-valor sobre el conjunto de validación para el modelo LSTM de una capa tras 3 ejecuciones de los experimentos

	units	dropout-rate	learning_rate	Val Loss
0	48	0.0	0.0010	0.004244
1	48	0.1	0.0100	0.008960
2	32	0.0	0.0001	0.009293
3	32	0.1	0.0100	0.009410
4	48	0.1	0.0001	0.010008
5	48	0.2	0.0100	0.010089
6	16	0.1	0.0100	0.010806
7	32	0.2	0.0001	0.011068
8	32	0.1	0.0001	0.011556
9	16	0.1	0.0001	0.012915

Figura 8: Valor de la función de pérdida para cada combinación hiperparámetro-valor sobre el conjunto de validación para el modelo LSTM con ventana temporal tras 3 ejecuciones de los experimentos

6.4.4. LSTM con Memoria

Las LSTM tienen memoria (recuerdan el estado). Normalmente esta memoria se borra al empezar otro lote de entrenamiento, o cuando se calcula una nueva salida para una nueva predicción. Una nueva estrategia es que esa memoria (estado) se conserve entre lote y lote de aprendizaje (stateful), con lo que se construirá un estado sobre la secuencia completa de entrenamiento (no lote a lote), y este estado es el que se usará para hacer la predicción. Los hiperparámetros a considerar son el número de elementos LSTM, la tasa de dropout y la tasa de aprendizaje. En la Figura 10 se muestran los resultados de la búsqueda para tres ejecuciones.

	hiddden layers	dropout rate	learning_rate	Val Loss
0	1	0.0	0.0010	0.005475
1	3	0.0	0.0010	0.005709
2	2	0.0	0.0010	0.006440
3	4	0.2	0.0010	0.006747
4	1	0.0	0.0001	0.006872
5	3	0.1	0.0010	0.007093
6	4	0.0	0.0001	0.007211
7	3	0.0	0.0001	0.007442
8	1	0.2	0.0100	0.014176
9	2	0.1	0.0100	0.016704

Figura 9: Valor de la función de pérdida para cada combinación hiperparámetro-valor sobre el conjunto de validación para el modelo con LSTMs apiladas tras 3 ejecuciones de los experimentos

	units	dropout rate	learning_rate	Val Loss
0	16	0.2	0.0010	0.004926
1	48	0.2	0.0001	0.006756
2	48	0.0	0.0001	0.006963
3	32	0.0	0.0001	0.007816
4	32	0.2	0.0100	0.011080
5	48	0.2	0.0100	0.011556
6	16	0.2	0.0100	0.011787
7	48	0.1	0.0100	0.012164
8	16	0.0	0.0100	0.012457
9	16	0.2	0.0001	0.013669

Figura 10: Valor de la función de pérdida para cada combinación hiperparámetro-valor sobre el conjunto de validación para el modelo con LSTM con memoria tras 3 ejecuciones de los experimentos

6.4.5. LSTM Apiladas con Memoria entre capas

Este modelo es una combinación de los dos anteriores. Se coge como número de elementos LSTM el mejor del modelo LSTM con Memoria, y se buscan en los mismos hiperparámetros que para el modelo de LSTM apiladas. En la Figura 11 se muestran los resultados de la búsqueda para tres ejecuciones.

6.4.6. GRU - 1 capa

Este modelo es igual al primero (LSTM de 1 capa), pero utilizando módulos GRU en lugar de LSTM, que tardan menos en entrenar al tener menos parámetros. En la Figura 9 se muestran los resultados de la búsqueda para tres ejecuciones.

	hiddden layers	dropout rate	learning_rate	Val Loss
0	1	0.1	0.0010	0.005284
1	3	0.0	0.0010	0.005295
2	2	0.0	0.0010	0.005371
3	3	0.1	0.0010	0.005404
4	4	0.0	0.0010	0.007296
5	3	0.0	0.0001	0.007448
6	1	0.0	0.0100	0.011854
7	2	0.0	0.0100	0.016311
8	2	0.2	0.0100	0.017312
9	3	0.0	0.0100	0.023779

Figura 11: Valor de la función de pérdida para cada combinación hiperparámetro-valor sobre el conjunto de validación para el modelo con LSTM con memoria apiladas tras 3 ejecuciones de los experimentos

	units	dropout-rate	learning_rate	Val Loss
0	32	0.0	0.0010	0.005256
1	16	0.0	0.0010	0.005839
2	48	0.2	0.0010	0.006574
3	32	0.2	0.0010	0.006675
4	48	0.0	0.0001	0.007145
5	48	0.1	0.0100	0.010899
6	16	0.1	0.0001	0.012423
7	16	0.2	0.0001	0.013104
8	48	0.0	0.0100	0.013271
9	32	0.0	0.0100	0.014460

Figura 12: Valor de la función de pérdida para cada combinación hiperparámetro-valor sobre el conjunto de validación para el modelo con GRU de una capa tras 3 ejecuciones de los experimentos

7. Evaluación de los Modelos

Una vez encontrados unos buenos valores para los hiperparámetros de cada modelo, se entrenará de nuevo el modelo con esos valores para los hiperparámetros durante 3 epochs. Una vez entrenado el modelo, se evaluará sobre el conjunto de test y se obtendrá el error cuadrático medio sobre dicho conjunto. Para dar validez a los resultados, se repetirán los experimentos 2 veces. Evidentemente, el número de epochs y repeticiones de entrenamiento de modelos es una variable configurable dentro del Notebook. De disponer de más tiempo, serían números más grandes.

En la Tabla 4 se muestran los resultados de la evaluación de los modelos, para cada modelo, cada ejecución, y la media por ejecuciónes.

A la vista de estos resultados, podemos concluir que un modelo con ventana temporal funciona mejor que los demás. Habría que verificarlo aumentando el número de epochs y de repetición de los

	Raíz del Error Cuadrático Medio (RMSE)		
Modelo	Ejecución 1	Ejecución 2	Media
LSTM 1 Capa	119.42372	110.77264	115.09818
LSTM Ventana Temporal (historia = 3)	82.83509	80.35518	82.59513
LSTM Apiladas	129.11170	112.17384	120.64277
LSTM con Memoria	93.30360	109.10974	101.20667
LSTM Apiladas con Memoria	98.11095	99.25191	98.68143
GRU 1 Capa	130.38722	126.16218	128.27470

Tabla 4: Raíz del Error Cuadrático Medio (RMSE) para cada Modelo tras 2 ejecuciones, 3 epochs cada una con los mejores valores de los hiperparámetros tras evaluar con el conjunto de test

modelos, para asegurarnos diferencias significativas en el rendimiento de los modelos.

También podemos pensar que habría que explorar modelos LSTM con ventana temporal, apiladas y con memoria. También habría que ver que tamaño de historia funciona mejor, es otro hiperparámetro cuyo valor habría que tratar de optimizar. Además, habría que seguir explorando modelos que usen módulos GRU.

8. Conclusiones

A lo largo de este trabajo se ha desarrollado un pequeño proyecto de Deep Learning desde cero, incluyendo el estudio de los datos, su preprocesamiento y la prueba de varios modelos, entrenándolos con los datos de entrenamiento y evaluados sobre el conjunto de test.

Para cada modelo se han estudiado valores posibles para algunos hiperparámetros, y luego se ha reentrenado cada modelo con los mejores valores de los hiperparámetros previamente obtenidos, y evaluado sobre el conjunto de prueba para comparar cada uno de los modelos.

Bibliografía

[1] Dive Into Deep Learning. Modern Recurrent Neural Networks. https://d2l.ai/chapter_recurrent-modern/index.html. Último acceso el 17/01/2024.

[2] Hadi Fanaee-T. Bike Sharing Dataset. UCI Machine Learning Repository, 2013. DOI: https://doi.org/10.24432/C5W894.

[3] seaborn. Plotting a diagonal correlation matrix. https://seaborn.pydata.org/examples/many_pairwise_correlations. Último acceso el 17/01/2024.

[4] National Institute of Standards and Technology. Measures of Skewness and Kurtosis. https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm. Último acceso el 17/01/2024.

[5] pandas. pandas.get_dummies. https://pandas.pydata.org/docs/reference/api/pandas.get_dummies. Último acceso el 17/01/2024.

[6] scikit-learn. sklearn.preprocessing.MinMaxScaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler. Último acceso el 17/01/2024.

[7] Jason Brownleee. Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/. Último acceso el 17/01/2024.

[8] Keras. Hyperparameter Tuning. https://keras.io/guides/keras_tuner/. Último acceso el 17/01/2024.

[9] Keras. RandomSearch Tuner. https://keras.io/api/keras_tuner/tuners/random/. Último acceso el 17/01/2024.

[10] Keras. Callbacks API. https://keras.io/api/callbacks/. Último acceso el 17/01/2024.

[11] Keras. EarlyStopping. https://keras.io/api/callbacks/early_stopping/. Último acceso el 17/01/2024.

[12] Keras. ReduceLROnPlateau. https://keras.io/api/callbacks/reduce_lr_on_plateau/. Último acceso el 17/01/2024.

[13] Jason Brownleee. Stacked Long Short-Term Memory Networks. https://machinelearningmastery.com/stacked-long-short-term-memory-networks/. Último acceso el 17/01/2024.