FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Organización de datos - 75.06/95.58

Trabajo práctico 2
Machine Learning

2º cuatrimestre 2019
Grupo 28 - Datoy Story

Apellido, Nombre	Nº Padrón
Mac Gaul, Pedro	101503
Macía, Tomás	99248
Perez, Cristian	95536
Riborati, Franco	99411

Link al repositorio: github.com/tomasmacia/orgaDatos

1. Introducción

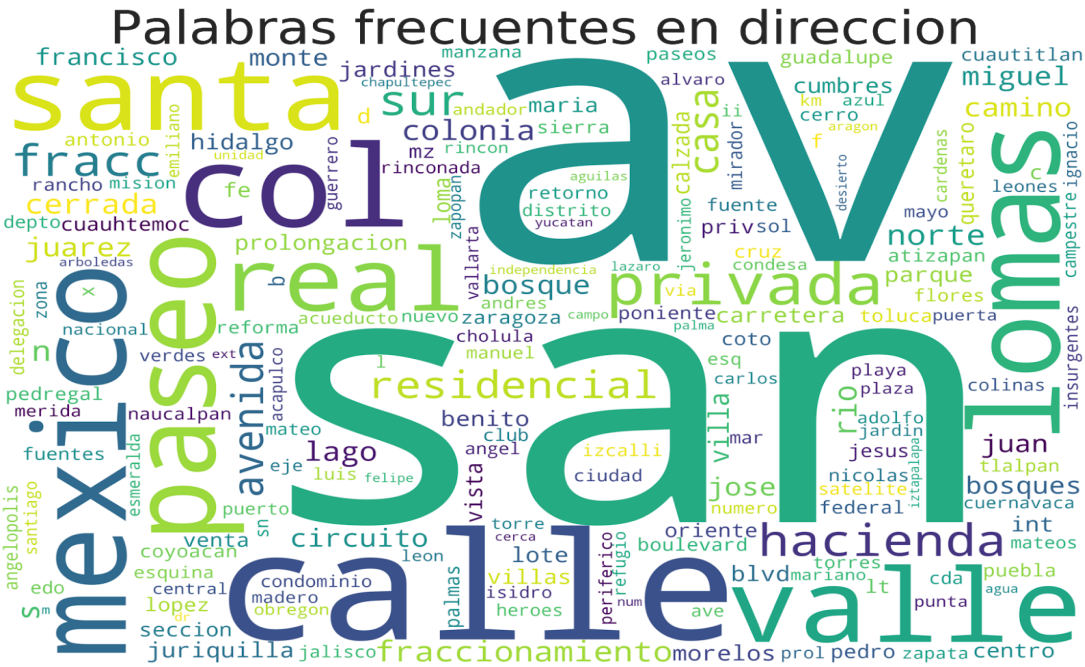
En este informe se mostrará el trabajo realizado durante el segundo cuatrimestre del 2019 de la materia Organización de Datos en la Universidad de Buenos Aires, el cual su objetivo fue realizar la mejor predicción, mediante distintos algoritmos de Machine Learning, el precio de las propiedades de la página ZonaProp en México.

El informe se dividirá en dos secciones, en la primer parte se desarrollará la parte del pre-procesamiento y limpieza de los datos, para luego realizar features sobre ellos y poder dar información de una manera única. En la segunda parte describiremos qué modelos utilizamos para producir las predicciones, cómo fueron mejorados y el resultados de los mismos.

2. Procesamiento de los datos

2.1 Limpieza de datos

En un principio se realizó la limpieza de las descripciones, títulos y direcciones de las publicaciones para poder obtener información relevante sobre el data set. Se limpiaron los formatos html, signos de puntuación, signos de acentuación, números y palabras cortas o de ningún significado importante para los features del texto (*stop words*). Una vez realizado todo este proceso, se procedió a generar features a partir de los datos con texto donde se encuentra determinada palabra, este proceso se realizó a mano, sin saber si aquella palabra iba a ser significante o tendria algun sentido. Algunas se agregaron a partir de la frecuencia de la palabra en las columnas del set de entrenamiento. Otras fueron agregadas porque nos parecieron palabras relevantes a la hora de buscar propiedades. En los siguientes gráficos se muestran las palabras más frecuentes según cada atributo con texto.

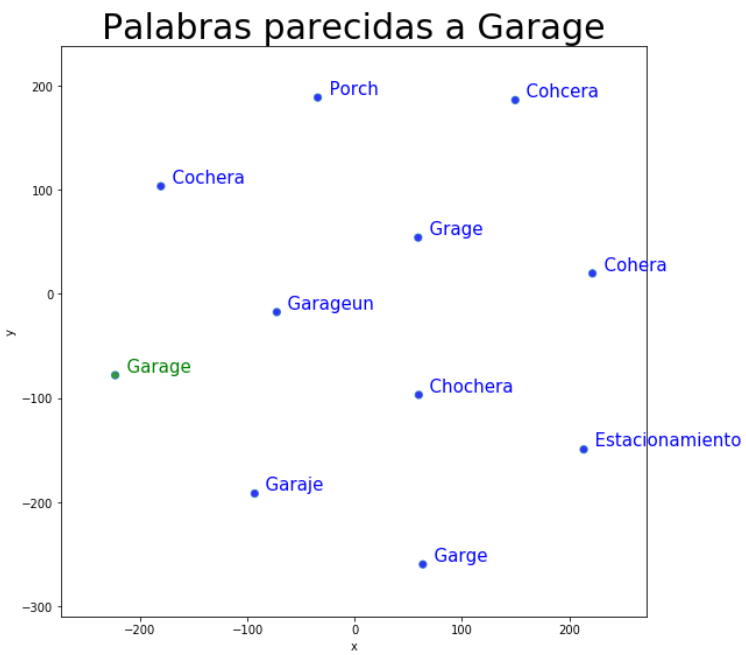


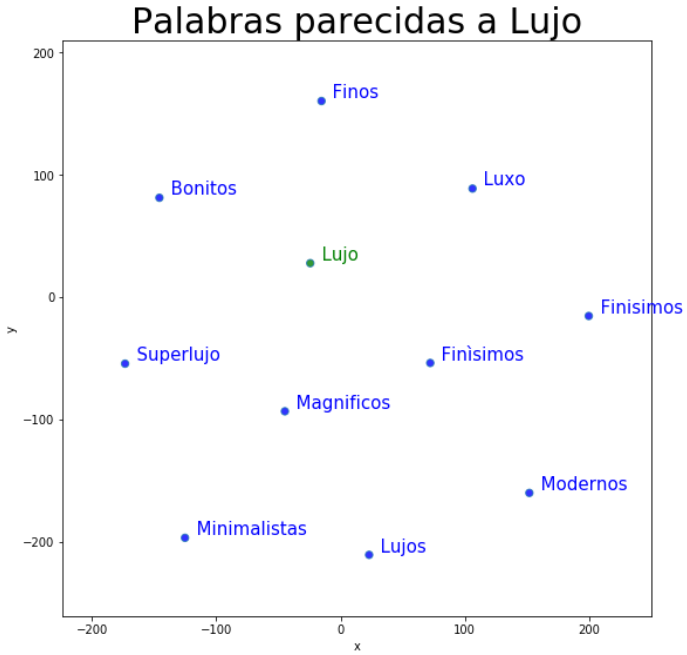
Continuamos haciendo este proceso eliminando los features de latitud y longitud, ya que, tiene una gran cantidad de nulos y no son representativos para el dataframe. Incluso, como vimos en el TP1, muchos de estos puntos geográficos se encontraban fuera del territorio mexicano. Además se agregaron, a partir de la fecha, los features del día, mes y año de la publicación.

Una vez realizada la limpieza, se procedió a buscar métodos para, a partir de un texto, generar un embedding: esto es, generar un vector que represente el texto. Decidimos utilizar un método llamado Word2Vec, el cual consiste en una red neuronal con 1 hidden layer. El output de esta capa son los pesos de los vectores que nos interesa. Aplicamos esta red neuronal y obtuvimos un vector de 300 dimensiones, representativo del

vocabulario que manejamos en nuestro set de datos. Nuestro objetivo era aprender si había algún feature escondido. Utilizamos Word2vec y luego para cada palabra presente en el documento, calculamos el promedio de cada una mediante su embedding correspondiente. De esta forma, si un documento tiene palabras muy similares a otro, su promedio sería similar. Al aplicar esta teoría, pudimos traducir el texto de las descripciones a un vector.

A continuación vemos, para ciertas palabras, vocabulario similar basado en la probabilidad de ocurrencia:





Podemos observar que hay resultados que se parecen, y tienen un significado parecido. Nuestra red neuronal funcionaria bien para poder predecir cuales palabras son parecidas a pesar de no ser iguales. Esto es importante, ya que nos ahorró el proceso de stemming y lemming para reducir el conjunto de palabras cuya base es similar. Y viendo el caso de “Garage”, vemos que hay casos bastante complicados de captar mediante estos métodos, ya que algunos son errores ortográficos, y otros son palabras mal escritas.

Comparamos todas las palabras, y se obtuvo un dataset a partir de esto, pero al ser tantas palabras y tantos vectores, se procedió a realizar una reducción de dimensiones probando distintas maneras de reducir las dimensiones del data set. Se utilizaron los algoritmos de PCA, Hashing trick y TSNE para poder ver si de alguna manera los datos utilizados se relacionaban de alguna manera. Se aplicaron distintas cantidad de dimensiones para poder ver si era un problema la energía perdida por el dataset. En los gráficos anteriores podemos ver que tan cercanas están las palabras según su parecido. El gráfico se realizó entrenando la red, utilizando PCA y TSNE para poder graficar la proximidad de las palabras.

Lamentablemente para nuestro caso, este largo proceso no obtuvo grandes beneficios como nosotros esperábamos. De los 150 features que obtuvimos al aplicar *Average Word2Vec,* mediante los algoritmos de reducción de dimensiones logramos reducirlo a 25, pero sin el efecto deseado (encontrar features escondidos que nos resulten de interés).

Las palabras que contenían determinadas descripciones o títulos nos brindaban más información que lo obtenido por este proceso; por eso mismo, no se realizó ningún submit al respecto del mismo.

2.2 Completando los datos

Al obtener muchos datos nulos, decidimos completarlos para poder obtener más información de las demás categorías que tenia la publicación con el dato nulo, estos fueron completados de distintas maneras mediante el Simple Imputer de la librería de Sklearn. Los completamos con la mediana, con el promedio, y los rellenamos con ceros a los datos numéricos. En el caso de los metros cuadrados, para los datos faltantes utilizamos completando los metros cubiertos con los metros totales y los metros totales con los cubiertos, para no perder información sobre la propiedad y no completar con un dato que sea poco representativo. Para los demás datos obtuvimos que la mejor aproximación se pudo realizar completando los datos con ceros.

Para los datos de texto abierto simplemente completamos con un string vacío y los categóricos, no los completamos porque utilizamos los Encoder para que se encargen de los mismos y sea una categoría más.

2.3 Features

A la hora de probar los algoritmos nos dimos cuenta que lo más importante para poder predecir con aun mayor precisión los precios, son los features que da información acerca del fenómeno observado, en este caso, la publicaciones de las propiedades y es una característica crucial la cual brinda información. Si únicamente nos dedicamos a seleccionar modelos y tunnearlos, muy lejos no íbamos a llegar, por esa razón dedicamos una extensiva búsqueda a los mismos probando diferentes alternativas y distintas maneras de representar los datos.

Pudimos ver cómo al buscar en los datos de texto abierto, la cantidad de publicaciones donde se encontraba la palabra no tenía relación alguna a la hora de ver si ese feature era importante o no. Esto quiere decir que en las publicaciones donde aparece la palabra eran pocas pero inciden con mayor precisión en el valor de la propiedad, con lo cual la cantidad de apariciones de la palabra no representaba nada.

2.3.1 Datos de texto abierto

Como mencionamos anteriormente, en la sección de limpieza de datos, sobre estos campos hicimos varias pruebas. Las primeras pruebas consistieron en hacer un análisis de palabras, mediante wordclouds y su frecuencia. Esto nos permitió conocer un poco el vocabulario que se utiliza para distintas características que pueden llegar a ser de interés.

Títulos y descripciones: Buscamos palabras frecuentes y otras que fuesen relevantes para nosotros. Por ejemplo las palabras más relevantes en el modelo que rescatamos fueron, lujo o lujosa, country o barrio cerrado, si tenía seguridad, o si era privada, si acepta créditos (infonavit, un tipo de crédito), entre muchas otras. Dichas palabras tuvieron mucho éxito y las encontramos varias veces entre los plots de feature importance. Las demás palabras no tuvieron tanto éxito.

Direcciones: Análogamente lo anterior lo realizamos para las direcciones, buscando cuales estaban en avenidas, cuales sobre la playa o cuales tenían la dirección privada. Acá nuestra sospecha era que no iban a ser relevantes, ya que no describen características de la vivienda (específicamente el caso de “avenida”), y luego de aplicarlos, nos encontramos con que efectivamente no tenían importancia estos features.

Otras de las pruebas que hicimos fue realizar la red neuronal Word2Vec sobre la descripción de las propiedades. Nuestra principal idea era poder reducir los datos de texto a un vector o embedding que represente en forma numérica los datos presentes. Utilizamos este método ya que, mediante la red neuronal, nos provee un vector de pesos para cada palabra (estos datos son el output de la primer y única hidden layer de la red neuronal), el cual podemos utilizar para comparar vocabulario. Con dichos vectores de cada palabra calculamos el promedio con todas las palabras de cada documento (nuestro documentos sería cada descripción de las palabras) y obtuvimos un vector que representa a la descripción. En nuestro caso, este vector fue de 150 dimensiones, el cual intentamos reducir a 25 mediante PCA, Hashing Trick y TSNE.

Lamentablemente estos 25 features que obtuvimos mediante esta técnica no resultaron mejores que los features que obtuvimos inicialmente en la búsqueda manual, por lo que después de varios intentos fallidos, los tuvimos que descartar. Ante la falta de nuevas ideas para probar con este campo, mantuvimos los features iniciales que logramos manualmente.

2.3.2 Datos categóricos

Para los features categóricos se probaron distintos tipos de Encoders. Se utilizaron Sum Encoder, Backward Difference Encoder, Helmert Encoder, Hashing Encoder, Binary Encoder, One Hot Encoder, Target Encoder y Label Encoder. Prácticamente, agarramos la sección de Scikit, category encoder y utilizamos los que más nos parecieron relevantes y nos quedamos con los que mejor resultados nos brindaban. Realizaremos una subdivisión en No-Target Encodes y Target Encoder.

No-Target Encodes:

Realizamos dos separaciones en este caso, ya que, había categorías las cuales tenían demasiados valores (idzona y ciudades), generaban una gran cantidad de features al punto de matar el kernel, no podíamos manejar tal cantidad de dimensiones. Solucionamos esto probando dos tipos de encodes, hashing encode y binary encode, resultando más eficiente el Binary encode. Por otro lado, a los demás datos categóricos (tipo de propiedad y provincia) les realizamos los demás encodes y resultando el más eficiente el One Hot Encode, aunque no por mucho ya que, Sum Encoder, Backward Difference Encoder, Helmert Encoder, resultaban ser muy parecidos a este, realizando un encode parecido dando un poco más de información acerca de lo encodeado pero en fin, terminaron prediciendo peor.

Target Encode:

Utilizamos target encode para poder ver el valor promedio por distintas categorías, este encode se realizó en Ciudades, Idzona, Tipo de propiedad, Provincia y en datos no categóricos cómo, habitaciones y ambientes(la suma de habitaciones y baños). Con distintos targets, como por ejemplo, la antigüedad, el precio, metros cubiertos, metros totales, metros cuadrados (la suma de metros cubiertos y metros totales), baños, habitaciones y ambientes. También transformamos los datos para obtener una distribución más uniforme en los mismos y con ello poder tener una mejor predicción.

Se obtuvo increíbles resultados de estos encodes, lo cual nos permitió romper con la barrera de los 520K de error, llegando a obtener 470K a partir de estos features.

2.3.3 Datos numéricos

Como vimos en el trabajo práctico 1, los metros de superficie de la propiedad estaban muy relacionados con el precio por propiedad, como era de esperarse. Se tuvo en cuenta a la hora de obtener features a partir de

este dato, ya que de alguna manera estaba directamente relacionado con el precio de la propiedad. Análogamente se realizó un análisis similar para los garajes, baños y habitaciones que será explicado con mayor detalle más adelante.

Como ya pudimos decir anteriormente, aplicamos distintas transformaciones para mejorar la distribución de los datos continuos (metros cubiertos, metros totales, precio), y para los datos discretos con poca variación utilizamos one hot encoder o sum encoder para poder apreciar más de ellos. Por ejemplo a los meses y años se realizo one hot encoding y para los garajes, baños y habitaciones se realizó sum encoding. También generamos datos a partir de estos como por ejemplo la cantidad de baños por habitación, los metros cubiertos-totales siendo estos la suma de los metros cubiertos y los totales, metros cuadrados siendo los metros totales por los cubiertos, etc, de esta manera obtuvimos varios features interesantes.

Además del trabajo práctico 1, podemos ver cómo los baños y las habitaciones dependen del precio de la propiedad, por eso procedimos a seleccionar el dato de baños por habitaciones, ya que tenía algo de interés a la hora de decidir el precio. Cómo podemos ver en el gráfico 2.2.3 hay una relación de los baños y la cantidad de habitaciones con el precio de metro.

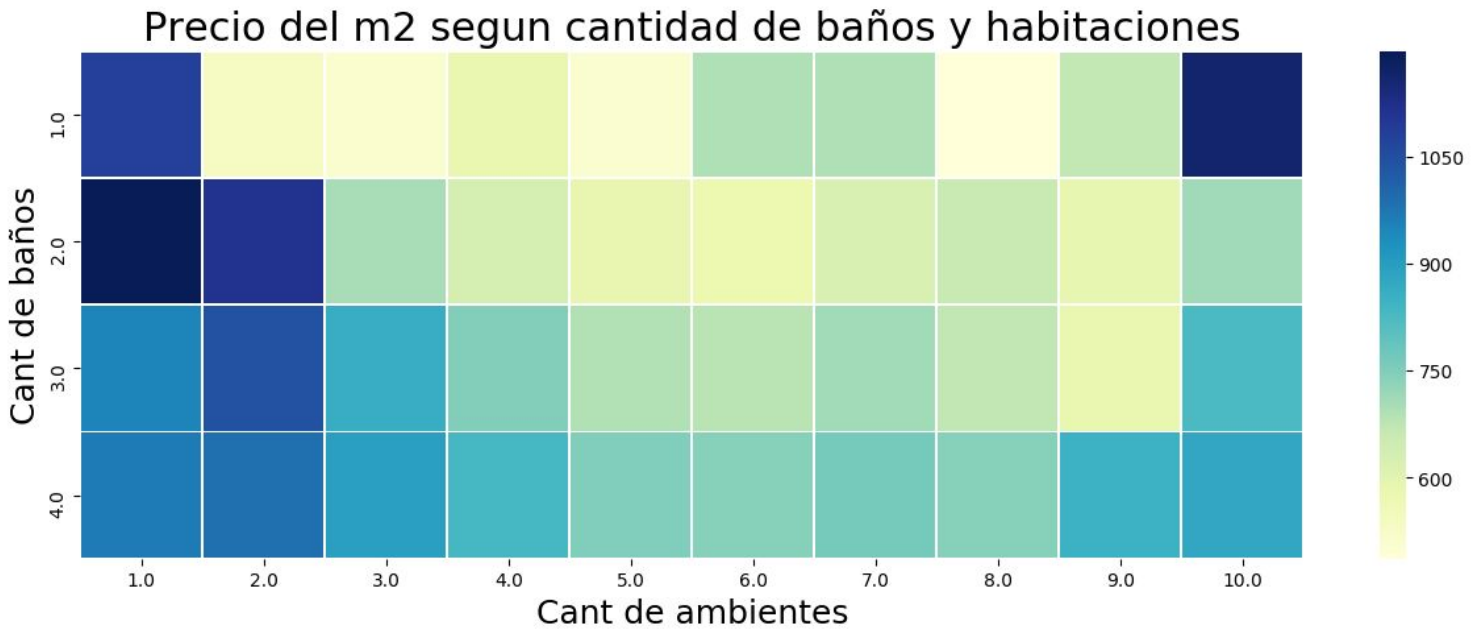


Gráfico 2.2.3

También podemos ver que en el gráfico de correlaciones del TP1 se puede ver a simple vista que a medida que aumentan los baños, garajes, aumenta el precio de la propiedad. Esto fue algo a tener en cuenta, se realizaron targets encodes en base a esto, buscando si se lograba ver algo a partir de ello. Utilizamos los siguiente features, metros totales sobre baños, Target encode siendo el target los baños, y mismo para las habitaciones.

3. Algoritmos utilizados

Utilizamos una gran cantidad de algoritmos, muchos de ellos fueron descartados o quedaron como borrador, ya que no fueron importantes los resultados obtenidos, consideramos que nos fueron importantes a los

que no brindan nueva información al respecto de nuestro modelo anterior o overfiteaban o no sabíamos cómo utilizarlos, cómo es el caso de ExtraTreesRegressor, NuSVC, SVC, AdaBoost, CatBoost, entre otros. De los cuales obtuvimos un resultado interesante de los siguientes modelos: KNN, RandomForest, XG Boost, Light GB y probamos los siguientes ensambles, Voting Regressor, Gradient Boost e Hist Gradient Boost. El mejor resultado fue obtenido por XG Boost, posicionándonos en la posición 3 de la competencia, con alrededor de 470K de puntaje.

Tuvimos muchas complicaciones a la hora de probar los algoritmos, por suerte, al empezar con tiempo pudimos solucionar estos errores, que fueron solucionados “a los golpes” esto nos demoró bastante el progreso de nuestros modelos. El problema fue que no estábamos simulando la situación real de nuestro problema, estábamos overfitteando realizando el split después de aplicar los features al set de datos lo cual no sería óptimo, ya que, uno no sabe los datos que va a probar, estábamos alejados de la situación real porque estábamos manejando los datos de test a nuestra manera para que queden bien en el modelo.

Una vez superado este error, tuvimos la suerte de que nuestras predicciones estaban muy acertadas a lo obtenido en Kaggle, estábamos a la par, lo cual es bueno porque sabíamos que nuestros modelos estaban prediciendo bastante bien en base a la cantidad de features obtenidos en el momento. Overfitteamos pocas veces en comparación a la cantidad de veces que estábamos a la par del valor de Kaggle.

Realizamos un archivo el cual tiene nuestras funciones más utilizadas (Featurizer.py) para tener al alcance y no repetir tanto código en los notebook y hacer más fácil la visualización de los resultados obtenidos. En este se encuentran los encodes, las maneras de predicción, el preprocesamiento para el test, entre otras cosas que nosotros encontrábamos útiles.

3.1 Regresores

3.1.1 KNN

Uno de los algoritmos utilizados fue K-Nearest-Neighbors. El algoritmo es bastante simple y consiste básicamente en, para cada uno de los puntos que queremos conocer el valor de la propiedad, buscar los k vecinos más cercanos, siendo k un hiper-parámetro que se pasa cuando se instancia el regressor. A partir de encontrar estos k vecinos más cercanos, evalúa cuántos vecinos pertenecen cerca a la publicación queremos predecir el valor, dando un valor cercano a estas. Este es uno de los algoritmos más simples de Machine Learning, pero se decidió de probarlo ya que suele arrojar buenos resultados sin tener mucha información. Fue nuestra primer vara a superar, nos quedamos sólo con categorías numéricas y encodear algunas de las categóricas, realizamos pocos features en este momento, ya que fue de los primeros algoritmos que utilizamos.

Se utilizó tres distancias distintas las cuales se obtuvieron resultados bastantes distintos. Utilizamos las distancias euclidiana, manhattan y chebyshev, cada una con K vecinos distintos siendo la distancia Manhattan la mejor distancia obtenida, con K=31. Obtuvimos en kaggle un error de 1092679.26063 de MAE. A continuación, tres tablas elaboradas en base a distintas iteraciones para métricas distintas, donde el valor ilustrado es la predicción del set de test.

Estos resultados son muy bajos porque no realizamos una transformación sobre los datos, para que las distancias sean normales. Como fue un modelo inicial, con la idea de ver cómo impactan los features, nos dio un modelo a superar pero no continuamos indagando en el mismo, ya que luego utilizamos Random Forest como método para análisis de los features iniciales.

3.1.2 Random Forest

Random Forest es un ensamble de árboles de decisión que usa para cada árbol una muestra de datos y features y genera árboles aleatorizados para luego promediarlos todos a la hora de predecir. El mejor puntaje logrado por random forest fue cercano al 616K, que representa un score de aproximadamente 80%. Fue uno de los métodos que utilizamos inicialmente para un rápido análisis de features y su importancia en el modelo. Si bien lo usamos con varios features, no los usamos con todos los target encoders, nos dio nuestra primer predicción para ir mejorando, lo que es muy bueno y nos hace pensar que de seguir su entrenamiento con más y mejores features habríamos podido mejorar mucho, pero sin mejorar nuestro mejor puntaje.

3.1.3 Light GBM

Light GBM es un algoritmo de gradient boosting que utiliza árboles. A diferencia de otros algoritmos de este tipo, Light GBM construye el árbol hoja por hoja y no por niveles, lo que puede ayudar a reducir el error aunque también puede resultar en un árbol desbalanceado, por lo que es importante fijar bien el parámetro de profundidad máxima y su relación las hojas. Una de las principales ventajas de Light GBM por lo que se suele usar es su velocidad y poco uso de memoria.

Hicimos más de 20 corridas de LightGBM explorando distintos parámetros y sets de datos y la amplia mayoría de nuestros mejores puntajes se deben al mismo. Entrenamos LightGBM con absolutamente todos los features que listamos previamente, y hemos realizado pruebas con menos features, de forma random y no random y pareciera ser que Light GBM se maneja muy bien en grandes dimensiones y que rara vez los features sobran. LightGBM es responsable de las predicciones más baratas que logramos, llegando a una buena precisión en kaggle con un entrenamiento de 10 iteraciones que se realiza en 4 minutos. Sobre estas corridas rapidas realizamos un grid search de parámetros para buscar los mejores. Los entrenamientos más largos de Light GBM no superaron la hora.

3.1.4 XGB

Este es un algoritmo de Boosting que genera un árbol para cada iteración, y es considerado como uno de los mejores algoritmos de regresión y para nosotros resultó el mejor. En un principio, Light GBM fue dejado un tanto de lado ya que con XGBoost se estaban obteniendo muy buenos resultados, pero la duración del entrenamiento eran más largas, cada entrenamiento del XGB duraban aproximadamente entre quince minutos a una hora dependiendo de la cantidad de parámetros utilizados.

Se comenzó utilizando los valores de estimadores bajos, ya que a medida de que se agrandaban la cantidad de estimadores, se hacia mas larga la espera del entrenamiento del mismo. Este árbol dentro de todo chico, se tuneo para ver cual era la mejor aproximación de hiperparametros para poder replicarlo en el grande, este proceso se realizó mediante Random Search y luego al mejor resultado aplicarle un Grid Search para poder mejorar su precisión. Se realizaron exhaustivas búsquedas de hiperparametros intentando continuamente de mejorar el resultado obtenido anteriormente modificando el modelo. Hay veces que tuvimos que tener cuidado porque al modificar mucho el algoritmo, este mismo overfitteaba, lo cual hay veces que nos ponía en duda algún resultado anómalo.

XGBoost fue en el que más se hizo énfasis, incluso con este algoritmo se logró el mayor de todos los resultados. Es fundamental destacar que sí o sí se necesitaron utilizar uno o más encodings. Por otro lado, fue el

algoritmo para el que más features se probaron, ya que se tomaron absolutamente todos aquellos desarrollados en la sección de features. Dado que XGBoost es un algoritmo que utiliza muchos recursos, nos llevó gran parte del tiempo poder mejorar este algoritmo pero siempre utilizando el 100% del data set, explicaremos más adelante porque decidimos utilizar todo el dataset.

3.2 Ensambles

3.2.1 Gradient Booster

Este ensamble es un conjunto de modelos de predicción débiles, típicamente árboles de decisión. Construye el modelo de forma escalonada como lo hacen otros métodos de boosting, y los generaliza permitiendo la optimización arbitraria de una función de pérdida diferenciable. Este algoritmo lo utilizamos ensamblado dos modelos, por un lado utilizamos un Random Forest y por otro lado utilizamos un LGBM, probamos algoritmos rápidos para poder ver la precisión del ensamble. Obtuvimos un error de 532K en el primer caso peor al obtenido por XGB pero ensamblando RF y con LGBM obtuvimos 496 K pero este obtener este resultado tardó una hora y media. Solo probamos este ensamble no lo tuneamos, ya que, era un modelo nuevo a utilizar y no fue visto en clase pero se puede ver su poder al predecir el precio de las propiedades.

3.2.2 Hist Gradient Booster

Es la versión rápida del ensamble anterior ya que usa histogramas para mejorar su velocidad, para datasets de gran tamaño y pudimos notar la diferencia. Al igual que Gradient Booster este algoritmo fue creado inspirado en LGBM. Ya que este algoritmo se caracteriza por su velocidad, probamos de utilizar gran cantidad de estimadores y generar árboles grandes, con muchas hojas para ver qué resultado obtenemos.

Al no ser un algoritmo que hayamos visto en clase, no profundizamos mucho en el, solo lo utilizamos y comparamos con el XGB que mayor puntaje nos dio y en poco tiempo obtuvimos un error de 484 K de MAE. Este ensamble no lo tuneamos y con un par de corridas al algoritmo logramos una precisión casi tan buena como la del XGB

3.2.2 Voting Ensamble

Este ensamble realiza una votación entre los algoritmos seleccionados proporcionando una solución entre todos los estimadores dados al ensamble. Este se probó con Hist Gradient Boosting, LGBM y XGB, obteniendo un puntaje parecido al de Hist Gradient Booster, 480 K de score MAE. Ya que los estimadores utilizados fueron simples a la hora de utilizar el algoritmo, este realizó una buena aproximación para no tunear ningún parámetro, ni realizaron más corridas con este ensamble.

3.3 Feature Selection.

En esta sección hablaremos sobre la performance de nuestro algoritmo. Nosotros al realizar demasiada cantidad de features, obtenemos un dataset de aproximadamente 220 features, lo cual a la hora de probar los estimadores con tal dataset, esto se volvía lento y agotador. Intentamos dos tipos de Feature Selection para poder mejorar esta situación, ya que si nos quedamos con la menor cantidad de features y nos resultaba un buen score, esos features serían representativos para llegar al score que necesitamos. Se utilizaron las selecciones de Select From Model y Select K Best, ademas se intento obtener los features mediante XGB, los que superan determinado valor y nos quedamos con ellos.

Una vez realizado este proceso de selección, se continuó corriendo nuevamente los algoritmos pero esta vez con menor cantidad de features. Obtuvimos una pequeña mejora de performance, realmente no fue

significativa pero lo más sorprendente fue que obtenemos un score mucho más elevado, creemos esto sucede por la falta de precisión a la hora de determinar el precio de la propiedad. Se optó por mantener el dataset cómo estaba, y continuamos realizando corridas de algoritmos del orden de la media hora o una hora de espera para ver el resultado con mayor precisión de nuestros features.

4. La mejor solución

Al realizar tanto énfasis en XGB Regressor se obtuvo el mejor resultado con el mismo, aunque muchos algoritmos no se quedan atrás de él. A mayor cantidad de estimadores que utilizamos, más tardaba nuestro algoritmo, pero más preciso era a la hora de predecir, se optó por utilizar 2000 estimadores, un valor que no fuese ni muy alto como para que en algún momento el algoritmo finalice, ni muy bajo para no obtener una buena precisión. El otro parámetro relacionado con la cantidad de estimadores es el Learning Rate, que fue tuneado con distintos valores obteniendo un valor de 0.04, ya que si era aun más bajo el algoritmo overfitteaba. Lo que observamos fue que al cambiar el learning rate, a medida que utilizamos más estimadores, este número tornaba a ser cada vez más bajo para mejorar el score porque a medida de que suavizaba la cantidad de estimadores que utilizabamos aprendiendo un poco de cada uno y no fuese alguno más predominante que otro.

Al mismo tiempo se realizaron limitaciones a los árboles creados entre los cuales se encuentra Max Depth con un valor de 11 de profundidad, Min Child Weight la cantidad mínima de nodos para realizar el split con un valor de 1, Col Sample By Tree que utiliza determinados features con un valor de 0.5, Gamma que ajusta la función de generación de nodos, para limitar aún más el crecimiento del árbol con un valor de 1. El tiempo que llevo entrenar este modelo fue de 32 minutos.

En cuanto a los features utilizados varios encodes en simultáneo, se generaron columnas duplicadas para poder realizar mediante distintas categorías a distintos targets, se realizaron 6 target encodes con los siguientes targets:

Targets utilizados

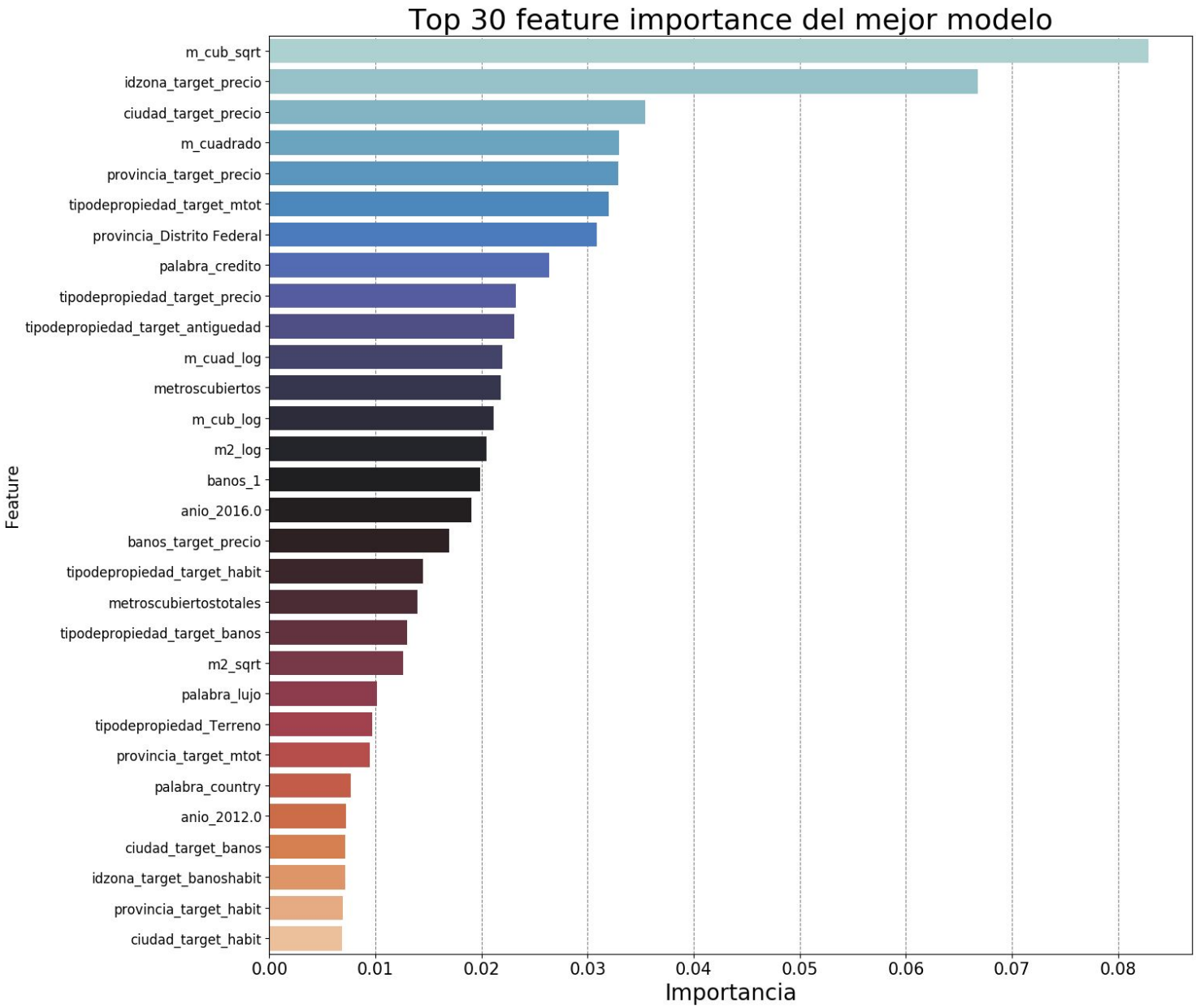
- Precio
- Antigüedad
- Metros totales
- Baños
- Habitaciones
- Baños por habitaciones

Categorías para encodear

- Idzona
- Tipo de propiedad
- Ciudades
- Provincia

A su vez, se realizaron features de One Hot Encode para las categorías tipo de propiedad y provincia, y también se realizó lo mismo para el año y el mes, por su poca variedad de valores. Para las categorías como ciudad e idzona se utilizó un Binary Encode obteniendo una menor cantidad de columnas, ya que, ambas categorías tienen un gran cantidad de valores. Por último, se realizó un Sum Encoding a las columnas como por ejemplo, baños, habitaciones, garaje, dando un resultado parecido al de One Hot Encoding pero dando más información al respecto.

Los hiperparametros más importantes para este algoritmo fueron los siguientes



5. Conclusiones

Podemos concluir que fue un trabajo que fue entretenido de hacer, teniendo la suerte de haberlo empezado con tiempo y pudiendo resolver todos los contratiempos posibles. Utilizando todo lo aprendido en clase para poder mejorar las técnicas de machine learning.

En cuanto a las publicaciones de los datos podemos concluir que el precio por la zona donde está ubicada la propiedad es importante a la hora de decidir el precio de la misma, que es algo obvio a la hora de definir el precio de la propiedad. Por lo que vimos en el TP1, había zonas mucho más caras que otras, las zonas de la capital y alrededores eran mucho más caras. También esta característica se puede denotar en la provincia de Distrito Federal en el cual se desarrolló un análisis con más atención, pudimos ver que hay ciudades dentro de la misma provincia las cuales son mucho más caras que otras, al subdividirlo en secciones más chicas (por

idzona) podemos denotar que resaltan más las zonas caras. Al haber utilizado a las zonas, estamos explicitándole al modelo categorías de propiedades, léase “estas viviendas son proclives a ser más caras/más baratas por pertenecer a cierta zona/barrio”. Al utilizar un target encoder, cuyo target es el precio, esta dependencia es aún más visible, ya que el modelo aprende del set de entrenamiento las zonas “más caras” o “ más baratas”.

Continuando otra característica importante de la publicación a la hora de definir el precio serian los metros cubiertos y los totales, como ya habíamos visto en el TP1 . Por un lado vemos que los metros cubiertos al transformarlo, obtenemos otra distribución menos agresiva hacia estos cambios, es más fácil separar por los que obtienen un mayor valor de uno menor. Por otro lado, es raro y sorprendente cómo los metros cuadrados (metros totales multiplicados los metros cubiertos) representa mejor el precio de la propiedad antes que otras características, como por ejemplo las habitaciones, los baños y la antigüedad. Está directamente relacionado con la cantidad de metros de la propiedad, lo cual tiene sentido pero creíamos que no sería de utilidad.

Pasando a las descripciones y títulos, claramente podemos ver que hay features escondidos en ellas, cómo es el caso de la palabra crédito o lujo en las propiedades; estas están directamente relacionadas con el precio de las propiedades. Por esta misma razón se intentó utilizar una red neuronal para poder encontrar esta información valiosa que nos da un poco más de precisión a la hora de definir el precio, aunque es propio de cómo una persona describe su propiedad, lo cual sería más fácil mediante un reconocimiento de imágenes.

Cómo vimos en el trabajo práctico anterior la página Inmuebles24 en México, al pasar de los años, la página se tornó aún más popular con un auge durante todo el 2016 en especial en diciembre. Para nuestros modelos esto fue importante a la hora de decidir si una propiedad es cara o barata, porque en el 2016 se realizaron precios de propiedades mas caras que los últimos años, marcando claramente la inflación que se fue generando año tras año.

El análisis que realizamos con los campos de texto consideramos que no fue suficientemente fructífero para la cantidad de datos que poseemos. Creemos que podríamos haber exprimido un poco mejor estos campos en búsqueda de mejores features, pero con nuestro approach con el método de Averaging Word2Vec no logramos obtener lo que quisimos. No pudimos indagar tanto como nos hubiese gustado, y consideramos que sería una fuente muy importante para extraer features.

También creemos que podríamos haber utilizado redes neuronales regresoras para predecir precios de propiedades, pero que no pudimos indagar demasiado en ellas más que para detección de features recién mencionados. Sin conocer a priori si serían mejor que los boosters utilizados, quizá podríamos haber utilizado para ver cómo se comparaba con nuestro modelo de XGBoost.