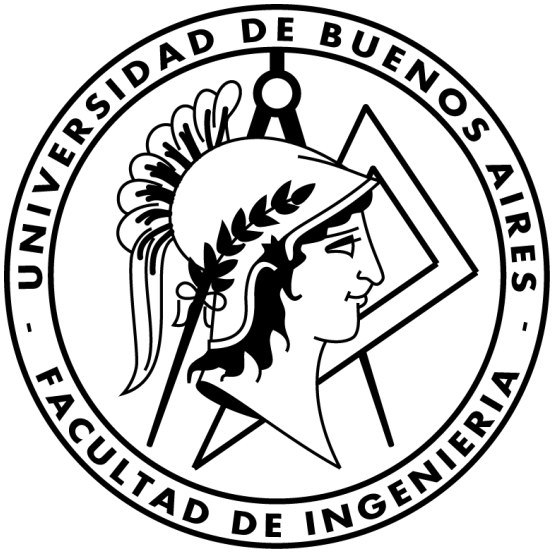


75.06/95.58 Organización de Datos - 2C 2019

Trabajo Práctico 2

Competencia de machine learning

Grupo 34: "DataTravellers"

Integrantes:

- Andrés Pablo Silvestri: 85881 (silvestri.andres@gmail.com)
- Juan Manuel González: 79979 (juanmg0511@gmail.com)
- Patricio Pizzini: 97524 (pizzinipatricio@yahoo.com.ar)

Link a repositorio de GitHub:
https://github.com/patopizzini/Organizacion_Datos_2C2019/tree/master/TP2

Fecha de entrega: 05/12/2019

Contenido

Introducción

En este informe se describe el trabajo realizado para poder llegar a los distintos resultados que se fueron entregando a la competencia de Kaggle: Inmuebles24.

La competencia consistía en estimar cúal es el valor de mercado de las propiedades publicadas en el portal ZonaProp de México, en los años 2012, 2013, 2014, 2015, 2016.

A partir de los datos proporcionados, se fueron extrayendo distintos features y probando distintos algoritmos de machine learning para obtener las mejores predicciones posibles. El proceso para obtener este resultado se describe a continuación.

Durante el trabajo, se utilizo el lenguaje de programación Python y algunas librerías extras como sklearn y pandas para facilitar el manejo de datos.

El trabajo se separó en dos etapas, la primera fue el armado de features o feature engineering y la segunda la prueba de algoritmos de machine learning y su “tueno” y combinación.

Esta división se aplicó también a nivel de codificación, dejando un notebook con el feature engineering (“ArmadoDatos”) para luego probar los algoritmos en sus respectivos archivos, tomando como entrada la salida del primer notebook mencionado.

En el link de la carátula se tiene acceso al repositorio de GitHub utilizado para el trabajo, ubicándose en la raíz el set completo de archivos utilizados para resolver el práctico, mientras que en el directorio “ENTREGA” se depositaron los notebooks utilizados para generar el submit final. Los directorios “DATA” y “SUBMITS” contienen los archivos con los datos procesados y los submits realizados a Kaggle respectivamente.

Transformación de datos y armado de features

Un primer trabajo de análisis consistió en analizar las columnas originales del dataframe y decidir cuáles descartar por no aportar valor predictivo. Seguidamente, se identificaron los tipos de datos óptimos para las columnas restantes, y la definición de una estrategia para para rellenar los valores que estaban nulos. Una vez hecho eso, se pasó a tipo 'category' todas las columnas que resultan acordes para ser categorizadas por tener una cantidad limitada de valores, lo que permite mejorar el rendimiento de las diferentes operaciones que involucren estas columnas.

Se describirá a continuación el armado de distintos features que se fueron probando, si bien no todos quedaron como definitivos para obtener el resultado final, fueron parte del proceso, coimo así también el formateo de cierta información, a fin de que nos quede todo numérico y poder utilizar algoritmos de tipo árbol que son los que elegimos para plantear la solución final de este problema.

Manejo de datos

Columnas descartadas

Descartamos las siguientes columnas del set de datos, por considerar que no aportaban valor al modelo planteado:

- Id
- Direccion
- Lat
- Long

Nulos

Sobre los nulos, decidimos separarlos según el tipo de datos que se trataba y analizar cómo resolver cada caso en particular, previo relevamiento de la cantidad de ocurrencias en cada columna.

Se manejaron de tres formas distintas y a la vez separados en dos grupos:

- Columnas numéricas:
    - Se completó con 0 o 1, en columnas donde esto era posible y no aportaba mucho a la solución, ejemplo cantidad de baños (se completó con 1,

suponiendo que muchas casas cuentan con solo 1 baño) o cantidad de garajes (se completó con 0).
    - En ciertos casos se buscó el promedio por ciudad para completar los NANs, por ejemplo en el caso de la antigüedad, los valores nulos fueron llenados por la antigüedad promedio que tenía cierta ciudad. Tomamos esta decisión porque creemos que utilizar un promedio más acotado se acerca más al valor real que el promedio de todo el set.

- Columnas no numéricas:

    - Fueron completadas con “Desconocido” o “Desconocida” para luego al momento de aplicar algoritmos de feature engineering no tenerlos en cuenta. Como este trabajo se realizó principalmente con variables de tipo categórico, se analizó agregar estos casos como alguna de las categorías ya existentes, por ejemplo “Otros” en el caso de los tipos de propiedad, pero decidimos finalmente no hacerlo para no alterar dichas categorías.

Fechas

Respecto a la fecha de publicación, ensayamos dos variantes. En ambas utilizamos solo el año, descartando tanto mes como año. En primer lugar la dejamos como una sola columna categórica, pero obtuvimos mejores resultados utilizando una segunda variante, que fue encodear el año utilizando one hot encoding.

Variables categóricas

Para las variables categóricas, se ensayaron dos variantes. Una vez tratados los NaNs y tipadas como ‘category’, las encodeamos tanto mediante one hot encoding como label encoding.

Para el caso de ‘tipo de propiedad’, ‘ciudad’ y ‘provincia’ decidimos dejar la solución con label encoding, dada la gran cantidad de columnas que generaba la primer alternativa ensayada. Por otro lado, el año, como se mencionó anteriormente, se dejó con one hot encoding.

Features utilizados

Luego de muchas pruebas, se fueron descartando/sumando features al modelo, con lo cual vamos a dividir en dos secciones esta parte: por un lado los features que finalmente quedaron en la solución final, y por otro lado los features que fueron descartados por no brindarnos valor a la hora del análisis. Vale aclarar que de estos features, siendo bastantes y quedando algunos en el camino, vamos a mencionar aquellos que a priori nos parecían más relevantes y que para nuestra sorpresa no nos dieron un valor significativo o positivo.

Inflación

Se implementó una lógica para poder reflejar la importancia y el impacto que podía tener la inflación en los precios de las propiedades, esto surgió por el hecho de tener una columna precio y una columna año que reflejaba la publicación, y las conclusiones obtenidas en el TP1 sobre el aumento interanual de los precios de las propiedades presentes en el set de datos.

Por ende, decidimos tomar el precio del año de la publicación y crear columnas que reflejaran cuál era ese mismo precio para los años anteriores y posteriores.

La información de la inflación fue una búsqueda externa, la cual pudimos conseguir a través del siguiente sitio web que es confiable con respecto a esta información: https://es.inflation.eu/tasas-de-inflacion/mexico/inflacion-historica/ipc-inflacion-mexico.aspx

De todas maneras, al no tener el precio en la parte del test, tuvimos que aprovechar y armar un agrupamiento sobre otras columnas claves como ser la ciudad, la provincia, el idzona, las habitaciones y otros features similares, para de esta manera obtener un precio estimado en base a la inflación tanto para el set de entrenamiento como para el set de test, cosa que luego se promedió y se aplicó con lógica de bandas, como explicaremos a continuación.

Agrupamiento por bandas

Varias categorías numéricas decidimos agruparlas por bandas para evitar valores sumamente exactos y potenciar overfitting en el modelo.

Por ejemplo, la antigüedad de una propiedad fue separada en categorías de a 10 años, es decir si una propiedad tenía 17 años de antigüedad, la incluimos en el grupo de 20 años, esto surge de manera transparente por una lógica sencilla provista por la cotidianeidad: uno normalmente la antigüedad de una casa la mide más genéricamente y no de manera tan particular y precisa, lo mismo sucedió por ejemplo con los metros cubiertos, los metros totales y el precio.

Esto quiere decir que formamos cuatro tipo de bandas, una para la antigüedad separada cada 10 años, los metros cubiertos y totales ambas dos separadas por bandas de 50 metros llegando hasta los 1000 metros y por último una banda sobre los precios, habiendo analizado que se podía llegar a más de doce millones de pesos y con muy pocos casos menos de 500 mil, por lo que optamos por usar este último valor como límite separados e ir avanzando de a medio millón.

Búsqueda de promedios y medianas

Probamos agrupando por ciertas categorías (idzona, tipo de propiedad, habitaciones) para luego sacar el precio promedio y las medianas. Esto, al probarlo, nos dio un resultado peor por lo que decidimos también agruparlas por bandas, como fue explicado anteriormente, para en ese caso sí obtener una mejora.

Búsqueda de features en 'titulo' y 'descripción'

Otro aspecto interesante en la búsqueda de features para el modelo fue el parseo de los campos 'titulo' y 'descripcion', entendiendo que podíamos encontrar muchísima información útil para predecir los precios. Al tratarse de columnas donde principalmente encontramos información ingresada por los usuarios, fue necesario realizar un tratamiento previo y agrupación de los datos, antes de su procesamiento final como features del modelo.

En primera instancia, sobre una copia de las columnas realizamos case folding y removimos todos los caracteres propios del idioma como los acentos. Finalmente también eliminamos los signos de puntuación. Acto seguido, procedimos a organizar las palabras resultantes en una lista y eliminar las más comunes, como 'de' y 'que', para quedarnos con los términos más significativos.

Sobre el resultado, ordenado por frecuencia decreciente, hicimos una selección de las palabras que consideramos más importantes, para ser incluidas como nuevos features.

Un último tratamiento consistió en agrupar las palabras obtenidas, por ejemplo para las palabras que identifican una nueva propiedad en título:

- NUEVA
- NUEVOS
- NUEVO
- NUEVOS
- NUEVOS
- ESTRENE
- ESTRENA
- ESTRENAR

Se definió una sola categoría, a la que nombramos 'NUEVA'. Esto fue realizado con muchas de las palabras encontradas, dado que al tratarse de columnas de texto libre existen muchas variaciones del mismo término.

Para agregar estos features al modelo, seguimos la misma estrategia que con el resto de las columnas categóricas. Se creó una nueva columna categórica para cada columna y se la encodeó mediante label encoding, y por otro lado se utilizó one hot encoding.

Finalmente nos decidimos por utilizar la segunda opción, ya que nos permitía complementar un feature dado, realizando un OR entre las columnas provenientes de título y las provenientes de descripción.

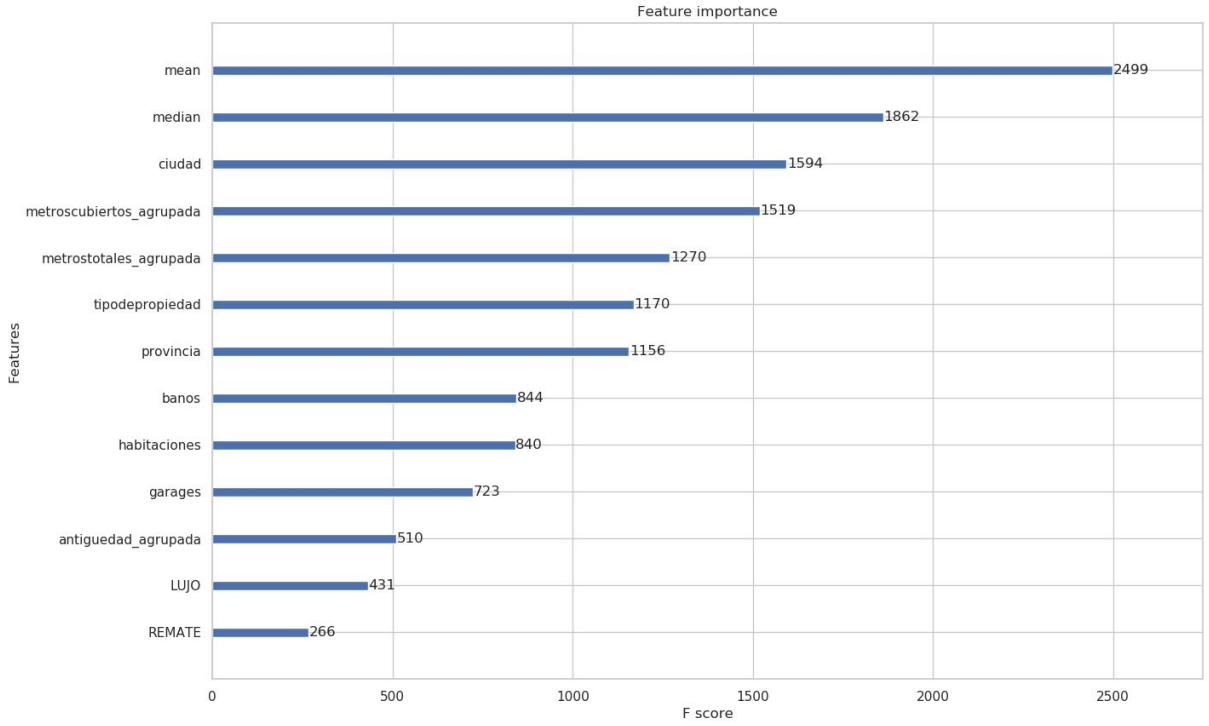
A modo de ejemplo, se realizó el OR entre 'NUEVA_x' (proveniente de 'titulo') y 'NUEVA_y' (proveniente de 'descripcion') y se guardó el resultado en 'NUEVA'. Esto permitió enriquecer estos features, dado que detectamos muchos casos donde la palabra que definia el feature estaba en una columna y no en la otra.

Feature Importance

En la búsqueda de features, una herramienta importante de la que hicimos uso para entender cómo estaba trabajando el modelo fue la creación de visualizaciones mostrando la importancia de los distintos features.

Por ejemplo, una prueba que hicimos fue dejar solamente los features más importantes, y ver si era posible mantener o inclusive mejorar el resultado obtenido previamente.

Estas pruebas fueron corridas en varias oportunidades y con varios de los algoritmos probados, en particular XGBoost.



En este ejemplo, que muestra el TOP de features, podemos ver como la predicción se basa fuertemente en lo que nosotros entendemos que define el precio de una

propiedad, como ser la ciudad (recordemos del TP1 que para el DF esto representaba el barrio), los metros de superficie, el tipo de propiedad y la cantidad de habitaciones.

Sin embargo, aquí podemos ver también como ‘mean’ y ‘median’ son dos features que más importancia tienen. Esto nos llamó la atención y decidimos hacer algunas pruebas adicionales que se explican en la sección de algoritmos (Lasso), a fin de mejorar el resultado.

Algoritmos utilizados

Random Forest

Como primer medida se decidió iniciar el trabajo para la predicción usando Random Forest. Entre otros motivos, podemos decir que la elección está basada en el simple hecho de que cuando un atributo es un buen predictor, sus árboles van a tener mejores resultados que aquellos que usan un conjunto de atributos que no son buenos predictores. Asimismo, sabemos que son invariantes a la escala de los atributos, es decir que no necesitamos normalizarlos, lo cual entre otras cosas nos facilita la operatoria.

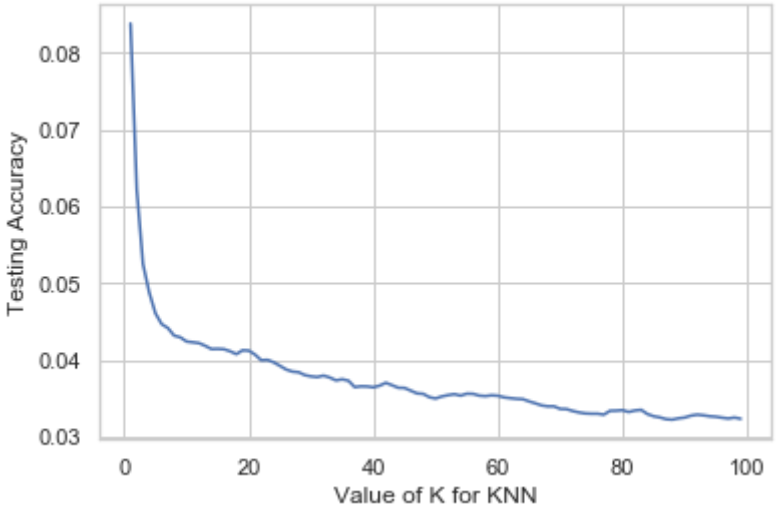
Con todo esto partimos con la idea inicial de utilizarlo para ir descubriendo qué features pueden resultar positivos y cuáles no, como ya explicamos.

KNN

A continuación, probamos KNN para determinar una cota inferior (KNN test), sabiendo que mínimamente deberíamos mejorar el resultado obtenido utilizando este algoritmo.

Logramos llegar al resultado: 1321473.45970, luego de tunear el valor de K. Sin embargo, aclaramos que este algoritmo fue corrido no muchas veces ya que era abundante el tiempo que tomaba cada ejecución.

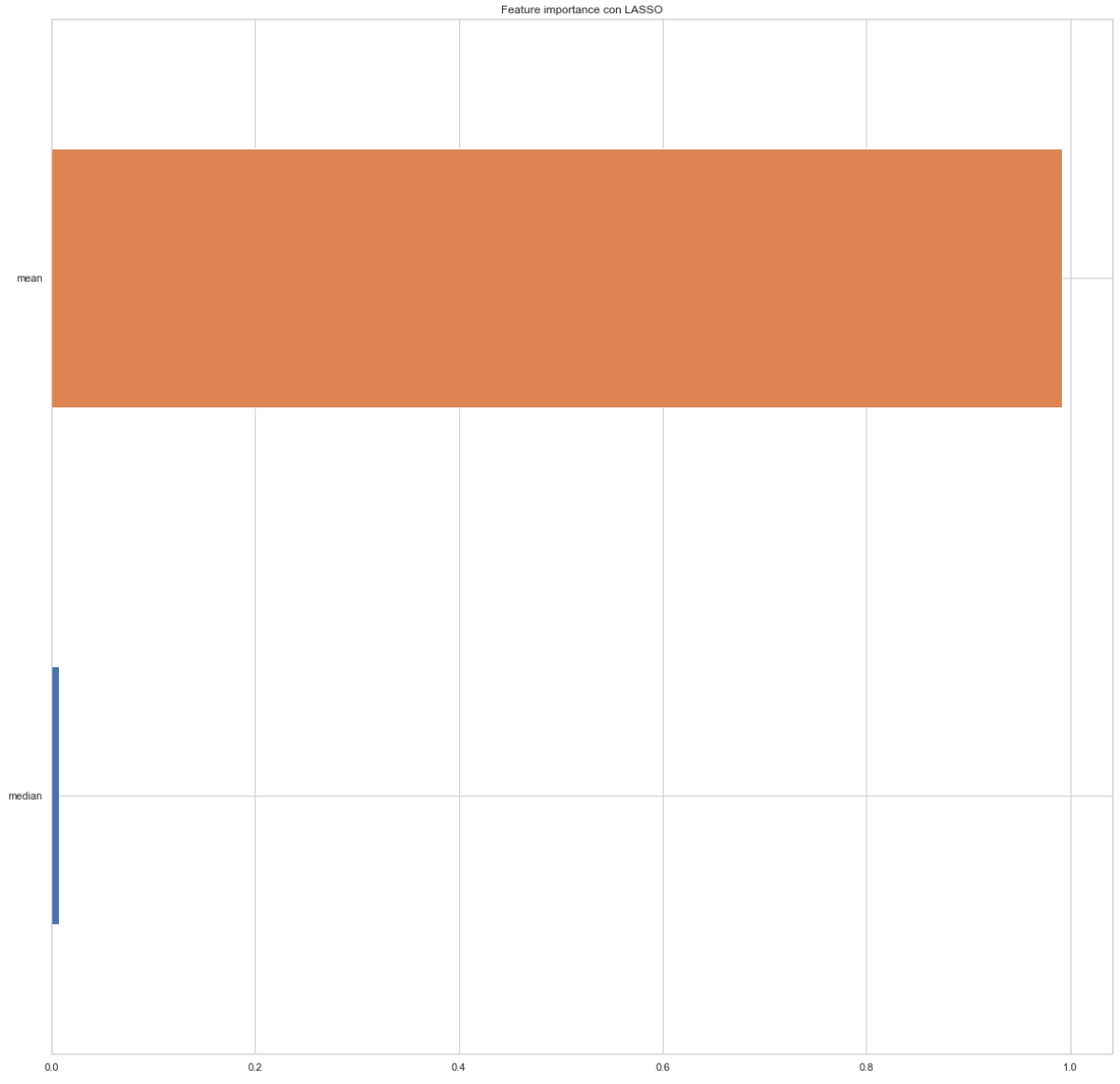
Para determinar el valor de K, hicimos un gráfico de Testing Accuracy, finalmente tomamos el valor k=10.



Lasso

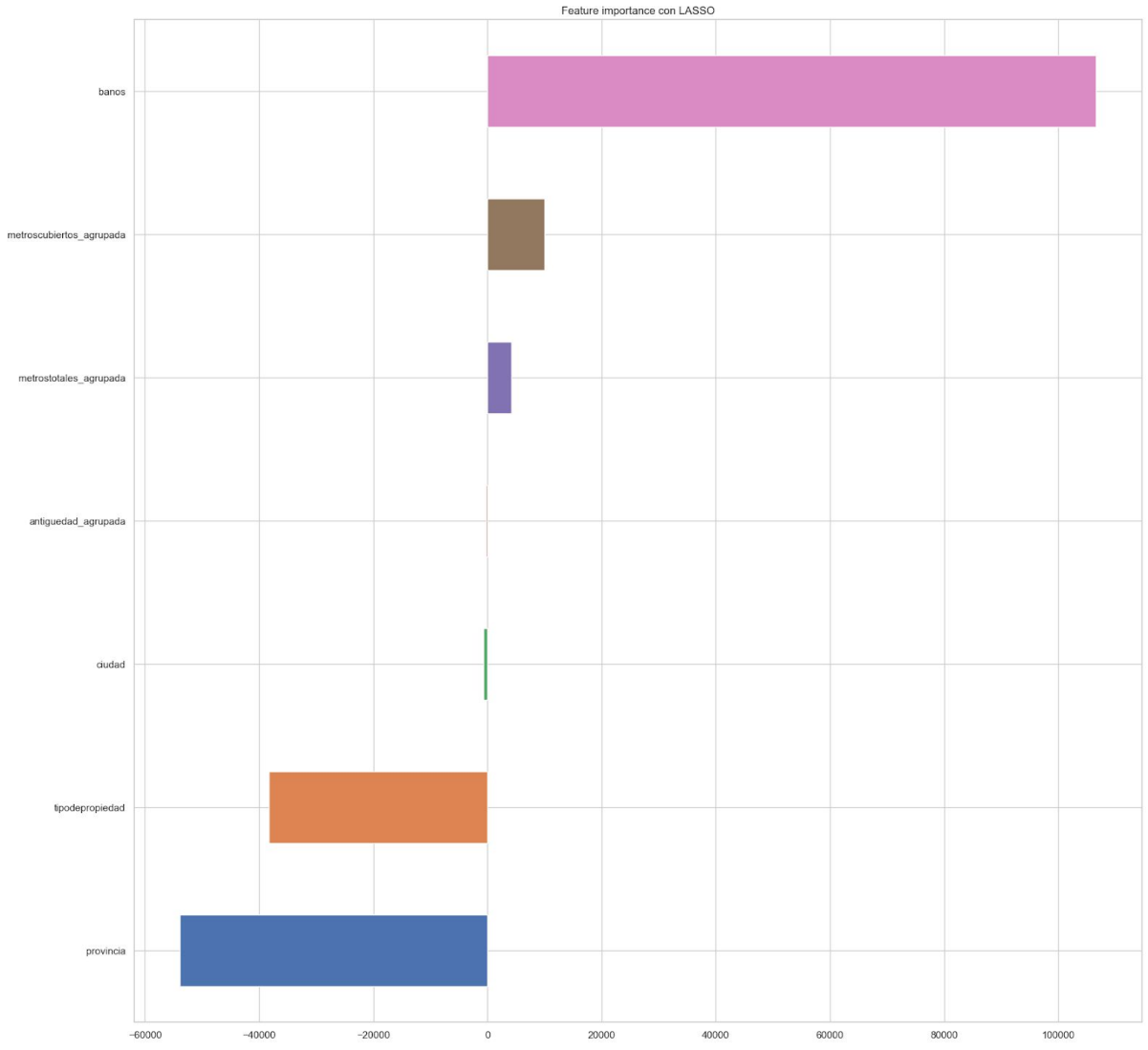
La intención inicial de este algoritmo fue utilizar una alternativa a las que teníamos como base, según lo visto en clase, y conseguir otra perspectiva y medio de análisis a sabiendas que quizás no nos fuese a brindar una predicción muy acertada. De todos modos, tomamos nuestro set de datos con los features armados (casi los definitivos) y nos pusimos a graficar a ver que nos representaba.

En particular, esta elección fue motivada también para mejorar la exactitud de las predicciones al alterar el proceso de construcción del modelo, seleccionando solamente un subconjunto de las variables provistas para usar en el modelo final, es decir ver con qué variables Lasso consideraba que era más óptimo hacer esta ejecución.



Esto nos llevó a un claro análisis, que luego también se replicó con otros algoritmos, y fue que tanto el promedio como la mediana para los precios terminaban absorbiendo toda la importancia de los features: de hecho Lasso simplemente dejaba de usar el resto de los features, lo que nos llevó a interpretar entre otras cosas que había que hacer algo para reflotar la importancia de otros features o bien buscar otros que ayuden a una decisión más acertada.

Por ende procedimos a quitar tanto el promedio y la mediana y volver a ejecutar y analizar el resultado con Lasso:



Esto nos permitió entender que otros features nos estaban dando mayor valor e importancia y cuáles podrían perjudicarnos, por lo que utilizamos dicho análisis complementar el armado de features, siempre teniendo a mano los análisis hechos para los otros algoritmos y situaciones.

XGBoost

Tomando como base los features que ya teníamos evaluados con Random Forest, decidimos aplicar XGBoost como la primer alternativa, a sabiendas que ya no podíamos encontrar muchas mejoras sobre lo que se obtenía con Random Forest. Es por eso que aunque sabiendo que este otro algoritmo no siempre es el mejor, al menos casi siempre da buenos resultados y muchas competencias se ganan usándolo como base. Por otro lado, se lo definió como 'estado del arte' en clase, lo que fue otra variable para probar utilizarlo.

En un primer momento no le pusimos mayor importancia a los hiper parámetros (no los trabajamos más que tomando algunos a base de pruebas) y probamos simplemente la ejecución del entrenamiento y la predicción. Tomamos como base un set de valores dados en clase como 'un buen punto de partida':

- n_estimators: 500
- learning_rate: 0.05
- gamma: 0.2
- lambda: 1
- max_depth: 5
- colsample_by_tree: 0.8
- subsample_by_tree: 0.8

Como función objetivo usamos 'reg:squarederror'. A partir de ese momento comenzamos a trabajar tanto con los hiper parámetros (que se verá más en detalle más adelante) como así también buscando otros features que nos pudiesen traer una mejora sustancial.

Ensamble de algoritmos

Luego tener los algoritmos empleados por separado y a sabiendas que de momento teníamos a random forest como la alternativa más precisa, decidimos emplear una combinatoria de los algoritmos probados para ver si esto nos generaba una mejora.

Hicimos una primera aproximación para entender el funcionamiento de los ensambles, abordando tres tipos diferentes de ensamble, a saber: Max Voting, Averaging y Weighted averaging, usando y aplicando diferentes combinaciones de algoritmos pero sin conseguir una mejora relevante, llegando al punto de no poder mejorar nuestra performance creada con random forest.

Aunque esto pasa a formar parte de otra sección del reporte vale aclarar que mientras entendíamos este tema empezamos casi en paralelo la optimización de los hiper parámetros, pero siendo esa parte sumamente pesada y al no tener los features

completamente definidos fuimos probando a medida que mejoraba o nos acercabamos al resultado esperado.

Luego, teniendo ya los features más definidos y los hiper parámetros más optimizados, nos topamos con un posible problema de overfitting puesto que en la predicción que hacíamos con el set de entrenamiento conseguimos valores muy altos, que luego no se reflejaban en el submit a Kaggle.

Esta situación nos llevó a tomar dos medidas para solucionarlo, por un lado tuvimos que revisar y retocar los features para poder mejorar los resultados, percibiendo por ejemplo que los features que representaban las medias y los medianas nos estaban robando toda la importancia de los features y terminamos optando en algunos casos por armar bandas de valores que recibieron valores de 1 o 0 según corresponda y así poder clarificar y mejorar el resultado de los algoritmos de decisión.

Por otro lado, tuvimos que retocar también el ensamble haciendo que la fragmentación de los datos de entrenamiento sea diferente para cada entrenamiento del modelo, como así también a pesar de usar varios algoritmos similares optamos por poner hyper parámetros diferentes, unos conseguidos a través de hyperopt y cross validation, como así también algunos básicos que teníamos anotados como ejemplo.

Terminamos optando por un ensamble con cinco algoritmos sumado a la técnica de averaging, simplemente porque fue con la que conseguimos obtener hasta el momento la mejor puntuación, siguiendo una combinatoria de 5 algoritmos, en este orden:

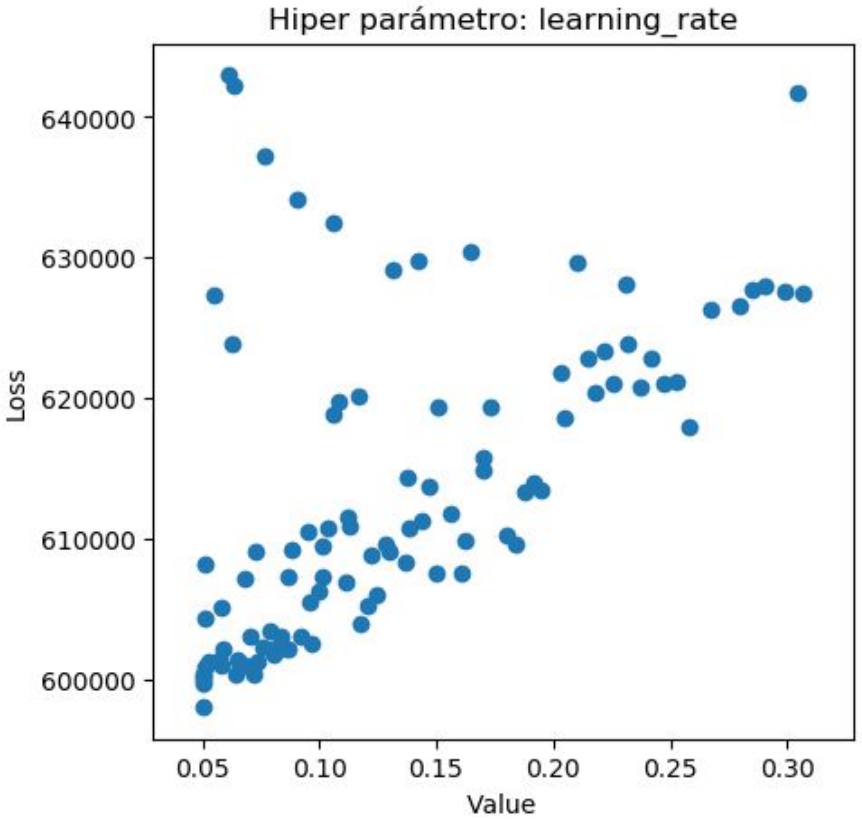
-XGBoost *(Hiper parámetros con tuneo de los más cercanos al óptimo)*
-XGBoost *(Hiper parámetros a base de pruebas)*
-XGBoost *(Hiper parámetros con tuneo óptimo)*
-Random Forest *(Hiper parámetros a base de pruebas)*
-XGBoost *(Similar al tercero)*

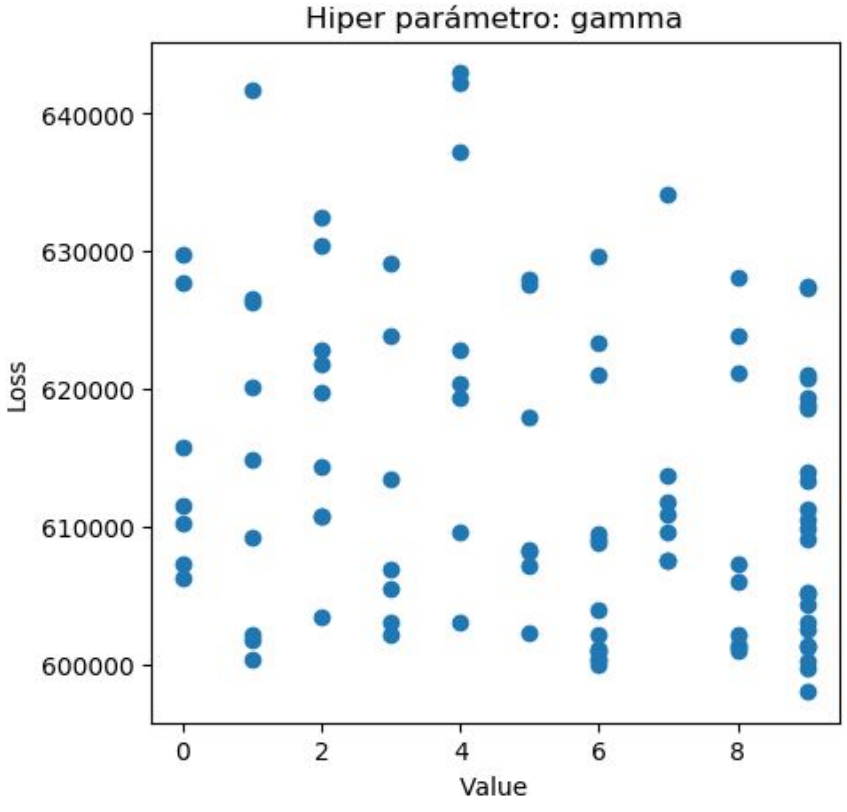
Tuneo de Hiperparametros

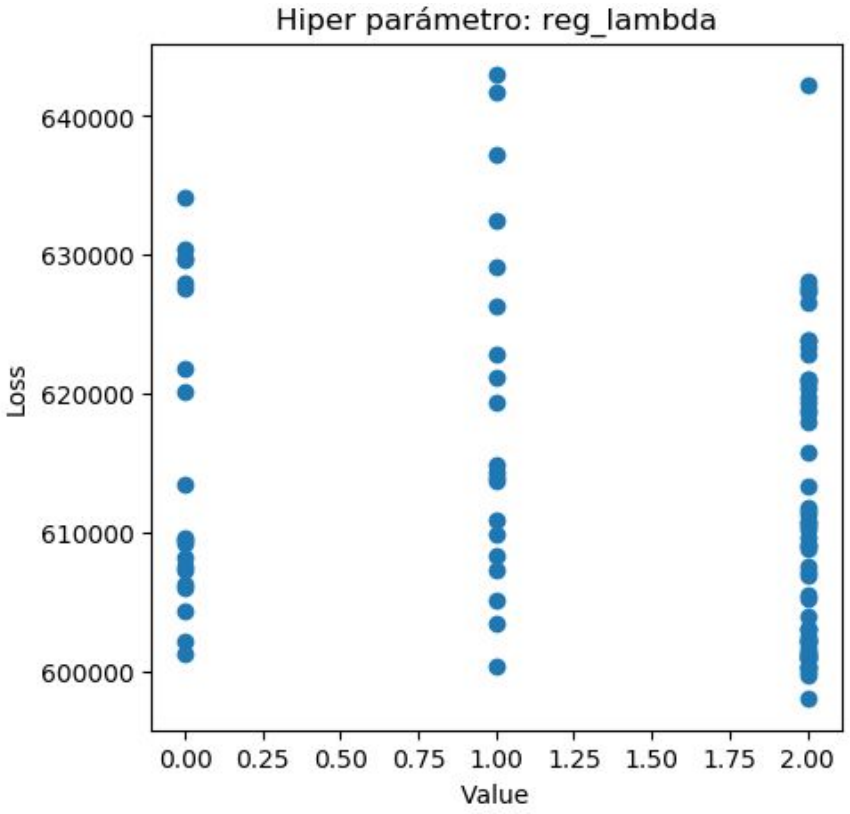
Para determinar los valores finales de los hiperparámetros de los algoritmos, se trabajó con HyperOpt.

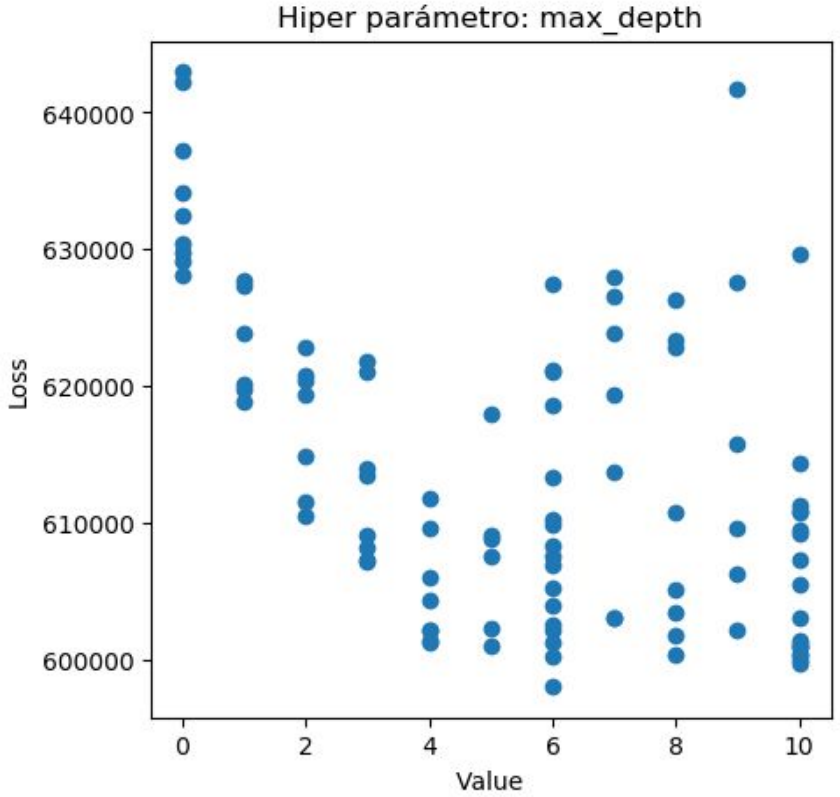
Se definió un espacio de variación para los hiperparámetros, tomando como base los valores ya mencionados en la sección anterior. Cada vez que se variaba el modelo se repetían las pruebas. Este trabajo se realizó con el algoritmo XGBoost.

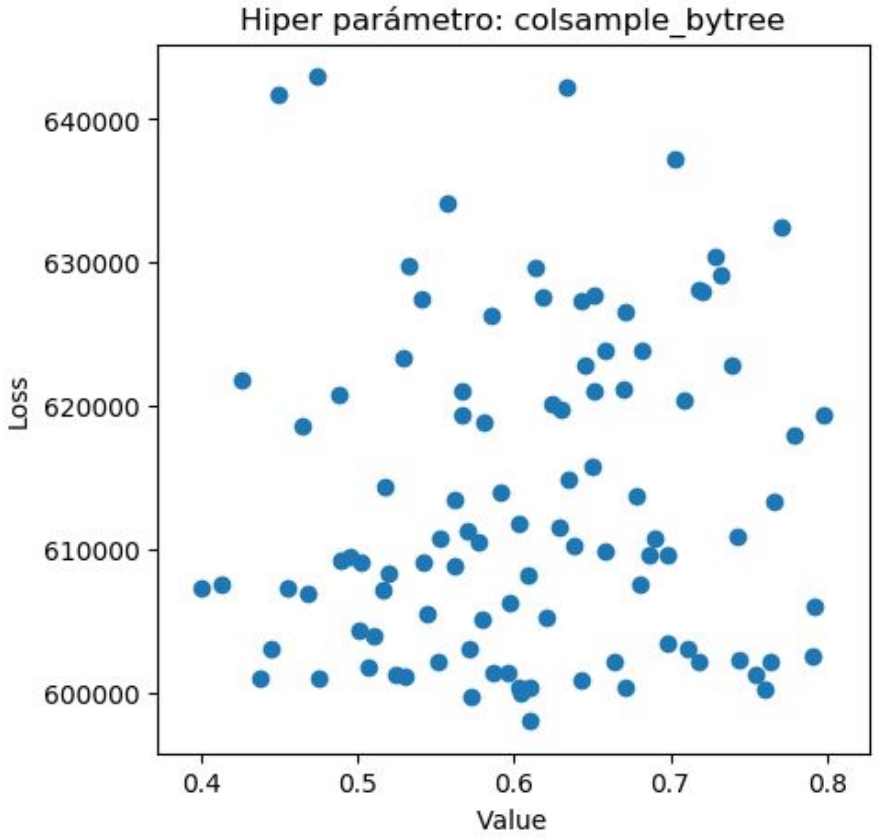
La mecánica utilizada consistió en ejecutar la optimización y graficar los valores obtenidos, para luego extraer conclusiones, y finalmente probarlos en el algoritmo. A modo de ejemplo, mostramos el set de resultados de una corrida:

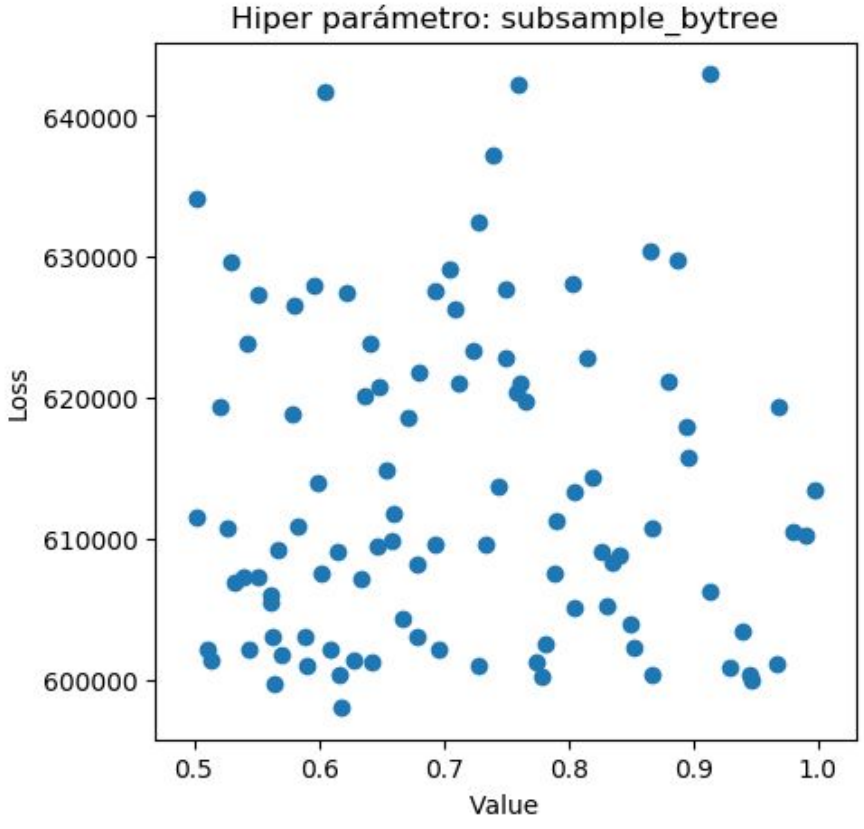

Hiper parámetro: learning_rate
Loss
640000
630000
620000
610000
600000
0.05
0.10
0.15
0.20
0.25
0.30
Value


Hiper parámetro: gamma
Loss
640000
630000
620000
610000
600000
0
2
4
6
8
Value


Hiper parámetro: reg_lambda
Loss
640000
630000
620000
610000
600000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Value


Hiper parámetro: max_depth
Loss
640000
630000
620000
610000
600000
0
2
4
6
8
10
Value


Hiper parámetro: colsample_bytree
640000
630000
620000
610000
600000
Loss
0.4
0.5
0.6
0.7
0.8
Value


Hiper parámetro: subsample_bytree
640000
630000
620000
610000
600000
Loss
0.5
0.6
0.7
0.8
0.9
1.0
Value

En esta prueba, los mejores valores obtenidos fueron:

- learning_rate: 0.05
- gamma: 9
- lambda: 2
- max_depth: 6
- colsample_by_tree: 0.6
- subsample_by_tree: 0.6

Como podemos ver, algunos de los hiperamétros tienden a sus valores iniciales, mientras que otros se ajustan más a las características de los datos, por lo que es importante hacer este tipo de experimentos, a fin de determinar cuáles son los valores óptimos.

Dado que esta operatoria consume mucho tiempo, hicimos búsquedas de distinta duración, pero no excediendo las 4hs, en las etapas de definición de modelo y experimentación inicial. Una vez determinado el modelo final, ejecutamos una prueba mucho más larga, de unas 10hs de duración, a fin de obtener los valores óptimos utilizados.

Conclusiones

Durante el desarrollo de este trabajo, se utilizaron distintas herramientas de machine learning adquiridas durante la cursada, probando diferente algoritmos como así también la construcción de distintos features para ir mejorando el score de predicción.

El trabajo realizado de adaptación de los datos y feature engineering nos permitió obtener un modelo más complejo en comparación al dataframe original, que entendemos agregó información valiosa a la predicción realizada.

Graficar la importancia de los features también tuvo su aporte, dado que gracias a esta técnica pudimos ver cómo trabajaban los algoritmos, y como vimos, decidir sobre los features a incorporar o descartar de la solución final.

Al probar los distintos algoritmos individualmente, verificamos que en este problema de ML, Random Forest nos dio un mejor score en relación a XGBoost, lo que nos llamo la atención ya que este ultimo se utiliza más para competencias de ML. El ensamble de ambos algoritmos, combinándolos en la forma descrita, nos permitió obtener el mejor puntaje, lo que destaca la importancia de esta estrategia.

Finalmente, notamos que si bien se obtenían leves mejoras al tunear los hiperparametros, esta operación puede consumir muchísimo tiempo, y las mejoras más significativas ocurrieron al agregar los features que aportaron valor al modelo. Sobre este último punto y en base al score obtenido en la competencia, entendemos que para mejorarlo tal vez faltó incorporar algunos features que no encontramos durante la exploración realizada sobre los datos.