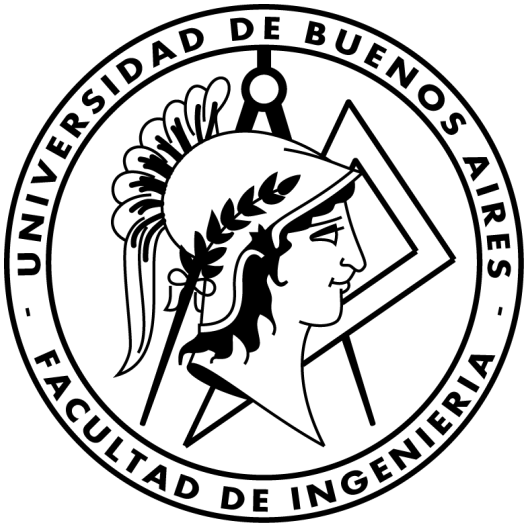
Facultad de Ingeniería
Universidad de Buenos Aires



Trabajo Práctico N°2
Organización de Datos 75.06

Grupo 24 “DatoSaurios”
Compuesto por:

Balestieri, Juan Diego - 101601
Sturla, Manuel - 100912
Fernández, Andrés - 102220

GitHub: https://github.com/diegobalestieri/TP2---Datos

Índice

Modelos probados

KNN

Este fue uno de los primeros modelos utilizados. La motivación para utilizarlo fue que tiene un solo hiperparámetro (“k”) y que se dijo en clase que daba una buena puntuación base sobre la cual los siguientes modelos a utilizar tienen que mejorar.
Para este no se hizo búsqueda de hiperparámetros dado que el modelo tardaba mucho tiempo en predecir y se prefirió luego de hacer la prueba de concepto seguir investigando sobre otros modelos.

Random Forest

Este modelo también fue de los primeros utilizados. Sin embargo, después de algunas pruebas, ajustando algunos hiperparametros y probando distintos métodos para codificar los features categóricos se dejó de utilizar este modelo. Ya que utilizando los mismos features obtuvimos un score mejor utilizando XGBoost en un tiempo mucho menor, mientras Random Forest necesitaba casi una hora para entrenar XGBoost lo hacía en 5 minutos. Esta diferencia de tiempo nos permite hacer más pruebas de manera más rápida y por lo tanto Random Forest dejó de ser utilizado para las predicciones.
Aun así, el modelo siguió siendo utilizado para medir la feature importance ya que es el modelo más adecuado para hacerlo. Además, para calcularlo pudimos reducir la cantidad de árboles y la profundidad de cada uno considerablemente sin afectar la medición. Esto, nos permite calcular la feature importance de manera mucho mas rapida.

XGBoost

XGBoost fue uno de los más utilizados por el equipo dado que nos permitió obtener buenos resultados de manera sencilla y rápida. Uno de los factores que nos hizo inclinarnos por este modelo fue saber que está basado en un algoritmo de boosting que según lo dicho en clase se encuentra entre los mejores algoritmos para competencias de machine learning .
A lo largo de la competencia fuimos tuneando los hiperparámetros buscando conseguir un mejor score cada vez. Un factor que colaboró a que este modelo fuera el más utilizado fue que puede sacar provecho del procesamiento por GPU, que es mucho más rápido. Con esta ventaja, un entrenamiento de 25 minutos con o más con XGBoost se lograba hacer en 5 minutos, ahorrandonos mucho tiempo y permitiéndonos probar más opciones de features e hiperparámetros.

CatBoost

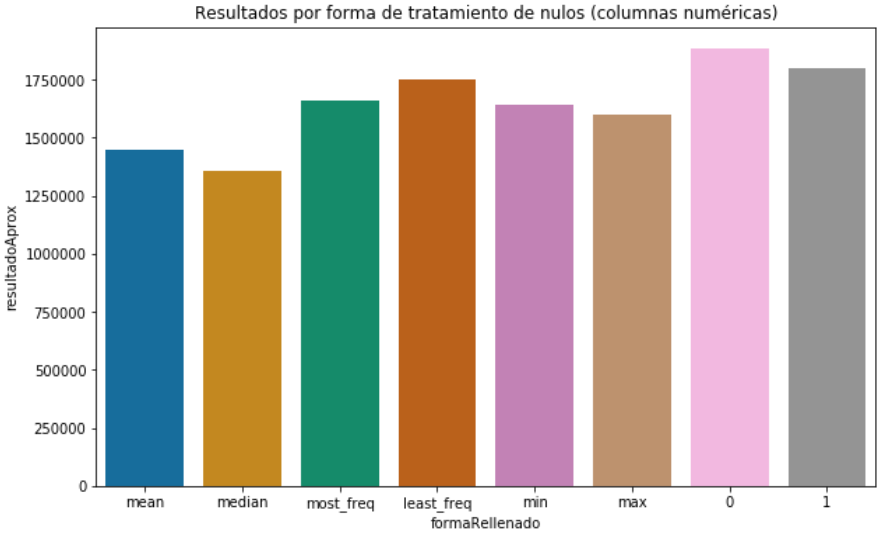
CatBoost fue utilizado al comienzo, pero obtuvimos resultados muy similares y ligeramente peores a los obtenidos con XGBoost que era el otro algoritmo que se estaba usando en ese momento. Por este motivo nos quedamos utilizando XGBoost. Es probable de todas maneras que dado que ambos modelos están basados en un boosting de árboles de decisión que podamos obtener resultados similares si optimizamos hiperparámetros.

Factorization Machines

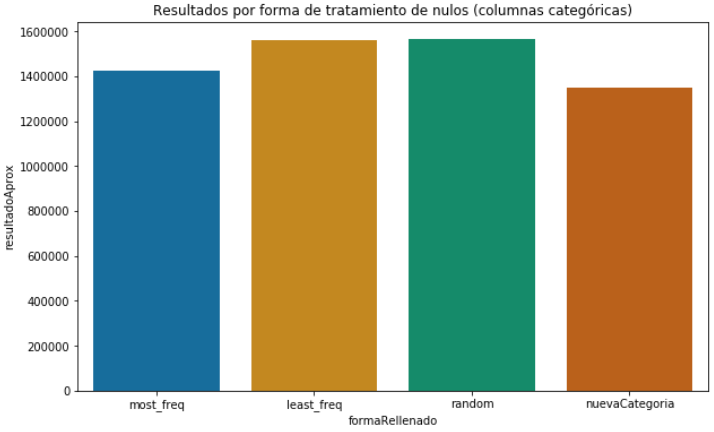
Este modelo se probó para predecir los precios como un modelo único sobre los mismos features que usábamos sobre XGBoost, obteniendo muy malos resultados.
Luego se intentó usarlo luego de un TF-IDF para que a partir del texto prediga los precios y utilizar estos como feature. Intentando aprovechar la capacidad de este modelo de encontrar variables latentes en los datos, que serían muy difíciles de obtener manualmente del texto.
Esta idea desgraciadamente aunque de manera local daba buenos resultados, no generaliza para nada bien. Y por lo tanto no aporta información al modelo sino que incluso lo empeora.

Tratamiento de nulos

Para los features que tenían nulos las primeras alternativas fueron rellenarlos utilizando un SimpleImputer, que nos permitía rellenar los nulos ya sea con un valor constante, o con el más frecuente o el promedio. La ventaja de usar este transformador de sklearn es que es totalmente integrable con los pipelines de esta misma biblioteca. Pero la desventaja es que resultaba conveniente tener un único SimpleImputer en todo el pipeline y no siempre buscábamos rellenar los nulos de la misma manera.
En primer lugar rellenamos los nulos a través del Simple Imputer con valores básicos: primero completando todas las columnas numéricas con 0, luego con su mínimo, con su máximo, con su promedio. Nuestras primeras estimaciones las realizamos dejando el tratamiento de nulos en segundo plano, aunque con cierta frecuencia cambiábamos la forma de rellenarla. Por supuesto, ninguna implicaba una mejora significativa en nuestras estimaciones.



En primera medida, pareció una buena idea implementar una especie de “Cross Validation” para el tratamiento de nulos. Generamos varias listas, que contenían valores con los que se podía rellenar los nulos de acuerdo a cada columna. Por ejemplo, para las categorías numéricas, la lista contenía (entre otros): “mean”,”median”,”min” y “max”. Para las columnas categóricas, por ejemplo, el tipo de propiedad, la provincia o la ciudad, la lista contenía “mostFrequent”,”leastFrequent”,”random”, teniendo en cuenta que los elementos de la otra lista no tendrían sentido para variables no numéricas.



A partir de ahí, utilizamos varios ciclos que cruzaran los diferentes elementos de cada lista todos contra todos para ver cuáles eran los valores que mejor funcionan para rellenar los nulos de cada columna. No tuvimos en primera instancia resultados muy conclusivos. Como los modelos que utilizamos fueron cambiando ciertos detalles, había combinaciones que funcionaban mejor para unos que para otros, pero ninguna mejoraba los resultados de forma suficientemente significativa como para definirla como la mejor.

En una segunda etapa, decidimos ser un poco más específicos con la forma de completar los nulos, de acuerdo a cada columna, analizando un poco más cuáles eran los factores que mejor podíamos tomar en cuenta para rellenarlas. Por ejemplo, a la hora de rellenar la antigüedad, consideramos que era una buena idea agrupar las propiedades por tipo, y a partir de ahí utilizar la antigüedad promedio de cada tipo para rellenar el dato faltante. O en otros casos rellenar con el promedio de esa columna en la ciudad o zona, suponiendo que las propiedades de una misma zona geográfica tendrían características en común. De manera relativamente análoga (cambiando algunos detalles), hicimos lo mismo para las columnas de baños, habitaciones, latitud, longitud e id de zona.

Pudimos observar, con esta forma de tratamiento de nulos (representada en la imagen de abajo) una mejora en las estimaciones que si bien no es tremendamente significativa, sí se mantiene constante a través de los diferentes modelos que fuimos utilizando para buscar mejorar nuestro puntaje.

COLUMNA	GROUP BY	CÁLCULO
ANTIGÜEDAD	CIUDAD	PROMEDIO
BAÑOS	TIPO DE PROPIEDAD	MEDIANA
HABITACIONES	TIPO DE PROPIEDAD	MEDIANA
IDZONA	CIUDAD	MEDIANA
LATITUD	CIUDAD	PROMEDIO
LONGITUD	CIUDAD	PROMEDIO
GARAGES	TIPO DE PROPIEDAD	MEDIANA
PROVINCIA	CIUDAD	-
CIUDAD	PROVINCIA	MOST FREQ
TIPO DE PROPIEDAD	NUEVA CATEGORÍA: TIPO VACÍO	

Features

Features Generados

- **Número de publicaciones por distintos grupos**:
    - Por ciudad: suponemos que saber cuantas publicaciones hay en una ciudad, te da una idea de la oferta de propiedades en la zona. Así como también informa del movimiento inmobiliario en cada una de las ciudades.
    - Por provincia: tiene un razonamiento similar al anterior.
    - Por tipo de propiedad: da información de la oferta que hay sobre cada tipo de propiedad.
    - Otros grupos: zona, cantidad de baños, habitaciones, antigüedad. Varios de estos fueron eliminados de los modelos, por generar en conjunto al resto de el preprocesamiento mucho overfitting.
- **Número de publicaciones que hubo en ese día, semana, 5 días hábiles, mes**. El objetivo de este feature es lograr dar información sobre cuánta actividad inmobiliaria hubo en el último tiempo suponiendo que esta impacte en el precio.
- **Promedio de ciertos features por zona y ciudad**: Este feature buscaba proporcionar información de cómo son las características de las propiedades de una zona o ciudad. Donde por ejemplo, una propiedad que tiene más metros que el promedio en su ciudad debería ser más cara. Los promedios calculados fueron de:
    - Antigüedad
    - Habitaciones
    - Metros Cubiertos
    - Metros totales
    - Precio
    - Precio por metro cuadrado

Estos features nos resultaron problemáticos porque resultaron en overfitteo del modelo. Sobre todo los promedios en función de los precios. Suponemos que estos problemas resultaba de la forma que los calculamos. En una primera instancia utilizabamos los precios del set de datos de Train para calcular estos promedios. Pero esto no daba cuenta de los valores que pueden llegar a tener el set de Test. Por lo tanto otra alternativa que se nos ocurrió fue utilizar dos modelos: uno que a partir de features independientes del precio, lo prediga. Y con este precio aproximado calcular features que lo utilicen. Finalmente con todos los features resultantes: aquellos independientes del precio y los derivados del precio aproximado, generar la predicción final.

- **Features del texto**: nos resultó muy difícil obtener información relevante del texto dada la dificultad de procesar un campo de entrada libre del usuario. Entre los features que pudimos obtener fueron:
    - Número de palabras
    - Número de caracteres
    - Longitud promedio por palabra: calculado como la cantidad de caracteres sobre la cantidad de palabras.
    - Número de apariciones de palabras más comunes: este resulta de la cantidad de palabras en la descripción que estén en el top 50 de palabras más

utilizadas. El objetivo de este feature era identificar atributos de las propiedades importantes como: estacionamiento, terraza, jardín. Que en caso de tenerlos, las propiedades aumentan de precio.

- Número de apariciones de palabras más comunes en las descripción por tipo de propiedad. Se calcula de manera similar al anterior pero buscan las palabras más comunes por tipo de propiedad. Supusimos que dependiendo de qué tipo de propiedad estuviera en venta las palabras con mayor relevancia serían distintas.

De estos features que obtuvimos el que resultó más relevante fue la longitud promedio por palabra.

- **Adicionales**: es un feature que agrupa todos los features booleanos, tanto los ya existentes en el Dataframe (piscina, gimnasio, usosmultiples, escuelascercanas y centroscomercialescercanos), como los que pudimos extraer de la descripción, que son los siguientes:
  - Jardín
  - Vigilancia (si la propiedad cuenta o no con vigilancia)
  - Jacuzzi
  - Lavadero
  - Ubicación (si la propiedad tiene o no una buena ubicación)
  - Barrio Cerrado (si la propiedad está o no en un barrio cerrado)

Asignamos un valor a cada elemento a través de un diccionario, por ejemplo, jardín tiene valor 2, piscina, 5, gimnasio, 4, barrio cerrado, 10, escuelas cercanas, 3. Estos valores los asignamos nosotros mismos de acuerdo a nuestra percepción del valor que debe tener cada adicional.

La realidad es que, a través de los distintos modelos que fuimos probando, la incorporación de este feature para realizar las predicciones terminó siendo relativamente insignificante: podía mejorar o empeorar el score, de acuerdo al modelo utilizado, pero no lo hacía de forma significativa en ninguno de los dos sentidos.

- **Promedio de precio por provincia**: no parece aportar ninguna información nueva dado que el score de nuestra prueba no cambio.
- **Puntaje en función de otros features**. Decidimos que para evitar que se aprenda demasiado las propiedades, en vez de mantener muchas columnas con distintos valores booleanos, asignar un puntaje conjunto en función de estos. Este luego de hacer la feature aunque acumulaba la importancia de
- **Precio del Metro Cuadrado:** consideramos que el precio por metro cuadrado sería sin duda un feature importante para la estimación de los precios, aunque la dificultad estaba en cómo utilizarlo: tenemos muchas manera de conseguir un promedio (o mediana), por lo que debemos determinar cuál es la mejor. Por ejemplo, por un lado nos pareció que sería demasiado general conseguir el precio del metro cuadrado por provincia, y por otro, conseguirlo por zona podría ser demasiado específico (hay muchas zonas que sólo tienen una o dos publicaciones). El precio de la ciudad podría ser la mejor solución en este sentido, aunque por supuesto seguía estando el riesgo de que fuera demasiado general en ciudades grandes o muy específico para ciudades con muy pocas publicaciones.

Nuestra idea entonces fue la de buscar la mejor estimación del precio del metro cuadrado posible, yendo de la más específica a la más general, pero sólo tomando como válidas aquellas estimaciones que contarán con una cantidad suficiente de datos. De esta manera, solamente utilizaremos la zona de la propiedad para obtener el precio si esa zona tiene una cantidad razonable de datos, para evitar el overfitting. Si la zona no cuenta con datos suficientes, entonces utilizaremos un filtro por ciudad y (de forma conjunta), en el siguiente orden, por trimestre, por semestre y por año, quedándonos con lo que consigamos primero, si tenemos suficientes datos. Por último, en el caso de que todavía no haya suficientes datos para tener una estimación del precio por metro cuadrado, utilizamos un filtro sólo por ciudad y luego uno por provincia y año, y por último uno sólo por provincia.

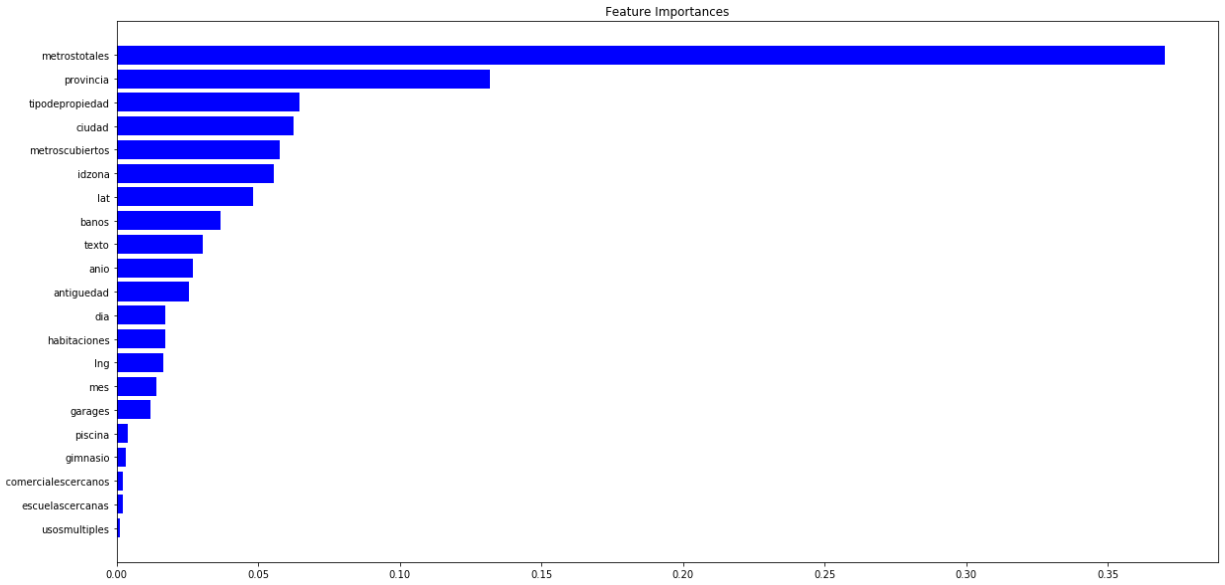
Estos pasos para la estimación del precio del metro cuadrado fueron los que mejores resultados nos dieron, y nos ayudaron a mejorar la estimación en alrededor de cinco mil puntos.

Feature Importance

Para analizar la feature importance utilizamos RandomForest y la función ***feature_importances_*.** Sin agregar features, se interpretaron los features categóricos y se rellenaron los nulos de la siguiente manera:

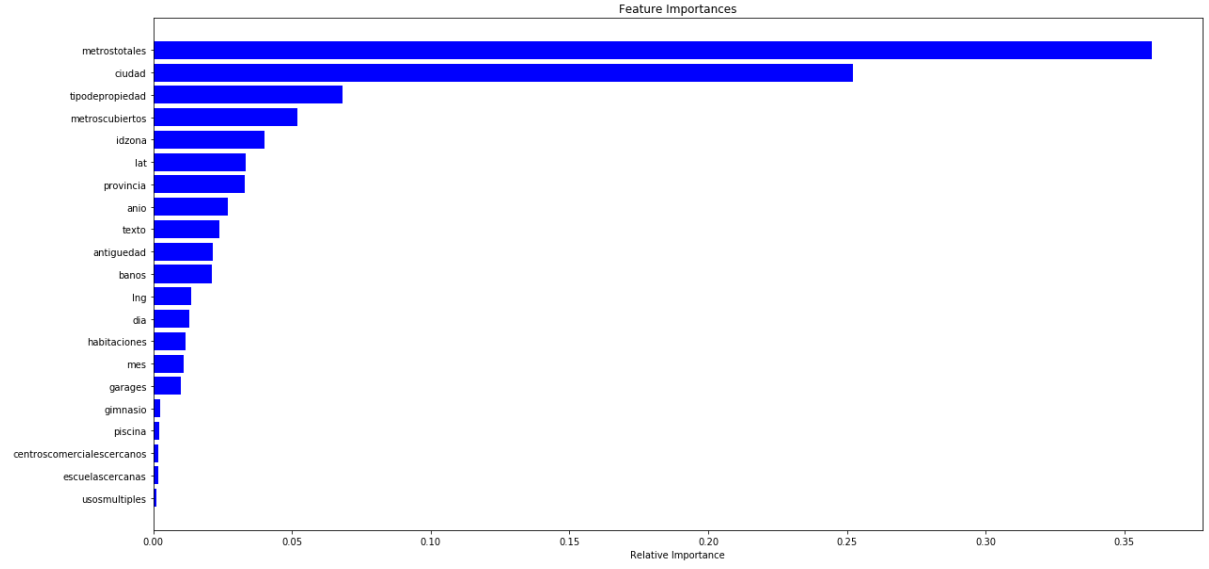
- Metros totales y metros cubiertos: primeramente se rellenaron todos los nulos con ceros y luego, utilizando las funciones min y max le asignamos el valor más alto a metros totales y el menor a metros cubiertos.
- Provincia: utilizamos CountEncoder.
- Tipo de propiedad: utilizamos CountEncoder.
- Ciudad: utilizamos CountEncoder.
- Id zona: rellenamos con la mediana del id de zona para la ciudad.
- Latitud y longitud: rellenamos con el promedio de la latitud y la longitud para la ciudad.
- Baños: rellenamos con 1, ya que es la cantidad mínima de baños por propiedad normalmente.
- Texto: usamos el largo de la descripción más el título.
- Fecha: separamos la fecha en dia, mes y año.
- Antigüedad: rellenamos con el promedio para la ciudad.
- Habitaciones: rellenamos por la mediana para el tipo de propiedad.
- Garages: rellenamos con 0, ya que consideramos que si no se informa que tienen garage no lo tiene.

Graficando la feature importance obtuvimos los siguientes resultados:

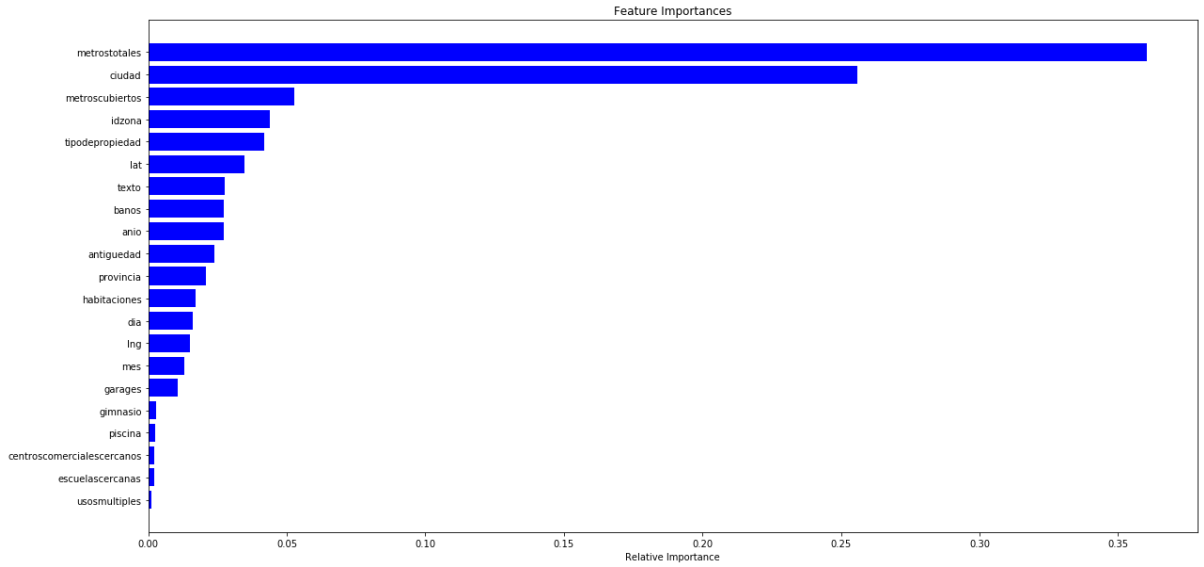


Podemos observar que tanto piscina, gimnasio, escuelas cercanas, centros comerciales cercanos, escuelas cercanas y usos múltiples son features que no aportan prácticamente nada. Mientras que los metros totales son el feature más relevante seguido por la provincia, algo lógico considerando que el tamaño y la ubicación de una propiedad afectan fuertemente en el precio.

Luego, probamos utilizando CatBoostEncoder para codificar las features categóricas. Aquí pudimos observar que la ciudad pasa a ser la más relevante de las categóricas, mientras que el tipo de propiedad mantiene la misma importancia. Por otro lado, la provincia pierde relevancia y pasa a ser superada por muchos otros features en importancia.



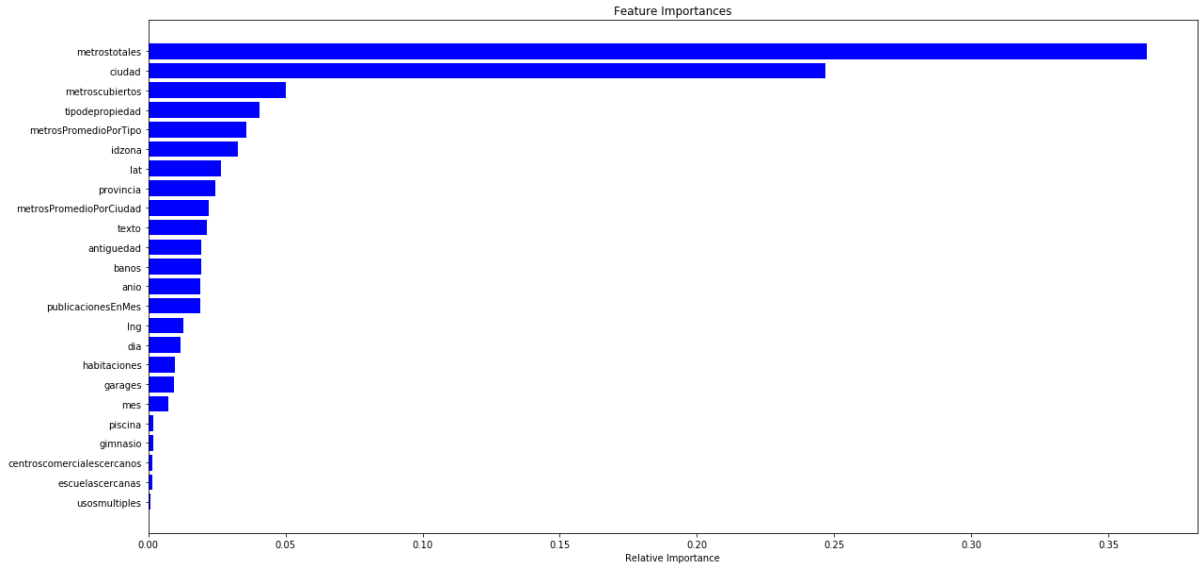
Con el objetivo de recuperar la importancia anterior de la provincia decidimos volver a codificar la provincia utilizando CountEncoder, lo mismo hicimos para el tipo de propiedad para observar si esto modifica su importancia.



Se puede ver que la importancia de tipo de propiedad y de provincia vuelve a decaer utilizando CountEncoder. Por lo tanto, concluimos que cuando posible es conveniente utilizar CatBoostEncoder para los features categóricos.

Continuando con el análisis decidimos crear alunos features para observar qué importancia tienen en comparación con los ya existentes. Primeramente creamos los siguientes:

- La cantidad de publicaciones por mes ('publicacionesEnMes')
- El promedio de metros totales por tipo de propiedad ('metrosPromedioPorTipo')
- El promedio de metros totales por ciudad ('metrosPromedioPorCiudad')

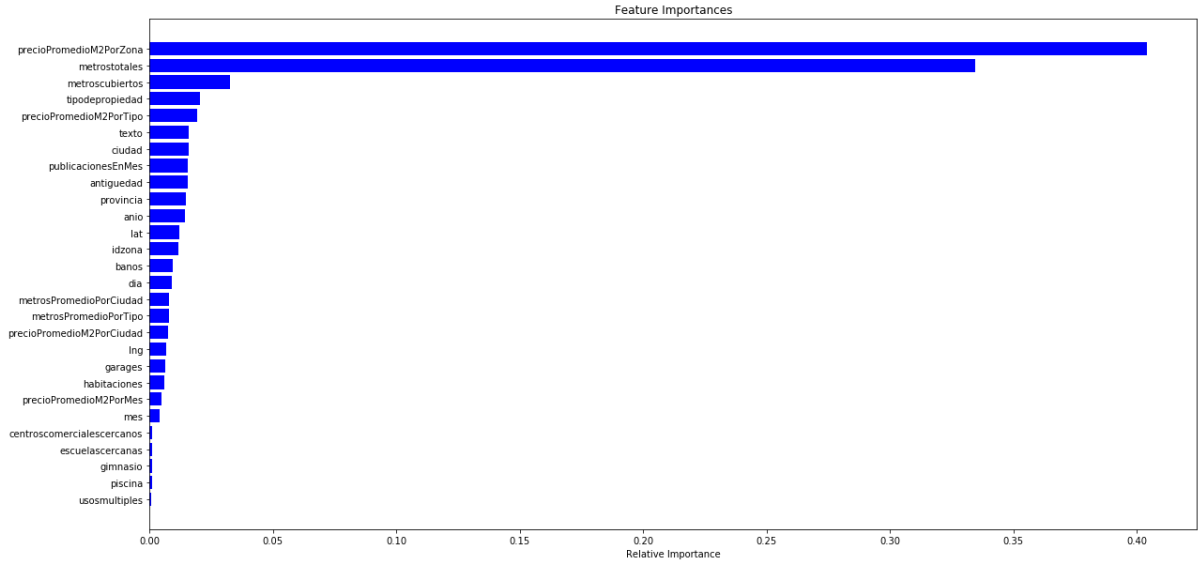


Podemos observar que el promedio de metros totales por tipo es el más relevante de los features creados. Pero también, que los otros dos features tienen una importancia razonable y por lo tanto merecen ser utilizados para realizar predicciones.

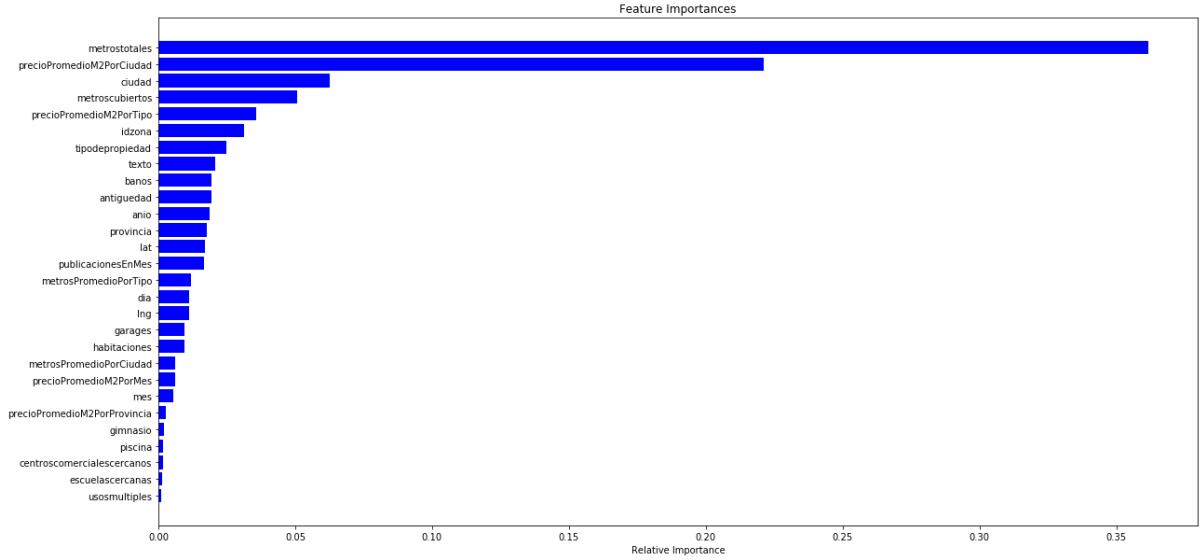
Agregamos mas features que se calculan en función del precio.

- El promedio del precio del metro total por ciudad (‘precioPromedioM2PorCiudad’)
- El promedio del precio del metro total por tipo de propiedad (‘precioPromedioM2PorTipo’)
- El promedio del precio del metro total por zona (‘precioPromedioM2PorZona’)

- El promedio del precio del metro total por mes (‘precioPromedioM2PorMes’)

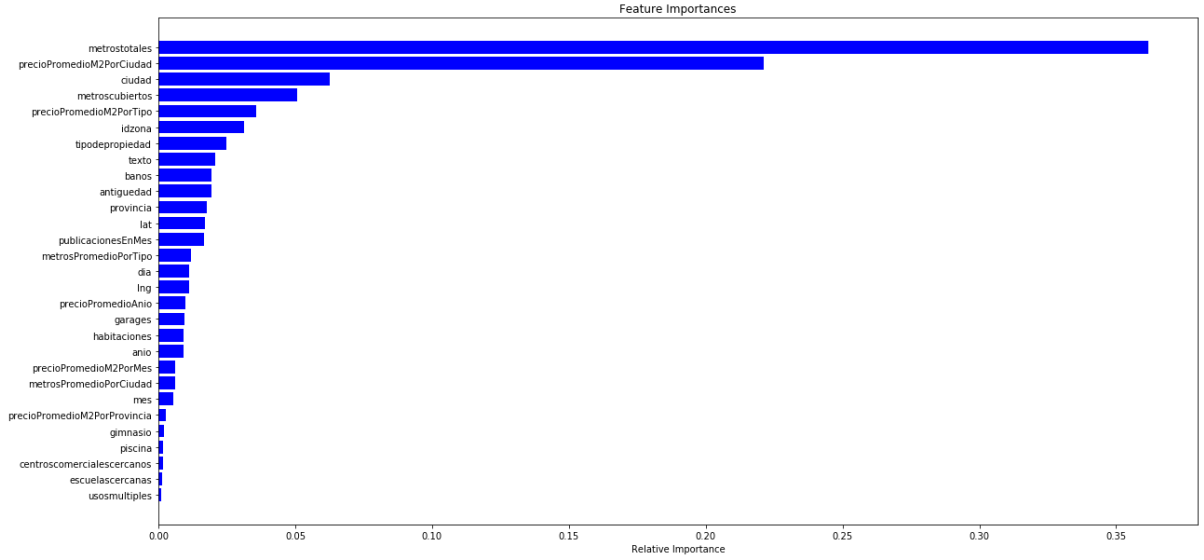


Aquí podemos observar que el precio promedio del metro cuadrado por zona pasa a ser el feature más relevante de todos. Esto se debe a que el id de zona es un feature muy específico que en muchos casos solo posee una propiedad en venta para esa zona lo que genera que utilizando los metros totales y el precio promedio podamos calcular el valor exacto de la propiedad. Esto, causa overfitting y por lo tanto, debe ser utilizado con mucho cuidado en las predicciones.

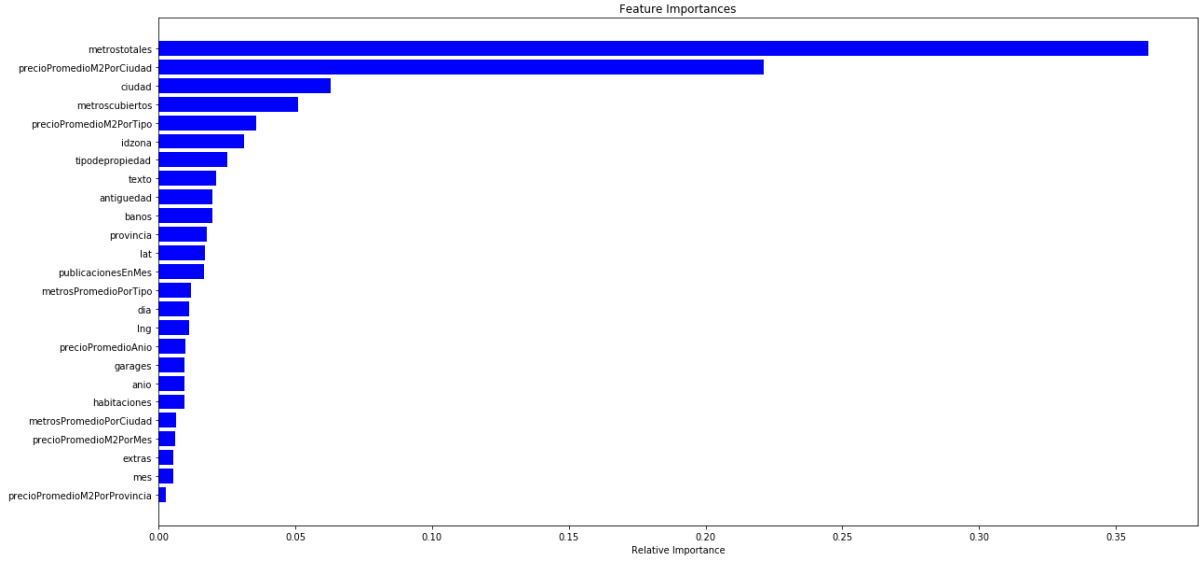


Eliminando el feature anterior, podemos analizar los demás features generados. El precio promedio del metro cuadrado por ciudad es el feature mas importante de los creados, pero también puede sufrir el mismo problema del precio por zona y debe ser utilizado con cuidado ya que si algunas ciudades aparecen muy pocas veces podemos tener overfitting. Sin embargo, el precio promedio por metro cuadrado por tipo de propiedad es un feature interesante y conviene ser utilizados, ya que no debería causar overfitting muy fácilmente ya que no tenemos demasiados tipos de propiedades y difícilmente tengamos alguna con muy pocas apariciones. Lo mismo no ocurre para ‘precioPromedioM2PorMes’ y ‘precioPromedioM2PorProvincia’.
Con el objetivo, de detectar la inflación agregamos, también, el promedio de precio por año y analizamos su importancia. Podemos observar que este feature no realiza lo esperado, ya

que consideramos que la inflación debe tener un impacto más elevado en el precio de las propiedades y aquí podemos ver que la importancia de este feature es baja.



Con la intención de aumentar la importancia de los últimos features (piscina, gimnasio, escuelas cercanas, centros comerciales cercanos, escuelas cercanas y usos múltiples) buscamos agruparlos y observar si de esta manera obtenemos un feature más relevante. De esta manera, creamos un nuevo features que llamamos extras, que posee una importancia un poco mayor pero que sigue siendo de los features menos relevantes.



Refinamiento Iterativo

Nuestra idea, a modo de cerrar un score final lo más bajo posible, fue la de, una vez encontrado un modelo que nos diera un resultado que pudiéramos considerar aceptable, utilizarlo de manera iterativa para lograr un refinamiento del resultado final. La idea principal de este sistema es la utilización de las predicciones de los precios del Dataframe de Test como precios “reales”, lo que nos permite concatenarlo y utilizarlo como parte del dataframe de Test, contando así con más datos sobre los que podemos trabajar.

El primer paso es la estimación inicial de precios en la que utilizamos sólo el dataframe de Train, evitando la creación de features relacionados con el precio, así como el algoritmo Cat Boost Encoder, que utiliza la columna “target”, en este caso el precio, para la codificación de las variables categóricas. Nos concentramos en esta primera iteración en la creación de otro tipos de features y en el uso de, por ejemplo, el One Hot Encoding, para la primer predicción de precios.

Como fue descrito anteriormente, lo que hacemos a continuación es concatenar los dataframes de Train y Test, contando ahora en el dataframe de Test con los precios estimados en la columna de los precios. A partir de aquí, realizamos una serie de 3 iteraciones en donde contamos con un dataframe más grande, de 300 mil filas, en las cuales nos concentramos en la creación de features relacionadas con el precio, siendo la más importante la estimación del precio del metro cuadrado. Además, utilizamos el Cat Boost para la codificación de algunas columnas, ahora que podemos replicar este proceso para la realización de predicciones.

Habíamos iniciado este proceso con una única segunda iteración, es decir, simplemente hacíamos dos pasos, uno sin concatenación y otro con ambos dataframes. Sin embargo, estimamos que si repetíamos el segundo paso una mayor cantidad de veces el resultado podría ir convergiendo hacia la solución real. Aclaramos que era una estimación, y que no contábamos con la certeza de que esto fuera a ocurrir.

En un primer momento no ocurrió, y las iteraciones posteriores arrojaban resultados levemente peores a la primera. Sin embargo, pudimos revertir esta tendencia modificando únicamente dos hiper-parámetros del modelo: los n-estimators y el learning rate. Iniciando la estimación con un número relativamente bajo de n-estimators y uno alto de learning rate. En las iteraciones posteriores, subimos la cantidad de n-estimators y disminuimos el learning rate simultáneamente y de forma progresiva, lo que terminó llevando a mejores resultados, por lo menos hasta la tercera iteración (sin incluir la primera predicción), a partir de la cual no logramos continuar mejorando la estimación.

Puntos Pendientes

Dentro del trabajo presentado, nos faltaron varios puntos que por falta de tiempo principalmente no logramos llevar a la práctica, o que a pesar de llevarlos a la práctica consideramos que nos hubiera requerido más tiempo implementarlos de manera satisfactoria.
Algunos de estos son los siguientes:

Tuneo de hiper-parámetros

Si bien hicimos muchas pruebas de modificación de hiper-parámetros, tanto de forma manual como de forma automática, sentimos que, al contar XG Boost con tantos hiper-parámetros diferentes, no pudimos hacer una cross-validation completa de todas sus combinaciones. Por un lado, por la demora realmente importante que implicaba probar esa cantidad de valores, y por otro, porque fuimos modificando distintos features y cuestiones externas al modelo en sí, lo que implicaba una nueva cross-validation en cada ocasión. Esta misma situación se repite para los otros modelos utilizados.

Combinación de modelos

Paralelización

Teníamos la idea de utilizar los diferentes modelos enumerados al principio del informe de forma paralela, una vez que ya hubiéramos creado los distintos features y codificado las diferentes columnas, para compararlos y ver cuál nos daba un mejor resultado.
Otra idea fue predecir con distintos modelos y realizar algún promedio ponderado de los mismos para lograr una mejor estimación.

Concatenación

Por otro lado, también queríamos “concatenar” diferentes modelos con la idea de que cada uno mejorara en cierta medida las estimaciones del anterior. Aunque no llegamos a implementarlo de forma total, nuestro notebook final es un primer acercamiento a esta idea, ya que utilizamos el mismo modelo de forma iterativa para ir mejorando las estimaciones que había sacado en la iteración anterior.

Crear features más relevantes

Mejorar el análisis de texto

Sentimos que en el análisis de texto realizado faltó profundizar con algún modelo que analice el texto únicamente. Dado que suponemos que hay varias características de las propiedades que se pueden sacar de sus descripciones y que afectan al precio que no pudimos obtener de manera concluyente debido a la dificultad de realizar un correcto procesamiento del texto. Algunas de estas ideas fueron:

- Realizar alguna clasificación de las descripciones mediante redes neuronales, o implementando algún tipo de puntaje de similaridad entre el texto y la propiedad, entre otros.
- Obtener de la dirección si está en alguna calle importante o céntrica.

Otros features

- Poder normalizar los precios o tener en cuenta la inflación de México.
- Incluir un índice de inseguridad, o similar para evaluar los precios.
- Agrupar propiedades por categorías como por ejemplo: rangos de precios e intentar sacar alguna información sobre las propiedades en esas categorías.

Conclusiones

Features

Nuestra búsqueda de features nos permite sacar ciertas conclusiones:
Los datos que tenemos permitían sacar mucha variedad de features, pero nosotros encontramos como los más importantes a la hora de inferir el precio, aquellos referidos a los metros del inmueble, su ubicación, y aquellos relacionados con el precio. Vimos poco impacto en aquellos features que daban características particulares de los inmuebles, dejándonos ante la complicada decisión de agregar features que filtraban nuestro target, mejorando considerablemente nuestro score, pero a la vez produciendo mucho overfitting.

Overfitting

Uno de las mayores dificultades que nos encontramos a lo largo del desarrollo del trabajo práctico fue el overfitting, tanto a la hora de detectarlo como a la hora de reducirlo. Para lo primero calculamos el score dividiendo el set de train para tener un subset con el cual validar y utilizamos gráficos que mostraban los scores sobre el set de train y el set de validación a lo largo de varias estimaciones de XGBoost. Estos no mostraron ser representativos del overfitting real, dado que los resultados que obtuvimos al mandar nuestras predicciones a la competencia fueron de un overfitting abrumadoramente mayor en varias ocasiones. Esto nos hizo pensar que teníamos features que filtraban datos del set de test al de entrenamiento.
Reducirlo fue realmente muy difícil, y en algunos casos imposible. En algunas ocasiones, estaba claro que el overfitting estaba directamente relacionado a un feature en particular, y encontramos la manera de reducirlo, pero fueron más frecuente las ocasiones en las que no pudimos ni predecir ni solucionar el overfitting de uno de los modelos utilizados. Incluso el último modelo utilizado, que es el que mejores resultados nos dio, cuenta con cierto overfitting, aunque en este caso entendemos que está fuertemente relacionado al uso del precio del metro cuadrado, y lo tomamos como parte necesaria de la obtención de nuestra mejor estimación.

Refinamiento Iterativo

Como pudimos observar en la sección de feature importance, utilizar el precio para las predicciones es algo que puede dar muy buenos resultados. Sin embargo, tiene un alto riesgo de causar overfitting ya que si contamos con alguna categoría que tiene una cantidad muy baja de apariciones estaremos agregando de manera indirecta el target al dataset de train. La manera de solucionar o de reducir este riesgo pensada por el grupo fue agregar varias iteraciones, una primera que no utilice el precio para la predicción y las demás que vayan mejorando este resultado utilizándolo.
De esta manera, no solo tenemos una mayor cantidad de datos que podemos utilizar, ya que tambien contamos con es dataset de test, si no que también reducimos el impacto del overfitting. Ya que en caso de que ocurra estaremos básicamente manteniendo la predicción anterior hecha sin utilizar el precio. Esta teoría en la práctica no tuvo tan buenos resultados

como los esperados, ya que el overfitting era alto y el score mejoró pero no demasiado. Aun así fue de esta manera que obtuvimos los mejores resultados y suponemos que tanto el overfitting como el score pueden ser mejorados utilizando mejores features que dependan del precio y tuneando los hiperparametros de los modelos utilizados.