Trabajo Práctico - Machine Learning: Regresión en Retail.

Profesor: Federico Pousa.

Boca, Martino.
De Santis, Octavio.
Noblea, Eugenio.

Introducción.

El presente informe aborda el **forecasting de demanda** para una cadena de retail con operaciones en diversos países europeos. El objetivo es predecir las ventas para las próximas seis semanas, lo cual es esencial para optimizar la gestión de recursos y mejorar la toma de decisiones dentro de la empresa. Para ello, se utilizarán datos históricos provenientes de una competencia en Kaggle, que se centró en la predicción de ventas a partir de información detallada sobre transacciones y características específicas de cada tienda.

La base de datos incluye un archivo con más de 1 millón de observaciones sobre ventas, abarcando el período desde el 1 de enero de 2013 hasta el 31 de julio de 2015. Además, se dispone de un conjunto que contiene información sobre las tiendas y otro destinado a realizar las predicciones. Este enfoque permite realizar un análisis exhaustivo y generar información adicional que facilite la identificación de patrones en las ventas.

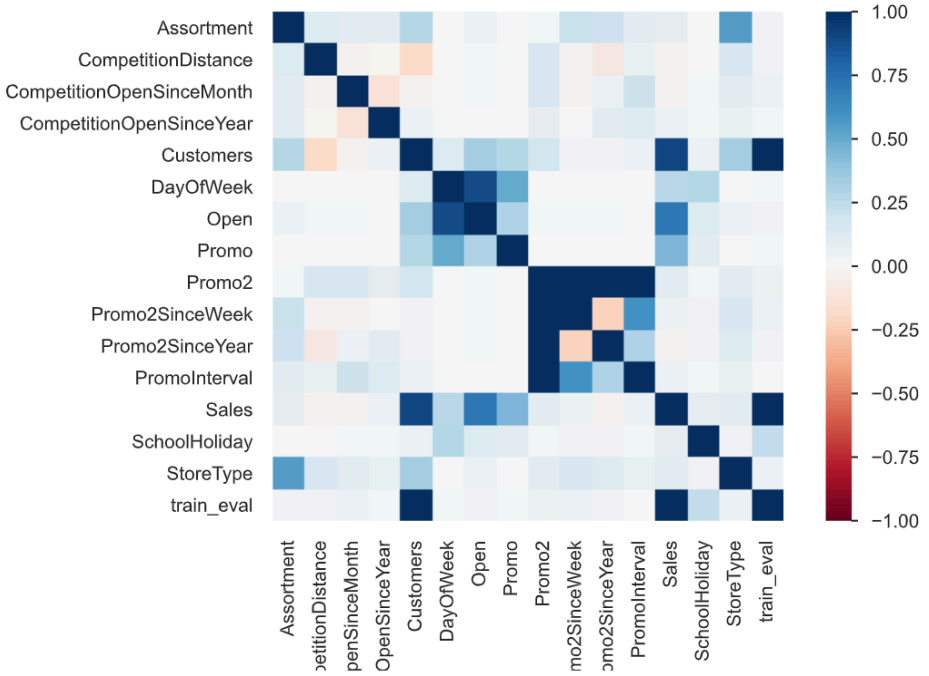
La adquisición de los datos se llevó a cabo mediante la descarga de tres archivos CSV proporcionados por los organizadores de la competencia. Estos archivos son:

- *Un conjunto de entrenamiento con datos históricos.*
- *Un archivo que detalla características específicas de cada tienda.*
- *Un archivo para las predicciones a realizar una vez entrenado el modelo.*

El rendimiento del modelo se evaluará utilizando el **RMSPE** (Root Mean Square Percentage Error) sobre el conjunto de test, lo que permitirá medir la efectividad del enfoque adoptado en este trabajo. Además, se explorarán diferentes técnicas y modelos para optimizar esta métrica, incluyendo un análisis preliminar y una ingeniería de datos adecuada.

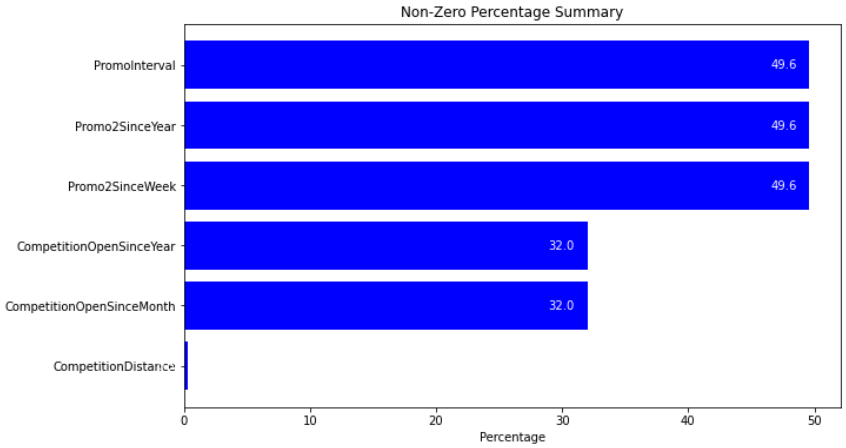
EDA & Feature Engineering.

Como primera aproximación a los datos, decidimos hacer una matriz de correlación entre las variables del dataset completo (train + test) luego de haberlo mergeado con “Stores”. En este mapa de calor podemos ver que la variable que cuenta con mayor colinealidad con la variable objetivo “Sales”, es Customers, con un $R^2 = 0.89$. Decidimos descartarla ya que permite mejorar la generalización del modelo, reducir su complejidad, hacer que los coeficientes sean más estables y facilita la interpretación.



Nulos

Identificamos las siguientes variables con nulidad a través del cálculo de porcentaje de nulos sobre el total de datos en la columna. Estas son ‘PromoInterval’, ‘Promo2SinceYear’, ‘Promo2SinceWeek’, ‘CompetitionOpenSinceYear’, ‘CompetitionOpenSinceMonth’, y ‘CompetitionDistance’. A continuación, detallaremos cuáles fueron las estrategias de rellenado de NaNs y procesamiento de cada una de ellas.

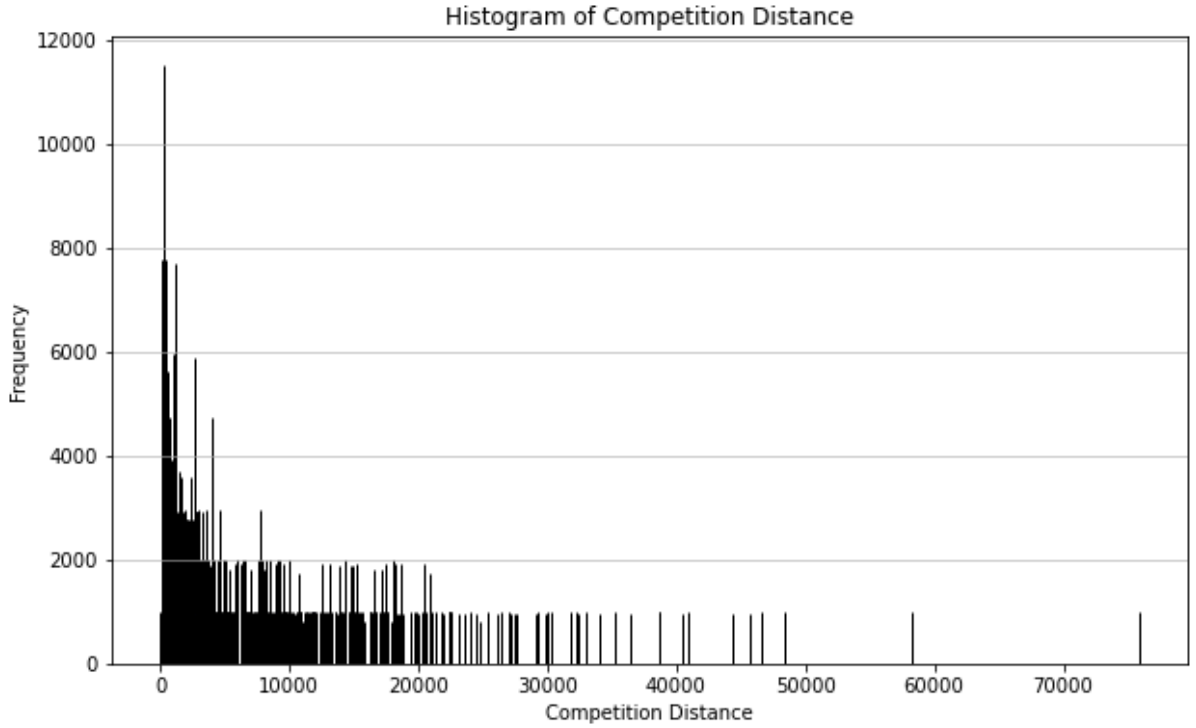


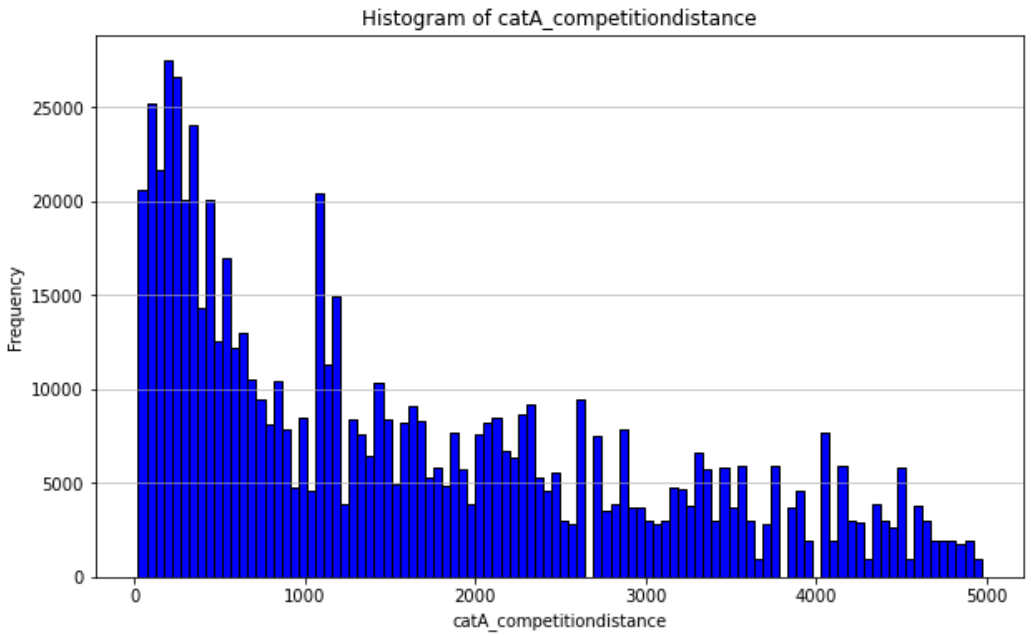
Variables Competition.

En cuanto a las variables de la competencia, identificamos que las variables Competition tenían un 32% de nulidad excepto por distance, que solo tenía datos faltantes para el 0.25%. En esta última variable, imputamos la distancia con un valor extraordinario para facilitar su discriminación posterior en el modelo (999999).

Luego, generamos CompetitionOpen para resumir la información de las “OpenSince”. Imputamos con el valor más común cuando no había información (año = 2013 y mes de apertura = enero), entonces completamos CompetitionOpen con 1. Por último, comparamos el año y mes de apertura con el año y mes del registro, en caso de que se haya abierto previo a la fecha, entonces completamos la variable con 1.

En el caso de CompetitionDistance, generamos una variable categórica separada en bins que representa la cercanía de la competencia. Primero generamos 3 grandes reglas, para las tiendas menores a 4000 generamos 10 bins, para las mayores a 4000 y menores a 20000 generamos otro bin y para las mayores a 20000 uno más. Totalizando 12 categorías. Los primeros 3 cortes se eligieron analizando el gráfico *“Histogram of Competition Distance”* y los siguientes cortes del primer corte con el gráfico *“Histogram of catA_competitiondistance”.*





Variables Promo.

En el caso de las promociones, creamos una nueva variable Promo2Active que representa si la promoción está activa o no. Aplicamos valores extremos para identificar una tercer categoría a la nulidad de datos (999). En el caso de que la fecha del dato sea posterior a la fecha del debut de la promoción y ademas nos encontramos en el mes en el que se activa, imputamos 1 en Promo2Active. Imputamos 0 para cualquier otro caso.

Variables Date.

La variable fecha fue descompuesta para tener mayor granularidad. Utilizamos la función add_datepart de la librería fastai que nos permite desglosar la fecha en diferentes columnas: 'Year', 'Month', 'Week', 'Day'. Y además genera las columnas booleanas: ‘Is_month_end', 'Is_month_start', 'Is_quarter_end', 'Is_quarter_start', 'Is_year_end', y 'Is_year_start'.

Otras Variables.

Tanto Assortment como StoreType y StateHoliday fueron generadas como categóricas ordinales para mostrar una posible diferencia cualitativa entre las tiendas. Se generó además una variable Open_Holiday para representar que la tienda está abierta en una fecha extraordinaria como una festividad.

Técnica de validación.

Como técnica de validación, utilizamos el enfoque Hold-out set, dividiendo nuestro data set en conjuntos de entrenamiento y validación según la fecha de las observaciones. En donde los datos se dividen en dos partes temporales: hasta 2014 para entrenamiento y 2015 para validación. Esta separación es crucial en problemas de series temporales, como la predicción de ventas, ya que evita mezclar información futura durante el entrenamiento, previniendo data leakage. Al evaluar en un período posterior al entrenamiento, se simula un escenario real donde el modelo, entrenado con datos históricos, predice el futuro.

Modelo 1 : Random Forest

El primer modelo a implementar es un Random Forest. El mismo es un modelo de ensamble compuesto por múltiples árboles de decisión, entrenados sobre diferentes subconjuntos aleatorios del conjunto de datos. Esta estrategia de muestreo y construcción de múltiples árboles reduce considerablemente el riesgo de overfitting, que es común en árboles de decisión simples.

A diferencia de un solo árbol de decisión, que puede ser altamente influenciado por las particularidades de los datos de entrenamiento, Random Forest mitiga este problema al promediar las predicciones de múltiples árboles. Esto no solo mejora la capacidad de generalización del modelo, sino que también aumenta su robustez. Los árboles en el bosque captan diferentes patrones presentes en los datos, y al combinar sus predicciones, Random Forest logra reducir la varianza del modelo y mejorar la precisión en datos no vistos.

Optimización de Hiperparámetros.

Utilizamos el método de RandomSearch para encontrar los mejores hiperparámetros, pero los resultados fueron peores que en el modelo original.
El modelo original dio como resultados:

- Train RMSPE: 0.09216
- Val RMSPE: 0.18758
- RMSPE Kaggle (private): 0.15613
- RMSPE Kaggle (public): 0.14323

Hiperparámetros:

- Número de estimadores (n_estimators): 100
- Profundidad máxima del árbol (max_depth): none
- Número mínimo de muestras para dividir un nodo (min_samples_split): 2
- Máximas características consideradas para cada división (max_features): 1

En cambio, para el modelo optimizado:

- Puntaje OOB (out-of-bag): -0.467
- Train RMSPE: 0.468
- Val RMSPE: 0.431
- RMSPE Kaggle (private): 0.41580

- RMSPE Kaggle (public): 0.40670

Hiperparámetros:

- Número de estimadores (n_estimators): 600
- Profundidad máxima del árbol (max_depth): 7
- Número mínimo de muestras para dividir un nodo (min_samples_split): 0.01
- Máximas características consideradas para cada división (max_features): 24

Importancia de Features.

Dado que el modelo original funcionó mejor, utilizaremos la importancia de este modelo. El método de importancia de características basado en el MDI, que evalúa la importancia de cada característica en función de la reducción del criterio de división en los árboles de decisión. A continuación, mostramos el top 15:

1. Open: 0.455539
2. Store: 0.096207
3. Promo: 0.072321
4. log_competitiondistance: 0.060137
5. CompetitionDistance: 0.059750
6. CompetitionOpenSinceYear: 0.043095
7. CompetitionOpenSinceMonth: 0.042315
8. Dayofyear: 0.029880
9. Dayofweek: 0.019487
10. StoreType: 0.019383
11. Day: 0.018617
12. DayOfWeek: 0.018061
13. Assortment: 0.015518
14. Week: 0.009352
15. cat_CompetitionDistance: 0.009223

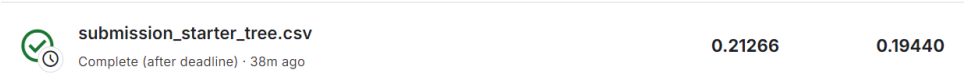
Modelo 2 : Decision Trees

Como segundo modelo decidimos implementar Decision Trees simples, con el objetivo de marcar la diferencia de performance que hay con el primer modelo Random Forest, siendo que este es un modelo de ensamble.

A pesar de que un árbol de decisión puede ofrecer interpretaciones claras y ser fácil de visualizar, su desempeño si los ponemos a competir, es inferior al de un modelo de Random Forest, especialmente cuando se trata de conjuntos de datos más complejos. Una de las principales razones de esta diferencia está en la tendencia del árbol de decisión a *overfitear* los datos de entrenamiento. Esto ocurre porque un único árbol puede adaptarse demasiado a las peculiaridades de los datos. En contraste, Random Forest, al estar compuesto por múltiples árboles que se entrenan en subconjuntos aleatorios de los datos (bagging), reduce significativamente el riesgo de *overfittear*. Esta estrategia de muestreo no solo mejora la capacidad de generalización del modelo, sino que también aumenta su robustez, ya que

diferentes árboles en el *bosque* pueden captar distintos aspectos de los datos. Como resultado, Random Forest tiende a ser menos sensible a variaciones en los datos, porque promedia las predicciones de varios árboles, lo que ayuda a reducir la varianza del modelo, y a su vez, mejora la precisión de las predicciones en datos no vistos.

Primero, se implementó el modelo Out Of The Box, a partir del Feature Engineering ya realizado y el score fue el siguiente:



Notemos que performa de peor manera al random forest, pero sin embargo podemos modificar algunos hiperparametros que limitarán la profundidad del árbol para lograr una mejor performance en el RMSPE.

Optimización de hiperparámetros

Para poder mejorar el score previo aplicamos una optimización de hiperparametros utilizando la misma estrategia que en random forest (Random Search) para los siguientes hiperparametros: ‘min_samples_split’, ‘min_samles_leaf’, ‘max_features’, ‘max_depth’.

Los mejores hiperparámetros encontrados fueron los siguientes: {'min_samples_split': 20, 'min_samples_leaf': 5, 'max_features': None, 'max_depth': None}

- min_samples_split: 20
  Indica el número mínimo de muestras necesarias para dividir un nodo. En este caso, el nodo solo se dividirá si tiene al menos 20 muestras. Este valor ayuda a prevenir que el árbol crezca demasiado profundo, lo que puede llevar a sobreajuste u overfitting.
- min_samples_leaf: 5
  Especifica el número mínimo de muestras que debe tener una hoja. Esto significa que cada hoja del árbol debe tener al menos 5 muestras. Valores más altos de min_samples_leaf pueden hacer que el modelo sea más general y eviten que el árbol se ajuste a ruido o anomalías en los datos.
- max_features: None
  Significa que en cada división, el modelo considerará todas las características disponibles. No hay ninguna restricción en cuántas características se pueden usar para determinar la mejor división en cada paso. Esto puede hacer que el modelo sea más preciso, pero también corre el riesgo de sobreajustarse a los datos.
- max_depth: None
  El árbol puede crecer hasta una profundidad ilimitada o hasta que las hojas contengan el número mínimo de muestras definidas por **min_samples_leaf**.

El resultado en kaggle luego de controlar por estos aspectos fue:

submission_starter_decision_tree.csv Complete (after deadline) · now		0.17688	0.16611

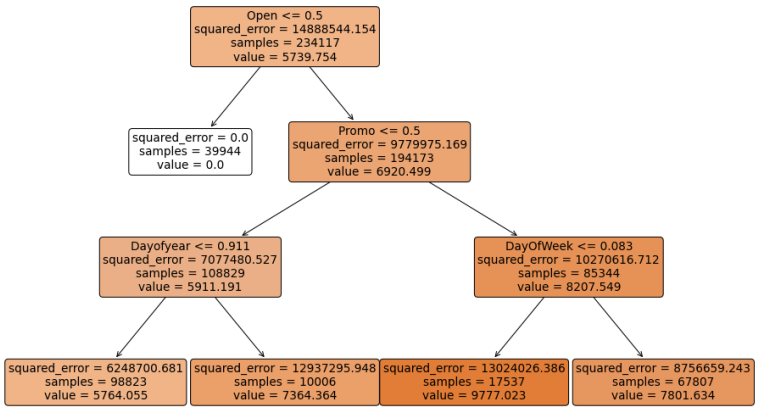
Como podemos notar, el score sigue siendo peor al de Random Forest, pero logramos una mejoría bastante significativa versus la aplicación Out-Of-The-Box.

Interpretación e Importancia de Features

A continuación, el listado de las variables más importantes luego de optimizar los hiperparametros del árbol de decisión:

	importance
Open	0.486230
Store	0.090180
Promo	0.077298
CompetitionDistance	0.058716
log_competitiondistance	0.056994
CompetitionOpenSinceMonth	0.043027
CompetitionOpenSinceYear	0.039278
Dayofyear	0.026291
Dayofweek	0.025271
StoreType	0.022000
Assortment	0.015620
DayOfWeek	0.010878
Day	0.009935
cat_CompetitionDistance	0.009750
Promo2	0.009380

Una de las ventajas principales de un árbol de decisión para regresión es su facilidad para ser interpretado.En ese sentido, para facilitar su interpretación y poder visualizar cómo particiona el árbol en un principio, limitamos su profundidad como máximo a 3. Lo que nos permitió generar este gráfico, utilizando el módulo *Tree* de sci-kit learn y la librería matplotlib:



El nodo superior corresponde a la variable Open, la cual tiene sentido que sea el primer criterio para particionar, cuando Open es <= 0.5 (osea cuando está cerrado, Open = 0) vemos que el error cuadrático es 0 ya que no hay ventas cuando la tienda está cerrada. Luego, si Open >= 0.5 (es decir =1) el árbol sigue dividiendo en base a si tiene o no Promoción en esa fecha, más hacia abajo DayOfYear o DayOfWeek lo cual no sugiere que ciertos días del año o días de la semana tienen impacto en las ventas. Y por último, vemos en las hojas los valores promedio de las predicciones.

Problema adicional.

Definición de Problema.

Elegimos como problema adicional la predicción de la variable “Customer”. Si bien está relacionada con “Sales”, la cantidad de clientes por periodo de tiempo puede llevar a tener una mejor optimización de distintas decisiones de negocio como puede ser el layout de la tienda, su stockeo y la gestión de acciones publicitarias con proveedores.

Modelo: Regresión Lineal.

Como baseline, proponemos una regresión lineal utilizando el mismo feature engineering proveniente de los modelos anteriores y como feature selection nos quedaremos con las variables más importantes, considerando que podrían ser buenas predictoras por la colinealidad existente entre Sales y Customers. También, podría proponerse un problema de clasificación, donde el objetivo es encontrar tiendas que se parezcan entre sí y en función de eso optimizar su operación.

Los resultados nos describen que hay oportunidad de mejora en el modelado, solo el 48% de la variabilidad de Customer es explicado por el top 15 de las variables utilizadas.

Otros puntos a analizar::

- Podría considerarse un filtro de “Open” dado que una tienda cerrada tendría 0 clientes. Esto hace sentido con la fuerza de su coeficiente.
- Se debería elegir solo una variable de competition distance dada su fuerte correlación.
- Se debería elegir solo una variable de Promoción.

```
Validation R-squared: 0.48502587499649163

Coefficients Table:
                     Variable  Coefficient
0                        Open   683.795853
1                       Store    20.573800
2                       Promo   133.046911
3         CompetitionDistance    92.548217
4     log_competitiondistance  -285.196938
5   CompetitionOpenSinceMonth   -40.413145
6    CompetitionOpenSinceYear  -120.333700
7                   Dayofyear    59.680637
8                   Dayofweek   -32.298261
9                   StoreType   -90.039435
10                 Assortment    54.192091
11                 DayOfWeek   -32.298261
12                       Day   -14.773011
13    cat_CompetitionDistance  -209.832452
14                     Promo2  -148.123634
```