Trabajo Práctico Grupal

7506/9558 Organización de Datos

Integrantes		
Nombre	**Padrón**	**Email**
Christian Cucco	99593	chris.cucco@gmail.com
Tomás Accini	99136	tomasaccini@gmail.com

1 Limpieza y preparación del set de datos

Para empezar este trabajo tomamos distintos set de datos de la página de Properati, filtrando con sus respectivas fechas para así formar un único dataset con las propiedades de Capital Federal y Gran Buenos Aires desde mitad de 2013 hasta mitad de 2017. También eliminamos ciertas columnas de información del dataset que consideramos no iban a aportar en nada al trabajo, como por ejemplo "properati_url" o "image_thumbnail", etc.

También en los diferentes datasets iniciales la superficie estaba descrita en forma diferente así que lo unificamos con un criterio común, eliminamos sobre los diferentes registros del dataset los datos irreales a nuestro criterio o con valores inválidos, por ejemplo propiedades demasiado grandes, demasiado chicas, o con precios por metro cuadrado demasiado altos o bajos.

Luego se preparó el set de datos a predecir para que nos quede con el mismo formato que el de propiedades.

A medida que fuimos probando modelos de predicción, fuimos eliminando registros y datos que considerábamos irreales.

Sin embargo, al contrario de lo que creíamos, al eliminar los datos que considerábamos irreales el resultado empeoraba más que cuando dejábamos todos los datos, por lo tanto consideramos que aunque haya propiedades con precios demasiado elevados para nuestro criterio y que no deberían ser tenidas en cuenta, las dejamos ya que posiblemente el set de datos que debíamos predecir no estaba filtrado y también tenía dato irreales, dándonos un error menor.

Una de las cosas más importantes que se hizo durante la preparación del set de datos fue solucionar los problemas de cuando la latitud o longitud era inválida o Nan, cosa que vimos que era muy común en el set de datos a predecir y nos iba a traer problemas. La solución a esto fue buscar el promedio de latitud y longitud por barrio y a partir de eso, con los datos que no tengan latitud o longitud le asignamos ese promedio de su barrio haciendo un merge. Haciendo este preprocesamiento, pudimos unificar toda la información de latitud, longitud, nombre del barrio y nombre de estado en coordenadas numéricas que nos facilitaba la búsqueda de los vecinos más cercanos.

Sin embargo llegando al final del trabajo nos dimos cuenta que tomando los últimos 4 años las predicciones eran normales, pero sin embargo si tomamos los últimos 2 años los resultados son mucho mejor, lo cual ahora lo pensamos lógico ya que los precios de las propiedades fueron subiendo año a año por lo que si tomamos dentro de los cálculos propiedades del 2013 hace que las predicciones sean mucho más bajas que los datos reales, por lo tanto solo tomamos los datos desde mitad de 2015 hasta mitad de 2017. Otra solución sin tener que eliminar datos pudo haber sido estudiar el porcentaje que aumentó el precio de las propiedades en cada año y corregir los precios según la fecha de publicación. Sin embargo, al investigar esto nos dimos cuenta que la variación de precios dependía de cada barrio, e incluso en zonas del mismo barrio había aumentos y descensos de precio, por lo que no tuvimos el tiempo necesario para hacer esta mejora.

2 Algoritmos probados

Para las pruebas de los diferentes algoritmos primero, armamos un set de prueba interno, utilizando la librería train_test_split, donde dividimos el set de datos en un 80% entrenamiento y un 20% prueba, con el fin de, mediante Grid-Search, encontrar aquellos hiperparámetros que mejor ajustan a cada algoritmo. La forma de medir que tan bien funcionaba cada algoritmo fue mediante una función creada por nosotros que calculaba el error cuadrático medio del dataset que predecimos.

2.1 Knn

Lo primero que probamos fue implementar un KNN nosotros. Jamás pudimos probarlo con un set de datos grande ya que, según nuestros cálculos, el algoritmo tardaría 84 horas en terminar una corrida del algoritmo con todos los datos.

Luego encontramos la librería Sklearn, y empezamos probando KNN, que en este caso sí terminaba muy rápido. Los resultados obtenidos en un principio eran muy malos, quedando siempre últimos en el leaderboard de Kaggle, pero luego de muchas pruebas con diferentes hiperparámetros, diferentes columnas utilizadas y varias optimizaciones, obtuvimos resultados más competitivos.

2.2 K-Means + Knn

Elegimos entonces aplicar K-Means y luego KNN, en este caso era un poco mayor el tiempo que tardaba el algoritmo, y entendimos que cuanto más chico sea el número de clusters armados por K-Means mejor resultado daba.

2.3 Random Forest + Knn

También probamos aplicar Random Forest y KNN pero los resultados eran peores que con K-Means, por lo tanto lo descartamos.

2.4 K-Means + Knn + Iteraciones

Luego además de esto le agregamos iteraciones, es decir que lo haga n veces y luego haga el promedio de todas esas veces. Esto nos daba menor desvío en las pruebas y permitía obtener resultado más precisos, pero no mejoró nuestra predicción. A continuación se ve una tabla de un grid search de knn + k-means con 10 iteraciones. Se puede observar que los resultados más chicos son con

un solo cluster. Esto tiene sentido, ya que si bien dentro de un mismo cluster se agrupan por propiedades similares, quedan afuera posibles vecinos cercanos, perdiendo precisión. Hubiese sido muy útil utilizar k-means si el set de datos fuese muy grande y no fuese viable hacer KNN contra todas las propiedades, pero como corría en un tiempo razonable, lo descartamos, y seguimos nuestras pruebas solamente con KNN.

	KNN										
KMEANS		1	2	3	4	5	6	7	8	9	10
	1	73168	71268	70440	70398	70145	70009	69981	70067	69675	69739
	2	74513	73251	70130	71590	70687	70457	70276	70901	70122	69640
	3	77223	74556	74728	73649	71793	71999	71277	71447	72361	71323
	4	76053	75468	74665	73572	72355	73256	73100	70857	72157	72551
	5	80041	78342	76003	75815	73477	74563	76321	73975	73695	74237
	6	80377	77610	76559	76388	76228	74296	74506	74160	74172	74598
	7	80440	78011	77083	75168	75152	75953	74645	73736	73660	74838
	8	80683	77040	74536	76609	74851	75201	74269	75157	74449	74436
	9	79585	77518	76815	77504	76745	74937	76125	74635	73809	75383
	10	81321	76233	78200	75669	76827	75063	75705	76505	75573	75884

2.5 Knn por peso

Después de eso modificamos KNN tal que no solo se tomen los vecinos y haga un promedio sino que le agrega peso, así puede aproximar mejor ya que importa cuán cerca está el vecino, por lo tanto los resultados mejoraron y mucho. A continuación un grid search de Knn con peso, con 10 iteraciones y 2 intentos para cada valor de k:

	KNN										
INTENTOS		1	5	7	10	15	20	30	40	50	60
	1	60625	57507	57403	57280	57253	57092	57306	57324	57584	57537
	2	60975	57455	57176	57269	56905	57006	57159	57357	57579	57588
	Promedio	60800	57481	57290	57275	57079	57049	57232,5	57340,5	57581,5	57562,5

Se puede observar que el mejor resultado nos da entre 15 y 20.

2.6 Agregamos ponderaciones a las columnas

Como mejora pensamos que si hiciéramos que alguna columna fuera más importante que el resto, por lo tanto agregamos una función de ponderaciones donde les pasamos los pesos o importancia que

queremos que tome cada columna. Mediante Grid-Search obtuvimos las combinaciones que mejores resultados nos daban, y luego promediamos varias combinaciones con varias iteraciones, disminuyendo aún más el error. Además, empezamos a probar con qué columnas convenía quedarnos, cuales eliminar y de las que quedaban cuáles eran más importantes. Como resultado general, vimos que la latitud y la longitud, la superficie en m2 y el tipo de propiedad eran claves para obtener un buen resultado.

En un principio probamos dándole diferentes ponderaciones a las columnas ['property_type', 'lat', 'lon', 'surface_in_m2', 'created_on', 'floor', 'rooms']. Con un grid search muy amplio vimos que el mejor resultado fue [6, 10, 10, 1, 0, 0, 0], con un error de 50811**, es decir, el tipo de propiedad tiene un peso de 6, latitud y longitud tiene un peso de 10 y la superficie un peso de 1. El resto de las columnas se les dio una ponderación de 0, por lo que las descartamos.**

A continuación se muestra una tabla en la que buscamos, con el conjunto de K={10,12,14,15,16,17,24} y para distintas ponderaciones para las columnas latitud y longitud aumentando cada 0,1, el promedio de 10 iteraciones. Es decir que si tenemos de 5 a 5.4 los numeros de ponderacion serian {5,5.1,5.2,5.3,5.4}, esos 5 valores se corren con los 7 mejores valores de knn, y con 10 iteraciones cada uno. No logramos ver ningún parámetro claro sobre los resultados, por lo que optamos por promediar el resultado de varias ponderaciones posibles, para obtener un resultado más general.

	ponderaciones	resultados
SOLO KNN con: K{10,12,14,15,16,17,24}	5 a 5.4	50804
	5.5 a 5.9	51243
	6 a 6.4	50919
	6.5 a 6.9	51288
	7 a 7.4	50839
	7.5 a 7.9	50872
	8 a 8.4	50855
	8.5 a 8.9	51349
	9 a 9.4	51517
	9.5 a 9.9	50660
	10 a 10.4	50866
	10.5 a 10.9	50690
	11 a 11.4	50732
	11.5 a 11.9	51203
	12 a 12.4	50868
	12.5 a 12.9	50524
	13 a 13.4	51096
	13.5 a 13.9	51008

2.7 Agregamos redondeo

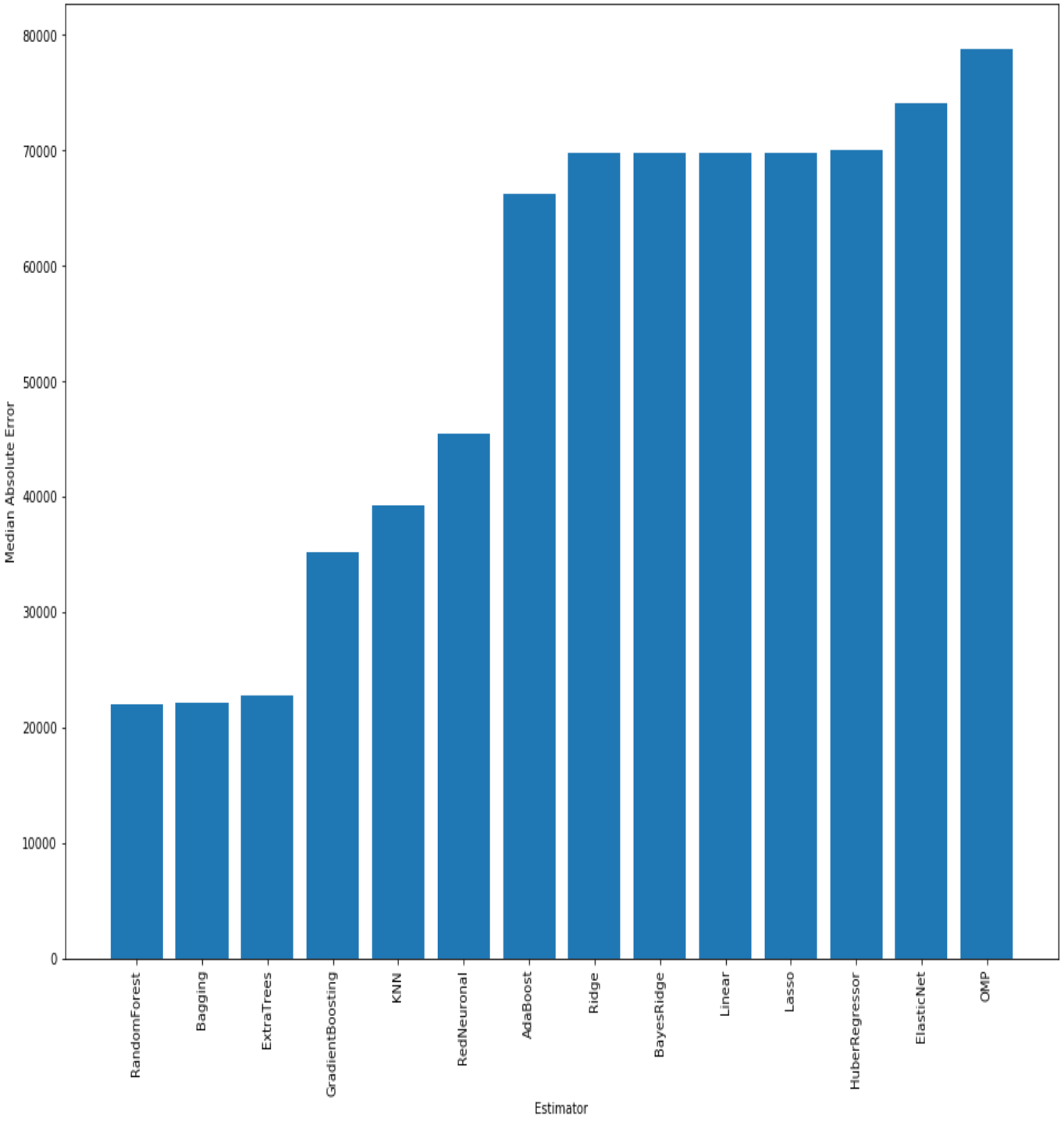
La última mejora fue aplicar una función de redondeo donde redondea las últimas tres cifra para arriba o abajo, lo cual hizo una mejora muy muy mínima pero mejora al fin. Para esto nos basamos en la idea de que el precio de una propiedad seguramente esté expresado en un valor exacto en miles de dólares. Luego de verificar que la gran mayoría de las propiedades tenían un precio final exacto (muy pocas propiedades tenían precio con centenas o decenas, ejemplo 120.040 o 120.200, el precio en general era 120.000 para ese ejemplo).

2.8 Algoritmo final utilizado

El algoritmo final utilizado es un KNN por pesos con muchos valores de K, a su vez con muchos valores de ponderaciones cada uno. Luego de todas estas iteraciones se busca el promedio y al resultado final se le aplica la función de redondeo de sus últimas tres cifras.

3 Pruebas de otros algoritmos

Ya en el final de la competencia, cuando las mejoras en el algoritmo de KNN eran muy pequeñas y costosas, decidimos ver qué otras alternativas teníamos. Para ello investigamos qué algoritmos de regresión nos proveía la librería sklearn, en particular las sub-librerías linear_model y ensemble. Así, probamos una gran variedad de modelos de predicción con sus valores por default, para tener una idea de qué tan bien funcionaban con nuestro set de datos, y ploteamos el error medio absoluto que obtuvimos con cada uno



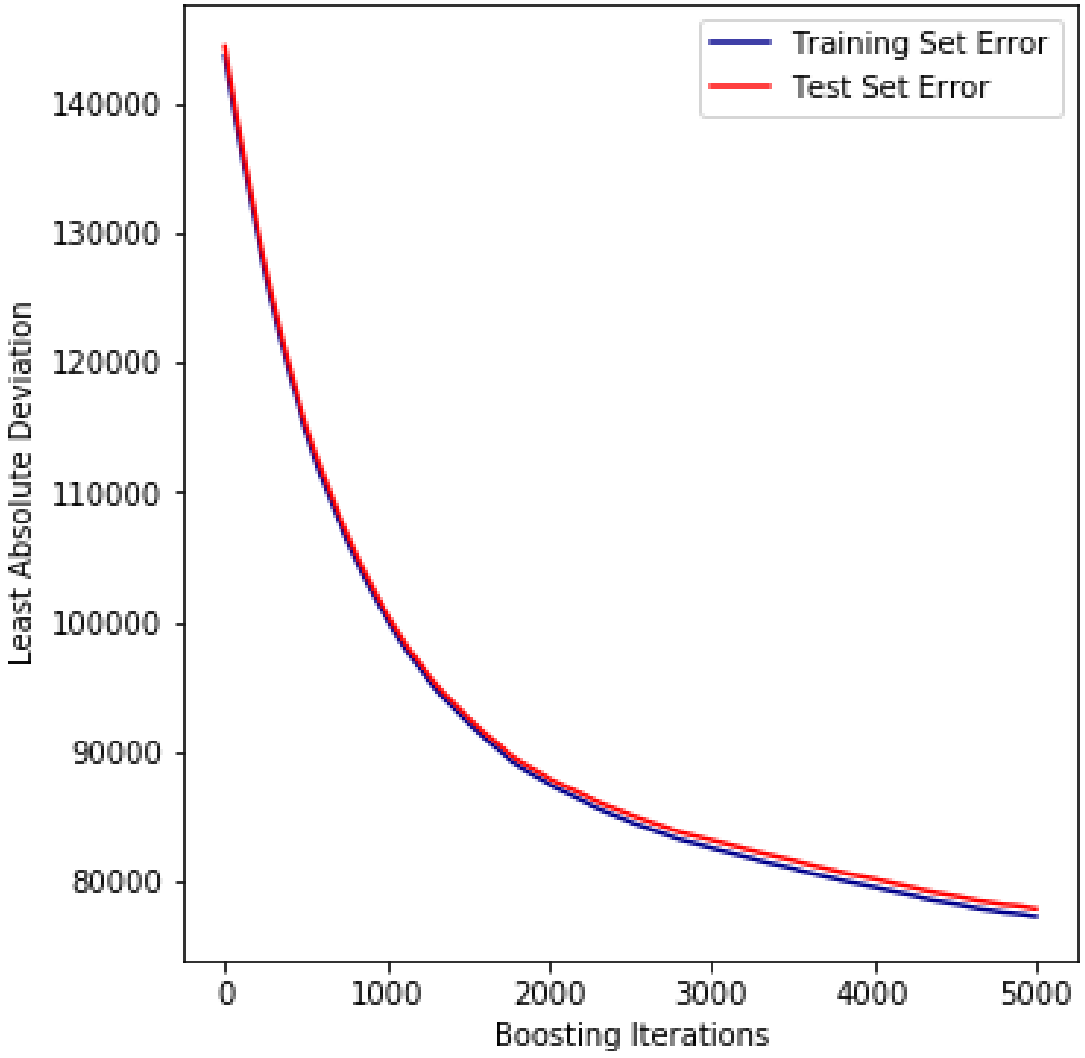
Como se puede observar, RandomForest, Bagging, ExtraTrees, GradientBoosting, KNN y Red Neuronal (MLPRegressor) tuvieron los menores errores. A continuación hicimos un análisis en mayor profundidas de cada uno de estos algoritmos.

3.1 Red Neuronal

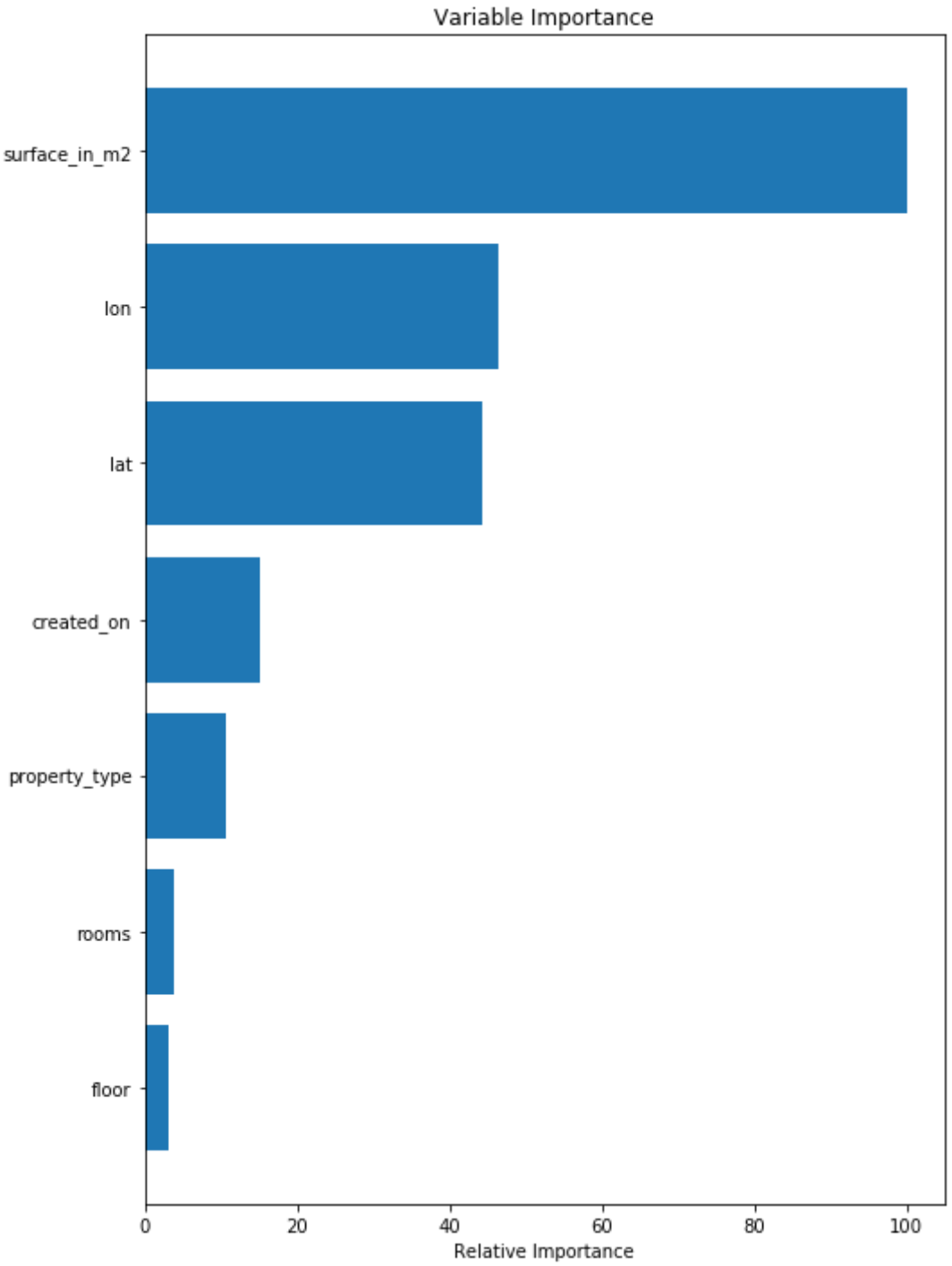
Probamos entrenando la red neuronal MLPRegressor o Multi-layer Perceptron regressor, provista por la librería sklearn.neural_network. La entrenamos con una gran variedad de iteraciones, los atributos de esta red neuronal, columnas y ponderaciones. Luego de muchos intentos de mejora y varios uploads a Kaggle, nos dimos cuenta que estabamos muy lejos del mejor resultado obtenido, por lo que desistimos y seguimos ajustando la predicción de KNN.

3.2 Gradient Boosting Regressor

Luego de hacer un Grid-Search para buscar los mejores parámetros (los cuales resultaron un learning rate = 0.001 y un max depth = 11), ploteamos el error en el set de pruebas y en el set de test en función de la cantidad de iteraciones, obteniendo el siguiente gráfico:



También quisimos verificar qué importancia le daba este algoritmo a cada columna:



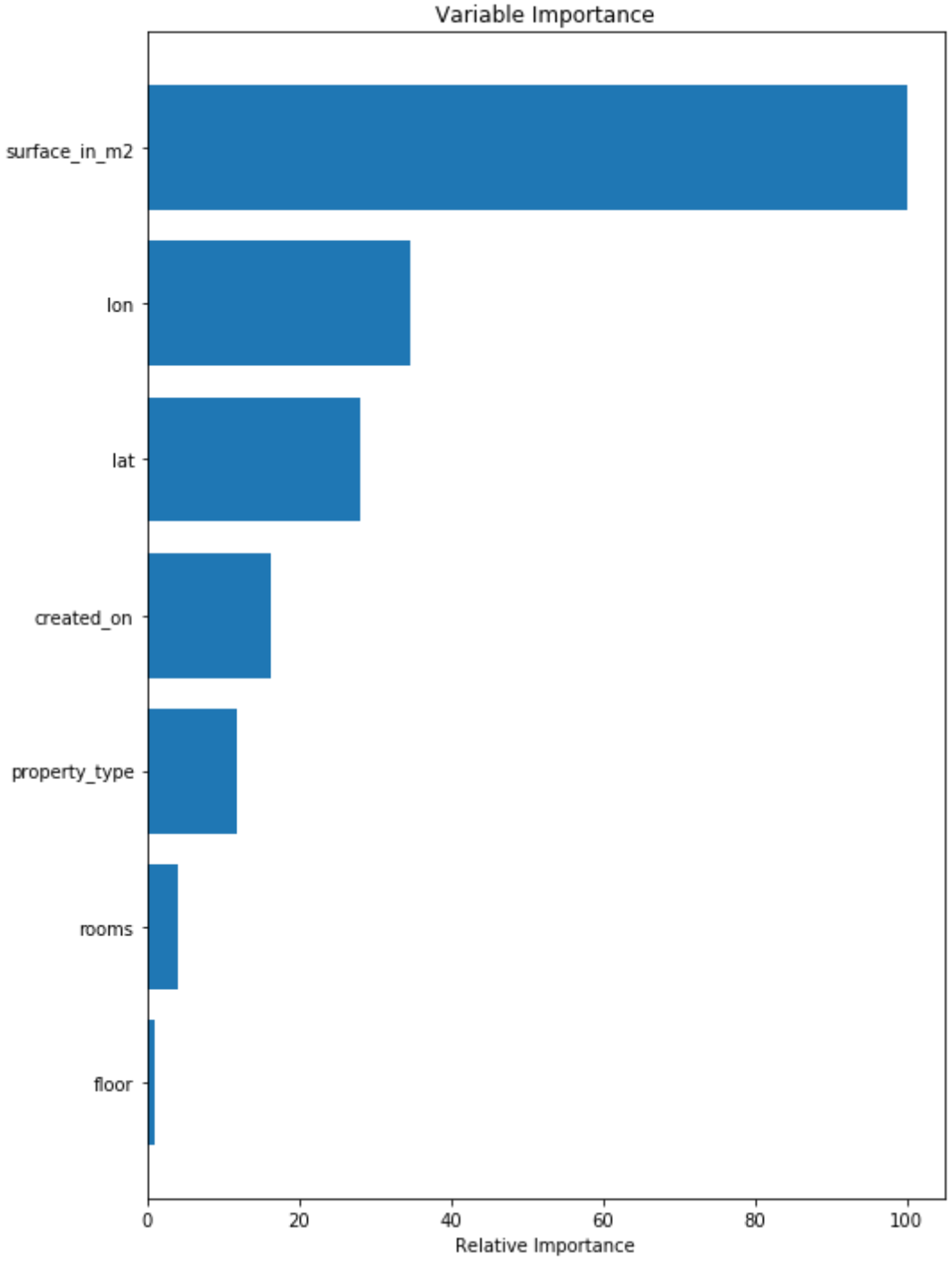
Al subir un upload a Kaggle con este algoritmo, el resultado seguía siendo peor (y por bastante) que el mejor que teníamos, hecho con KNN.

3.3 Bagging

Probamos varios valores para el parámetro n_estimators, que según lo que habíamos averiguado era el parámetro más influyente. El que minimizaba el error era 80, por lo que lo utilizamos para la predicción. Si bien daba un muy buen resultado en las pruebas locales, al momento de predecir el precio del dataset provisto por la cátedra y subir los resultados a Kaggle seguía siendo peor que el de KNN, aunque esta vez más cercano.

3.4 Random Forest Regressor

Por último probamos el Random Forest Regressor, que había dado el error medio absoluto más chico de todos los algoritmos probados. Luego de probar varios valores para el número de estimadores y quedarnos con el que mejor valor nos dió, ploteamos la importancia que le dió este algoritmo a cada columna:



Se puede observar que es muy similar a la importancia que le dio a cada columna Gradient Boosting Regressor. Es digno de destacar que le de más peso a la longitud que a la latitud, y la gran importancia que se le da a la superficie por m2.

4 Conclusiones

Algunas conclusiones generales luego de realizar nuestra primer competencia de machine learning para predecir el precio de propiedades:

- Una gran parte del tiempo se nos fue en tratar de curar los datos recibidos, eliminar registros inválidos y poner a punto el set de datos para poder empezar a hacer predicciones.
- Luego de implementar un KNN propio que tardaba 84 horas en correr, aprendimos la lección de que es mucho más eficiente buscar alguna librería ya testeada que haga lo que necesitamos (hace el código más legible, más eficiente y ahorra tiempo).
- En muchas situaciones nos enfrentamos a cambios que si bien no eran lógicos para nosotros, mejoraban los resultados en mayor o menor medida. Por ejemplo, nos parecía obvio que eliminando las propiedades con precios irreales mejoraría el algoritmo de predicción, pero esto no fue así.
- Si bien los plots de las importancias de las variables nos daba que la superficie por metro cuadrado era lo más relevante a la hora de predecir el precio de una propiedad, cuando hicimos el grid search para las ponderaciones de las columnas para KNN los mejores resultados le ponían un peso muy bajo a la superficie.
- El hecho de haber hecho todo el análisis de datos previo a la competencia de predicción de precios, nos sirvió para entender en qué cosas había que prestar más atención y a entender los datos y los posibles resultados que nos daba nuestro algoritmo.