FACULTAD DE INGENIERÍA - U.B.A.

75.06 ORGANIZACIÓN DE DATOS

2DO. CUATRIMESTRE DE 2017

Trabajo práctico 2

Machine Learning

MATIAS LEANDRO FELD, PADRÓN: 99170
feldmatias@gmail.com

AGUSTÍN ZORZANO, PADRÓN: 99224
aguszorza@gmail.com

GRUPO DE KAGGLE: GRUPO DOBLE

Github: https://github.com/feldmatias/tpDatos

Índice

1. Introducción

El objetivo del trabajo es predecir el precio de algunas propiedades en venta, aplicando algoritmos de Machine Learning, y teniendo como datos las ventas de propiedades de los últimos años proporcionados por la empresa properati. Para ello, se utilizara el lenguaje Python, ya que cuenta con la librería sklearn que contiene varios algoritmos ya implementados.

2. Análisis previo y preparación del set de datos

El set de pruebas que contiene las propiedades para adivinar el precio tiene las siguientes columnas: Tipo de propiedad, dirección, latitud, longitud, superficie, fecha, descripcion, piso, habitaciones y expensas. Analizando los datos, se puede ver que hay muchas propiedades con datos importantes faltantes, como la superficie o la latitud y longitud. Teniendo en cuenta lo analizado en el TP anterior, sabemos que estos datos son importantes e influyen en el precio. Por eso, creemos necesario calcularlos de alguna forma para poder utilizarlos. Decidimos que lo más conveniente sería obtenerlos con un promedio.

Para el caso de la latitud y la longitud, los calculamos como el promedio de aquellos que estén en la misma zona. Por ejemplo, si la propiedad esta en Villa Crespo, en caso de que no tenga su latitud o longitud las calcularemos como el promedio de todas las latitudes o longitudes de propiedades de Villa Crespo. Esto lo hacemos así porque los valores no varían mucho con pequeñas distancias. Para el caso de la superficie obtuvimos la media de las superficies de las propiedades que estén en la misma zona y además son del mismo tipo (casa, departamento, etc).

Con respecto a los datos utilizados para entrenar los algoritmos, utilizaremos los mismos datos utilizados en el TP anterior.

Viendo los resultados del TP anterior, donde analizamos que las distintas comodidades (pileta, gimnasio, seguridad, etc) pueden influir en el precio, decidimos tenerlas en cuenta, ya que contamos con las descripciones de las propiedades. En un principio utilizamos los mismos datos analizados en el TP anterior: presencia de seguridad, gimnasio, pileta, aire o elementos de climatización, y cochera. En algunos casos agregamos algunas características más para ver como varían los resultados. Entre ellas, agregamos si tiene parrilla, jardin o quincho, si la propiedad tiene más de un piso, o si está cerca de algún hospital, clínica, shopping, centro comercial o estación de tren.

Con respecto a las distancias a distintos lugares, la idea era utilizarlas también, pero en principio no las usamos debido a que no logramos conseguir estos datos para propiedades que no sean de Capital Federal.

Por último, hay que tener en cuenta que los algoritmos no aceptan strings como parámetros, por lo que fue necesario transformar los tipos de propiedad y lugares a números. Para ello, se utilizó el Label Encoder, proporcionado por la librería, que a cada cadena le asigna un número distinto.

Otro tema analizado, también en el tp anterior, es la presencia de puntos anómalos. Se puede ver que en el set de datos hay propiedades con superficie de 44000 metros cuadrados o longitud

-122, lo que claramente no es posible en la realidad. Para ello decidimos eliminar estos casos utilizando las siguientes restricciones:

$$10 \leq superficie \leq 300$$

$$-35 \leq latitud \leq -34$$

$$-59 \leq longitud \leq -58$$

$$30000 \leq precio(usd) \leq 3 * 10^6$$

2.1. Linealidad de los datos

Otro tema a analizar en los datos, previo a la ejecución de los algoritmos, es ver si los datos son lineales o no. Esto nos permitirá saber si podremos aplicar SVD para reducir las dimensiones o no, y además debemos tenerlo en cuenta para algunos algoritmos que requieren únicamente datos lineales.

Utilizando las siguientes columnas: Superficie, place name, tipo de propiedad, seguridad, gimnasio, aire, pileta y cochera, aplicamos SVD para reducirlo a dos dimensiones y obtuvimos el siguiente resultado:

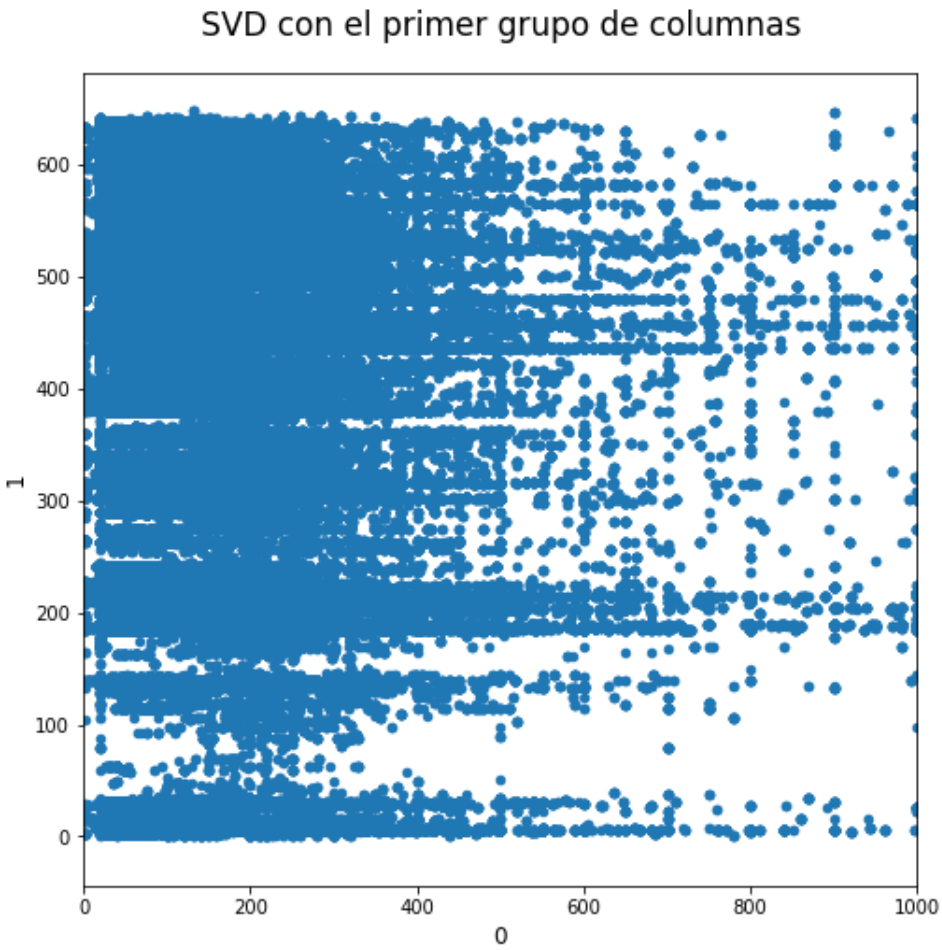

Figura 1: SVD inicial

Según lo observado en la figura 1 se puede concluir que utilizando estas columnas los datos no son lineales.

Sin embargo, si en lugar de place_name utilizamos la latitud y longitud, obtenemos el siguiente resultado:

Figura 2: SVD con el segundo grupo de columnas

Donde se puede ver que los datos son más o menos lineales. Por último, teniendo en cuenta la eliminación de los puntos anómalos, obtuvimos el siguiente gráfico, donde se ve claramente que los datos se ubican en una recta.

Figura 3: SVD con menos datos

3. Algoritmos utilizados

Cuando se intenta resolver un problema mediante el uso de machine learning se debe analizar qué tipo de problema es. Este podría ser de clasificación o de regresión. En este caso, nos encontramos con un problema de regresión, pues no se trata de decir si la propiedad es de un tipo o de otro, sino que se intenta obtener su precio a través de las características que tiene.

Por otro lado, hay que tener en cuenta que cada algoritmo recibe distintos parámetros, haciendo que el resultado sea diferente según cuáles sean estos. Por eso, para cada algoritmo realizamos pruebas con distintas combinaciones de parámetros, y nos quedamos con aquellos que nos dieron el mejor resultado.

3.1. Métricas utilizadas

3.1.1. Precisión

Para calcular la precisión de un algoritmo se utilizó la siguiente fórmula:

$$p = (1 - \frac{\sum(y_{true} - y_{pred})^2}{\sum(y_{true} - y_{true}.mean())^2} * 100$$

3.1.2. Error

Para calcular el error cometido al predecir utilizamos la misma métrica utilizada por la competencia de Kaggle, que en este caso es MSE y responde a la siguiente fórmula:

$$e = \frac{1}{n} * \sum^{n} (y_{true} - y_{pred})^2$$

3.2. Procedimiento de prueba de los algoritmos

El procedimiento que realizamos para probar los algoritmos es similar para todos, por lo tanto lo describiremos a continuación.

En primer lugar realizamos pruebas utilizando los datos de las siguientes columnas: Superficie, place name, tipo de propiedad, seguridad, gimnasio, aire, pileta y cochera. El objetivo es encontrar los mejores hiperparametros para predecir el precio de las primeras propiedades de Julio de 2017, utilizando como set de datos todas las demás propiedades. Sabemos que se trata de un caso de overfitting porque intentamos que se ajuste siempre a los mismos datos, pero nos sirvió para tener una intuición de cómo funcionan los algoritmos y cuanto tardan, además de los parámetros que reciben.

En un segundo paso, decidimos realizar lo mismo pero reemplazando la columna place name por las de latitud y longitud. En general, en la mayoría de los algoritmos obtuvimos mejores resultados que antes. Esto es debido a que la latitud y longitud son números, mientras que los otros son cadenas de texto, por lo que dos lugares podrían tener nombres parecidos pero estar muy lejos, haciendo que haya mas error. Además, implícitamente la latitud y longitud incluye en qué barrio se encuentra la propiedad.

Luego, utilizamos Grid Search y Cross Validation para encontrar los mejores parámetros del algoritmo. Para eso, utilizamos una función de la librería sklearn que usa K-fold, es decir, divide a los datos en k bloques y realiza k iteraciones, cada una con k-1 bloques como entrenamiento y el restante como validación. Luego, toma como resultado el promedio de las mismas. Además, recibe una lista con los valores de los parámetros que se quiere probar, realizando pruebas con todas las combinaciones posibles y quedándose con los que dan mejor resultado.

Por último, basándonos en el principio de bagging, y utilizando los parámetros encontrados por grid search, realizamos las predicciones calculando el promedio de varias iteraciones, donde en cada una usamos una porción al azar del set de datos o de las columnas.

3.3. Otras pruebas realizadas

3.3.1. Normalizar los datos

Consiste en normalizar las columnas, es decir, transformarlas para que tengan promedio 0 y desviación estándar 1. En general no obtuvimos mejores resultados.

3.3.2. SVD

Probamos aplicando SVD a los datos, en distintas dimensiones, para ver cómo cambiaban los resultados obtenidos. También probamos reducir los datos a dos dimensiones, y luego estas

agregarlas como columnas adicionales al set de datos original. Por último, probamos normalizar los datos antes de aplicar SVD. Utilizando SVD no obtuvimos buenos resultados.

3.3.3. Agrupar por zonas y por tipo de propiedad

Debido a que para las propiedades de Capital Federal tenemos calculadas las distancias al subte, villas, estadios, etc (del tp1), pero para Gran Buenos Aires no las tenemos, probamos separar el set de pruebas en dos y predecir los precios por separado. Para las propiedades de Gran Buenos Aires utilizamos las columnas mencionadas anteriormente, y para las de Capital agregamos las distancias. Entre ellas está la distancia a monumentos, a villas, a alguna estacion de subte o a algún estadio.

Luego, subdividimos nuevamente, pero por tipo de propiedad, utilizando para las predicciones únicamente los datos que fueran del mismo tipo de propiedad y además de la misma zona.

3.3.4. No completar datos faltantes

Como se mencionó en la primer sección, muchas propiedades a predecir tienen datos importantes incompletos, como la superficie o la latitud y la longitud.

Sin embargo, esto significa que habrá un error en la predicción. Por eso, probamos realizar la predicción de cada propiedad en función de los datos con los que cuenta. Por ejemplo, si una propiedad tiene latitud y longitud pero no superficie, utilizamos un estimador que fue entrenado con datos sin tener en cuenta la superficie. En cambio, para otra propiedad que si tiene la superficie si se tuvo en cuenta.

3.3.5. Encodear los datos de otra forma

Como se explicó anteriormente, los datos que son strings hay que convertirlos a números. Lo que hicimos en la mayoría de los casos es asignarle a cada cadena un número distinto. Pero otra forma sería usando One hot encoder. Lo que hace es, teniendo n cadenas distintas, asignarle a cada una un vector de n posiciones que tendrá todos 0 excepto en la posición correspondiente a la cadena que será 1. Esto lo utilizamos para encodear el tipo de propiedad, ya que tenía pocas opciones distintas.

3.3.6. Utilizar datos de distintas fechas

Como se analizó en el tp1, los precios fueron cambiando con el tiempo. Por eso, los distintos algoritmos los fuimos probando utilizando como set de entrenamiento únicamente los de 2017, o usando los de 2016 y 2017, etc.

En general, los mejores resultados se obtuvieron usando únicamente los de 2017, ya que son los más cercanos temporalmente a los datos a predecir.

3.3.7. Agregar una columna correspondiente a la fecha

Además de ir evolucionando año a año, los precios también evolucionaron mes a mes. Por eso, agregamos una columna a los datos que tenga en cuenta esto. El valor agregado es un número de 6 cifras con el formato aaaamm. De esta forma, los datos de Julio serán más cercanos

a los de Junio que a los de Febrero. En general, los resultados fueron similares utilizando esta columna y sin utilizarla.

3.3.8. Agregar muchas más columnas a los datos

Teóricamente, cuantos más datos y más columnas mejor serán los algoritmos. La forma que encontramos de agregar muchas columnas fue tomar palabras que aparecen en las descripciones de las propiedades.

Para eso, se hizo un análisis de la cantidad de apariciones de cada palabra en todas las descripciones de las propiedades del set de datos a predecir. De todas las palabras se tomaron aquellas que más apariciones tienen y que además tienen algún sentido. Con cada una de éstas se agregó un feature booleano nuevo que indicaba si esa palabra estaba presente en la descripción o no, además de utilizar las mencionadas anteriormente.
Haciendo esto logramos los mejores resultados obtenidos.

3.4. Algoritmos

3.4.1. Decision Tree

El árbol de decisión se trata de tomar una decisión en cada paso según los datos, y en función de ésta se continúa por alguna rama del mismo. Cuando se llega a una hoja se obtienen los datos en ella y se calcula el resultado. Entre los parámetros que variamos se encuentra el criterio de split, que puede ser mse (mean squared error) o friedman_mse que es una mejora. Por otro lado, también está la profundidad máxima del árbol, la cantidad mínima de datos para realizar un split, la cantidad mínima de datos que puede haber en una hoja, y la cantidad máxima de features a utilizar.

Realizando Grid Search obtuvimos que los mejores parámetros son: criterio mse, 80 % de features, y 100 como máxima profundidad del árbol, con una precisión de 96,39 %.

3.4.2. Random Forest

Este algoritmo consiste en realizar varios árboles de decisión con una fracción de los datos al azar, y luego obtener el resultado final promediando los resultados de cada árbol. Los parámetros que se pueden variar son los mismos de los arboles de decision, ademas de la cantidad de árboles a usar.

El mejor resultado lo obtuvimos con 20 árboles y 40 % de features, obteniendo una precisión de 93,72 %.

3.4.3. Extra Trees

Este algoritmo es muy similar al anterior, aunque tiene algunas diferencias que lo hacen aún más randomizado.

Con grid search obtuvimos los siguientes valores de los parámetros: criterio mse, 500 árboles, sin máximo de profundidad del árbol y utilizando el 40 % de los features. En este caso obtuvimos una precisión de 97,58 %.

3.4.4. KNN

Se trata de un algoritmo que obtiene los K vecinos más cercanos y luego su resultado, en el caso de regresión, es el promedio de estos K vecinos. Es decir, que si queremos estimar el precio de una propiedad, lo que se hace es obtener las K propiedades más cercanas y calcular el promedio de sus precios. La librería Sklearn presenta un algoritmo específico para KNN de regresión, al cual se le pueden variar ciertos parámetros y se puede obtener la precisión de una estimación. En nuestro caso, analizamos los resultados obtenidos para distintas cantidades de vecinos y para distintos valores del parámetro p. Este último parámetro, está relacionado con el cálculo de las distancias. Si p es igual a 1, la distancia calculada se corresponde a la distancia Manhattan, si p es igual a 2 se corresponde a la distancia Euclidiana, para un valor arbitrario de p, la distancia calculada se denomina distancia de Minkowski.

Los mejores resultados los obtuvimos cuando p = 1 o 2 y k = 1 o 2, con una precisión de 86.96 %. Este resultado es extraño puesto que la utilización de un único vecino es un caso de overfitting debido a que no generaliza bien, sin embargo, se puede ver que al aumentar la cantidad de vecinos la precisión disminuye notablemente, llegando a menos de 70 % cuando k = 5.

Luego se probó normalizando los datos, para ver si los resultados variaban, pero se obtuvo un resultado similar.

3.4.5. Bagging regressor

Consiste en dividir el dataset en varios grupos, tomando elementos al azar. Luego, cada grupo obtiene su predicción y la predicción final se obtiene como el promedio de estas o bien se obtiene por votación.

Los parámetros utilizados fueron la cantidad de estimadores, el máximo de muestras y el máximo de features. Por grid search se obtuvo el mejor resultado con 1000 estimadores, y utilizando todos los datos y features. La precisión obtenida es de 97,09 %.

3.4.6. Perceptrón multicapa

Es un algoritmo que aprende una función $f(\cdot) : R^m \rightarrow R^o$, donde m es la dimensión del dataset y o es la dimensión de la estimación. La ventaja de este algoritmo es la capacidad de aprender modelos no lineales y la capacidad de aprender modelos en tiempo real utilizando partial_fit. Una de sus desventajas sería el tiempo de aprendizaje y la gran cantidad de parámetros que hay que probar.

Los parámetros utilizados son la función de activación, la función solver y la tolerancia. El mejor resultado obtenido fue utilizando la función solver “lbfgs”, la función de activación logistic y una tolerancia de 10-10, con una precisión de 16,6 %. Lamentablemente, en este caso el resultado obtenido no es muy preciso.

3.4.7. Ada Boost

Consiste en entrenar un estimador en el dataset original y luego entrenar copias adicionales del estimador con el mismo dataset pero los pesos de la instancias son ajustadas de acuerdo al error de la predicción anterior. Los parámetros utilizados fueron la cantidad de estimadores, la

función loss y el learning rate. Con grid search los parámetros son 20 estimadores, un learning rate de 0.1 y la función loss “exponential”, con una precisión de 46,13 %.

3.4.8. Gradient Boosting

Este algoritmo construye un modelo aditivo, en donde en cada etapa entrena a un árbol de regresión. Por lo tanto, uno de sus inconvenientes es su escalabilidad. Los parámetros utilizados fueron la cantidad de estimadores, la función loss, la profundidad y el learning rate. Con grid search obtuvimos que la función loss a utilizar es “ls” con un learning rate de 0,5. El mejor resultado se obtuvo con 100 estimadores, siendo la precisión 75,65 %. Creemos que aumentando la cantidad de estimadores el resultado será mejor, pero el tiempo requerido por el algoritmo también aumenta mucho.

3.4.9. Catboost

En este caso se trata de un algoritmo que no está presente en la librería de sklearn, pero está basado en el algoritmo de gradient boosting. Los parámetros utilizados fueron la profundidad, el learning rate, la función loss y el método de estimación de hoja. El mejor resultado obtenido fue usando como parámetros una profundidad de 9, un learning rate de 0.7, la función loss “RMSE” y el método de estimación de hoja “Newton”.

También probamos normalizando los datos con standard scaler, obteniendo un resultado similar.

3.4.10. XGBoost (Extreme Gradient Boosting)

Se trata de otro algoritmo que no es parte de sklearn. Este algoritmo está basado en el algoritmo de gradient boosting. Los parámetros utilizados fueron la profundidad, el learning rate, la cantidad de estimadores y la función booster. El mejor resultado obtenido fue con una profundidad de 10, 15000 estimadores, un learning rate de 0.5 y la función booster “gbtree”.

3.4.11. LightGBM

Se trata de otro algoritmo que no es parte de sklearn y que está basado en gradient boosting. Se trata de una versión más optimizada de gradient boosting, tanto en performance como en uso de memoria.

Los parámetros que recibe el algoritmos son el tipo de boost, el learning rate y la cantidad de estimadores. Utilizando el tipo de boosting “goss”, un learning rate de 0.5 y 250000 estimadores obtuvimos una precisión de 95,34 %.

3.4.12. SVR

Support Vector Regression es la versión de SVM para problemas de regresión, y es un algoritmo que se basa en encontrar un hiperplano que separe los datos para obtener el resultado. En este caso, con estos datos, el algoritmo tuvo una performance muy pobre, tardando mucho y obteniendo malos resultados, utilizando cuatro kernels distintos (‘linear’, ’poly’, ’rbf’ y ‘sigmoid’), por lo que decidimos abandonarlo.

3.4.13. Cross Decomposition

Este algoritmo trabaja a los datos como matriz para obtener el resultado final realizando algunas operaciones basadas en la reducción de dimensiones.

Entre los parámetros que recibe están la tolerancia y la cantidad de componentes con los que se queda. En general los resultados no variaron al modificar los parámetros.

3.4.14. Algoritmos lineales

Realizamos pruebas con distintos algoritmos lineales pero obtuvimos muy malos resultados con todos ellos. Por lo tanto, concluimos que con los datos que tenemos no podemos resolver el problema linealmente.

Los algoritmos lineales probados son Ridge, Stochastic Gradient Descent, Lasso, Orthogonal Matching Pursuit y Elastic Net.

3.4.15. K-means

K-means no es un algoritmo de regresión, sino que es de clustering. Sirve para agrupar los puntos en k clusters, en función de los datos de cada uno.

El uso que le dimos a este algoritmo fue dividir el set de datos en varios cluster, y para cada uno de ellos entrenar un estimador diferente. Luego, cada punto del set de pruebas lo predecimos únicamente usando los datos que pertenecen al mismo cluster. Esto lo combinamos con otros algoritmos como Decision Tree o Gradient Boosting.

3.4.16. Ensambles con promedios de algoritmos

Tomando la idea de algoritmos como Random Forests o Bagging Regresor que utilizan varios estimadores y luego hacen un promedio, decidimos realizar lo mismo con los algoritmos que probamos. Es decir, agarramos todos los algoritmos y realizamos un promedio de los resultados obtenidos. Para ello, probamos todas las combinaciones posibles entre los algoritmos, y nos quedamos con aquella que da menor error.

3.4.17. Ensambles utilizando Blending

Blending consiste en entrenar un conjunto de estimadores los cuales serán utilizados para predecir el precio del set de test, donde cada predicción agregará un nuevo feature al set de test. Luego se tomará el set de test con estos nuevos features y se lo utilizará para entrenar otro estimador. Este último estimador será el utilizado para predecir el precio de las propiedades. Para ello, utilizamos decision tree, KNN y Bagging Regressor como estimadores aplicando en cada caso gridsearch. Luego utilizamos el set de test para entrenar a otro Decision Tree con gridsearch para obtener el estimador final.

3.4.18. KNN propio

Además de usar los algoritmos ya implementados en sklearn, decidimos implementar nuestra propia versión de KNN. Para calcular los vecinos más cercanos utilizamos la distancia usada en el TP1, es decir la distancia de Haversine, que permite calcular la distancia entre dos puntos dadas sus latitudes y longitudes.

Debido a que calcular las distancias todos contra todos es muy costoso, decidimos realizar algunas optimizaciones. Teniendo en cuenta el análisis del TP1, sabemos que el precio está muy influido por el tipo de propiedad y la superficie de la misma. Por eso, agrupamos los datos por tipo de propiedad, y dentro de cada grupo los agrupamos por superficie. Luego, dentro de cada grupo, guardamos los datos en un árbol de n dimensiones (en este caso n = 2) implementado por nosotros, que lo que hace es en cada nodo realizar un split según una de las n dimensiones. En este caso los splits se realizaron por latitud y longitud. Finalmente, se calcula la distancia con todos los datos agrupados en el nodo hoja del árbol correspondiente.

A pesar de no haber obtenido buenos resultados, nos dio mejores valores que con algunos de los algoritmos implementados en la librería.

4. Algoritmo final

Como algoritmo final elegimos Decision Tree, debido a su gran performance y los buenos resultados que obtuvimos. Los hiperparametros utilizados son:

- criterion = 'mse'
- max_depth = 100
- max_features = 0.8

Las columnas utilizadas son: Superficie, latitud, longitud, property type, seguridad, gimnasio, aire, pileta, cochera, transporte, comercio, servicios, doble piso y jardín.

Además, agregamos 81 columnas, obteniéndolas con palabras de las descripciones, como se explicó anteriormente.

El resultado final se obtiene promediando el resultado de predecir el precio 10 veces, donde en cada iteración se utiliza el 80 % de las columnas de descripciones.

La precisión obtenida con un set de validación al azar del 20 % del set de datos es aproximadamente 95 %.

5. Conclusiones

Luego de haber probado muchos algoritmos, podemos ver que en la mayoría de ellos obtuvimos una precisión muy alta (haciendo cross validation mediante K-fold), lo que significa que la mayoría de las propiedades las predecimos correctamente. Sin embargo, analizando los errores obtenidos, vemos que son valores muy grandes, ya que son del orden de 10^{10}. Esto significa que las pocas propiedades que se predicen mal, se predicen con mucho error, haciendo que el error cuadrático sea muy grande. El alto valor de los errores se puede explicar teniendo en cuenta que los precios de las propiedades pueden estar en un rango muy grande y variable de precios, haciendo que pueda haber errores del orden de 10^6 aproximadamente.

Por otro lado, podemos ver que los resultados con el set de datos que hay que predecir tienen un error mayor, generalmente dos órdenes de magnitud más, alrededor de 10^{12}. Esto puede deberse a que,como explicamos en la primer sección, algunos de los datos de las propiedades los tuvimos que completar manualmente. También puede deberse a que los datos desconocidos no están tan relacionados con los datos conocidos, o que hay mucho datos anómalos en el set de test que obviamente no podemos filtrar porque los debemos predecir.

Con todo esto, podemos concluir que predecir el precio de una propiedad es una tarea compleja, ya que hay que tener muchos factores en cuenta y además el error de la predicción puede ser demasiado alto.

Referencias

[1] Información de Machine Learning: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

[2] Libreria utilizada: http://scikit-learn.org/stable/