Trabajo Practico 2
Predicción de precios

Manuel del Carril, *Padrón 100772*
manueldelcarril@gmail.com

Lucas Verón, *Padrón 89341*
lucasveron86@gmail.com

Federico Zugna, *Padrón 95758*
mail@mail.com

GitHub: https://github.com/Alaneta/Datacones

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

Índice

1. Introducción

El presente trabajo tiene como objetivo la implementación de un modelo predictivo capaz de predecir precios de casas en base a sus caracteristicas. A continuación se describen todos los detalles de dicho modelo, además de otras opciones contempladas que por un motivo u otro no formaron parte de la resolución final.

2. Modelo

El modelo implementado es un ensamble homogéneo, es decir, como unidad predictiva se implementa el mismo algoritmo pero se varía su entrenamiento para evitar que realicen todos la misma predicción. Como unidad, se utiliza un XGBoost y para variar su entrenamiento, se toman subconjuntos del set de entrenamiento de igual con tamaño con repetición.

Luego, estos "pequeños" XGBoosts realizan sus predicciones con las cuales se entrena el algoritmo ensamblador es el encargado de realizar la predicción final.

La cantidad de sub algoritmos fue variando. En un principio se utilizaban 200 pero debido a lo lentos que eran los ensambladores para entrenar dificultando la búsqueda de hiper parámetros, se redujo su cantidad a 100, además de que el hecho de que "n" sea mayor, no se traducía en una mejor puntuación.

Anteriormente, se había probado utilizar un modelo de Random Forest (cuando todavía no había un ensamble implementado) pero los resultados eran muy malos y creemos que se debe principalmente a que a la hora de llenar los valores faltantes, se estaba induciendo error. Se intentó llenar dichos valores de la forma más coherente posible, como utilizar el promedio según el tipo de propiedad y demás, pero aún así no fue suficiente. Si se desea observar dicho modelo, en el GitHub se debe ir al commit "41660af" del dia 14 de Noviembre del 2019, puesto que después fue eliminado del repositorio.

Antes de plantear la idea del ensamble y en vistas del fracaso de Random Forests, todo lo que se tenía en dicho modelo de Random Forests, se pasó a un XGBoost. Este modelo también presentaba malos resultados. Probablemente debido a la presencia de malos features y el relleno de NULLs (aunque después se probó eliminando los algoritmos encargados de "fillear" los datos faltantes). Se puede encontrar en el commit "f12c765" del día 26 de Noviembre del 2019, luego fue eliminado. Ante la frustración del fracaso, se decidió arrancar con un nuevo modelo de cero, solamente con las columnas numéricas del dataset como features.

Este nuevo modelo con pocos features era un XGBoost, el cual se convertiría en la unidad de ensamble presentada anteriormente.

No se contempló la idea de el uso de redes nueronales debido a que se trataba de un problema de datos de tipo estructurado, donde a priori, los modelos basados en árboles dan mejores resultados además de que los modelos basados en árboles son mucho más sencillos de utilizar.

2.1. Unidad de modelo

Se eligió como unidad del ensamble a XGBoost principalmente por la presencia de NULLs dentro del set de datos, los cuales si bien solo conformaban el 10 % de los mismos, también los había en el set a predecir para Kaggle, haciendo que sea imposible ignorarlos.

Además había columnas que eran fundamentales para el cálculo predictivo como el "id_ zona" pero era imposible llenarlos, puesto que uno no puede inventar una ubicación para la propiedad o su antigüedad por ejemplo.

Teniendo en cuenta esto y que además no había otro algoritmo que sea capaz de soportar la presencia de NULLs, se decidió por XGBoost.

Este modelo fue perfeccionándose gracias a la búsqueda de hiperparametros tanto por grid-search como random-search. Además se fueron agregando features a los datos con los cuales se les logró bajar el el error (ver más adelante en la sección "Features").

Los resultados de la unidad de modelo fueron los siguientes:

Cantidad de estimadores	Iteración de búsqueda de hiperparametros	score train	score test	score Kaggle
200	0	*No hay registros*	*No hay registros*	730017.948
200	0	*No hay registros*	*No hay registros*	727375.630
200	0	*No hay registros*	*No hay registros*	1238008.9
200	1	*No hay registros*	*No hay registros*	658460.118
200	1	*No hay registros*	*No hay registros*	639646.998
200	1	*No hay registros*	*No hay registros*	621038.668
400	1	749.37	768.15	592821.13
1600	2	690.15	744.70	-
2400	2	665.67	739.48	-

Como comentario, es importante recordar que entre distintas corridas también cambiaban los features agregados. En la primer fila, se tenía XGBoost con los hiperparámetros por "default" y con features que eran las columnas del DataFrame original que se podían importar de forma directa (metroscubiertos, metrostotales, id zona, lat, lng, etc). Para la siguiente fila, se habían agregado los tipos de propiedad codificados mediante One-Hot-Encoding (ver más adelante en la sección "Features"). Esto se tradujo en una mejora del modelo predictivo.

Lo sucedido en la tercer fila, fue la adición de varios features que no contribuían al modelo y al parecer lo "overfitteaban" dado que la puntuación local daba más baja. A partir de ahí se tomaron mayores precauciones con respecto a cómo implementábamos los features.

En las siguientes filas, ya se fueron adicionando distintos features que mejoraron bastante la predicción del mismo. Entre ellos, la codificación de provincias y ciudades, la presencia de palabras claves tanto en título como descripción y algunos más, además de aumentarse la cantidad de estimadores internos del XGBoost.

Con respecto a la búsqueda de hiperparámetros, en un principio se realizó un Grid-Search, el cual dio muy buenos resultados. Posteriormente, debido al incremento del tamaño del algoritmo, se realizó un RandomSearch pero los valores obtenidos fueron casi que los obtenidos hace tiempo por medio de Grid-Search. Por ende, no se prosiguió con su búsqueda.

2.2. Ensamblado

La elección del ensamble fue algo más complicada. Se fueron probando varias opciones de forma casi paralela hasta dar con el resultado final. A continuación se presenta una breve descripción de las mismas y de los resultados obtenidos:

2.2.1. Random Forests

Como primera opción, se encaró un Random Forests debido a su sencillez a la hora de utilizarse. Sufrió varias modificaciones en sus hiperparámetros. La cantidad de estimadores fue aumentando con el tiempo, para obtener mejores resultados. En un principio se trabajaba con 200, luego 400 hasta finalmente quedar en 1600.

Para búsqueda de hiperparámetros se intentó la técnica de Grid-Search Cross Validation pero debido a los tiempos de entrenamiento, fue imposible. Por esta razón, se optó por Random-Search Cross Validation, donde se probaban 10 combinaciones aleatorias de los hiperparámetros propuestos.

Se realizaron sucesivos Random-Search para obtener los mejores valores, prestando especial atención a aquellos que quedaban en los extremos de los conjuntos propuestos. En un primer momento, este ensamble "overfitteaba" bastante, y gracias al ajuste de hiperparámetros se logró reducir esta condición. Aún así, sigue teniendo

una diferencia entre el score de train y test.

Este ensamble siempre fue el que mejor resultados obtuvo, imponiéndose así como el ensamblador definitivo.

Los resultados obtenidos fueron los siguientes:

Cantidad de sub-XGB	Cantidad de estimadores	Iteración de búsqueda de hiperparametros	score train	score test	score Kaggle
10	30	0	*No hay registros*	*No hay registros*	633174.26
200	30	0	*No hay registros*	*No hay registros*	617294.06
200	400	0	*No hay registros*	767	591626.12
100	200	0	530	794	-
100	200	1	575.42	792.19	-
100	200	2	576.71	792.19	-
100	400	3	551.41	765.86	589073.79
100	1600	3	494.27	729.56	537860.78
100	1600	3	*No hay registros*	723	529480.13

Nota: Como a medida que se tuneaba el ensamblador también se mejoraban y agregaban más features al modelo individual, el descenso de valores no se debe enteramente al tuneo propio del algoritmo en cuestión. En especial en las últimas filas, donde el modelo unidad sufrió grandes cambios debido al aumento en su tamaño y cantidad de features. Por ejemplo, la búsqueda de hiperparámetros número 3 no fue la causante del descenso en la puntuación sino más bien una mejora en el modelo individual y debido al aumento de estimadores del Random Forests.

2.2.2. XGBoost

Se contempló la idea también de que otro XGBoost decidiera sobre las predicciones del resto. Al implementarse, se obtuvieron resultados comparables a los de Random Forests pero siempre por detrás.

Se hizo búsqueda de hiperparámetros por medio de Random-Search CV. Esto ayudó mucho al modelo puesto que en un principio "overfitteaba" muchísimo. Lo cual en cierta medida daba esperanzas, porque significaba que, en caso de reducirse, el modelo iba a predecir muy bien. Lo interesante es que si bien esto se logró (llegando a tener una relación de score entre train y test incluso mejor que Random Forest), no consiguió superarlo. Aún así siempre se mantuvo actualizado por si acaso obtuviese una mejor puntuación, puesto que no estaba muy lejos.

Los resultados obtenidos fueron los siguientes:

Cantidad de sub-XGB	Cantidad de estimadores	Iteración de búsqueda de hiperparametros	score train	score test	score Kaggle
100	200	0	269.48	797.48	-
100	200	1	672.42	788.22	-
100	200	2	714.68	775.23	604757.85
100	1600	2	419	742	-

En este caso, vale la misma aclaración que en el caso de Random Forest.

2.2.3. Bagging

Se hizo uso del BaggingRegressor como técnica de ensamblaje. Si bien prometía, nunca estuvo cerca de los líderes (Random Forests y XGBoost). Con la primera corrida se pensó que era un problema de hiperparámetros. Pero aún después de buscarse los valores deseados mediante, una vez más, Random-Search CV, el resultado casi que ni mejoró.

Los resultados obtenidos fueron los siguientes:

Cantidad de sub-XGB	Cantidad de estimadores	Iteración de búsqueda de hiperparametros	score train	score test	score Kaggle
100	100	0	*No hay registros*	*No hay registros*	-
100	100	1	811.978	816.73	-
100	100	1	706.00	740.53	-

Como aclaración, este modelo casi que ni mejoraba la puntuación respecto del modelo individual. Quizás la bajaba en 2 o 3 puntos. Por esta razón, casi que quedó descartado desde el principio, aunque por las dudas cada tanto se lo corría para ver si en una de esas daba resultados más competitivos.

2.2.4. AdaBoost

Se intentó implementar un AdaBoost después del fracaso de BaggingRegressor pero curiosamente y a pesar de provenir de la misma librería que Bagging, no soportaba la presencia de NULLs en el dataset cuando se le pasaba como estimador un XGBoost. Se invirtió tiempo buscando soluciones en internet pero no se encontraron respuestas claras. Por esta razón, esta implementación nunca vio la luz.

2.2.5. Redes Neuronales

Se contempló la idea de diseñar una red neuronal como ensamble, porque el dataFrame que ingresa al ensamblado son solamente las predicciones de precios lo cual significa que todas las columnas tienen valores cuya unidad es la misma (haciendo muy fácil el escalado de los datos). Esto es un contexto bastante favorable para dicho modelo. No se llegó a implementarse por cuestiones de tiempo y porque los algoritmos que se tenían hasta el momento prometían bastante y eran mucho más sencillos de utilizar, pero hubiera sido interesante haber hecho aunque sea una prueba de concepto al respecto.

3. Features

A continuación se enumeran todos los features calculados para el modelo unidad:

- **antiguedad**: Es la antigüedad de la propiedad en años. Proviene del dataset original. Mejoró el score.
- **habitaciones**: Es la cantidad de habitaciones de la propiedad. Proviene del dataset original. Mejoró el score.
- **garages**: Es la cantida de garages de la propiedad. Proviene del dataset original. Mejoró el score.
- habitaciones: Es la cantidad de habitaciones de la propiedad en años. Proviene del dataset original. Mejoró el score.
- **banos**: Es la cantidad de baños de la propiedad. Proviene del dataset original. Mejoró el score.
- **metroscubiertos**: Es la cantida de metros cubiertos de la propiedad. Proviene del dataset original. Mejoró el score.

- **metrostotales**: Es la cantidad de metros totales de la propiedad. Proviene del dataset original. Mejoró el score.
- **idzona**: Es el id único de la zona en la que se encuentra la propiedad. Proviene del dataset original. Mejoró el score.
- **lat**: Es la latitud de la ubicación de la propiedad. Proviene del dataset original. Mejoró el score.
- **lng**: Es la longitud de la ubicación de la propiedad. Proviene del dataset original. Mejoró el score.
- **gimnasio**: Si el edificio de la propiedad tiene gimnasio. Proviene del dataset original. Mejoró el score.
- **piscina**: Si el edificio de la propiedad tiene piscina. Proviene del dataset original. Mejoró el score.
- **escuelascercanas**: Si la propiedad tiene escuelas cerca. Proviene del dataset original. Mejoró el score.
- **centroscomercialescercanos**: Si la propiedad tiene centros comierciales cerca. Proviene del dataset original. Mejoró el score.
- **tipodepropiedad_i**: Todas estas columnas son la codificación de tipo One Hot Encoding de la variable "tipodepropiedad" que corresponde, como su nombre indica, al tipo de propiedad. Mejoró mucho el score.
- **provincia_i**: Todas estas columnas son la codificación de tipo One Hot Encoding de la variable "provincia" (es la provincia en la que está ubicada la propiedad). Mejoró mucho el score.
- **ciudad_i**: Todas estas columnas son la codificación de tipo Binary de la variable "ciudad" (ciudad en la que se encuentra la propiedad). Se utilizó esta codificación debido a que la cantidad de ciudades distintas eran muchas para One Hot Encoding. Este feature mejoró mucho el score.
- **anio_publ**: Corresponde al año en que se realizó la publicación de la propiedad. Se obtuvo a partir de la variable "fecha" del dataset. Mejoró de forma moderada el score.
- **mes_publ**: Corresponde al mes en que se realizó la publicación de la propiedad. Se obtuvo a partir de la variable "fecha" del dataset. Mejoró un poco el score.
- **dia_publ**: Corresponde al día en que se realizó la publicación de la propiedad. Se obtuvo a partir de la variable "fecha" del dataset. Empeoraba el score (muy poco igual) así que se descartó.

La importancia de features según XGB es la siguiente:

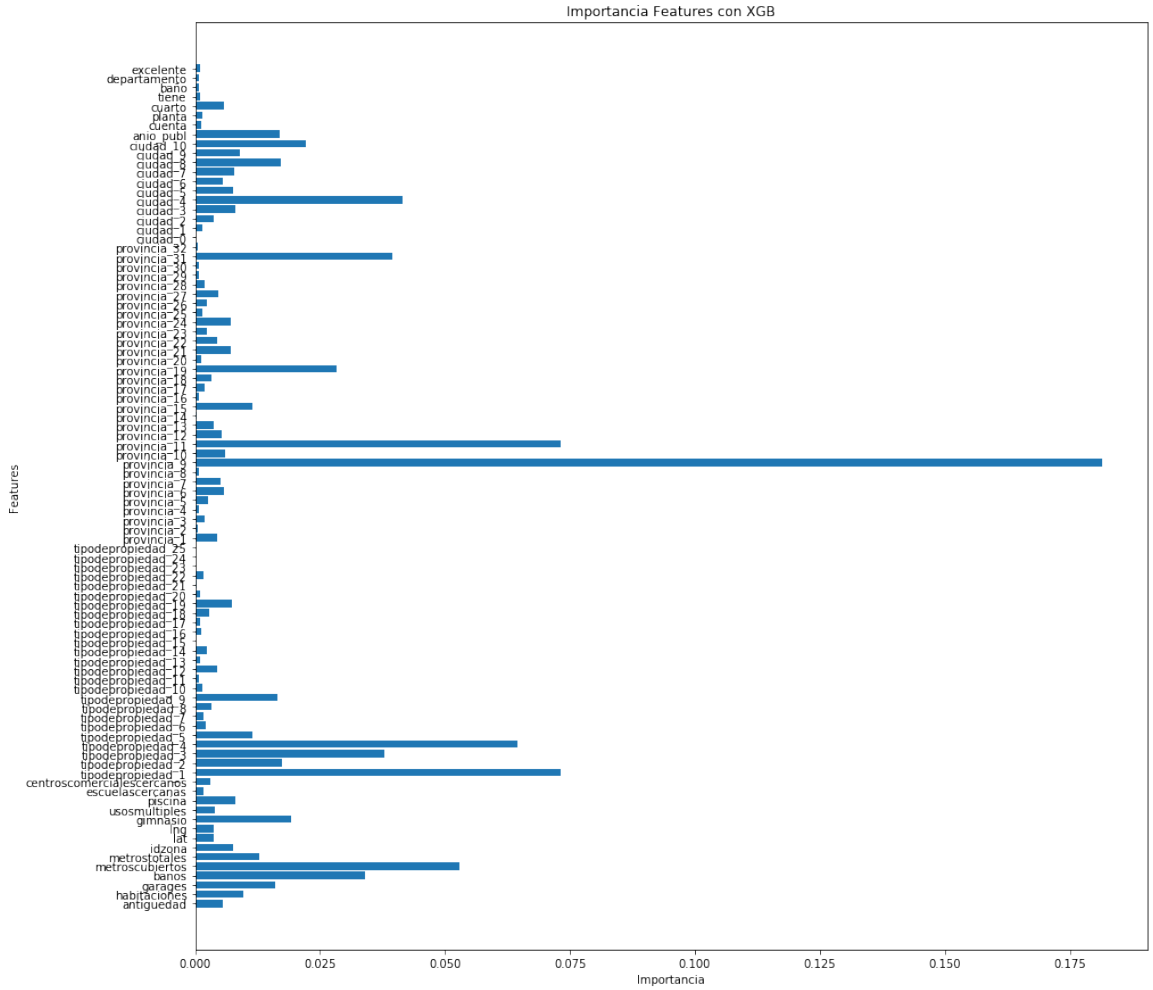

Figura 1: Importancia de features de XGBoost.

Si bien el análisis de importancia de features sólo es relevante para el cálculo interno de XGBoost, es interesante ver que prima a la hora de decidir. Se puede observar que tanto en las provincias como en los tipos de propiedades, hay unas pocas a las que se le da mucha importancia respecto a todo el resto. Esto probablemente se deba a que para esos valores, debe haber una alta correlación con el precio a diferencia de las otras. Es interesante ver que en el caso de las ciudades, esta diferencia no es tan marcada, quizás se deba por el tipo de codificación también puesto que es distinto.

Con los features que vienen por default, no se ve ninguna sorpresa, exceptuando por el idzona lo cual uno esperaría una importancia mayot que otros features que uno suele considerar menos relevantes (como gimnasio, piscina, antiguedad, etc). Incluso, cuando se tenían menos features, este id zona resaltaba por sobre los otros.

Si entrenamos un Random Forest con los datos (ignorando los NULLs por supuesto, por lo cual pasamos a tener muchos menos registros así que ciudado...), el resultado obtenido es el siguiente:

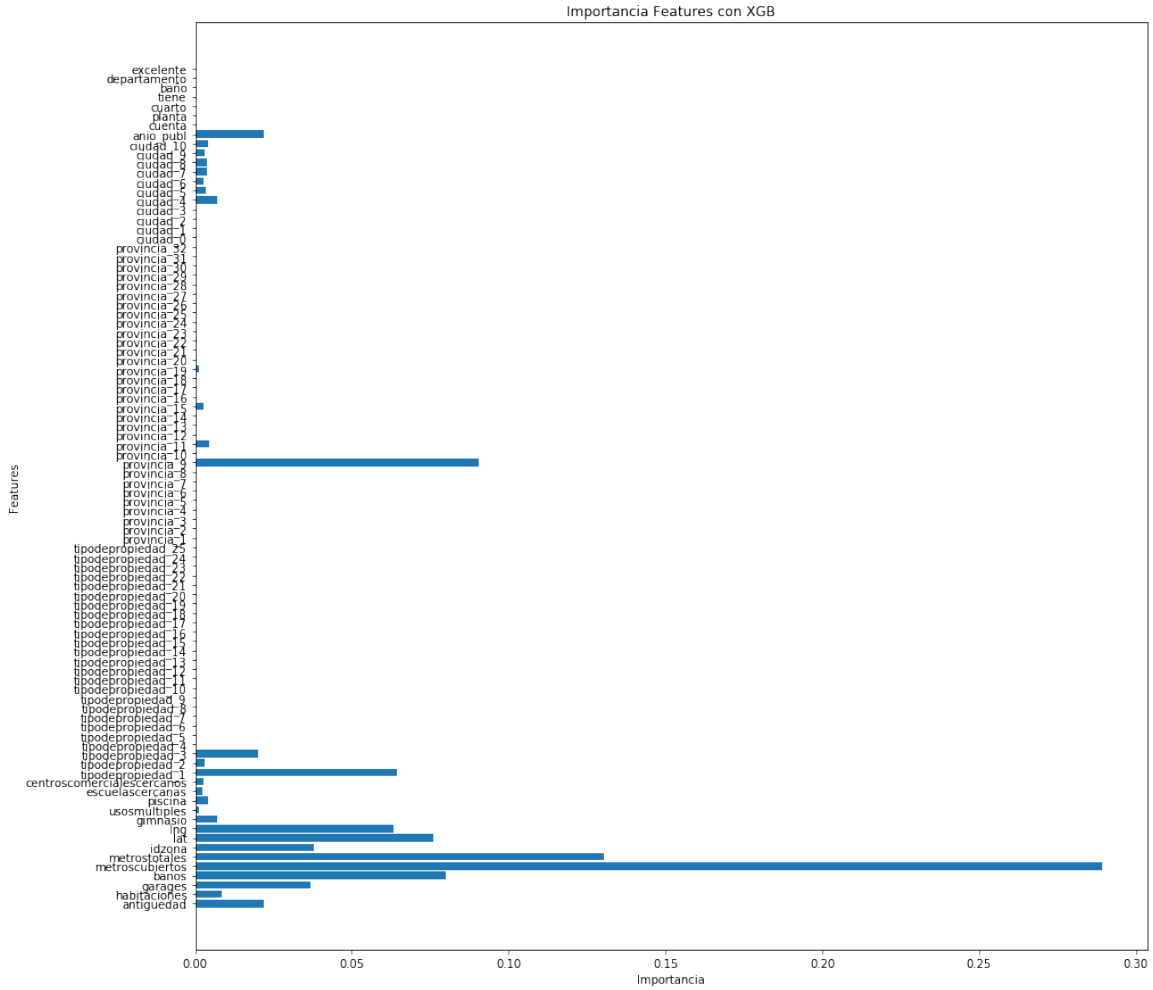

Figura 2: Importancia de features de Random Forest.

El gráfico es distinto al anterior. En el caso de las variables codificadas, aquellas que aparecían como relevantes en el XGBoost se mantienen como tales aunque su importancia no es tan alta con respecto a los otros features. El resto de los valores de las variables codificadas casi que no presentan relevancia en este caso. Las ciudades al igual que antes, presentan barras de igual altura aproximadamente.

Se puede observar como el feature de metros cubiertos es el más relevante (y con diferencia) respecto a los demás, lo cual es esperable y tiene sentido. Luego se tienen los metros totales, la cantidad de baños y la posición dada por la latitud y longitud.

La importancia de Random Forest es algo más esperable y con bastante más sentido a comparación con la de XGBoost. Aún así hay que recordar que para realizarla, se eliminaron bastantes registros debido a la presencia de NULLs.

3.1. Minado del campo "descripción"

Dado que no se obtuvieron buenos resultados con varios de los features analizados en la primer parte del trabajo(Tp1 - exploración de datos) se decidió armar features a partir de las palabras más frecuentes dentro del campo 'descripcion' para cada uno de las propiedades registradas. Cómo primer medida se fueron separando las palabras de cada descripción(utilizando espacios como split word). Luego se agruparon las palabra y se ranqueó el conjunto quedandonos con la palabra que más repeticiones tenia para la descripcion. Se forma una tupla de la forma: (word,count_word), donde "word" conteniene la palabra y "word_count" contiene la cantidad de repeticiones de la palabra. A continuación separó la tupla para formar 2 columnas. Una con la palabra y otra con la cantidad de esa palabra. Por último se generó 1 columna para las n palabras mejor rankeadas entre el total de palabras. Cada columna indica si la misma se encuentra dentra de la "descripcion" o no(1—0).

Cómo primera aproximación se utilizó un n=3, dando buenos resultados. A continuación se eligió subir el n

a 5. Se obtuvo otra mejora. Entonces luego se incrementó el n a 10. Ya en esta última iteración los resultados no fueron alentadores. No se obtuvo una mejora del modelo y hasta se obtuvo una desmejora. Se las mejores 5 palabras concentran la información más rica para el aprendizaje del modelo.

3.2. Minado del campo "título"

Teniendo en cuenta los buenos resultados obtenidos con el campo "descripcion" se pensó utilizar el mismo proceso pero aplicado al campo 'titulo', el cuál es contiene una cantidad de palabras importante, luego del campo "descripcion". El proceso fue el mismo. El n "óptimo" fue: n = 5.

3.3. Features desarrollados no implementados

A continuación se listan un conjunto de features que fueron desarrollados, pero forman parte del algoritmo final por cuestiones de puntaje. Varios de los features no funcionaron de la manera esperada. Los features son los siguientes:

Feature correlacion precio vs provincia: Para este feature se buscó la correlación entre precio de las propiedades según la provincia. Se generó un columna con la correlación correspondiente.

Feature correlacion garage vs precio por ciudad De manera similar a como se buscó la correlación por provincia se encontró la correlación por ciudad. Para cada registro se seteó la correlación.

Feature total de antiguedad por anio Para este feature se tuvo en cuenta la antiguedad total de la publicación. Un problema a resolver es que algunas propiedades no tenian seteada la antiguedad. En estos casos se dejo en Nan. Para el resto de los casos se incorporó una columna con la antiguedad por año.

Feature antiguedad por provincia Antiguedad por provincia contiene la antiguedad promedio por provincia. No funcionó como se esperaba.

Feature Usos multiples piscina gimnasio según tipo de casa Este feature seteaba un factor (entre 1 o 0.5) que incorporaba información sobre si la propiedad tenia o no usos multiples y pisicina según la propiedad. Se abandonó la idea al ver que los resultados no eran alentadores.

Feature Relación existe entre el precio, los metros cuadrados y la cantidad de habitaciones de los departamentos Para este feature se generó un factor(1 o 0.5) que setteaba un valor según el precio de la propiedad y la cantidad de habitaciones de los departamentos a partir de un límite(5) para determinar si correspondia un factor de 1 o 0.5. La existencia del precio en este feature parece que no permitió que el algoritmo mejore ya que el mismo overfiteaba.

3.4. Cosas que se podrían probar

Dado que uno de los features implementados que dió buenos resultados fue el analisis y parseo de las palabras dentro de las descripciones de la publicación; una de las ideas que se abarajo fue intentar hacer un analisis de frases en vez de simplemente palabras. Por ejemplo buscar el ranking de palabras más frecuentes. En vez de utilizar sólo la palabra 'bonito' utilizar la frase 'ambiente bonito'. Otro de los features pensados fué rankear o clasificar las palabras según la frecuencia de aparición en diferentes localidades/provincias,etc.

4. Modo de ejecución

Para entrenar el algoritmo, se deben ejecutar los Notebooks que ese encuentran en el directorio /Datacones/tp2/tp2_final de acuerdo al orden impuesto por los números en sus nombres:

- El primero lo que hace es generar el DataFrame con todos los features calculados a partir de train.csv y test.csv.
- El segundo, entrena los "n" XGBoost independientes y los guarda en disco. Además, calcula el dataFrame necesario con todas las predicciones de "train.csv" para que el ensamble de Random Forest pueda entrenar.

- El tercero entrena el modelo de Random Forest y lo guarda en disco.
- El cuarto calcula la predicción a subir a Kaggle levantando el Random Forest guardado y todos los XGBoosts.

5. Conclusiones

Se tiene un algoritmo capaz de predecir con cierta precisión el precio de distintas casas acorde a sus características presentadas en sus respectivas publicaciones. Resumiendo puntos podemos concluir en que:

- Los NULLs que presenta el dataset suponen un problema para el entrenamiento y predicción de valores. Para el modelo que se desarrolle es necesario tratarlos o elegir un algoritmo que los soporte. Nosotros nos decantamos por la segunda opción.
- Si bien el ensamble fue crucial para reducir el error de la predicción, era muy importante también mejorar el modelo a nivel individual mediante "tunning" o agregando más features.
- A nivel features, no se espera que se descubra ese feature "mágico" que mejora la predicción debido a su alta correlación con el precio. Todos ellos son bastante lógicos y simples además de ser esperables a la hora de tasar una propiedad. Esto se ve reflejado en la importancia de features donde no hay nada que le gane a los metros cubiertos.
- Si bien se le sacó partido a la columna de descripción, un análisis más exhaustivo sobre la misma, podría traer mejores resultados puesto que al parecer hay una correlación bastante fuerte entre ella y el precio.