Predicción de ventas futuras aplicando modelos de Machine Learning

Master Universitario en Business Analytics & Big Data

Ficha de entrega : 22/09/2020

Estudiante: Hang Li

Docente: Juan Manuel Moreno Lamparero

Índice

Tipo de trabajo

Este es un trabajo de Data Science, orientado a la adquisición, procesado y analítica sobre datasets interesantes.
Los datos adquiridos son proveídos por una de las compañías de software más grandes de Rusia, 1C company, que corresponde un historial de la venta diaria de 60 tiendas diferentes durante el periodo entre enero de 2013 y octubre de 2015.
Procesando estos datos, se pretende predecir la cantidad total de productos que serán vendidos durante el noviembre de 2015, por cada una de las tiendas y por cada una de los productos.
Para más información:
https://www.kaggle.com/c/competitive-data-science-predict-future-sales/overview

Introducción

Actualmente, el estudio de la cantidad de ventas futuras de cada una de los productos tiene un gran interés para reducir el coste de la producción, del almacenamiento y de la logística en el Supply Chain. La producción excesiva comporta una pérdida irrecuperable de tiempo y recursos la cual todas las empresas quieren evitar este problema. Si es posible predecir la cantidad de los productos vendidos para un futuro próspero, todas las empresas podrán optimizar su producción y suministro de tal manera que maximiza sus beneficios.

Objetivos de aprendizaje

El objetivo principal es aplicar los conocimientos adquiridos durante el estudio del Máster en datos del mundo real, y estudiar cuál es el modelo de machine learning más apropiado para predecir ventas futuras.

Resultados esperados

Como resultado se espera poder obtener un modelo eficaz para predecir la cantidad total de productos que serán vendidos, por cada una de las tiendas y por cada una de los productos, analizando los datos de ventas históricas.

Asignaturas relacionados con los objetivos y resultados

- Fundamentos de Tratamiento de Datos para Data Science
- Modelos y Aprendizaje Estadístico usando R
- Aprendizaje Automático Aplicado
- Inteligencia de Negocio y Visualización
- Almacenamiento e Integración de Datos
- Valor y Contexto de la Analítica Big Data
- Aplicaciones Analíticas

Métodos, materiales y tecnologías de uso potencial.

- Análisis exploratorio de datos
- Probabilidad e inferencia estadística
- Modelos lineales y aprendizaje estadístico
- GLMS y series temporales
- Modelos supervisados
- Etc.

Memoria explicativa de los proceso del desarrollo

Estudio de los datos

Para implementar nuestro modelo de predicción, es muy importante entender los datos antes de nada.

En total tenemos 6 archivos csv. A continuación vamos a examinarlos uno en uno:

sales_train.csv : este es el archivo que contiene los datos históricos de venta de todas las tiendas de todos los productos entre 1/2013 i 10/2015.
items.csv : este archivo nos indica por cada ID del artículo el nombre del artículo y el ID de la categoría que pertenece.
item_categories.csv : por cada ID de la categoría, el nombre de la categoría correspondiente.
shops.csv : por cada ID de la tienda, el nombre de la tienda.
sample_submission.csv y test.csv : nos dice el formato de los resultados que queremos predecir para ser evaluado por Kaggle.

Entre los 6 archivos csv, podemos destacar los siguientes variables:
ID : un id que representa una tupla de tienda y artículo en el test set
shop_id : número identificador de tienda
item_id : número identificador de artículo
item_category_id : número identificador de categoría del artículo
item_cnt_day : artículos vendidos en el día correspondiente
item_price : precio del artículo en el momento de la venta
date : dia de la venta en formato dd/mm/yyyy
data_block_num : un número consecutivo que indica el mes, contando desde 01/2013
item_name : nombre del artículo
shop_name : nombre de la tienda
item_category_name : nombre de la categoría del artículo

Entre todas estas variables, podemos clasificarlas en siguiente grupos:

- Números identificadores (Id)
- Fecha
- Nombre
- Variable numérico

Grupo	Variable	Tipo de data	Tipo de variable
Id	ID	int	Categórico
Id	shop_id	int	Categórico
Id	item_id	int	Categórico
Id	item_category_id	int	Categórico
Fecha	date	datetime	Ordinal
Fecha	data_block_num	int	Ordinal
Nombre	item_name	string	Categórico
Nombre	shop_name	string	Categórico
Nombre	item_category_name	string	Categórico
Variable	item_price	float	Numérico
Variable	item_cnt_day	int	Numérico

A primera vista, solo tenemos 2 variables numéricas que se relacionan con las ventas que queremos predecir, pero con tratamientos apropiados, las variables categóricas y ordinales también contienen mucha información.

Estudio gráfico

Después de importar librerías, convertimos los datos de los archivos en pandas.dataframes.

```
#Read all csv files to dataframe
df_items = pd.read_csv("../input/competitive-data-science-predict-future-sales/items.csv")
df_sales_train = pd.read_csv("../input/competitive-data-science-predict-future-sales/sales_train.csv")
df_item_categories = pd.read_csv("../input/competitive-data-science-predict-future-sales/item_categories.csv")
df_test = pd.read_csv("../input/competitive-data-science-predict-future-sales/test.csv")
df_shops = pd.read_csv("../input/competitive-data-science-predict-future-sales/shops.csv")
df_sample_submission = pd.read_csv("../input/competitive-data-science-predict-future-sales/sample_submission.csv")
```

y por cada uno miramos sus contenidos.

```
df_shops.head()
```

:

	shop_name	shop_id
0	!Якутск Орджоникидзе, 56 фран	0
1	!Якутск ТЦ "Центральный" фран	1
2	Адыгея ТЦ "Мега"	2
3	Балашиха ТРК "Октябрь-Киномир"	3
4	Волжский ТЦ "Волга Молл"	4

```
df_sales_train.head()
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
0	02.01.2013	0	59	22154	999.00	1.0
1	03.01.2013	0	25	2552	899.00	1.0
2	05.01.2013	0	25	2552	899.00	-1.0
3	06.01.2013	0	25	2554	1709.05	1.0
4	15.01.2013	0	25	2555	1099.00	1.0

```
df_item_categories.head()
```

	item_category_name	item_category_id
0	PC - Гарнитуры/Наушники	0
1	Аксессуары - PS2	1
2	Аксессуары - PS3	2
3	Аксессуары - PS4	3
4	Аксессуары - PSP	4

```
df_items.head()
```

	item_name	item_id	item_category_id
0	! ВО ВЛАСТИ НАВАЖДЕНИЯ (ПЛАСТ.) D	0	40
1	!ABBYY FineReader 12 Professional Edition Full...	1	76
2	***В ЛУЧАХ СЛАВЫ (UNV) D	2	40
3	***ГОЛУБАЯ ВОЛНА (Univ) D	3	40
4	***КОРОБКА (СТЕКЛО) D	4	40

```
df_test.head()
```

	ID	shop_id	item_id
0	0	5	5037
1	1	5	5320
2	2	5	5233
3	3	5	5232
4	4	5	5268

```
df_sample_submission.head()
```

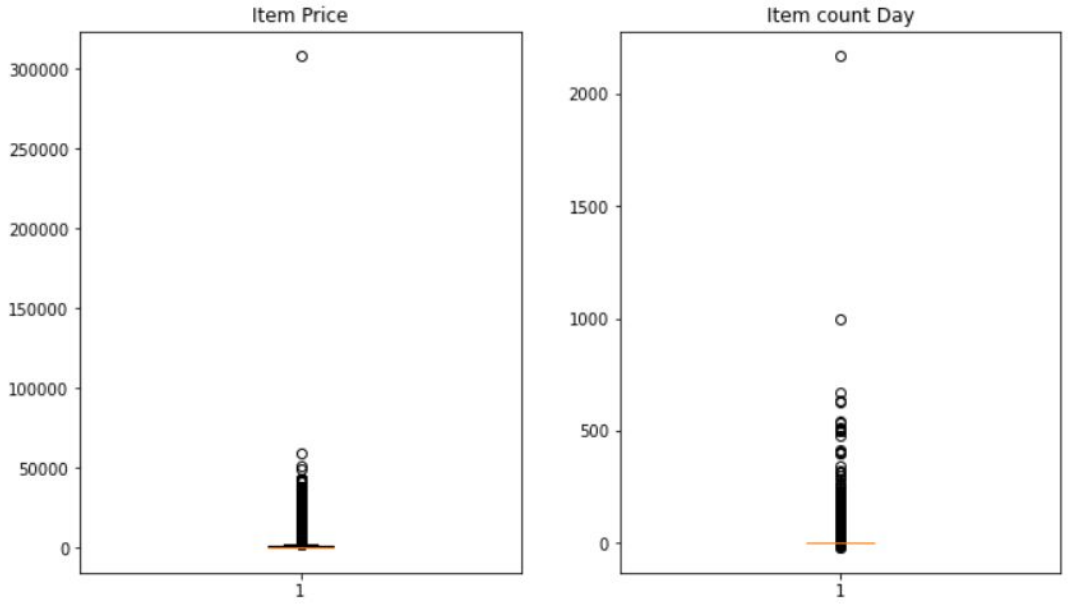
:

	ID	item_cnt_month
0	0	0.5
1	1	0.5
2	2	0.5
3	3	0.5
4	4	0.5

Estudio de irregularidades

Examinamos las 2 variables numéricas que tenemos : item_nt_day y item_price con 2 boxplot

```
plt.figure(figsize=(10,6))
plt.subplot(121)
plt.boxplot(df_sales_train['item_price'])
plt.title('Item Price')
plt.subplot(122)
plt.boxplot(df_sales_train['item_cnt_day'])
plt.title('Item count Day')
plt.tight_layout(pad=3)
plt.show()
```



Y vemos que la mayoría de los artículos tienen precios entre 0 y 60000, y la mayoría de artículos se vende menos de 1000 por dia, pero hay artículos con precios exagerados o vendas excesivas.

```
negp = df_sales_train[df_sales_train['item_price']> 50000]
print(negp)
```

```
             date  date_block_num  shop_id  item_id  item_price  \
885138   17.09.2013              8       12    11365     59200.0
1163158  13.12.2013             11       12     6066    307980.0
1488135  20.03.2014             14       25    13199     50999.0

         item_cnt_day
885138            1.0
1163158           1.0
1488135           1.0
```

Listamos los artículos más caros i comparamos con su precio mediano

```
median = df_sales_train[(df_sales_train.item_id==11365)]['item_price'].median()
print(median)
```

```
1203.75
```

```
median = df_sales_train[(df_sales_train.item_id==6066)]['item_price'].median()
print(median)
```

```
307980.0
```

```
median = df_sales_train[(df_sales_train.item_id==13199)]['item_price'].median()
print(median)
```

```
50999.0
```

El resultado nos dice que el artículo con id = 11365 tiene un precio muy menor, mientras que el artículo con id = 6066 y id = 13199 no sabemos si su precio es una irregularidad porque solo ha sido vendido una vez y no tenemos más datos. No sabemos las unidades de item_price por lo tanto no podemos ver si el precio es realmente razonable. A pesar de que sea así, eliminaría estas 3 filas porque considero que son resultados inusuales y por lo tanto no serían buenos para hacer regresiones.

El mismo argumento por los valores de las vendas, eliminaría los que son demasiados altos por que los considero como resultados no reproducibles.

```
#clean irregular data
df_sales_train = df_sales_train[(df_sales_train['item_price']<50000) & (df_sales_train['item_cnt_day']<1000)]
```

Además de estos, vemos valores negativos en las dos gráficas, las ventas negativas pueden ser interpretados como devoluciones, pero los productos con precios negativos no pueden ser interpretados, los eliminamos del dataframe.

```
#Negative values
df_sales_train = df_sales_train[df_sales_train['item_price'] > 0]
```

Estudio de shop_name

```
#read the shop names
df_shops['shop_name'].groupby(df_shops.shop_id).value_counts()
```

12]:

```
shop_id  shop_name
0        !Якутск Орджоникидзе, 56 фран                    1
1        !Якутск ТЦ "Центральный" фран                    1
2        Адыгея ТЦ "Мега"                                 1
3        Балашиха ТРК "Октябрь-Киномир"                   1
4        Волжский ТЦ "Волга Молл"                         1
5        Вологда ТРЦ "Мармелад"                           1
6        Воронеж (Плехановская, 13)                       1
7        Воронеж ТРЦ "Максимир"                           1
8        Воронеж ТРЦ Сити-Парк "Град"                     1
9        Выездная Торговля                                1
10       Жуковский ул. Чкалова 39м?                       1
11       Жуковский ул. Чкалова 39м²                       1
12       Интернет-магазин ЧС                              1
13       Казань ТЦ "Бехетле"                              1
14       Казань ТЦ "ПаркХаус" II                          1
40       РостовНаДону ТРК "Мегацентр Горизонт" Островной  1
41       РостовНаДону ТЦ "Мега"                           1
42       СПб ТК "Невский Центр"                           1
43       СПб ТК "Сенная"                                  1
44       Самара ТЦ "Мелодия"                              1
45       Самара ТЦ "ПаркХаус"                             1
46       Сергиев Посад ТЦ "7Я"                            1
47       Сургут ТРЦ "Сити Молл"                           1
48       Томск ТРЦ "Изумрудный Город"                     1
49       Тюмень ТРЦ "Кристалл"                            1
50       Тюмень ТЦ "Гудвин"                               1
51       Тюмень ТЦ "Зеленый Берег"                        1
52       Уфа ТК "Центральный"                             1
53       Уфа ТЦ "Семья" 2                                 1
54       Химки ТЦ "Мега"                                  1
55       Цифровой склад 1С-Онлайн                         1
56       Чехов ТРЦ "Карнавал"                             1
57       Якутск Орджоникидзе, 56                          1
58       Якутск ТЦ "Центральный"                          1
59       Ярославль ТЦ "Альтаир"                           1
Name: shop_name, dtype: int64
```

En la tabla de df_shops, en total tenemos 60 filas, miramos y vemos que algunas de ellas son repetitivas, la tienda 10 es la misma que la 11, la 0 y 1 son las mismas que 57 y 58. Por este motivo combinamos estos datos para no calcularlas muchas veces.

```
#We found repetitive names, so we can combine them
df_sales_train.loc[df_sales_train.shop_id == 0,'shop_id']=57
df_test.loc[df_test.shop_id == 0,'shop_id']=57
df_sales_train.loc[df_sales_train.shop_id == 1,'shop_id']=58
df_test.loc[df_test.shop_id == 1,'shop_id']=58
df_sales_train.loc[df_sales_train.shop_id == 10,'shop_id']=11
df_test.loc[df_test.shop_id == 10,'shop_id']=11
```

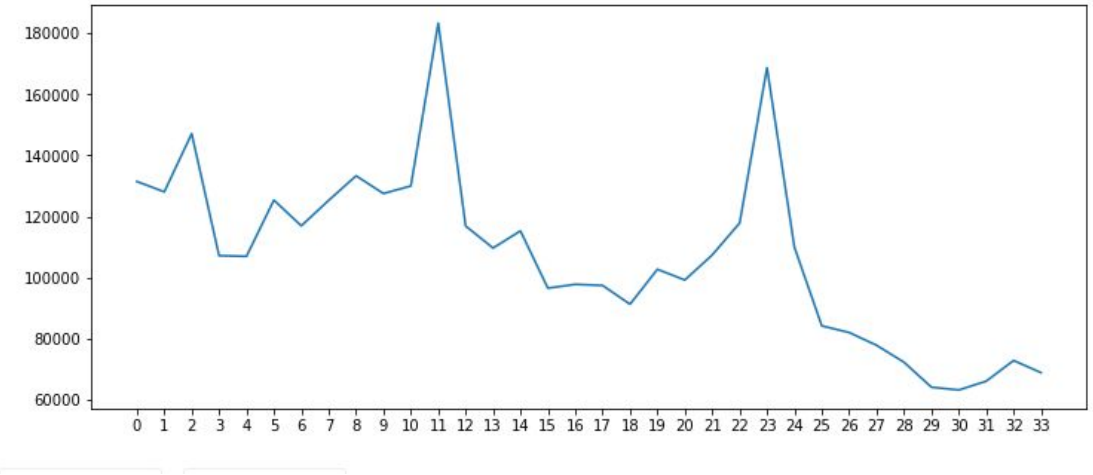
Estudio de la estacionalidad y tendencia

Para hacer cálculos, combinamos el dataframe de df_sales_train con df_items.

```
#add the item_ID we want to predict
df_sales_train = pd.merge(df_sales_train,df_items,how='left', on=['item_id'])
df_sales_train.fillna(0,inplace=True)
```

```
df_new = df_sales_train.groupby(['date_block_num'])["item_cnt_day"].sum()
plt.figure(figsize=(12,5))
plt.plot(df_new)
plt.xticks(np.arange(0, 34, 1.0))
```

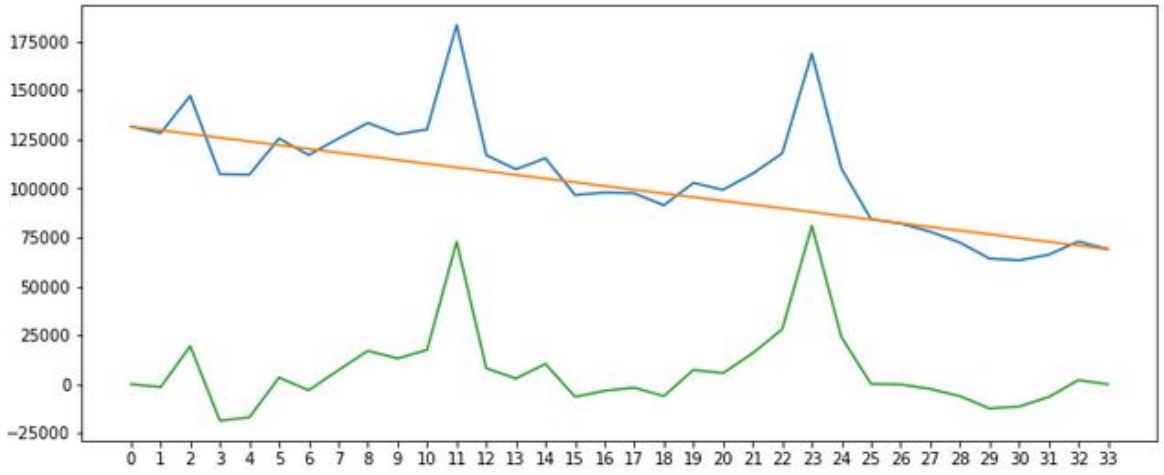
Hacemos un recuento de todo los artículos vendidos y vemos que hay 2 picos cuando date_block_num = 11 y 23, estos 2 valores corresponden diciembre de 2013 y diciembre de 2014, época del año con muchos festivos y vacaciones. Además, la tendencia de la venta se va bajando, se ha vendido menos en 2014 que en 2013. El motivo de la bajada de la cantidad de venta no es objetivo de nuestro estudio, puede ser causado por subida de precios, impacto de venta online, etc. Pero de momento lo dejamos a parte. Lo que nos interesa es saber que existe estacionalidad en la venta de productos.



Dibujamos una línea interpolando el valor de primer mes con el último, y obtenemos la tendencia y la estacionalidad de los valores.

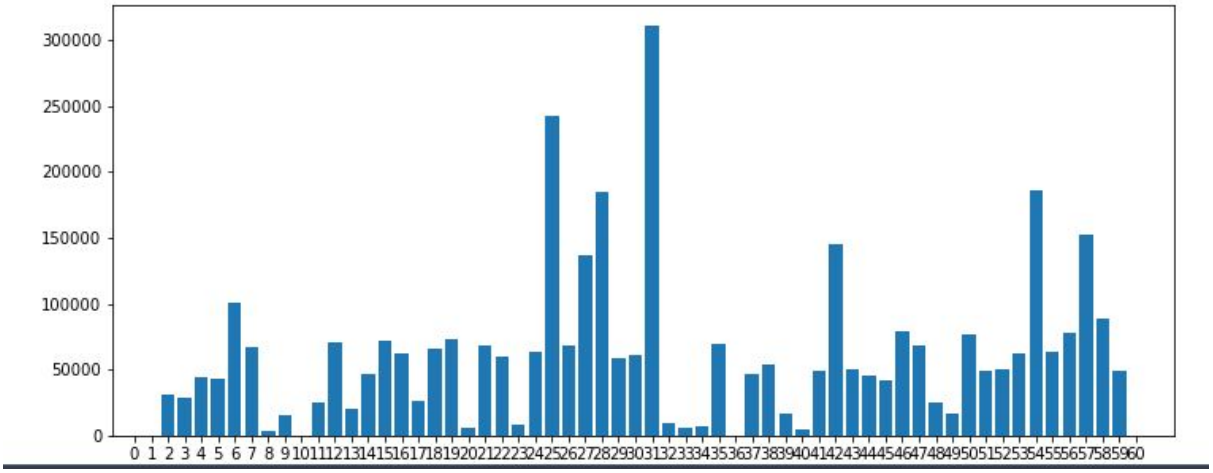
```
b = (month_sale[33,1]-month_sale[0,1])/(month_sale[33,0]-month_sale[0,0])
month_sale_trend = np.zeros((34,2))
month_sale_average = np.zeros((34,2))
for x in range(34):
    y = x*b+month_sale[0,1]
    month_sale_trend[x,0] = x
    month_sale_trend[x,1] = y
    month_sale_average[x,0] = x
    month_sale_average[x,1] = month_sale[x,1] - y
del b,x,y
```

```
plt.figure(figsize=(12,5))
plt.plot(month_sale[:,0],month_sale[:,1])
plt.plot(month_sale_trend[:,0],month_sale_trend[:,1])
plt.plot(month_sale_average[:,0],month_sale_average[:,1])
plt.xticks(np.arange(0, 34, 1.0))
plt.show()
```



Estudio de la tienda más visitada

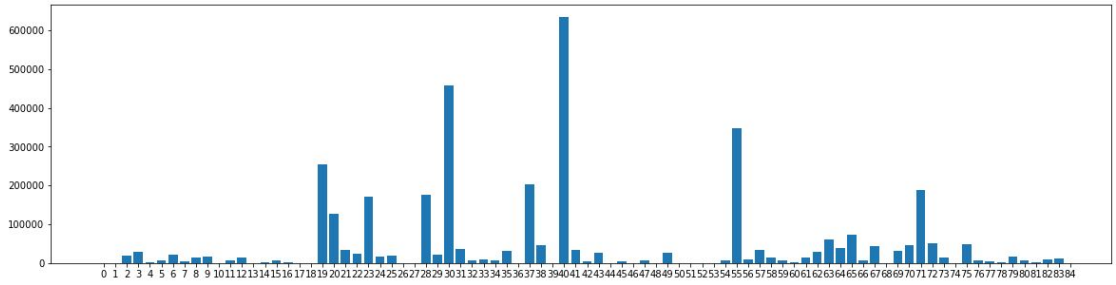
```
df_new = df_sales_train.groupby(['shop_id'], as_index = False)["item_cnt_day"].sum().to_numpy
plt.figure(figsize=(12,5))
plt.bar(df_new[:,0],df_new[:,1])
plt.xticks(np.arange(0, 61, 1))
plt.show()
```



Aquí vemos que unas tiendas venden más que otras, quizá porque están mejor situadas, quizá porque son tiendas que hacen mucha publicidad, lo que nos importa es que el shop_id lleva información interesante.

Estudio de categoria_id

```
df_new = df_sales_train.groupby(['item_category_id'], as_index = False)["item_cnt_day"].sum().to_numpy()
plt.figure(figsize=(20,5))
plt.bar(df_new[:,0],df_new[:,1])
plt.xticks(np.arange(0, 85, 1))
plt.show()
```

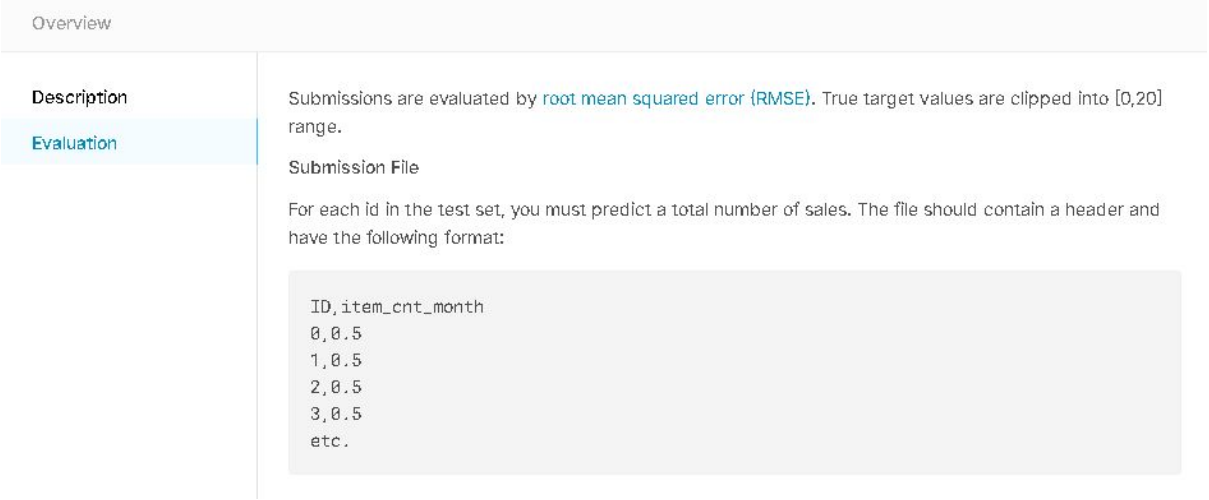


El Id de la categoría del producto también se tiene que tener en cuenta.

Implementación del modelo

Validación de los modelos

Antes de entrenar un modelo, es importante saber cómo comprobar su eficacia. En las competencias de Kaggle, se puede comprobar el resultado entregando los resultados según el formato indicado.



En esta competencia, los resultados serán evaluados con algoritmo RMSE.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Aunque es posible evaluar los resultados con el propio Kaggle, evaluando solo mirando los valores de leaderboard puede causar overfitting. Así que crearemos un subset de datos para evaluar nuestros modelos, para hacer validación, lo importante es que los datos de la validación sea similar que los datos que queremos predecir. En este caso, como que los datos tiene componente cronológico, no se pueden escoger aleatoriamente, cogeremos los datos del octubre del 2015 (date_block_num = 33) como dataset de validación.

Tratamiento de los datos

¿Cómo podemos predecir las ventas del próximo mes? Mi primera intuición es hacer una regresión con los datos de volumen de venta de los meses anteriores.
Añadimos el date_block_num 34, el número identificador del mes que corresponde a noviembre de 2015.

```
#add the test ID we want to predict
df_test['date_block_num'] = 34
df_sales_train = pd.concat([df_sales_train,df_test],ignore_index=True,
                           sort = False,keys = ['date_block_num','shop_id','item_id'])
df_sales_train.fillna(0,inplace=True)
df_sales_train.tail()
```

:

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_name	item_category_id	ID
3150038	0	34	45	18454	0.0	0.0	0	0.0	214195.0
3150039	0	34	45	16188	0.0	0.0	0	0.0	214196.0
3150040	0	34	45	15757	0.0	0.0	0	0.0	214197.0
3150041	0	34	45	19648	0.0	0.0	0	0.0	214198.0
3150042	0	34	45	969	0.0	0.0	0	0.0	214199.0

Una vez hemos creado date_block_num 34, creamos una nueva dataframe que contiene el id de todas las tiendas y todos los artículos por cada mes.

```
# Create "grid" with columns
index_cols = ['shop_id', 'item_id', 'date_block_num']

# For every month we create a grid from all shops/items combinations from that month
grid = []
for block_num in df_sales_train['date_block_num'].unique():
    cur_shops = df_sales_train.loc[df_sales_train['date_block_num'] == block_num, 'shop_id'].unique()
    cur_items = df_sales_train.loc[df_sales_train['date_block_num'] == block_num, 'item_id'].unique()
    grid.append(np.array(list(product(*[cur_shops, cur_items, [block_num]])),dtype='int16'))

# Turn the grid into a dataframe
grid = pd.DataFrame(np.vstack(grid), columns = index_cols,dtype=np.int32)
grid.head()
```

9]:

	shop_id	item_id	date_block_num
0	59	22154	0
1	59	2552	0
2	59	2554	0
3	59	2555	0
4	59	2564	0

Contamos los artículos vendidos por cada tienda en cada mes, y añadimos todo los datos que queremos procesar en un dataframe llamado all_data.

```
#Create a column with month count
df_new = df_sales_train.groupby(index_cols,as_index=False)["item_cnt_day"].sum()
df_new = df_new.rename(columns = {"item_cnt_day": "target"})
all_data = pd.merge(grid, df_new, how='left', on=index_cols).fillna(0)
all_data.head()
```

2]:

	shop_id	item_id	date_block_num	target
0	59	22154	0	1.0
1	59	2552	0	0.0
2	59	2554	0	0.0
3	59	2555	0	0.0
4	59	2564	0	0.0

Para añadir más variables para entrenar, considero que shop_id y item_id son variables valorables. Las tiendas situadas en grandes ciudades atraen más visitantes, y los artículos de uso diario, pilas, por ejemplo, se venden más que artículos durables. Pero estas son variables categóricas, una tienda con índice 50 no es necesariamente más rentable que una tienda con índice 5. Para transformar variables categóricas en variables numéricas, el método más usado es el one-hot-encoding, que consiste en la creación de variables dummies de 0 y 1 para cada tipo de valor categórico diferente. Pero tenemos un problema para usar one-hot-encoding: tenemos miles y miles de valores diferentes en item_id! Crearemos un dataframe con diez mil columnas de 0 y 1 si usamos one-hot-encoding, no tenemos una computadora con suficientes capacidades. Para evitar este problema he optado por otro método llamado “target encoding”.

Target encoding

El método target encoding, también llamado mean encoding, consiste en reemplazar valores de una característica categórica en valores cuantitativos usando el valor de la “target”. Explicaré con ejemplos:

```
#Create a column with mean encoded shop_id
df_new = df_sales_train.groupby(['shop_id', 'date_block_num'],as_index=False)["item_cnt_day"].sum()
df_new = df_new.rename(columns = {"item_cnt_day": "target_shop"})
all_data = pd.merge(all_data, df_new, how='left', on=['shop_id', 'date_block_num']).fillna(0)
all_data.head()
```

]:

	shop_id	item_id	date_block_num	target	target_shop
0	59	22154	0	1.0	2017.0
1	59	2552	0	0.0	2017.0
2	59	2554	0	0.0	2017.0
3	59	2555	0	0.0	2017.0
4	59	2564	0	0.0	2017.0

Cada mes, contamos los productos totales vendidos por cada tienda, y creamos una columna llamado target_shop, cada valor del target_shop corresponde un valor de la columna shop_id, pero los valores del target_shop son valores cuantitativos, porque es el número de artículos vendidos en ese mes. Una tienda con target_shop= 200, ha vendido el doble que una tienda con target_shop=100. De este modo convertimos la característica shop_id, una variable categórica no apto por regresión, a una variable numérica cuantitativa, preparada para una regresión.

Lo mismo por item_id:

```
#Create a column with mean encoded item_id
df_new = df_sales_train.groupby(['item_id', 'date_block_num'],as_index=False)["item_cnt_day"].sum()
df_new = df_new.rename(columns = {"item_cnt_day": "target_item"})
all_data = pd.merge(all_data, df_new, how='left', on=['item_id', 'date_block_num']).fillna(0)
all_data.head()
```

4]:

	shop_id	item_id	date_block_num	target	target_shop	target_item
0	59	22154	0	1.0	2017.0	18.0
1	59	2552	0	0.0	2017.0	0.0
2	59	2554	0	0.0	2017.0	1.0
3	59	2555	0	0.0	2017.0	2.0
4	59	2564	0	0.0	2017.0	5.0

Ahora tenemos 3 columnas nuevas para entrenar el modelo.

```
#Names of new columns
cols_to_rename = list(all_data.columns.difference(index_cols))
cols_to_rename
```

```
['target', 'target_item', 'target_shop']
```

Para estudiar el volumen de ventas de un mes, alistamos los volúmenes de ventas de los meses anteriores en la misma fila. Hemos cogido los datos de 5 meses más recientes y el mismo mes del año pasado, teniendo en cuenta la estacionalidad de los datos.

```
shift_range = [1, 2, 3, 4, 5, 12]

for month_shift in shift_range:

    train_shift = all_data[index_cols + cols_to_rename].copy()

    train_shift['date_block_num'] = train_shift['date_block_num'] + month_shift

    foo = lambda x: '{}_lag_{}'.format(x, month_shift) if x in cols_to_rename else x
    train_shift = train_shift.rename(columns=foo)

    all_data = pd.merge(all_data, train_shift, on=index_cols, how='left').fillna(0)

del train_shift
```

El resultado de tal procesamiento es el siguiente:

```
all_data.tail().transpose()
```

	11127951	11127952	11127953	11127954	11127955
shop_id	45.0	45.0	45.0	45.0	45.0
item_id	18454.0	16188.0	15757.0	19648.0	969.0
date_block_num	34.0	34.0	34.0	34.0	34.0
target	0.0	0.0	0.0	0.0	0.0
target_shop	0.0	0.0	0.0	0.0	0.0
target_item	0.0	0.0	0.0	0.0	0.0
target_lag_1	1.0	0.0	0.0	0.0	0.0
target_item_lag_1	2.0	1.0	5.0	2.0	3.0
target_shop_lag_1	702.0	702.0	702.0	702.0	702.0
target_lag_2	0.0	0.0	0.0	0.0	0.0
target_item_lag_2	1.0	3.0	3.0	3.0	5.0
target_shop_lag_2	654.0	654.0	654.0	654.0	654.0
target_lag_3	0.0	0.0	0.0	0.0	0.0
target_item_lag_3	3.0	0.0	4.0	7.0	1.0
target_shop_lag_3	710.0	0.0	710.0	710.0	710.0
target_lag_4	0.0	0.0	0.0	0.0	0.0
target_item_lag_4	12.0	0.0	4.0	2.0	2.0
target_shop_lag_4	675.0	0.0	675.0	675.0	675.0
target_lag_5	0.0	0.0	0.0	0.0	0.0
target_item_lag_5	19.0	0.0	8.0	4.0	2.0
target_shop_lag_5	622.0	0.0	622.0	622.0	622.0
target_lag_12	0.0	0.0	0.0	0.0	0.0
target_item_lag_12	0.0	0.0	9.0	0.0	6.0
target_shop_lag_12	0.0	0.0	1251.0	0.0	1251.0

Para visualizar el nombre de todas las columnas hemos transpuesto el dataframe.

En la dataframe que hemos creado, para cada combinación de artículo, tienda i mes, tenemos una serie de datos de su volumen de venta de meses previos.Pero como que no tenemos datos del 2012, no podemos predecir nada del 2013. Eliminamos las filas del año 2013 del dataframe.

```
# Don't use old data from year 2013
all_data = all_data[all_data['date_block_num'] >= 12]
```

Antes de entrenar el modelo, definimos los columnas que no serán entrenados.

```
# We will drop these at fitting stage
to_drop_cols = ['target', 'target_shop', 'target_item', 'date_block_num']
```

La columna date_block_num es una columna con valores ordinales que el mes que va pasando desde 01/2013. No podemos comparar nada sabiendo que una fila que tiene date_block_num=30 y otra date_block_num=15.
Las columnas target, target_shop, target_item son columnas de las cuales no tenemos información de su valor y las deseamos predecir, por lo tanto también las quitamos cuando hacemos entrenamiento.

Para añadir más información, adjuntamos la columna de item_category_id a la tabla y guardamos los datos en un fichero pickle.

```
# Category for each item
item_category_mapping = df_items[['item_id','item_category_id']].drop_duplicate
s()

all_data = pd.merge(all_data, item_category_mapping, how='left', on='item_id')
all_data.to_pickle("data.pkl")
del all_data
```

Entrenamiento de los modelos

Aunque la columna date_block_num no sirve para entrenar datos, sirve para separar el dataset en train y test.

```
# Save `date_block_num`, as we can't use them as features, but will need them t
o split the dataset into parts
all_data = pd.read_pickle("data.pkl")
dates = all_data['date_block_num']
```

```
dates_train = dates[dates < 33]
dates_target  = dates[dates == 33]

X_train = all_data.loc[dates <  33].drop(to_drop_cols, axis=1)
X_test =  all_data.loc[dates == 33].drop(to_drop_cols, axis=1)

y_train = all_data.loc[dates <  33, 'target'].values
y_test =  all_data.loc[dates == 33, 'target'].values
```

Modelo regresión lineal

Primero de todo, entrenamos los datos por regresión lineal. El modelo de regresión lineal es el modelo más básico de todos. Es muy rápido y consume pocos recursos, ideal para una prueba superficial.

```
#we will run linear regression on numeric columns and get predictions for the l
ast month.

lr = LinearRegression()
lr.fit(X_train.values, y_train)
pred_lr = lr.predict(X_test.values)
mse = mean_squared_error(pred_lr, y_test, squared=False)
print(mse)
```

```
1.9408528795254338
```

Obtenemos un RMSE=1.94, no sabemos si este valor es bueno o no, así que entrenamos otro modelo.

Modelo light GBM (gradient boosting)

Aplicamos un algoritmo basado en árboles de decisión: lightGBM. Los algoritmos basados en árboles de decisión divide los datos con clasificación continua según sus diferentes características, por lo tanto, su algoritmo es muy diferente de las regresiones, que estudian la dependencia del variable objetivo con los variables entrenados.

Los métodos de gradient boosting, es una técnica basada en árboles de decisión, que consiste en la optimización gradual haciendo que cada modelo sea construido teniendo en cuenta el previo resultado.

Dentro de los métodos de gradient boosting, hemos escogido lightGBM porque es conocido como una librería de alta eficiencia y gran velocitat.

```
#lgb model
lgb_params = {

             'feature_fraction': 0.75,
             'metric': 'rmse',
            'nthread':1,
            'min_data_in_leaf': 2**7,
            'bagging_fraction': 0.75,
            'learning_rate': 0.03,
            'objective': 'mse',
            'bagging_seed': 2**7,
            'num_leaves': 2**7,
            'bagging_freq':1,
            'verbose':0
           }

model = lgb.train(lgb_params, lgb.Dataset(X_train, label=y_train), 100)
pred_lgb = model.predict(X_test)
mse = mean_squared_error(pred_lgb, y_test, squared=False)
print(mse)
```

```
1.987378854420746
```

El resultado de aplicar el modelo de lightGBM es RMSE=1,98. El resultado es más o menos parecido al modelo de la regresión lineal.

Modelos de segundo nivel

Para mejorar los resultados, podemos probar de combinar los diferentes modelos que hemos probado, aplicando modelos de segundo nivel.

Los modelos de segundo nivel, como su nombre indica, son modelos entrenados con resultados de modelos del primer nivel. En otras palabras, después de predecir unos valores aplicando a varios modelos, podemos usar estos valores para hacer otro entrenamiento, para ver cual modelo es el mejor en una situación y otra para darle más peso.

Guardamos los resultados de la predicción de regresión lineal y lightGBM.

```
X_test_level2 = np.c_[pred_lr, pred_lgb]
X_test_level2
```

```
array([[0.12629944, 0.05817722],
       [0.79221693, 0.82198447],
       [0.9967755 , 0.5318341 ],
       ...,
       [0.10347263, 0.21777319],
       [0.12692807, 0.06040953],
       [0.0498505 , 0.0438579 ]])
```

Para hacer entrenamiento de segundo nivel, necesitamos obtener más predicciones para hacer validación. Haremos predicción para los meses contando desde marzo de 2015 hasta septiembre de 2015.

```
# get target for the 2nd level dataset
y_train_level2 = y_train[dates_train.isin([26, 27, 28, 29, 30, 31, 32])]
```

Creamos una matriz para guardar resultados de predicción.

```
#And here we create 2nd level feeature matrix, init it with zeros first
X_train_level2 = np.zeros([y_train_level2.shape[0], 2])
```

Creamos las meta-características, que son resultados de predicción de validación.

```
#training with metafeatures
xposition = 0
# Now fill `X_train_level2` with metafeatures
for cur_block_num in [26, 27, 28, 29, 30, 31, 32]:

    X_train = all_data.loc[dates <  cur_block_num].drop(to_drop_cols, axis=1)
    X_test =  all_data.loc[dates == cur_block_num].drop(to_drop_cols, axis=1)

    y_train = all_data.loc[dates <  cur_block_num, 'target'].values

    lr.fit(X_train.values, y_train)
    pred_lr = lr.predict(X_test.values)

    model = lgb.train(lgb_params, lgb.Dataset(X_train, label=y_train), 100)
    pred_lgb = model.predict(X_test)

    dates_train_level2 = np.c_[pred_lr, pred_lgb]

    X_train_level2[xposition:(xposition + X_test.shape[0])]=dates_train_level2
    xposition = xposition + X_test.shape[0]
```

Para cada mes indicado, entrenamos los datos de meses previos con modelo de regresión lineal y con modelo de lightGBM, y guardamos los resultados en la matriz X_train_level2.

Una vez tenemos la matriz X_train_level2, podemos entrenar nuevos modelos, llamados meta-modelos, o modelos de segundo nivel, a partir de ellos.

El modelo más sencillo es hacer una combinación lineal. Su ecuación es:

$$mix = \alpha \cdot \text{linreg\_prediction} + (1 - \alpha) \cdot \text{lgb\_prediction}$$

Como que no sabemos la alpha de la combinación, probamos con mil valores equiespaciados entre 0 y 1, el resultado con menos error a la matriz de validación es la mejor alpha.

```
#simple convex mix
alphas_to_try = np.linspace(0, 1, 1001)

error = 1000
for a in alphas_to_try:
    mix = a * X_train_level2[:,0] + (1 - a) * X_train_level2[:,1]
    mse = sklearn.metrics.mean_squared_error(mix, y_train_level2)
    if mse < error:
        best_alpha = a
        r2_train_simple_mix = sklearn.metrics.r2_score(y_train_level2, mix)
        error = mse

print('Best alpha: %f; Corresponding r2 score on train: %f' % (best_alpha, r2_
train_simple_mix))
```

```
Best alpha: 0.107000; Corresponding r2 score on train: 0.252832
```

Una vez tenemos la alpha que da mejores resultados, que alpha=0,107, la probamos con X_test_level2.

```
test_preds = best_alpha * X_test_level2[:,0] + (1 - best_alpha) * X_test_level2
[:,1]
mse = mean_squared_error(test_preds, y_test, squared=False)
print(mse)
```

```
1.9567470406261278
```

Vemos que los resultados no han mejorado mucho.

Probamos entrenar con los meta-características, para crear un modelo de segundo nivel. Esta vez no es necesario usar modelos complicados, porque los resultados del primer entrenamiento ya son muy correlacionados con nuestro objetivo. Hacemos una regresión lineal.

```
#stacking
lr.fit(X_train_level2, y_train_level2)
```

```
LinearRegression()
```

```
test_preds = lr.predict(X_test_level2)
mse = mean_squared_error(test_preds, y_test, squared=False)
print(mse)
```

```
1.9462985274604279
```

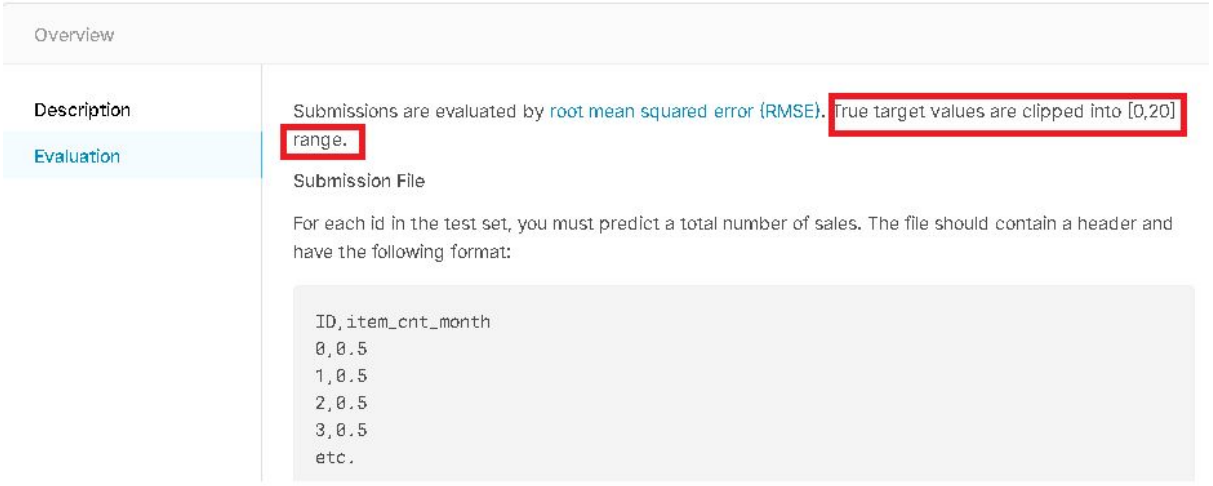
Resultado

Aplicamos nuestro modelo a noviembre de 2015. Con la misma dataframe, extraemos los datos donde date_block_num=34 y establecimos los meses previos para el entrenamiento.

```
# Save `date_block_num`, as we can't use them as features, but will need them t
o split the dataset into parts
all_data = pd.read_pickle("data.pkl")
dates = all_data['date_block_num']
```

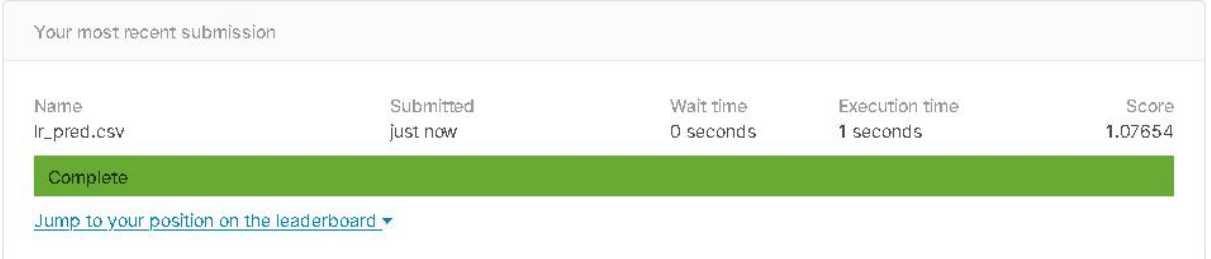
```
dates_train = dates[dates < 34]
dates_target  = dates[dates == 34]

X_train = all_data.loc[dates <  34].drop(to_drop_cols, axis=1)
X_test =  all_data.loc[dates == 34].drop(to_drop_cols, axis=1)

y_train = all_data.loc[dates <  34, 'target'].values
y_test =  all_data.loc[dates == 34, 'target'].values
```

Esta vez aplicamos un filtro para los valores que son menores de 0 y mayores de 20, porque el enunciado nos dice que los valores se encuentran en este rango.



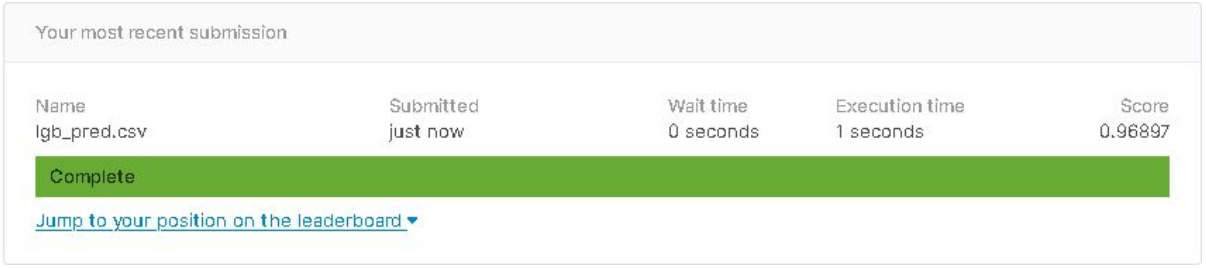
Modelo regresión lineal

```
#we will run linear regression on numeric columns and get predictions for the l
ast month.

lr = LinearRegression()
lr.fit(X_train.values, y_train)
pred_lr = lr.predict(X_test.values)
df_new = pd.DataFrame(pred_lr)
df_sample_submission["item_cnt_month"] = df_new
#we create a filter for values bigger than 20 or smaller than 0
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_mont
h"].where(df_sample_submission["item_cnt_month"] < 20, 20)
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_mont
h"].where(df_sample_submission["item_cnt_month"] > 0, 0)
df_sample_submission.to_csv('lr_pred.csv',index=False)
```



Modelo lightGBM

```
#lgb model
lgb_params = {

            'feature_fraction': 0.75,
            'metric': 'rmse',
           'nthread':1,
           'min_data_in_leaf': 2**7,
           'bagging_fraction': 0.75,
           'learning_rate': 0.03,
           'objective': 'mse',
           'bagging_seed': 2**7,
           'num_leaves': 2**7,
           'bagging_freq':1,
           'verbose':0
          }

model = lgb.train(lgb_params, lgb.Dataset(X_train, label=y_train), 100)
pred_lgb = model.predict(X_test)
df_new = pd.DataFrame(pred_lgb)
df_sample_submission["item_cnt_month"] = df_new

df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].wher
e(df_sample_submission["item_cnt_month"] < 20, 20)
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].wher
e(df_sample_submission["item_cnt_month"] > 0, 0)

df_sample_submission.to_csv('lgb_pred.csv',index=False)
```



Modelo de segundo nivel - combinación simple

```
X_test_level2 = np.c_[pred_lr, pred_lgb]
```

```
# we get target for the 2nd level dataset
y_train_level2 = y_train[dates_train.isin([27, 28, 29, 30, 31, 32, 33])]
```

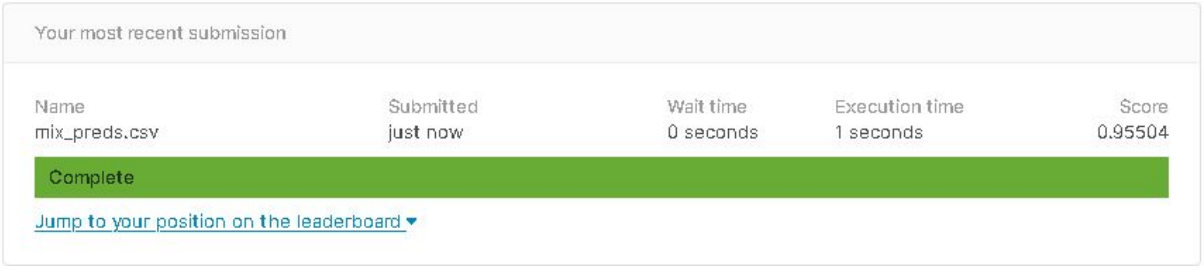
```
#And here we create 2nd level feeature matrix, init it with zeros first
X_train_level2 = np.zeros([y_train_level2.shape[0], 2])
```

```
#training with metafeatures
xposition = 0
# Now fill `X_train_level2` with metafeatures
for cur_block_num in [27, 28, 29, 30, 31, 32, 33]:

    X_train = all_data.loc[dates <  cur_block_num].drop(to_drop_cols, axis=1)
    X_test =  all_data.loc[dates == cur_block_num].drop(to_drop_cols, axis=1)

    y_train = all_data.loc[dates <  cur_block_num, 'target'].values

    lr.fit(X_train.values, y_train)
    pred_lr = lr.predict(X_test.values)

    model = lgb.train(lgb_params, lgb.Dataset(X_train, label=y_train), 100)
    pred_lgb = model.predict(X_test)

    dates_train_level2 = np.c_[pred_lr, pred_lgb]

    X_train_level2[xposition:(xposition + X_test.shape[0])]=dates_train_level2
    xposition = xposition + X_test.shape[0]
```

```
#simple convex mix
alphas_to_try = np.linspace(0, 1, 1001)

error = 1000
for a in alphas_to_try:
    mix = a * X_train_level2[:,0] + (1 - a) * X_train_level2[:,1]
    mse = sklearn.metrics.mean_squared_error(mix, y_train_level2)
    if mse < error:
        best_alpha = a
        r2_train_simple_mix = sklearn.metrics.r2_score(y_train_level2, mix)# YOUR CO
DE GOES HERE
        error = mse

print('Best alpha: %f; Corresponding r2 score on train: %f' % (best_alpha, r2_train
_simple_mix))
```

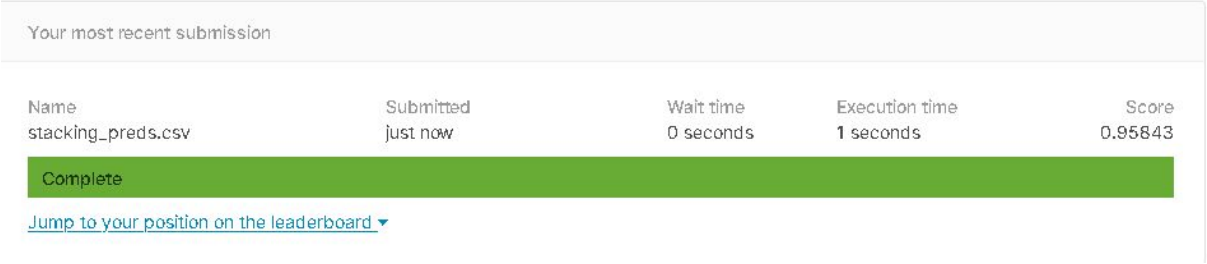
```
Best alpha: 0.205000; Corresponding r2 score on train: 0.289841
```

```
test_preds = best_alpha * X_test_level2[:,0] + (1 - best_alpha) * X_test_level2[:,1]
#r2_test_simple_mix = sklearn.metrics.r2_score(y_test, test_preds)
df_new = pd.DataFrame(test_preds)
df_sample_submission["item_cnt_month"] = df_new

df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].wher
e(df_sample_submission["item_cnt_month"] < 20, 20)
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].wher
e(df_sample_submission["item_cnt_month"] > 0, 0)

df_sample_submission.to_csv('mix_preds.csv',index=False)
```



Modelo de segundo nivel - meta-modelo

```
test_preds = lr.predict(X_test_level2)

df_new = pd.DataFrame(test_preds)
df_new.head()
df_sample_submission["item_cnt_month"] = df_new

df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"]
                                                .where(df_sample_submission["item_cnt_month"] < 20, 20)
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"]
                                                .where(df_sample_submission["item_cnt_month"] > 0, 0)

df_sample_submission.to_csv('stacking_preds.csv',index=False)
```

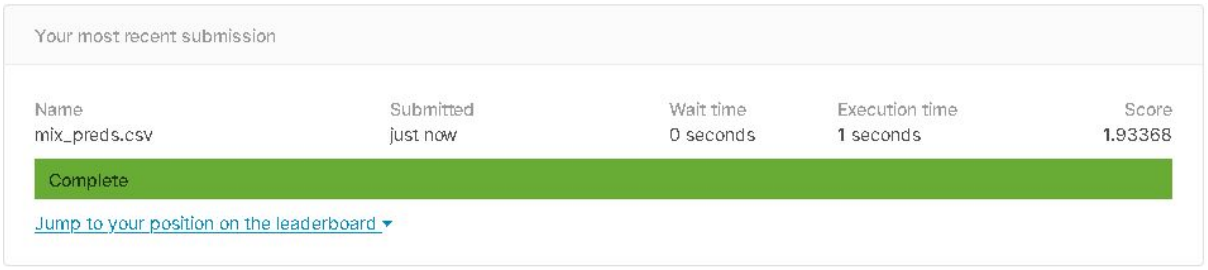


La utilidad del filtro

Como podemos comprobar, los modelos para noviembre de 2015 (date_block_num=34), su error es aproximadamente RMSE=1. Mientras que para octubre de 2015 (date_block_num=33), el error es aproximadamente RMSE=2. Eso es porque sabemos que los valores están entre 0 y 20 y hemos aplicado un filtro para valores fuera del rango. Si no aplicamos el filtro para los modelos, los resultados obtienen una puntuación similar a los de modelos de date_block_num=33.

```
Best alpha: 0.205000; Corresponding r2 score on train: 0.289841
```

```
test_preds = best_alpha * X_test_level2[:,0] + (1 - best_alpha) * X_test_level2[:,1]
#r2_test_simple_mix = sklearn.metrics.r2_score(y_test, test_preds)
df_new = pd.DataFrame(test_preds)
df_sample_submission["item_cnt_month"] = df_new

#df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].whe
re(df_sample_submission["item_cnt_month"] < 20, 20)
#df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].whe
re(df_sample_submission["item_cnt_month"] > 0, 0)

df_sample_submission.to_csv('mix_preds.csv',index=False)
```



Visto la utilidad del filtro, hemos intentado aplicarlo a los modelos para date_block_num=33. Lamentablemente no han mejorado los resultados.

```
#we will run linear regression on numeric columns and get predictions for the last month.

lr = LinearRegression()
lr.fit(X_train.values, y_train)
pred_lr = lr.predict(X_test.values).clip(0,20)
mse = mean_squared_error(pred_lr, y_test, squared=False)
print(mse)
```

```
2.52270713567186
```

+ Code + Markdown

```
#lgb model
lgb_params = {

              'feature_fraction': 0.75,
              'metric': 'rmse',
             'nthread':1,
             'min_data_in_leaf': 2**7,
             'bagging_fraction': 0.75,
             'learning_rate': 0.03,
             'objective': 'mse',
             'bagging_seed': 2**7,
             'num_leaves': 2**7,
             'bagging_freq':1,
             'verbose':0
            }

model = lgb.train(lgb_params, lgb.Dataset(X_train, label=y_train), 100)
pred_lgb = model.predict(X_test).clip(0,20)
mse = mean_squared_error(pred_lgb, y_test, squared=False)
print(mse)
```

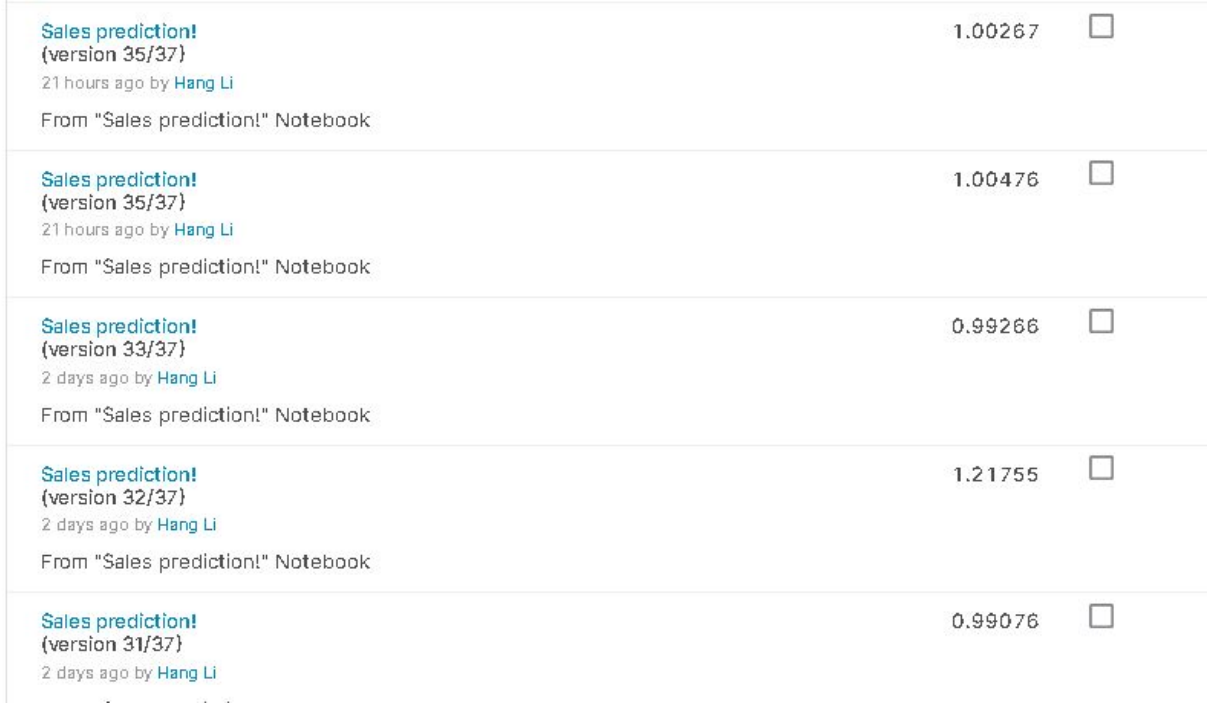
```
2.507335809822127
```

Otras características

Debido a la diferencia entre el dataset de validación y el dataset que deseamos predecir (a uno podemos aplicar un filtro pero no a otra), la validación de los modelos no es tan eficaz. Hemos añadido nuevas características a los modelos, como el precio promedio de los artículos de cada mes o la categoría de los productos. Algunos han mejorado ligeramente la puntuación de la validación pero ninguna ha mejorado la puntuación del leaderboard.

```
df_price = df_sales_train.groupby(['item_id', 'date_block_num'],as_index=False)["ite
m_price"].mean()
df_price = df_price.rename(columns = {"item_price": "target_price"})
all_data = pd.merge(all_data, df_price, how='left', on=['item_id', 'date_block_num
']).fillna(0)
del df_price
```

```
# Category for each item
all_data = pd.merge(all_data, df_items, how='left', on=['item_id']).fillna(0)
df_new = df_sales_train.groupby(['item_category_id','date_block_num'],as_index=Fals
e)["item_cnt_day"].sum()
df_new = df_new.rename(columns = {"item_cnt_day": "target_category"})
all_data = pd.merge(all_data, df_new, how='left', on=['item_category_id', 'date_bloc
k_num']).fillna(0)

all_data = all_data.drop(columns = ["item_category_id", "item_name"])
```



Otros modelos

También hemos probado otros modelos, como random forest regresor o knn regresor, no han mejorado mucho la puntuación, pero sus resultados pueden ser meta-características para modelos de segundo nivel. Debido a la limitación de recursos computacionales no hemos probado esta posibilidad.

```
rf = RandomForestRegressor()
rf.fit(X_train.values, y_train)
pred_rf = rf.predict(X_test.values)
df_new = pd.DataFrame(pred_rf)
df_sample_submission["item_cnt_month"] = df_new
#we create a filter for values bigger than 20 or smaller than 0
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].wher
e(df_sample_submission["item_cnt_month"] < 20, 20)
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].wher
e(df_sample_submission["item_cnt_month"] > 0, 0)
df_sample_submission.to_csv('rf_pred.csv',index=False)
```


```
knn = KNeighborsRegressor(n_neighbors=20)
knn.fit(X_train.values, y_train)
pred_knn = knn.predict(X_test.values)
df_new = pd.DataFrame(pred_knn)
df_sample_submission["item_cnt_month"] = df_new
#we create a filter for values bigger than 20 or smaller than 0
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].wher
e(df_sample_submission["item_cnt_month"] < 20, 20)
df_sample_submission["item_cnt_month"] = df_sample_submission["item_cnt_month"].wher
e(df_sample_submission["item_cnt_month"] > 0, 0)
df_sample_submission.to_csv('knn_pred.csv',index=False)
```

Conclusión

Nuestro objetivo es obtener un modelo para predecir las ventas futuras a partir de los datos históricos. Hemos aplicado diferentes modelos: regresión lineal, regresión KNN, random forest, gradient boosting, y modelos combinados. Y como resultado tenemos un modelo capaz de predecir la cantidad de productos vendidos por cada tienda y por cada producto, con un error promedio menos de 1 unidad. Reducirían mucho los costes de producción y los costes de transportación de los productos sabiendo la cantidad vendida en cada lugar. ¡Bien!