Reto "*Atmira Stock Prediction*" - UH 2021

Ánalisis del dataset (*src/exploracion.ipynb*)

Primero comenzamos explorando el *dataset* que nos fue entregado. Para ello usamos la librería pandas del lenguaje de programación *python*. Esta librería nos permite hace un análisis profundo del dataset y poder manipularlo libremente.
Al cargar el dataset como variable de tipo *Dataframe* de pandas mediante el método *read_csv*() nos dimos cuenta de que algunas fechas tenían la posición del mes y del día cambiados, por lo que añadimos al método *read_csv*() un *parser* de fechas para leer todas las fechas con el mismo formato. Mediante el parámetro "*parse_dates*" conseguimos que las fechas sean del tipo *datetime*.

Una vez cargado el *dataset* analizamos su estructura y sus atributos uno a uno para ver su importancia para las predicciones. El *dataset* contiene un total de 4045022 entradas y cada entrada tiene 10 atributos y la variable a predecir, que es las unidades vendidas. Hay un total de 7 atributos, numéricos contando la variable a predecir, 3 de tipo objeto y 1 de tipo fecha. Se procede a analizar los distintos atributos.

Analizando el atributo fecha se puede ver que el dataset recoge un total de 487 días. Se busca si hay algún tipo de relación entre el día de la semana y se observa que los viernes, sábados y domingos se venden más productos de media que el resto de los días. También se busca relación con los meses del año y se observa que en noviembre, diciembre y enero se venden más productos de media, que puede deberse a eventos como el *Black Friday* o la Navidad. Por último se busca si existe relación con que sea principio o final de mes debido al pago de las nóminas, pero se encuentra que no hay mucha relación.

En los atributos que se habían leído como objeto se encuentra el precio, lo cual es erróneo ya que el precio debería ser numérico. Esto es debido a que se utiliza la coma en vez del punto como separador de euros y céntimos, por lo que pandas lo lee mal. Para arreglar este asunto se reemplaza la coma por el punto y asigna el tipo de este atributo a *float*. Los otros dos atributos de tipo objeto son *categoria_uno* y *estado*. Estos atributos son cambiados a tipo *category* que es un tipo más eficiente para tratar con atributos nominales en pandas.

El atributo *categoria_uno* tiene 13 valores distintos y se observa que los productos que más visitas tienen son de los más vendidos. El atributo *categoria_dos* tiene 182 valores distintos y no hemos encontrado una relación directa con el número de unidades vendidas.

El atributo *campaña* toma dos valores distintos que indican si hay campaña o no. Se puede observar que cuando hay campaña no hay casi roturas en comparación con cuando no hay campaña. Esto puede deberse a que cuando hay campaña se está más preparado para las posibles numerosas ventas que se puedan hacer. También se observa que en campaña no

hay días con menos demanda de la habitual y que no es un factor muy determinante para que haya más demanda de la habitual. Hay que resaltar que cuando hay campaña se venden muchos más artículos de media que cuando no hay campaña. Se intenta ver si hay una relación entre el stock del artículo y que sea un día atípico pero no tienen mucha relación.

Por último, se construye una matriz de correlación para ver si algún atributo tiene correlación con las unidades vendidas y como es de esperar el atributo con mayor correlación son las visitas, con casi un 0,4 de correlación, por lo que será un atributo determinante a la hora de entrenar

Preprocesamiento de los datos (*src/prediccion.ipynb*)

Los atributos estado y *dia_atipico* son transformados a codificación *One-Hot* creando un atributo nuevo para cada valor de estos atributos, por lo que se añaden 6 atributos y se quitan 2 del modelo.

Viendo que en el dataset se repetían algunos ids de producto para un mismo día se llegó a la conclusión de que el dataset contenía datos duplicados, en concreto 677904 casos con todos los valores duplicados en una o más filas , haciendo un total de 2004985 filas duplicadas, las cuales eliminamos del dataset, quedando éste reducido a la mitad.

Comprobamos que los atributos *categoria_dos*, *precio* y *antiguedad* tienen *missing values.*
El atributo *categoria_dos* tiene un 0.215% de valores nulos, precio un 68.62% y antigüedad un 23.57%.
En la información del *dataset* nos dicen que para precio hay que completarlo con el valor de la fecha anterior más cercana. Al realizar esta operación se comprueba que hay algunos valores con el precio nulo que no tienen valor anterior no-nulo más próximo. Para esos valores se completa con el precio más antiguo que se tenga constancia.
El atributo *antigüedad* se descarta del dataset porque no hay forma de calcular los valores para los ejemplo que son null y para los ejemplos no null, el valor de *antiguedad* es fijo para el mismo producto independientemente de la fecha.
El atributo *categoria_dos* se rellena mediante *IterativeImputer*, el cual estima el posible valor con la información de las filas de los valores que no son nulos.

Por último, estandarizamos los atributos con *StandardScaler* para tener la misma escala en los atributos.

Tras este proceso se guardan los datos en un fichero intermedio *Modelar_Intermedio_UH2021.txt* que servirá de punto de partido para el script *prediccion.ipynb* que se explica en el siguiente apartado.

Creación de modelos (*src/prediccion.ipynb*)

Previo a la creación de modelos, se realiza una separación de conjuntos para poder ejecutar una serie de entrenamientos y validaciones. Queda de la siguiente manera:

- Entrenamiento – 70%
- Testing – 20%
- Validación – 10%

Se descartan los atributos *unidades_vendidas*, al ser lo que se pretende predecir, *fecha* al no ser un atributo sobre el que se pueda entrenar a un modelo e *id* al no ser un atributo relacionado directamente con cada caso.

Después de la separación de conjuntos se crean tres funciones para poder calcular la función de pérdida sobre la que se juzgará el proyecto. Se utilizará como función de optimización más adelante. Se utilizan las siguientes ecuaciones:

$$loss = (0.7 * rRMSE) + (0.3 * (1 - CF)) \quad \text{(Equation 1)}$$

$$rRMSE = \frac{\sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}}}{\bar{y}} \quad \text{(Equation 2)}$$

$$CF = \frac{\sum_{n}^{i} y_i <= \widehat{y_i}}{n} \quad \text{(Equation 3)}$$

Tras ello, se prueban varios modelos y se incluyen en el script dos que mejores resultados obtienen para los datos. Estos son los modelos de ensamblado *Random Forest Regressor* y Extreme Gradient Boosting.

A su vez, se prueban varios modelos basados en redes neuronales secuenciales (con distinto número de capas y distinto número de neuronas para cada una) hasta encontrar uno que obtiene unos resultados que superan al resto. Consiste en un modelo de 3 capas ocultas que se puede ver con más detalle en el diagrama de la derecha.

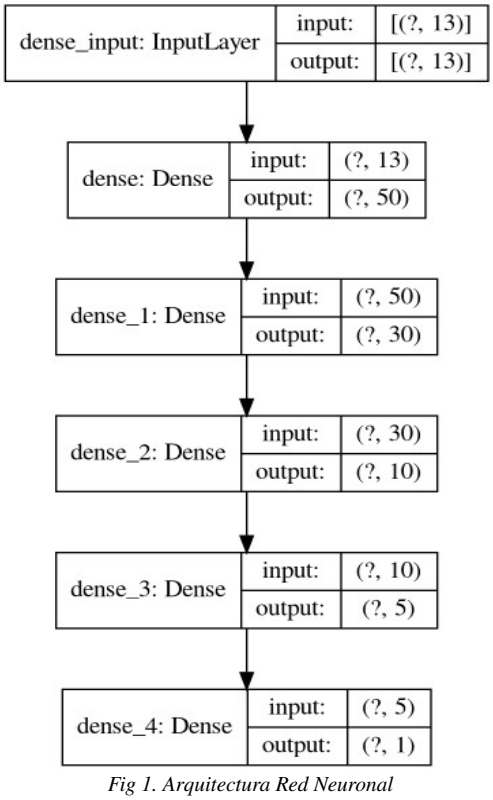

Fig 1. Arquitectura Red Neuronal

Tras un *tuneo* de hiperparámetros del modelo (entre los que tenemos el "learning rate" del optimizador de nuestra

red neuronal) observamos que se obtienen unas predicciones que suelen pasarse en cuanto al número de unidades vendidas. Esto tiene sentido al analizar la ecuación 1 anterior: para reducir esta función se deberá maximizar los casos favorables (ecuación 3) por lo que siempre será óptimo no quedarse corto. Esto penalizará a rRMSE (ecuación 2), pero al dividirse por la media de los valores reales, la penalización queda reducida.

Por ello, para obtener unos resultados más próximos a la realidad se debe sustituir rRMSE por RMSE, o dicho de otro modo, no dividir por la media de los valores reales. Por otro lado, en las instrucciones del concurso se indica sobre la ecuación 1 que "*[...] Está será la métrica a minimizar: **(0.7 * rRMSE) + (0.3 * (1 - CF))**, siendo los mejores trabajos los que consigan los valores más reducidos de dicha métrica. [...]*". Por ello, el equipo decide mantener esta función como función de pérdida con el objetivo de obtener mejor calificación.

En los siguientes gráficos podemos observar una comparación entre utilizar rRMSE en la ecuación 1 con RMSE:

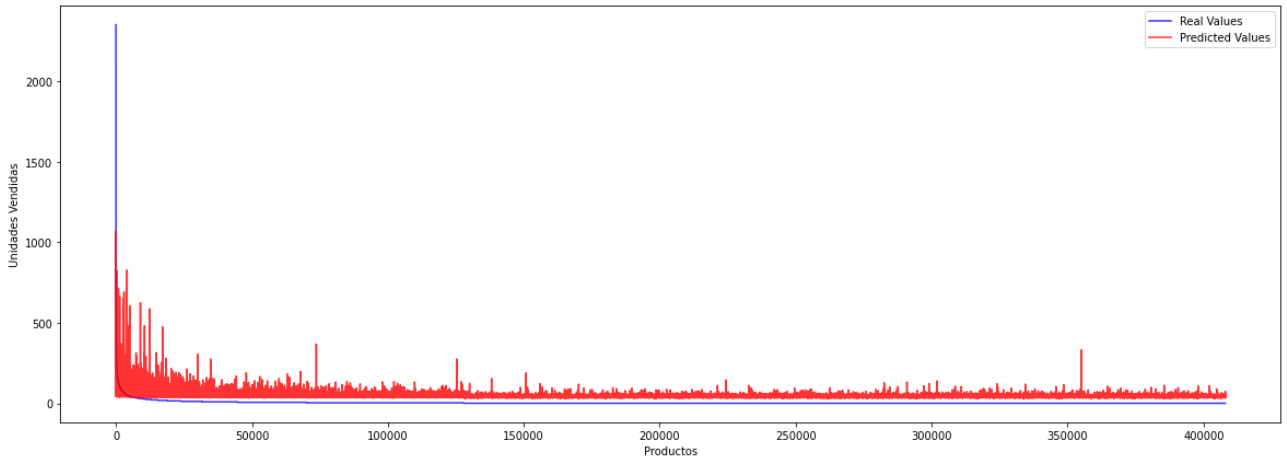

Fig 2. rRMSE en ecuación 1

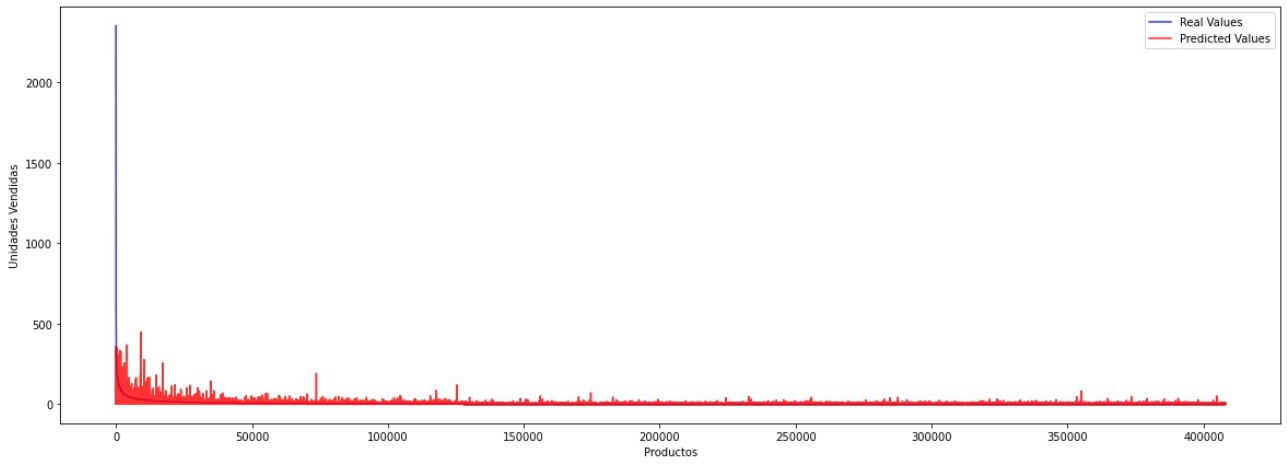

Fig 3. RMSE en ecuación 1

Se observa fácilmente como para la figura 3 se obtienen unos resultados más próximos a los reales. Aunque este método obtiene peores resultados para la ecuación 1.

A la hora de predecir los datos de "*Estimar2.txt*" se debe de volver a realizar el preprocesamiento de datos explicado en el apartado 2, tras lo cual se utiliza nuestro modelo de la figura 1 para entrenarse con los datos de entrenamiento.

Futuro trabajo

Como posible futuro trabajo se plantean los siguientes objetivos:

- Automatizar el proceso de preparación de datos.
- Aumentar la precisión de las predicciones intentando concentrar el esfuerzo de los modelos en la medida rRMSE (y menos en CF) para así no siempre obtener una cifra de unidades mucho mayor a la real.
- Buscar nuevos modelos basados en redes neuronales no sólo reduciendo a redes neuronales secuenciales.
- Buscar nuevas funciones de pérdida que puedan aumentar la precisión del modelo.

Nota

- Los ejecutables deben situarse dentro de un directorio *src/* dentro de la carpeta raíz y los ficheros de datos en otro *data/* en el directorio raíz.
- Recomendamos analizar detalladamente los scripts de jupyter para poder ver el proceso de realización en más detalle.
- Las principales dependencias del proyecto son:
  - Python 3.8
  - Numpy
  - Pandas
  - Matplotlib
  - Keras / Tensorflow
  - Seaborn
- En el caso de tener alguna duda, escribir a:
  - j.enriquezballe@gmail.com / jaime.enriquez@estudiante.uam.es
  - jorgedl25@gmail.com