Trabajo Práctico 2
Predecir precio de propiedades en México

Di Maria, Franco Martín, *Padrón 100498*
franki.dimaria@gmail.com

Raveszani, Nicole Denise, *Padrón 100193*
nraveszani@gmail.com

Kristal, Juan Ignacio, *Padrón 99779*
jkristal@fi.uba.ar

Marchesini, Ariel, *Padrón 100625*
arielmrchsn@gmail.com

Repostorio : https://github.com/juankristal/7506-Datos

2do. Cuatrimestre de 2019
75.06 - Organización de Datos
Facultad de Ingeniería, Universidad de Buenos Aires

Diciembre 2019

Contents

1 Introducción: 3

2 Herramientas (y librerías) utilizadas 3
2.1 Entorno remoto . . . 3

3 Limpieza set de datos 3

4 Features probados 4
4.1 Caracteristicas basicas . . . 4
4.2 Encoding . . . 5
4.2.1 Categorias: tipodepropiedad, ciudad, provincia . . . 5
4.3 Explotando top feature: metroscubiertos . . . 5
4.4 Calculando distancias . . . 6
4.5 Explotando Target : precio . . . 6
4.6 Strings : titulo, descripcion, direccion . . . 7
4.6.1 Features masivos . . . 7
4.6.2 Clasificando descripciones . . . 7
4.7 Aumentar peso de pequeños features . . . 8

5 Modelos de predicción 9
5.1 Modelos probados . . . 9
5.2 RandomForestRegressor . . . 9
5.3 LightGBM . . . 11
5.4 XGBRegressor . . . 12

6 Tuneo de XGBRegressor 13

7 Conclusiones: 15
7.1 A continuar . . . 16

1 Introducción:

El objetivo de este trabajo es predecir los precios pertenecientes a propiedades en venta durante el periodo comprendido entre 2012 y 2016, ubicadas en el territorio de México. Para ello, se han utilizado los métodos de Machine Learning vistos en clases, como la elección de un modelo y su tuneo, la obtención de features y qué importancia tienen para nuestro set de datos y la elección de encodings.

Las predicciones se encuentran basadas en los datos otorgados por ZonaProp.

2 Herramientas (y librerías) utilizadas

Todos los codigo de entrenamiento, tunneo, 'feature engineering' y 'feature selection' se realizaron en jupyter-notebook, utilizando 'pandas' y 'numpy'.

Las herramientas usadas para el 'tunning' de hyper-parámetros fueron:

- sklearn.RandomSearchCV
- sklearn.GridSearchCV
- hyperopt.fmin

En un principio se utilizó mucho RandomSearchCv, por su velocidad de tuneo. GridSearchCv se utilizó poco, puesto que utilizar la librería hyperopt resultaba ser mas eficiente. Esta última se uso mucho.

La librera hyperopt permite hacer tunneos donde los algoritmos se encargan de acotar el intervalo de búsqueda de hyper-parámetros, alrededor de aquellos donde se obtuvo mejores resultados.

Librerias para modelos de predicción base:

- skelearn
- xgboost
- lightgbm

Además, sklearn cuenta con muchos mas algoritmos que sirvieron para la creación de features, y la selección de ellos, entre otras cosas.

2.1 Entorno remoto

Se descubrió que con el mail de la Facultad de Ingeniería de la Universidad de Buenos Aires (@fi.ub.ar), se obtiene acceso con cierta libertad a jupyter-notebooks de Google (Google Colab).

Estos notebooks vienen con librerías pre-instaladas (sklearn, xgboost, etc), y corren de forma remota en equipos de esta empresa. Estos equipos parecen ser mejores de los que se utiliza en forma local (equipos de los integrantes del grupo), por lo que permite ahorrar tiempo en tuneos que, de tardar 8 horas en equipos locales pasan a tardar 3 horas en equipos remotos. Además, permite ejecutar hasta un máximo de entre 4 y 5 notebooks al mismo tiempo, lo que permite tunnear varios modelos en simultaneo. La única demanda que se le exige a los equipos locales es acceso continuo a internet, mientras se ejecuten los notebooks. Los equipos locales se mantienen encendidos, pero no se consume sus recursos.

3 Limpieza set de datos

Con el fin de facilitar los tuneos, y creación de nuevos features, se limpiaron los sets de datos: train.csv y test.csv, reemplazando NaN por diversos valores.

Para empezar se aprovechó los datos que proveen ambos sets, combinándolos para crear un nuevo set de datos con mas registros, es decir, con mas información, sin la columna de 'precio'.

Las columnas con valores faltantes son:

- titulo
- descripcion
- direccion
- tipodepropiedad
- provincia

- ciudad
- habitaciones
- banos
- garages
- antiguedad
- metrostotales
- metroscubiertos
- idzona
- lat
- lng

Las columnas título, descripción y dirección, se pre-procesaron a aparte, pero básicamente, se completo los NaN, con strings 'nan', 'e', o cadenas vacías.

Las columnas tipodepropiedad y idzona, se completaron con la estadística 'moda' por ciudad. En caso de faltar alguna, se calcula la moda por provincia, y en última instancia por la moda del país.

Las columnas antiguedad, habitaciones, banos, garages, se completaron con la estadística 'moda' por tipodepropiedad, ciudad, provincia y país.

Las columnas metrostotales y metroscubiertos se completaron una con la otra. Donde faltaba metrostotales, se colocaron los metroscubiertos, y viceversa pues fue observado en el análisis exploratorio que había una fuerte relación lineal entre ellos. No hizo falta calcular ninguna estadística. Un error cometidos aquí fue que no se tuvo en cuenta el tipo de propiedad, por lo que de haber un terreno con metros cubiertos en NaN, se habrán completado con el total de metros totales, en lugar de con cero metros que es lo mas lógico.

La columna ciudad se completo con la moda por provincia.

La columna provincia se completo con la moda del país.

Las columnas lat, y lng, se completaron con datos externos, correspondientes a las latitudes y longitudes del centro de las ciudades y provincias. En caso, de faltar alguna, se utilizó el promedio de latitudes y longitudes por idzona, y en última instancia se completa con la latitud y longitud del centro del país.

4 Features probados

4.1 Caracteristicas basicas

- antiguedad : antigüedad en años
- habitaciones : cantidad de habitaciones
- garages : cantidad de garages
- banos : cantidad de baños
- metroscubiertos : metros cubiertos
- metrostotales : metros totales
- idzona : id de la zona donde se encuentra
- lat : latitud
- lng : longitud
- gimnasio : 1 si tiene gimnasio; 0 en caso contrario
- usosmultiples : 1 si tiene usos multiples; 0 en caso contrario
- piscina : 1 si tiene piscina; 0 en caso contrario
- escuelascercanas : 1 si tiene escuelas cercanas; 0 en caso contrario
- centroscomercialescercanos : 1 si tiene centros comerciales cercanos; 0 en caso contrario

- servicios_cercanos : suma de features: escuelascercanas + centroscomercialescercanos
- utilidades_extra : suma de features: gimnasio + ususmultiples + piscina
- cantidad_espacios : sumatorio de features: habitaciones + garages, banos + gimnasio + ususmultiples + piscina
- año : año de publicación
- mes : mes de publicación
- dia : día de publicación

4.2 Encoding

4.2.1 Categorias: tipodepropiedad, ciudad, provincia

La idea es codificar las variables categoricas: tipodepropiedad, ciudad, provincia, en base a los promedios de las 'caracteristicas basicas'.

Para hacer al encoding mas 'jugoso' en contenidos, se calcularon promedios por grupos, que se van agrandando en base a la categoría:

Para cada tipodepropiedad se calculó el promedio por característica básica, dentro de grupos de la misma provincia, y ciudad. En caso de que falte algún tipodepropiedad por grupo, se toma el promedio de la siguiente capa del grupo, es decir, el promedio por provincia, y en ultima instancia el promedio general para todo México.

De la misma forma se codificaron ciudad y provincia.

- Promedio de antiguedad, habitaciones, garages, banos, metroscubiertos, utilidades y servicios cercanos según tipodepropiedad, ciudad y provincia.
- Latitud promedio por ciudad y provincia
- Longitud promedio por ciudad y provincia.
- Producto interno canónico entre las coordenadas de la propiedad y las coordenadas del centro de la ciudad, provincia y país a la que pertenece. Se calcula como: (xLAT,yLNG)*(CentroLAT, CentroLNG)

Otros encodings probados fueron One Hot Encoding, LabelEncoding, y TargetEncoding, pero no se obtuvo tan buenos resultados como en las anteriores codificaciones.

4.3 Explotando top feature: metroscubiertos

Durante el análisis exploratorio se observó que las propiedades mas caras solían ser aquellas con una mayor cantidad de metros cubiertos. Por ese motivo se decidió usar como métrica a la hora de visualizar el valor del metro cuadrado para diferentes lugares. Esto resultó ser útil durante la elaboración de los modelos predictivos.

Esperadamente, en los análisis de importancia de los modelos usados, el feature 'metroscubiertos' es que el suele tener mayor peso en general.

Se toma los features que suelen pesar menos: habitaciones, garages, banos, utilidades_extra y combinarlos con esta top feature, para balancear el modelo.

Cabe destacar, que 'metrostotales' también es un feature con gran peso en los modelos, pero los análisis realizados en el TP1 nos revelaron una relación cuasi-lineal entre el mismo y 'metroscubiertos'. Bajo este análisis se concluye que los features obtenidos a partir de uno podrían ser muy similares al obtenido del otro, por consiguiente, solo se crearon features a partir de 'metroscubiertos'.

- metros_no_cubiertos : metros totales - metros cubiertos
- ratio_metros_cubiertos : metroscubiertos / metrostotales
- metros_x_espacio : metroscubiertos / cantidad_espacios
- metros_x_habitaciones : metros_x_espacio * habitaciones
- metros_x_garages : metros_x_espacio * garages
- metros_x_banos : metros_x_espacio * banos
- metros_x_utilidades_extra : metros_x_espacio * utilidades_extra

4.4 Calculando distancias

Tras la corrección del TP1, comprendimos que la cercanía a zonas comerciales, era un factor que nos faltó analizar y que resultaba tener una importante relación con el precio de las propiedades. Si tenemos en cuenta que las oficinas y locales comerciales suelen generar zonas donde el valor inmobiliario aumenta (suposición), entonces es lógico suponer el precio de las propiedades variara en función de la distancia a estas zonas.

Para el cálculo de las distancia se utilizo NearestNeighbors. En principio, se intento utilizar LSHForest, pero hubo problemas con la librería sklearn, puesto que para ciertas versiones este objeto no se encontraba disponible. Se decidió no perder tiempo, y continuar con NearestNeighbors.

Aquí se cometieron 2 errores. Por un lado, suponer que la cercanía a tipos de propiedad comerciales resultaría mejor que la cercanía a otros tipos de propiedad, sin tener un análisis previo. Y, por otro lado, se calculó la distancia a la propiedad más cercana de cada uno de estos tipos de propiedad, olvidándose que de que esa propiedad podría no estar rodeada de otras propiedades del mismo tipo, sino que ser un caso anómalo. En otras palabras, se estaba overfiteando la cercanía a zonas comerciales, usando KNN con K igual a 1.

Para solucionar estos problemas, se reconstruyeron los features creados utilizando la distancia promedio a las 10 propiedades mas cercanas (K = 10) para cada tipo de propiedad incluyendo las propiedades no comerciales. En caso de aquellas propiedades que contaban con menos de 10 registros en el set de datos, se utilizó la distancia al propiedad mas cercana de dicho tipo (K = 1).

Además, a este nuevo set de datos, también se le agrega distancia promedio a propiedades cercanas, con elevado valor inmobiliario, así como distancia promedio a propiedades cercanas, con bajo valor del mismo.

Un último feature agregado en este set es la distancia al centro de la provincia Distrito Federal. En el TP1 vimos que el precio de las propiedades tiende a ser mayor en esta provincia, por lo que las distancia de la propiedad al centro de la misma, podría darnos buenos resultados.

A pesar de la reconstrucción de los features anteriores, los mejores resultados obtenidos se mantuvieron con la distancia para el vecino mas cercanos (K = 1), por lo mantuvimos estas features en el último set de datos. Para distancias, nos referimos a la distancia Euclidiana.

Cuando hablamos de propiedades comerciales nos referimos a Terrenos, Locales, Oficinas, Locales en Centros comerciales, Bodegas, Garages, Hospedaje, Huertas y Lotes)

- Distancia mínima a las propiedades comerciales
- Distancia promedio a las 10 propiedades comerciales mas cercanas
- Distancia promedio a las 10 propiedades no comerciales (es decir, el resto de los tipos de propiedad) más cercanas.
- Distancia al centro de la ciudad, provincia y país.
- Distancia promedio a las 10 propiedades con precio mayor a $ 5.305.000,00 mas cercanas
- Distancia promedio a las 10 propiedades con precio menor a a $ 631.386,25 mas cercanas.
- Distancia a Distrito Federal.

4.5 Explotando Target : precio

Se utiliza el mismo target para calcular features que contengan el precio promedio según diversas asociaciones. Se utilizo a la ciudad como representación geográfica suficiente para cada propiedad.

- banos_preciopromedio_ciudad : precio promedio por ciudad por cantidad de baños.
- habitaciones_preciopromedio_ciudad : precio promedio por ciudad por cantidad de habitaciones.
- garages_preciopromedio_ciudad : precio promedio por ciudad por cantidad de garages.
- banos_preciopromedio_metroscubiertos : precio promedio por metros cubiertos por cantidad de baños.
- habitaciones_preciopromedio_metroscubiertos : precio promedio por metros cubiertos por cantidad de habitaciones.
- garages_preciopromedio_metroscubiertos : precio promedio por metros cubiertos por cantidad de garages.
- precio_x_m2 : precio promedio por metro cuadrado por ciudad.
- tipodepropiedad_mean_precio : precio promedio por tipo de propiedad.

4.6 Strings : titulo, descripcion, direccion

4.6.1 Features masivos

Se procesa cada uno de estas características: titulo, descripcion, dirección, contando y eliminando los signos de puntación, asi como las 'stopwords' para el idioma español. Posteriormente, se generan features que cuentan distintas repeticiones.

Las siguientes features fueron aplicadas a todos los campos de texto (titulo, descripcion y direccion)

- Cantidad de palabras y caracteres. Para palabras se evitaron contar las stopwords y para caracteres los signos de puntuación.
- Longitud media de palabras.
- Cantidad de stopwords y signos de puntuación.
- Cantidad de palabras en mayúsculas (se evitaron las direcciones para dicho feature)
- Cantidad de palabras, prefijos y post-fijos (de largo 2) pertenecientes al top K de frecuencia (K = 50) y al bottom K (K = 10000)

4.6.2 Clasificando descripciones

Se procesa la 'descripcion' eliminando los signos de puntación, tildes, expresiones en HTML, así como las 'stopwords' para el idioma español e ingles.

Se dividen a las publicaciones en 7 grupos según su precio:

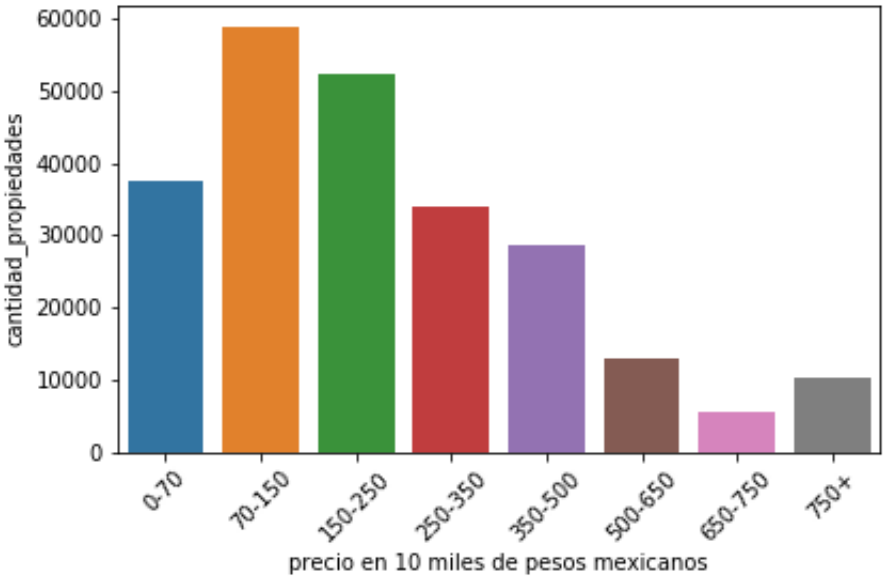

Figure 1: Gráfico de propiedades por bloque de precio

Posteriormente, se vectoriza a las descripciones con TfidfVectorizer, y se busca los que mas peso tienen. Para ello se utiliza la distribución chi2, obteniendo distintos conjuntos de bi-terminos que parecen tener una fuerte correlación con la categoria de precio.

Conjuntos de dos palabras mas correlacionados:

0. $ [0 - 70] $*10^4$ - patio servicio, cochera auto, cocina tarja, credito infonavit
1. $ [70 - 150] $*10^4$ - principal vestidor, lavado cuarto, servicio baño, cuarto servicio
2. $ [150 - 250] $*10^4$ - recamaras principal, recamaras baño, credito infonavit, principal baño
3. $ [250 - 350] $*10^4$ - principal vestidor, cochera auto, servicio baño, cuarto servicio
4. $ [350 - 500] $*10^4$ - lavado cuarto, patio servicio, servicio baño, cuarto servicio
5. $ [500 - 650] $*10^4$ - salon fiestas, family room, salon juegos, cuarto servicio
6. $ [650 - 750] $*10^4$ - recamaras vestidor, salon juegos, cuarto servicio, family room
7. $ [+ 750] $*10^4$ - cualquier tipo, cto servicio, cristal templado, cuarta recamara

Vemos que muchos de estos términos correlacionados por grupo, se repiten en varios de ellos, por lo que es probable que la clasificaciones que se obtengan de estos datos no resulte muy buena.

Se prueba rankear las descripciones utilizando CountVectorizer y TfidfTransformer, para luego clasificarlas con MultinomialNB (Naive Bayes), pero los resultados fueron desastrosos, obteniendo una precision del 33%.

Finalmente, se obtuvieron mejores resultados simplemente con la frecuencia de cada bi-termino por cada una de las categorías antes mencionadas (0,1,2,3,4,5,6,7)

4.7 Aumentar peso de pequeños features

Durante la etapa de 'feature selection', se intento combinar features de poca importancia, para nutrir a los modelos de features con mayor contenido.

- distancia_euclideana_al_origen : norma euclidiana de la posicion: $\sqrt{LAT^2 + LNG^2}$.
- distancia_minima_comercial : distancia euclidiana mínima entre Local en centro comercial, Bodega comercial, Oficina comercial.
- producto_interno_maximo_ciudad_pais : valor máximo entre producto_interno_centro_ciudad y producto_interno_centro_pais .
- ciudad_mean_antiguedad_sobre_provincia_mean_antiguedad : ciudad_mean_antiguedad / provincia_mean_antiguedad.
- tipodepropiead_mean_utilidades_extra_sobre_ciudad_mean_utilidades_extra : tipodepropiead_mean_utilidades_extra / ciudad_mean_utilidades_extra
- antiguedad_sobre_tipodepropiedad_mean_antiguedad : antiguedad / tipodepropiedad_mean_antiguedad
- direccion_cantidad_al_menos_una_mayuscula : cantidad de palabras que estan en mayuscula o su primer letra es mayscula
- direccion_cantidad_fijos_top_k : direccion_cantidad_prefijos_top_k + direccion_cantidad_postfijos_top_k (K = 50)
- titulo_cantidad_fijos_top_k : titulo_cantidad_prefijos_top_k + titulo_cantidad_postfijos_top_k (K = 50)
- titulo_palabras_top_k_sobre_total_palabras : cantidad de palabras que aparecen en el top K (K = 50) de la palabras mas frecuentes, sobre la cantidad de palabras totales.
- ciudad_distancia_al_origen : norma euclideana de la posicion promedio de la ciudad: $\sqrt{CiudadMeanLAT^2 + CiudadMeanLNG^2}$.
- ciudad_mean_mean_todas : promedio de los promedios por ciudad de habiaciones, banos, garages, utilidades_extra, servicios_cercanos
- ciudad_mean_antiguedad_sobre_mean_metrocubiertos : ciudad_mean_antiguedad / ciudad_mean_metrocubiertos

5 Modelos de predicción

5.1 Modelos probados

Se probaron de los siguientes modelos:

- K Nearest Neighbors Regressor (sklearn.neighbors.KNeighborsRegressor)
- Random Forest Regressor (sklearn.ensemble.RandomForestRegressor)
- XGBoost Regressor (xgboost.XGBRegressor)
- XGBoost + Random Forest Regressor(xgboost.XGBRFRegressor)
- Adaboost Regressor + Random Forest Regressor
  (sklearn.enemble.AdaboostRegresor + sklearn.ensemble.RandomForestRegressor)
- Lightgbm (lightgbm.LightGBMRegressor)

El primer modelo probado, en instancias muy tempranas del trabajo fue KNN, sin features agregadas, con encodings simples (One Hot Encoding, TargetEncodign) e imputando los NaN's de manera simple (promedio). Sin embargo los resultados estuvieron muy por debajo de Random Forest y XGBoost, por lo que se descartó este modelo. Más adentrados en el trabajo, se realizó un tuneo de KNN con features añadidas y mejores formas de codificar e imputar valores, el desempeño fue mucho mejor que en la primera iteración pero seguía estando por debajo de los anteriores mencionados. Por lo que se descartó completamente este modelo.

Luego se continuo el trabajo utilizando solamente RandomForestRegressor y XGBRegressor, y en un principio, con muy pocos features, obtuvimos mejores resultados con RandomForest que con XGBoost.

Dado nuestra base teórica, sabiamos que xgboost debería darnos mejores resultados. Se investiga y se encuentra que existe un xgboost regressor que estaba combinado con el random forest. Dados nuestros buenos resultados con random forest se supuso que en combinación con xgboost debería mejorar los resultados aun más. Lamentablemente los resultados fueron desastroso: los precisión del modelo fue menor y el tiempo de ejecución aumento.

También se probó la combinación AdaboostRegressor utilizando RandomForestRegressor como estimador base, pero los resultados fueron similares a los de XGBboost junto a Random Forest.

Se opto por descartar estos modelos, y continuar tuneando RandomForestRegressor, así como XGBRegressor.

Eventualmente, con la creación de diversos features, y mejores hyper-parámetros, se logro un mejor resultado con XGBRegressor.

Por último, con varios sets armados que ya se habían probado con anterioridad con XGBoost, se utilizó un nuevo modelo: LightGBM. Con los mismos sets, no se logró obtener un mejor resultado que con XGBoost, pero se lograron mejores resultados que con Ranfom Forest.

5.2 RandomForestRegressor

Mejores hyper-parametros:

- max_depth = 17,
- max_features = 13,
- min_samples_leaf = 8,
- min_samples_split = 5,
- n_estimators = 456

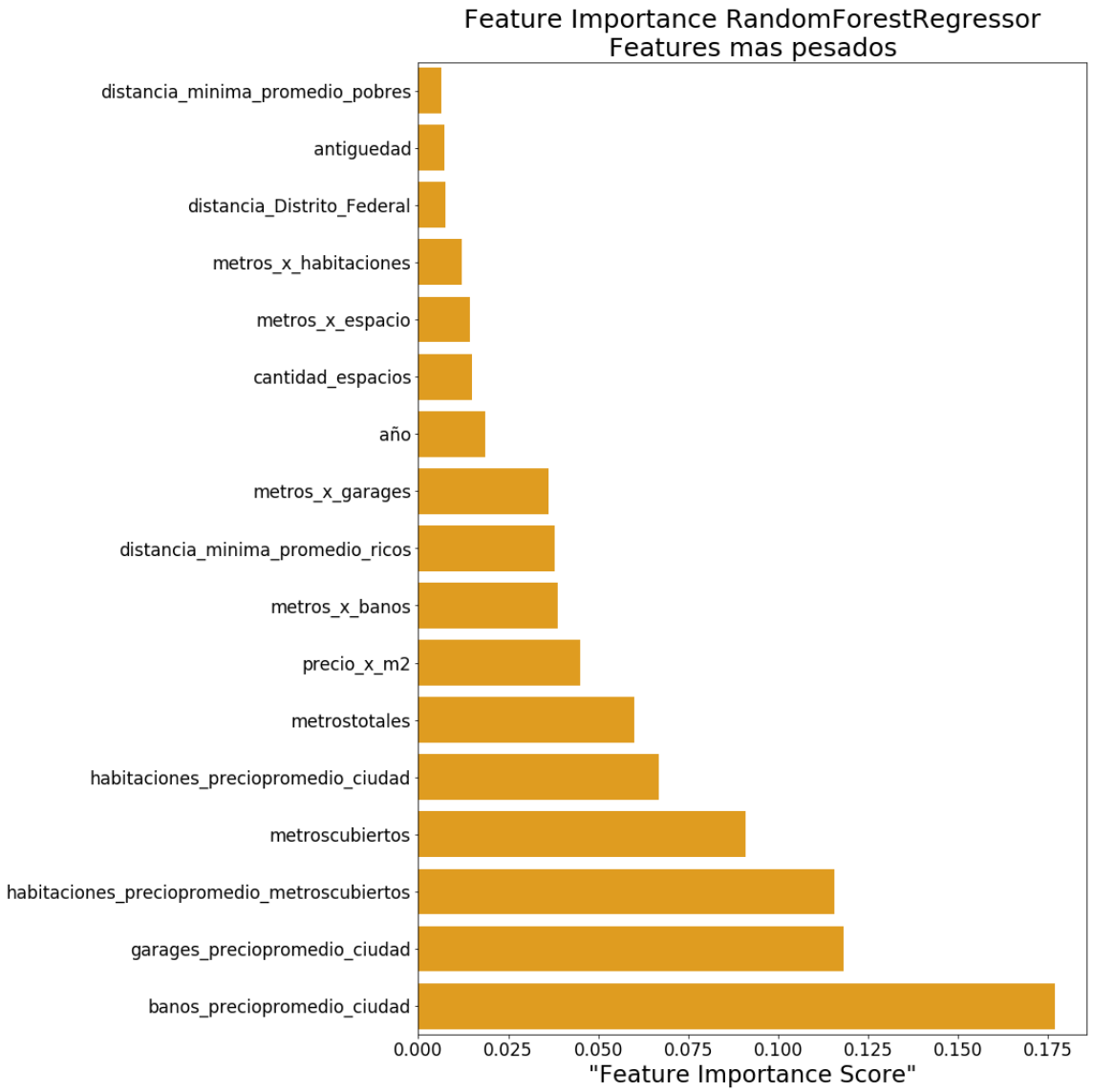

Figure 2: Gráfico de feature importance del submit final

Este es uno de nuestros ultimos intentos con el RandomForestRegressor, utilizando las ultimas features agregadas. Lamentablemente muchas de ellas obtuvieron un peso bastante bajo en el modelo, por lo que no se muestran en el grafico.

Los features faltantes son:

- distancia_promedio_Local_Comercial_10_mas_cercanos
- distancia_promedio_Terreno_comercial_10_mas_cercanos
- distancia_promedio_Casa_uso_de_suelo_10_mas_cercanos
- distancia_promedio_Edificio_10_mas_cercanos
- distancia_promedio_Duplex_10_mas_cercanos
- distancia_promedio_Quinta_Vacacional_10_mas_cercanos
- distancia_promedio_Villa_10_mas_cercanos
- distancia_promedio_Inmuebles_productivos_urbanos_10_mas_cercanos
- distancia_promedio_Bodega_comercial_10_mas_cercanos
- distancia_promedio_Casa_en_condominio_10_mas_cercanos

- distancia_promedio_Local_en_centro_comercial_10_mas_cercanos
- distancia_promedio_Terreno_10_mas_cercanos
- distancia_promedio_Apartamento_10_mas_cercanos
- distancia_promedio_Lote_10_mas_cercanos
- distancia_promedio_Rancho_10_mas_cercanos
- distancia_promedio_Casa_10_mas_cercanos
- distancia_promedio_Otros_10_mas_cercanos
- distancia_promedio_Oficina_comercial_10_mas_cercanos
- distancia_promedio_Departamento_Compartido_10_mas_cercanos

Una mejora en este modelo, podría ser reducir la cantidad de max_features para darle oportunidad a otros features de resaltar más. Además, el max_depth es bastante alto, por lo que también se podría reducir.

5.3 LightGBM

Mejores hyper-parámetros:

- reg_lambda = 10,
- reg_alpha = 7,
- num_leaves = 30,
- n_estimators = 400
- min_child_weight = 100,
- max_depth = 60
- learning_rate = 0.3

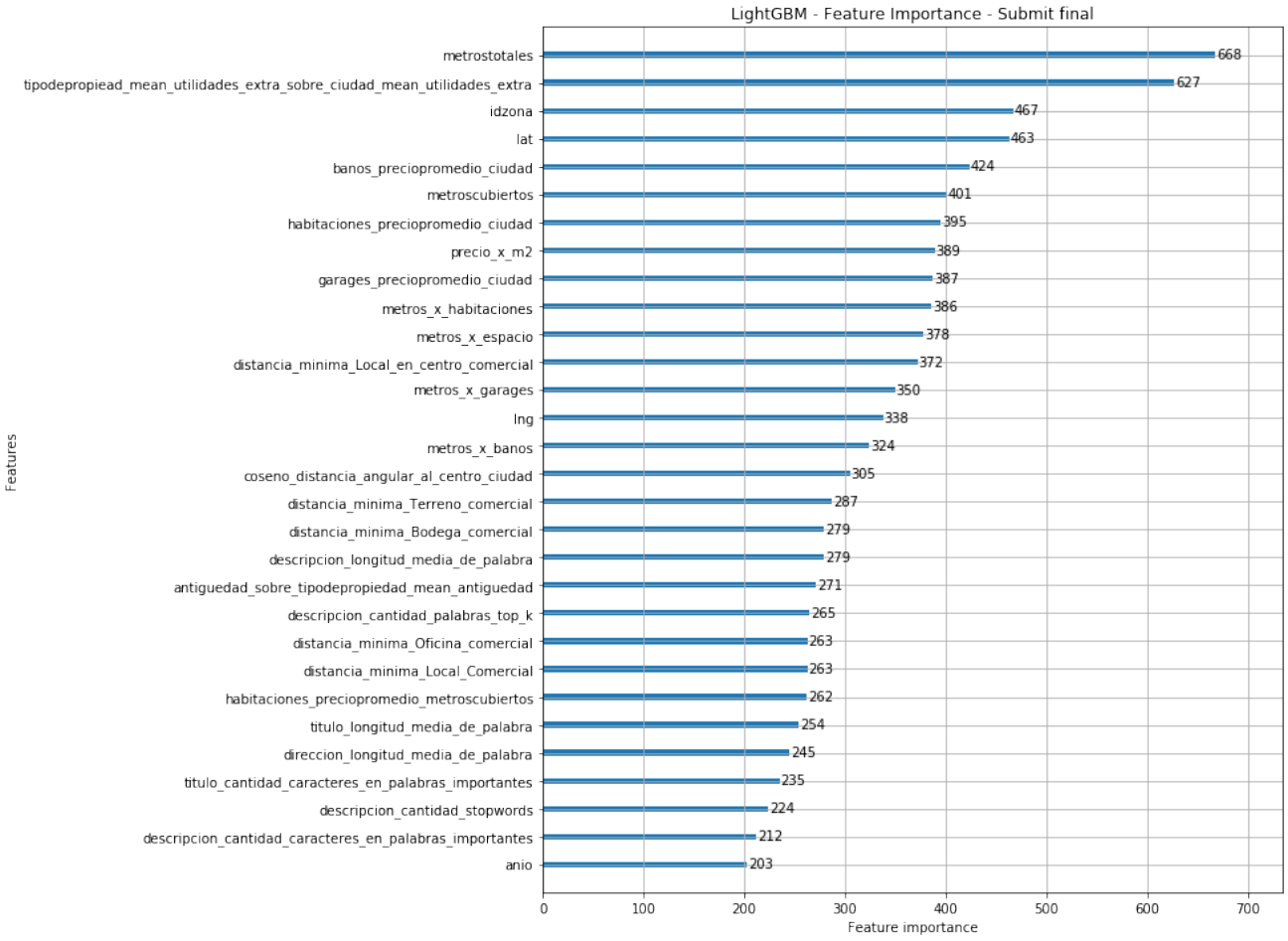

Figure 3: Gráfico de feature importance del submit final

Se puede ver que, en su mayoría, las features relacionadas a las descripciones son las que menor importancia tienen.

Unas mejoras que se podrían probar, serían aumentar n_estimators y num_leaves teniendo cuidado de no causar overffiting.

5.4 XGBRegressor

Mejores hyper-parametros:

- colsample_bytree = 0.5391492353697619,
- gamma = 1.9431723255672306,
- learning_rate = 0.06832619437670823,
- max_depth = 10,
- min_child_weight = 1,
- n_estimators = 943,
- objective = 'reg:squarederror',
- reg_lambda = 0.1654632901595984,
- subsample = 0.9627179797311021

Grafico de 'Feature Importance'

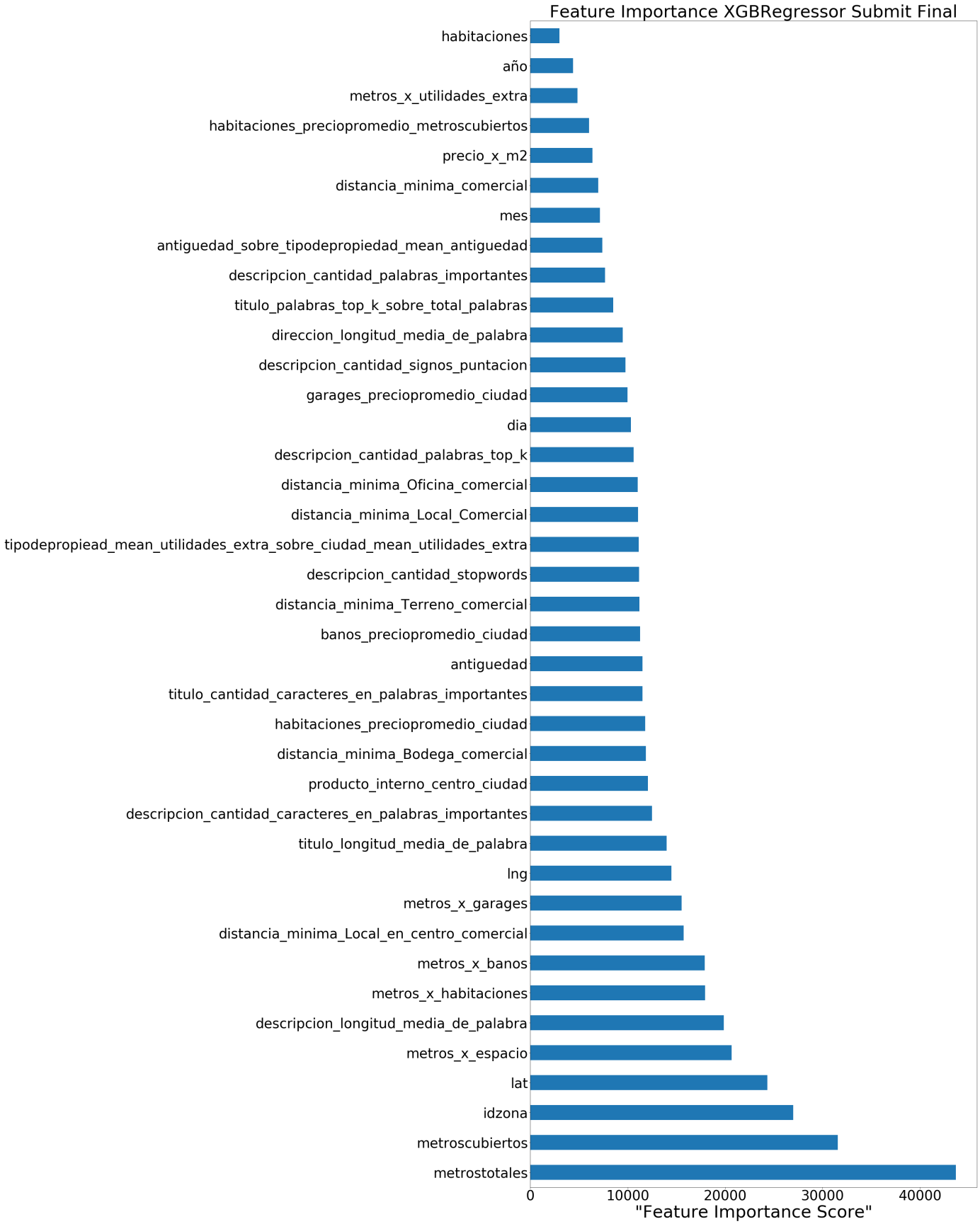

Figure 4: Gráfico de feature importance del submit final

6 Tuneo de XGBRegressor

En principio se tuneo un modelo de manera simple, con LabelEncoding y sin ninguna features agregada, solo quitando features sin procesar, como: lat, lng, idzona, titulo, descripcion, etc. Inmediatamente se decidió

aumentar un poquito la complejidad del tuneo dando la posibilidad de distintos tipos de encoding, se puso a prueba en el mismo contexto del anterior tuneo pero con la opción de hacer LabelEncoding o TargetEncoding. Como era de esperarse TargetEncoding dio mejores resultados y mas consistentes.

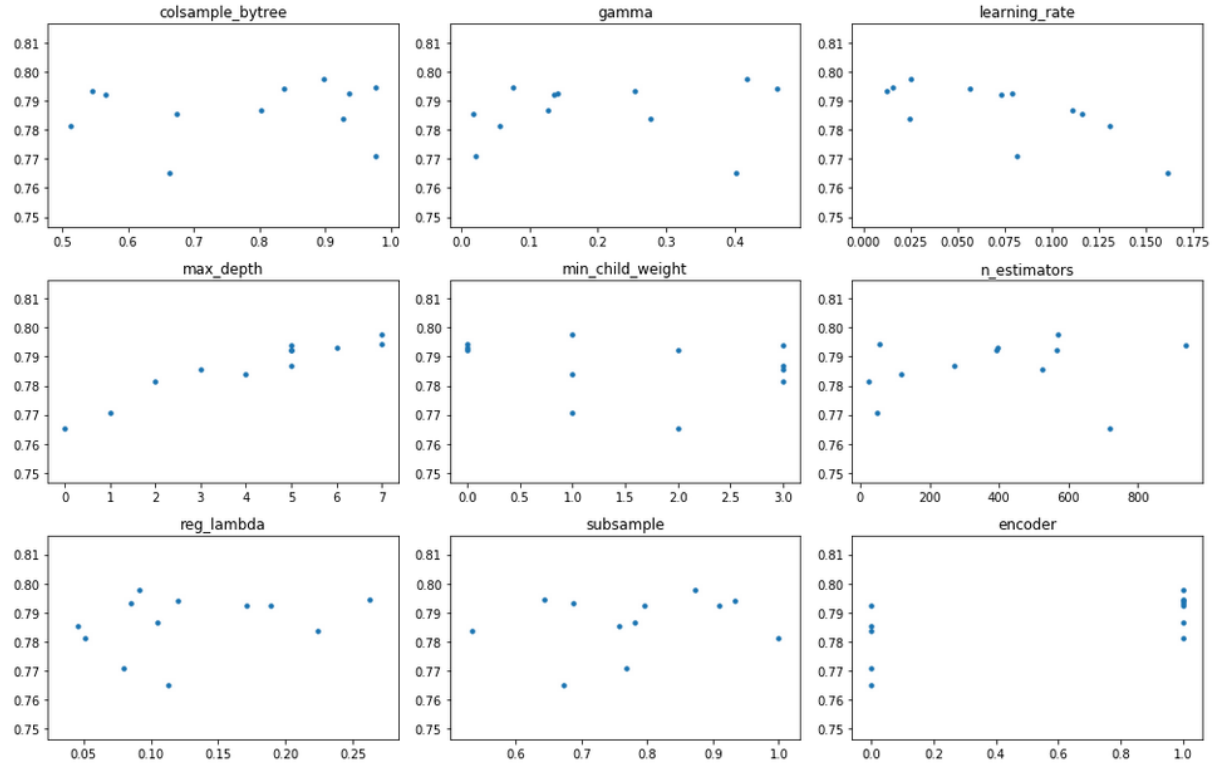

Figure 5: Gráfico de resultados según cada hyperparametro por separado

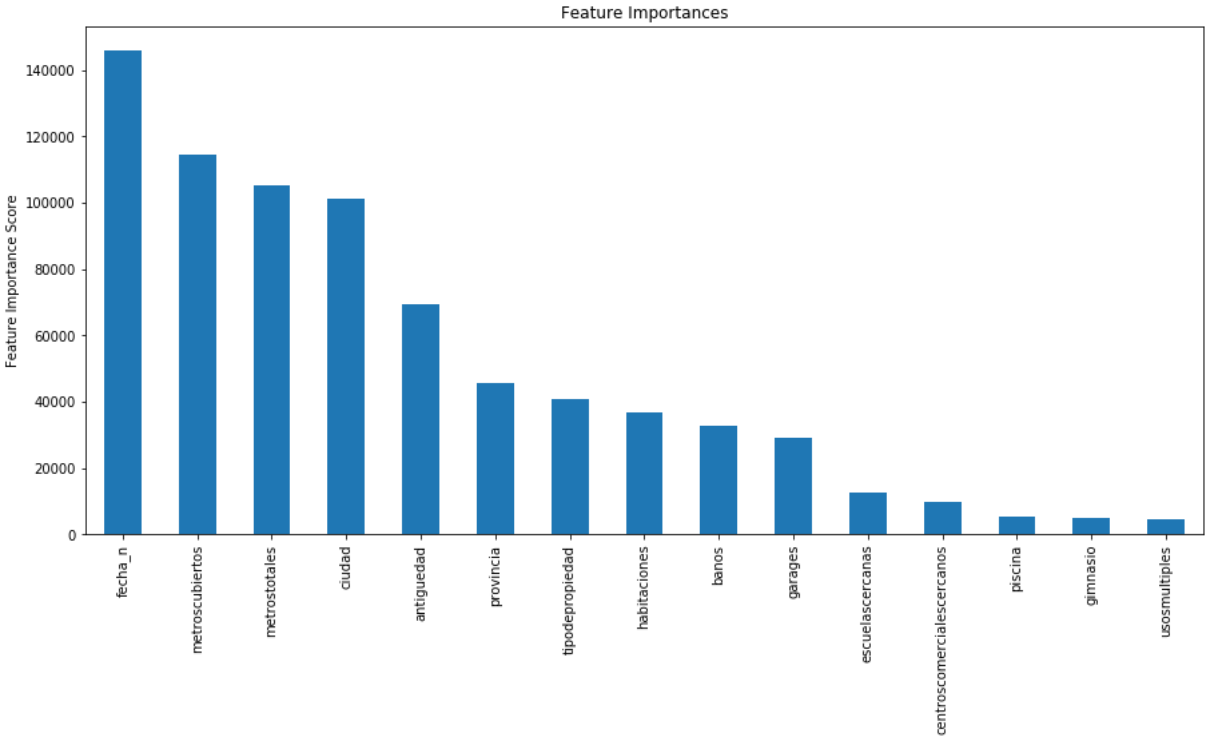

Figure 6: Gráfico de feature importance

Aquí se puede ver lo antes mencionado con los encoders (0 representa LabelEncoding y 1 representa TargetEncoding).

Las conclusiones que se sacaron de la figura 5 es que se podría achicar el rango del learning rate por la tendencia que tiene entre los valores 0.01 y 0.1. También el hyperparametro max_depth tiene tendencia a crecer, por lo que se podría aumentar el rango para ver si esa tendencia sigue hacia mayores valores. Da la figura 6 se llegó a separar de mayor importancia a menor los siguientes grupos, grupo 1: "ciudad", "metroscubiertos", "metrostotales"; grupo 2: "fecha_n", "provincia", "tipodepropiedad", "antiguedad"; grupo 3: "habitaciones", "banos", "garages"; grupo 4: "piscina", "gimnasio", "escuelascercanas", "centroscomercialescercanos", "usosmultiples".

Para el siguiente tuneo se decidió eliminar las features de importancia 4 y se agregó la posibilidad de agregar o no las features de importancia 3. Ademas se agregaron opciones en el encoding, estas eran inicialmente: TargetEncoder, One hot encoder, Count Encoding, CatBoost Encoding. Mas tarde se descartó One Hot Encoding porque el tuneo fue demasiado pesado para mi computadora.

Los resultados fueron:

El encoder con mejores resultados fue CountEncoding, y con respecto a las features, dejar las de importancia 3 dio mejores resultados.

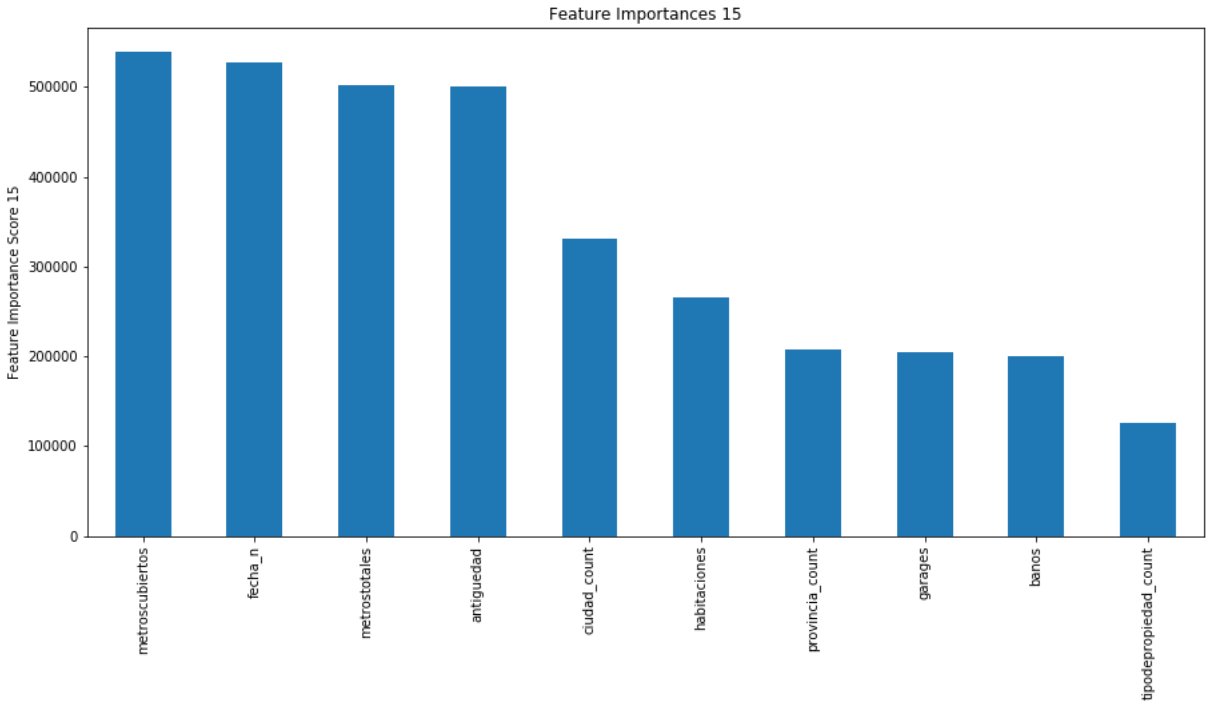

Figure 7: Gráfico de feature importance

Los resultados de este segundo tuneo fueron mejores que los resultados del primero, pero luego haciendo un submit llegamos a la conclusión de que estabamos overfitteando.

Para el siguiente tuneo ya se tenía a disposición muchas features nuevas interesantes ("dima_train_with_features_6.csv"). La estrategia fue simple, se realizó un tuneo simple con todas ellas y se graficó el feature importance de ese modelo entrenado (La imagen es muy grande para este formato, además lo importante fueron las conclusiones, y no la imagen en sí); De aquí se dedujeron las 39 features mas importantes. Luego se realizó otro tuneo.

Para este tuneo se decidió achicar más el rango del learning rate y se aumento el de los estimadores. Este tuneo resultó en el predictor que mejores resultados dió hasta el momento en que se escribe este informe (Mie. 4 de Dic. de 2019, 17:50hs)

7 Conclusiones:

RandomForestRegressor resulto un buen modelo inicial, puesto que no requirió de poco tuneo inicial para obtener buenos resultados. Hasta muy ultimo momento se mantuvo con mejores resultados que XGBRegressor. Cuando este ultimo lo supero finalmente en score, se descontinuo el tunneo del RandomForestRegressor para enfocarnos en xgboost.

XGBRegressor necesita de un mayor tiempo de tuneo, puesto que tiende a overfitear, con los hiper-parámetros incorrectos. Además, tener presente en que afecta cada hyper-parametro de este modelo, en la forma de entrenar, ayuda a perfilar los tuneos en direcciones, que por lo menos tengan sentido.

Por último, a pesar de que LightGBM tuvo peores resultados que XGBoost, logró una mejor predicción que Random Forest, y se destacó por poseer un tiempo de tuneo bastante menor a ambos. Además, cabe mencionar que no se tuvieron problemas de overfitting.

7.1 A continuar

Variar el procesamiento, utilizando otras estadísticas o valores, intuitivamente con sentido, para llenar los NaN, podría generar resultados mejores, posteriormente.

De las características básicas, los metros totales y cubiertos, son los que mas peso tuvieron en los modelos. Se intento crear varios features a partir de los mismos, pero aun así mantuvo su peso en los modelos. Creemos que estos features se pueden explotar aun mas.

No se obtuvieron features muy pesados con las columnas titulo, descripcion, y direccion, pero se cree que se pueden explotar más. Se podría probar con otros algoritmos de clasificación, y dado que se tratan de strings, desarrollar una red neuronal podría dar mejores resultados. Por otro lado, tampoco se analizaron variantes de las features actuales. Se tomaron top K's con K igual a 50 y 10000 de forma arbitraria. De la misma, manera el largo de los prefijos y postifjos usados fue arbitrario. Una corrección y posible mejora, seria tunear, probando con otros valores para estos features, además de realizar un análisis previo de estas caracteristicas.

Las columnas de lat y lng podrían explotarse mas, calculando distancias a otras ciudades o provincia, como por ejemplo, Distrito Federal donde los precios suelen aumentar. También podría probarse otro tipo de distancia además de la Euclidiana.

Agregar features basados promedios de los precios, potencio las predicciones. Se podría probar mas estadísticas relacionadas, como moda, desvió estándar, máximo, mínimo por tipo de propiedad, ciudad, y provincia.

Se generaron muchas features a partir la distancia entre distintos tipo de propiedad, ciudades y/o provincias, pero la distancia elegida fue arbitraria (Euclidiana). Analizar y probar otro tipos de distancias (angular, alguna variante de Minkowski, etc), podría mejorar los resultados de los features en la predicción. De la misma manera, se podría tunear el valor K para la cantidad de propiedades cercanas, según el tipo de propiedad, en lugar de tomar el mismo valor para todas ellas.

Muchos de los features de distancia no llegaron a tunnearse en mucha profundidad, obteniendo resultados, pero no tan buenos como los demas features usados. Por ejemplo, distancia_promedio_<tipodepropiedad>, distancia_Distrito_Federal, etc. Con más tiempo de tuneo en estos features, se podría realizar una mejor selección de ellos, así como, de hyper-parámetros, que podría mejorar la predicción actual.