ORGANIZACIÓN DE DATOS

Trabajo Práctico 2

Predicción de Precios Zonaprop utilizando algoritmos de Machine Learning

Prof Corrector:

INTEGRANTE	PADRÓN	E-MAIL
Taiel Colavecchia	102510	tcolavecchia@fi.uba.ar
Santiago Klein	102192	sklein@fi.uba.ar
Yuhong Huang	102146	yhuang@fi.uba.ar

GRUPO 10

Introducción **2**

Link al repositorio **2**

Notas sobre el set de datos **3**

Pre-procesamiento de los Datos **3**

Feature Engineering **3**

Test de Modelos Varios **3**

Elección del modelo para predicción **3**

Tuneo de Hiper-Parámetros **3**

Modelo Final **3**

Introducción

Este trabajo práctico fue realizado para la materia Organización de Datos de la Facultad de Ingeniería de la Universidad de Buenos Aires.

En el presente informe, se detallarán los resultados obtenidos a partir de la utilización de técnicas de Machine Learning en pos de predecir el valor de propiedades del registro histórico de publicaciones del sitio https://www.zonaprop.com.ar**.**

El set de datos incluye el registro de todas las publicaciones realizadas entre los años 2012 y 2016 en México, y puede ser obtenido desde:

`metadata.fundacionsadosky.org.ar/media/navent/train.csv`

Link al repositorio

`github.com/tonyhuang07/Datos_TP2`

Notas sobre el set de datos

A partir del trabajo de análisis del set de datos que se hizo en el Trabajo Práctico 1, se concluyeron algunos aspectos que consideramos importantes:

- Valores Nulos: deben ser procesados de alguna forma para algoritmos que no soporten los mismos.
- Distribución de Precios: los precios responden a una distribución de "cola larga" por lo que para algunos algoritmos se buscará transformar la distribución para buscar una mejora.
- Dependencia de la variable 'precio' con las otras columnas: durante el trabajo práctico anterior verificamos que con varios de los otros *features* tiene una fuerte dependencia y se explorará buscar la importancia con algunos modelos, basados en insights extraídos del mismo.

Pre-procesamiento de los Datos

Valores Nulos

Se comenzó por observar a partir del TP1 las *features* que tenían valores nulos que necesitaban ser reemplazados para los modelos que no los soportaran.

A continuación se hace un resumen de los mismos:

- `'metros_cubiertos'` y `'metros_totales'` : se completaron los valores faltantes de un campo al otro. Esta estrategia alcanzó para agotar los nulos.
- Variables categóricas y numéricas: se utilizaron las funciones `'CategoricalImputer'` y `'SimpleImputer'` de la biblioteca `sklearn` para insertar valores faltantes.
- `'lat'` y `'lng'`: se llenaron los valores calculando el promedio de los mismos para los valores de `'idzona'` que le correspondiera a cada registro.

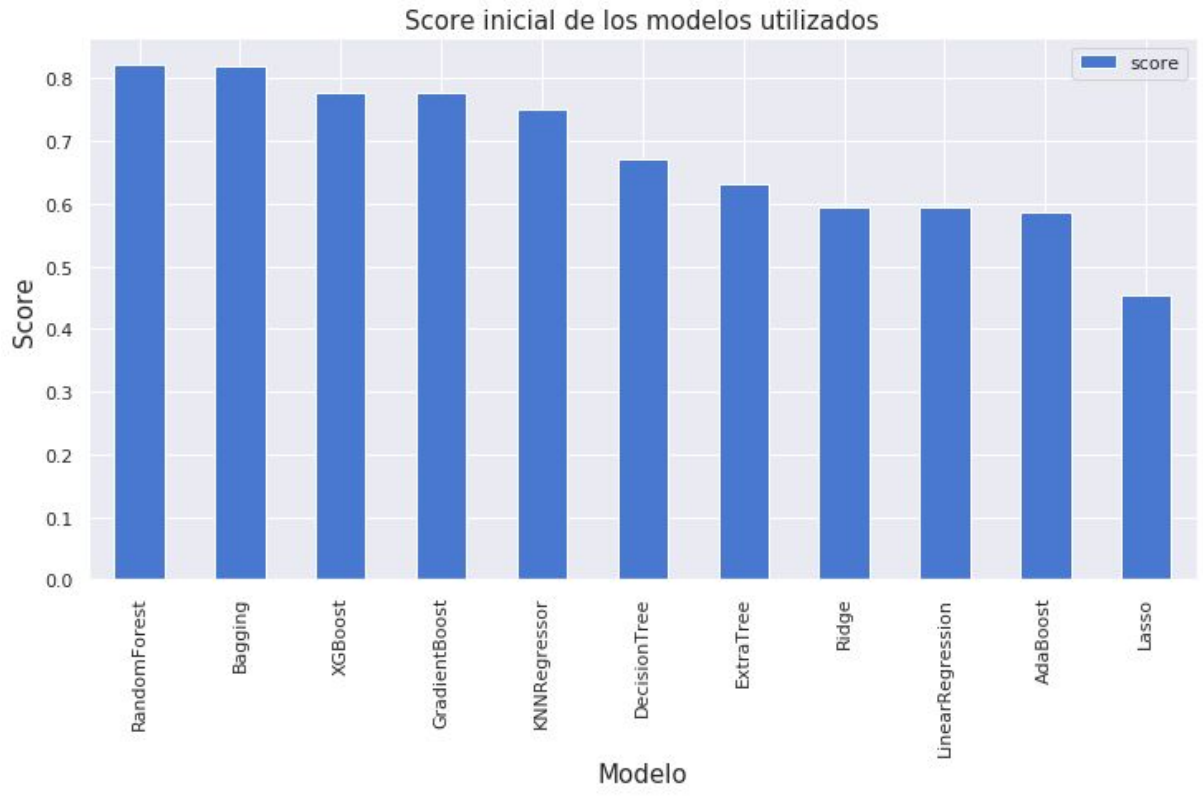
Redistribución de los precios

Al encontrar que los precios tenían una distribución que podría no ser amigable a algunos modelos, agregamos la posibilidad de utilizar el logaritmo de los precios como label para los modelos.

Test de Modelos Varios

Una vez realizado el preprocesamiento de datos (en principio, rudimentario, para ir mejorándolo a medida que avanzabamos), nos enfocamos en buscar un modelo robusto, que se adapte a las características del problema en cuestión.

Dado que no estábamos familiarizados con la utilización de los diferentes modelos, en aras de obtener un pantallazo general de su funcionamiento, la primer acción fue realizar una predicción utilizando los modelos: `'LinearRegression'`, `'KNNRegressor'`, `'Ridge'`,`' Lasso'`, `'DecisionTree' , 'ExtraTree'`, `'XGBoost'`, `'RandomForest'`, `'AdaBoost'`, `'GradientBoost'`, y `'Bagging'`. Todos ellos fueron utilizados con sus hiper-parámetros default.



Como se puede observar, aquellos que obtuvieron mayor score fueron los modelos basados en árboles y boosting. A raíz de ello, los primeros submits a Kaggle fueron hechos a partir de predicciones de RandomForest.

Asimismo, analizamos el comportamiento de los distintos algoritmos al dejar el precio en escala, al aplicarle logaritmo, y al estandarizar. Los mejores resultados se obtuvieron

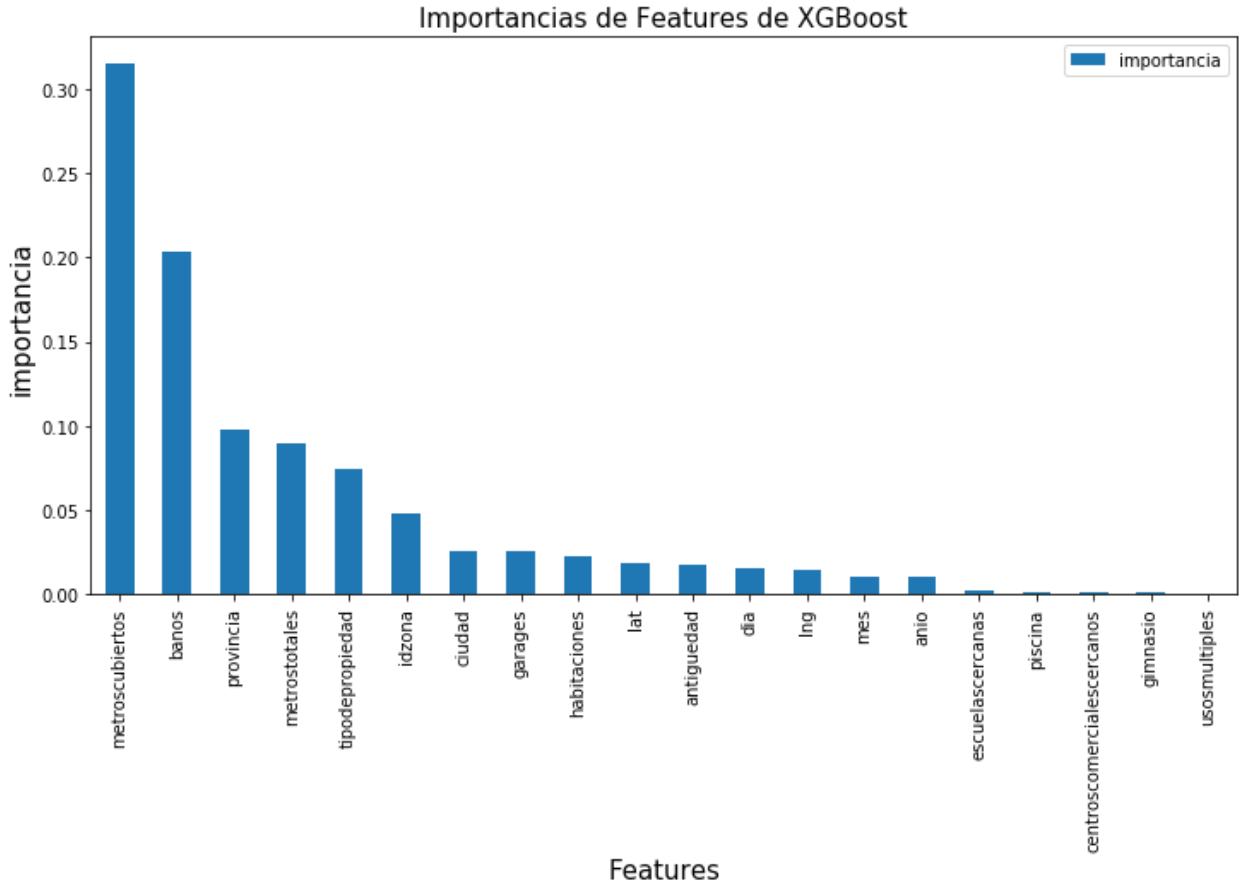
aplicando logaritmo, lo cual es entendible, ya que la distribución de precios es de cola larga y obedece a la *ley de potencias*.

No obstante, a medida que se fue desarrollando la etapa de feature engineering, sobresalió un modelo por sobre el resto.

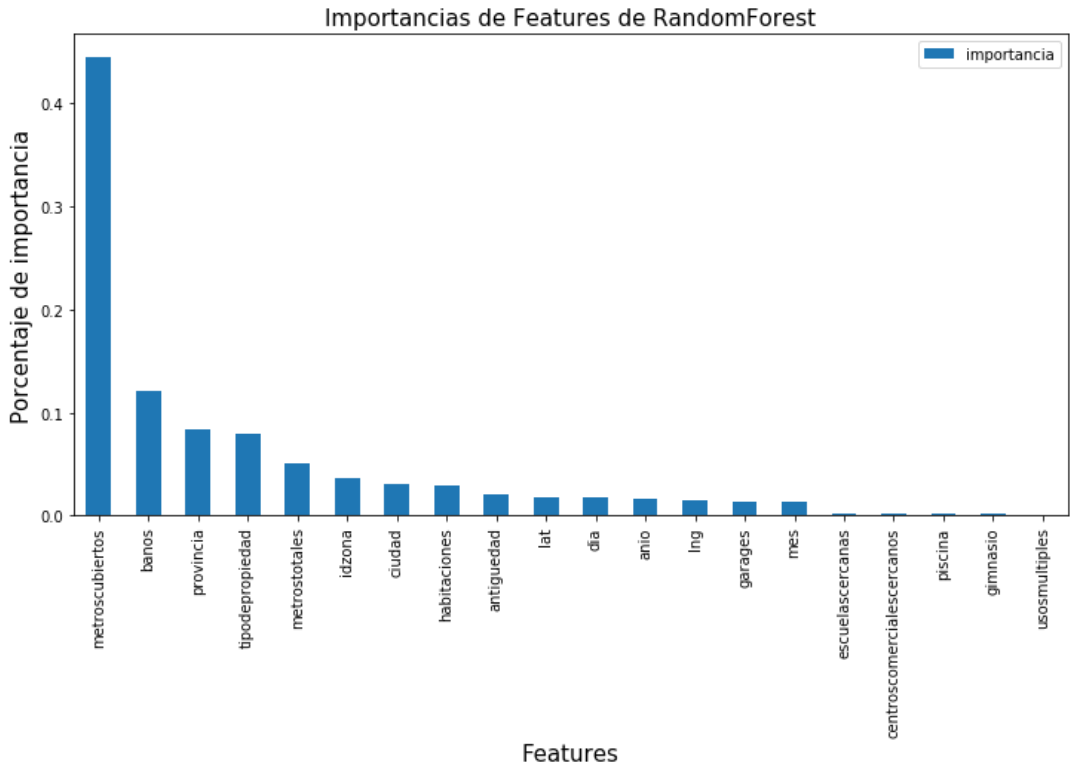
Feature Engineering

Feature importance - Inicial

A partir del pre-procesamiento de los datos, se analizó la importancia de las columnas originales de los datos utilizando el modelo *XGBRegressor*, y se obtuvieron los siguientes resultados:



En el caso del *XGBRegressor* se utilizó el parámetro - 'total_gain' de *feature importance* para medir la importancia de las features.



En el caso de *RandomForest*, se utilizó la función *feature_importance* del modelo para medir la importancia de las features.

Después de comparar ambos gráficos, se puede ver que ambos modelos tienen una distribución de la importancia de las features parecida.

Luego de haber analizado la importancia de las features existentes, procedimos a agregar nuevas features, analizando la conveniencia en cada caso. Para determinar la utilidad o no de una nueva feature, se corrió el modelo correspondiente y se analizó la mejora de los resultados.

Feature Engineering - metros cubiertos/totales

Con respecto a los metros cubiertos y totales de las propiedades, se crearon dos nuevas features. En primer lugar, **tiene_sup_descubierta**, la cual representa si la propiedad posee balcón/jardín/etc. Su valor es 1 si existe diferencia en el valor absoluto de metros_cubiertos y metros_totales, y 0 en otro caso.

Además, se agregó como feature **diff_metros_cubiertos_y_totales**, que representa, en valor absoluto, el valor de dicha diferencia entre metros cubiertos y totales, es decir, la superficie descubierta de la propiedad.

Feature Engineering - tipos de propiedades

Se aplicó one-hot-encoding a los tipos de propiedades, lo cual llevó a una mejora sustancial en los resultados de predicción.

Feature Engineering - ciudad/provincia/id_zona

Se utilizó ordinal encoding para generar features nuevas, en las cuales se les asigna un orden a los diferentes tipos de ciudad, idzona, y provincia, según su ranking en el precio promedio por metro cuadrado, resultando en las features: ***ciudad_ordinal***, ***idzona_ordinal*** y ***provincia_ordinal***. Cabe destacar que idzona es una feature relevante original e indica las ubicaciones de las propiedades de una manera más precisa que ciudad y provincia, con lo que se obtiene más información.

Por otro lado, también se generaron las features ***ciudad_promedio_m2*** y ***idzona_promedio_m2*** que indican el precio promedio de la ciudad y zona de cada propiedad. Si bien el agregado de dichas features puede incurrir en un leaking, consideramos oportuna su inclusión dada la sustancial mejora que se dieron en las predicciones. No se realizó lo mismo para las provincias, pues a raíz del TP1 se puede ver que la precio de las propiedades varían mucho según la zona que estén, por lo que generar esta feature puede generar mucho ruido. De todas maneras, fue constatado empíricamente, y después realizar las predicciones, la feature ***provincia_promedio_m2*** no ayudó a mejorar los resultados.

Feature Engineering - lat/lon

Ya que la exactitud de las posiciones ubicadas de las propiedades es un factor importante al predecir los precios, después de haber analizado las columnas de latitud y longitud se puede ver que cada propiedad tiene valores diferentes para las mismas, por lo tanto, consideramos redondear los valores latitud y longitud con dos decimales, la razón de elegir dos decimales fue que las latitudes y longitudes con dos decimales son

suficientes para determinar la zona, y también se facilita la agrupación por posición de la forma ***(lat, lon).***

Con ello, se calcularon los precios promedios de cada posición, y se generó una nueva feature: ***posicion_promedio_m***.

Feature Engineering - agrupacion

En el análisis exploratorio del TP1, pudimos observar que los factores `'gimnasio'`, `'usosmultiples'` y `'piscina'` tienen una relación similar con el precio de la propiedad. Por ello, se generó una nueva feature sumando dichos valores, concentrando así la cantidad de extras que ofrece la propiedad en una sola feature: ***suma_gimnasiousosmultiplespiscina.***

Análogamente, se generó una feature según la cercanía a escuelas y centros comerciales: ***suma_escuelascercanascentroscomercialescercanos.*** Asimismo, a partir del trabajo anterior pudimos determinar que la cercanía con escuelas y centros comerciales no afecta significativamente en el valor de la propiedad, por lo que dejar las features por separado no generaba ganancia alguna.

Feature Engineering - análisis de descripciones y títulos

En el análisis exploratorio del TP1, pudimos obtener los términos más frecuentes en las descripciones y títulos, se analizaron los términos más frecuentes en las descripciones de las propiedades más caras (decidimos el precio 3 millones en pesos mexicanos como borde) pero que no aparecen en las más baratas, y se obtuvo una lista de términos importantes. Con la lista de términos se generó un nuevo feature: **importancia de palabras**, en esa columna se acomula valores por termino, su suma 1 a la importacia por cada término importante que aparezca en la descripción y el título de cada registro.

Al final de realizar esa feature, el resultado de la predicción se mejoró poco, pero decidimos dejarlo como una parte de la feature engineering.

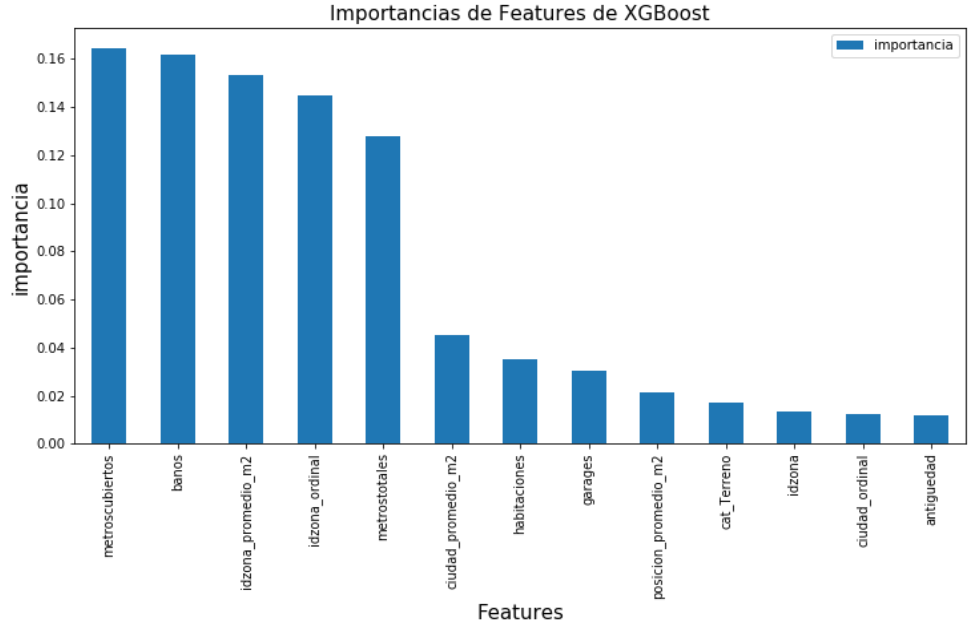
Feature Engineering - Idea fallada

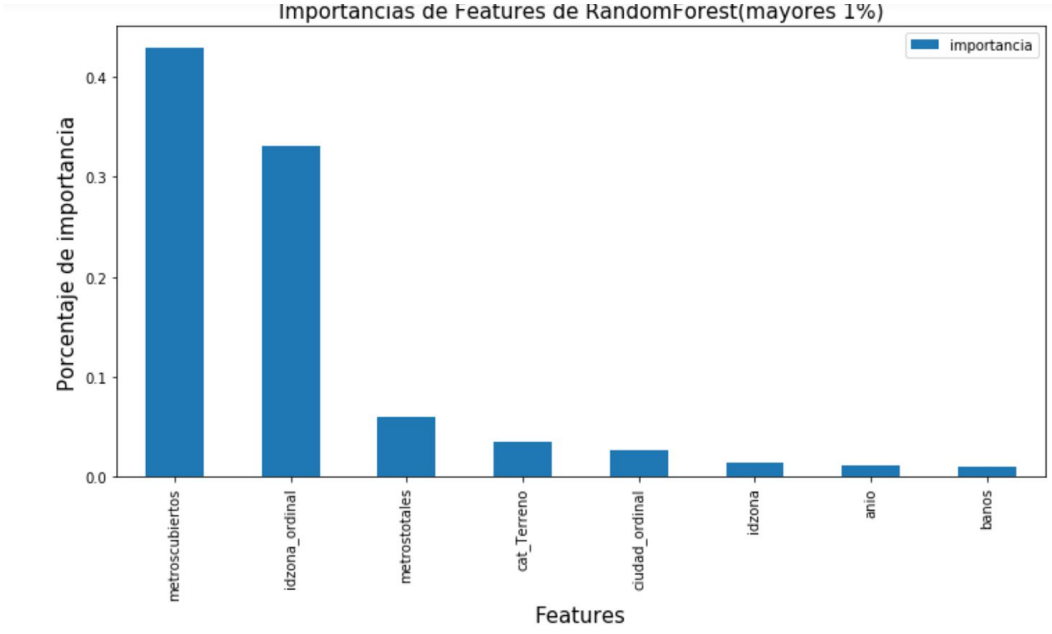
Se consideró que la inflación del país puede un factor importante para predecir el precio de las propiedades, y decidimos generar una nueva feature: **inflación por mes.**

En la predicción obtenida, la feature de inflación no mejoró el resultado, lo cual es razonable, ya que el modelo tiene información de las fechas y precio, es posible que el modelo haya aprendido el factor de la inflación, y esta feature resultó como información repetida por lo que no mejora los resultados.

Feature Importance - Final

Después de aplicar feature engineering, se obtuvieron los siguientes resultados:





Se puede observar que las features que relacionan metros cubiertos y idzona tienen una importancia muy significativa.

Cabe destacar, que durante el proceso se realizaron pruebas de agregado y eliminación de nuevas features de manera constante, determinando si inclusión o no en el modelo final según el mejoramiento de los resultados o no.

Elección del modelo para predicción

Una vez madurado el proceso de feature engineering, el modelo que sobresalió por sobre el resto fue *XGBoost*, aunque, por su parte, RandomForest no le fue a la zaga.

Este algoritmo es muy poderoso para lidiar con irregularidades en los datos, dado que utiliza la técnica de regularización.

En particular, resultó muy útil al dejar los valores nulos, pues presenta una rutina interna que los maneja, frente a los cuales realiza diversas acciones. Se realizó una comparación, primero imputando los valores nulos, y luego dejándolos así como están. El resultado fue apreciablemente mejor en el segundo caso.

Además, su soporte para el procesamiento paralelo de los datos a la hora de entrenar el modelo redujo los tiempos de entrenamiento de manera considerable.

Ante los excelentes resultados que fue presentando, nos concentramos en tunear sus numerosos hiper-parámetros (que resultó una tarea difícil), proceso del que hablaremos en la próxima sección, dado que se trata de un algoritmo muy sofisticado.

Tuneo de Hiper-Parámetros

Habiendo elegido *XGBoost* como modelo definitivo, se procedió a tunear sus hiper-parámetros con una librería de sklearn. Utilizamos la función 'GridSearchCV' para comenzar, como prueba se utilizó sólo para `'min_split_loss'` y `'max_depth'`. Aquí notamos que llevaba mucho tiempo en ejecutarse (ya que por default usa 3-fold Cross Validation y las combinaciones de los parámetros dados), por lo que decidimos hacer una búsqueda más simple para el resto de los parámetros.

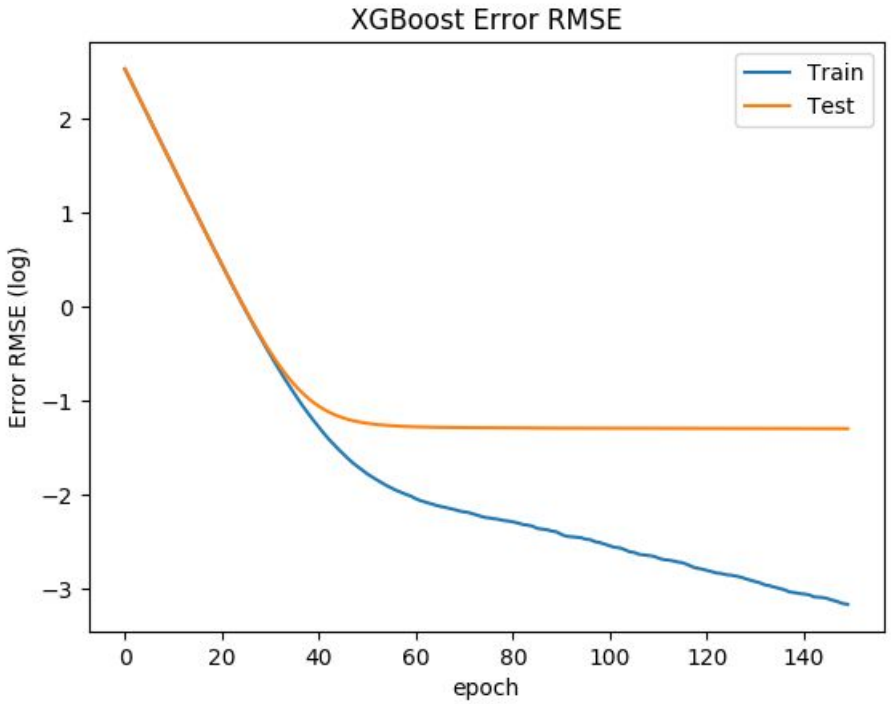
Se hizo una búsqueda automatizada que recorría sobre las opciones de cada uno de los parámetros restantes por separado. A pesar de que esta forma simple no utiliza *Cross-Validation* y no prueba combinaciones entre los mismos, obtuvimos una mejora pequeña, pero significativa, de performance del modelo.

De esta forma, decidimos sobre los siguientes valores para los parámetros (para algunos de ellos se encontró que el mejor valor era el default):

`min_split_loss (gamma)`	0 (default)
`max_depth`	21
`n_estimators`	150
`min_child_weight`	2
`learning_rate (eta)`	0.07
`subsample`	0.9
`reg_alpha`	0 (defualt)
`reg_lambda`	0 (defualt)
`colsample_bytree`	0.8

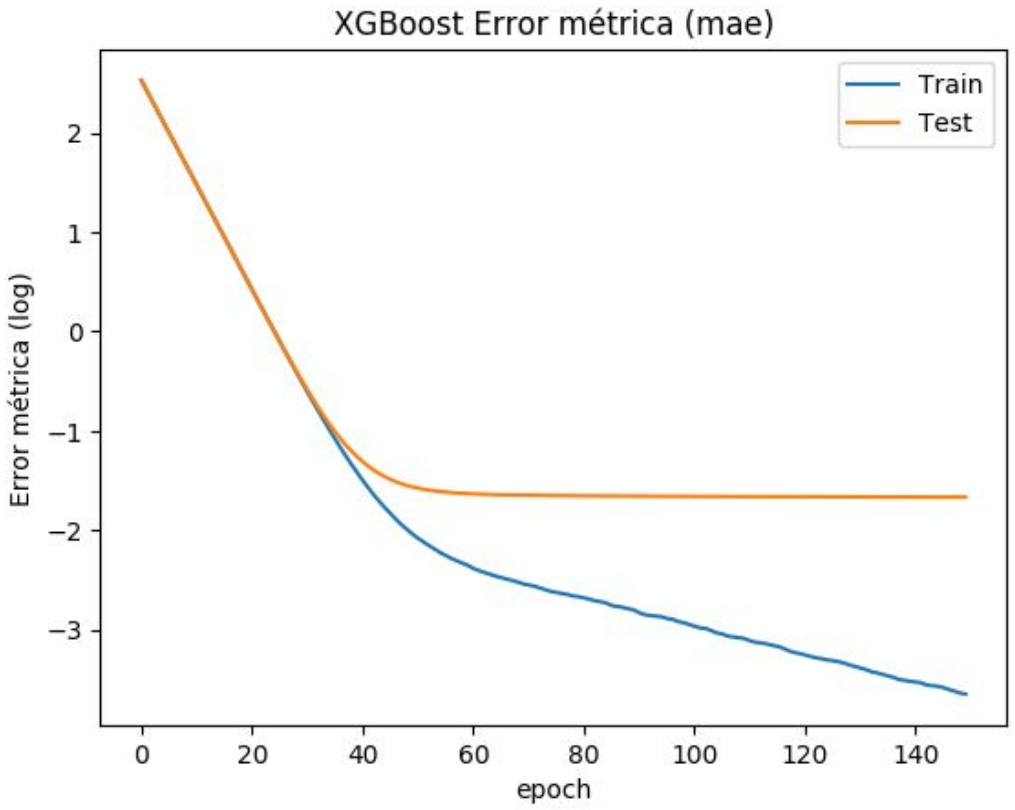
Modelo Final

Para los parámetros obtenidos de *XGBoost* se ejecutó el algoritmo para obtener cómo mejoraba el aprendizaje a lo largo de las iteraciones del mismo.



En este primer gráfico, se muestra la métrica que utiliza el algoritmo para corregir el aprendizaje, el error cuadrático medio. A pesar de que el modelo consigue un error muy bajo sobre el set de **Train**, puede observarse que la mejora sobre los valores del set de **Test** mejoran muy poco a partir de (aproximadamente) la iteración 60.

Podemos ver además, que este resultado se mantiene sobre la métrica elegida para la competencia del Trabajo Práctico:



A pesar de que se ve que el error para **Test** casi no mejora, se puede observar que entrenar más tiempo no resulta en un *overfitting* del modelo.

Para obtener el resultado final de las predicciones, a último momento se decidió realizar un promedio del precio de las predicciones de *XGBoost* y *RandomForest*, y ofreció una considerable mejora respecto de submits anteriores.

Conclusiones

El presente trabajo práctico fue de suma utilidad para la exploración del mundo de Machine Learning desde cero, y el aprendizaje del uso de modelos predictivos, feature engineering, parameter tuning, etc.

El enfoque inicial nos permitió conocer, en primera instancia, los distintos modelos y sus funcionamientos, pudiendo descartar muchos de ellos que no resultaban tan útiles

para el problema en cuestión. Experimentalmente, pudimos constatar el poder de predicción que tienen los algoritmos de árboles y boosting, que no sorprendentemente representan el estado del arte.

Así, nos enfocamos principalmente en el manejo de *XGBoost* y *RandomForest* (en una etapa temprana también incluimos Bagging, pero luego fue descartado), a medida que, íbamos descubriendo nuevas features y trabajando sobre el dataset de entrenamiento, concentrados en la importancia de las mismas.

Si bien intentamos experimentar con redes neuronales, tuvimos dificultades en su implementación, y por cuestiones de tiempo decidimos descartarlas, aunque creemos hubiera sido fructífera su inclusión. Asimismo, desistimos de ensamblar distintos modelos, a pesar de que creíamos que esto iba a mejorar significativamente los resultados (lo cual no se reflejó empíricamente).

La última etapa fue dedicada casi exclusivamente a la obtención de los hiper-parámetros que maximicen la performance de los modelos. Para ello, comenzamos utilizando la técnica de cross-validation, empero, debido al tiempo que tomaba, decidimos hacer una búsqueda automatizada de los mismos, singularmente, a intervalos discretizados, en lo cual también debimos invertir una considerable cantidad de tiempo.

Referencias

- Documentación de Sklearn: https://scikit-learn.org/stable/documentation.html
- Documentación de XGBoost: https://xgboost.readthedocs.io/en/latest/index.html