UNIVERSIDAD DE BUENOS AIRES FACULTAD DE INGENIERÍA

Trabajo Práctico N.º 2

Estimación de precios de las propiedades de Mexico México 2012 - 2016

Segundo Cuatrimestre de 2019

Repositorio: github.com/BraianV/Datos2019-2doCuatri

Alumno	Padrón	Mail
Cenizo, Facundo	96657	facundocenizo@gmail.com
Cruz, Pablo	97553	cruzpa95@gmail.com
Villalba, Braian	97641	braian.e.a.villalba@gmail.com

Contenido

MODELOS

Modelos usados:

- *KNN Regressor*
- *Random Forest Regressor*
- *Xgboost Regressor*

Metrica utilizada:

- *MAE: Mean Absolute Error*
- *RMSLE: Root Mean Squared Logarithmic Error*

KNN Regressor:

➔ *Conclusiones*: modelo introductorio, probado con métricas simples y con el único propósito de “ver” y “empezar a entender” el flujo de trabajo.

También sirvió como medida de piso para un modelo, sabiendo que KNN da siempre resultados cuanto menos, aceptables.

➔ *Resultados*:

- Fue un algoritmo que sirvió de base para poder entender como usar algunas métricas.
- Fue un template para los demás experimentos.
- No obtuvo buenos resultados en la competencia, aunque de manera local tenía resultados medianamente aceptables (probablemente un overfitting).

➔ *Observaciones*: fue usado como ya se explico, con hiperparametros simples sin mucha complejidad, aquellos datos faltantes (Nan) fueron rellenados dependiendo de la lógica para cada feature.

➔ *Hiperparametros utilizados*:

- Distancia:
  - ✔ euclidean
  - ✔ minkowski
  - ✔ manhattan
- K:
  - ✔ 1, 5, 10, 15, 30, 50, 70, 90

➔ *Métricas utilizadas*:

- RMSLE
- MAE

Random Forest Regressor:

➔ *Conclusiones*: modelo usado para evaluar ciertas features extraídas, la evaluación del modelo se hizo mediante algunos hiperparametros usados “popularmente”, posteriormente se buscaron hiperparametros obteniendo mejores resultados mediante algunos random search y grid search.

El algoritmo permitió que se empiece a introducir cierta lógica para el tratado de valores anómalos (anteriormente eran rellenados con 0 por simplicidad).

➔ *Resultados*:

- Modelos útiles tanto para scores de manera local, como scores en la competencia.
- Mucho overfitting hasta que se fueron puliendo ciertas features y la mejor utilización de ciertos hiperparametros.

- Primeras observaciones sobre el rendimiento de la algunas features en el modelo.

➔ *Observaciones*: sirvió para hacer las primeras subidas/experimentos en la competencia de kaggle, obteniendo valores no satisfactorios en un principo (debido al mal uso de los hiperparametros y a la inexperiencia en el trato con los mismos), aunque gradualmente se mejoró con features encontradas, depuración y limpieza de datos más finas (con mejor criterio) y mejor utilización de los hiperparametros.

➔ *Hiperparametros utilizados*:

- max_depth = 100
- random_state = 0
- n_estimators = 400
- criterion = 'mse’
- min_samples_leaf = 1
- min_samples_split = 2

➔ *Métricas utilizadas*:

- MAE

Xgboost Regressor:

➔ *Conclusiones*: es hasta el momento el mejor modelo, se utilizaron hiperparametros usados razonables, que luego fueron mejorados mediante un random search.

➔ *Resultados*: mejores scores, tanto de manera local como en la competencia.

➔ *Observaciones*: se utilizaron las mismas features que en random forest, dando una mejora sustancial. El grid search se demoró aproximadamente unas 16 horas.

➔ *Hiperparametros utilizados*:

- objective = reg:squarederror
- colsample_bytree = 0.5
- learning_rate = 0.1
- max_depth = 15
- alpha = 10
- n_estimators = 300
- eval_metric = 'mae'
- subsample = 1
- min_child_weight = 25
- gamma = 5

➔ *Métricas utilizadas*:

- MAE

Transformación de datos:

Uno de los inconvenientes que conlleva el transformar features en más features es la repetición de código y la reutilizacion de métodos útiles que facilitan dichas transformaciones. Una manera de solucionar este problema de andar llevando códigos y métodos de una notebook a otra, fue la utilización de ciertos scripts, facilitando tanto el mantenimiento de dichos scripts y su uso, es decir, darle un uso más práctico, efectivo y elegante al flujo de trabajo.

Todas las transformaciones del set de datos original se hace mediante un scripts llamado “featurizer” en dónde se realizan todas las transformaciones de datos, desde depuración de datos anómalos y el agregado de features nuevas.

Desde el script featurizer es mucho más fácil darle de baja o de alta a una feature, siendo esta una gran ventaja a la hora de correr algún experimento con algún modelo en particular.

Una manera muy efectiva de trabajar a la hora de crear ciertos features fue la de aislar todo el microprocesamiento de datos para obtener ciertas features en notebooks y lograr un output que se representaría como archivo csv en el que luego será leído por el featurizer, y posteriormente agregado al set de datos original como una nueva feature.

Este procedimiento te permite aislar y manejar de manera mucho más práctica ciertas transformaciones de datos para luego poder pasarlas en limpio en nuestro set de datos original.

Los criterios de transformación para aquellos datos anómalos (nan) fueron muy variados. Desde el rellenado por columnas con valores como su mediana, o la de rellenarlos con un valor 0, o en el caso de aquellas features con descripciones o ciertas secuencias de palabras fueron agregadas como valores “sin datos”.

Feature engineering

Features:

- *Beneficios en las propiedades :* se encarga de cuantificar aquellas propiedades que tienen algún beneficio y se las suma, es decir, aquellas propiedades que tengan gimnasio, sala de usos multiples, piscina, escuelas cercanas o centros comerciales aledaños, serían propiedades consideradas con algunos beneficios, dando como resultado la suma total esas cualidades.

  Observacion: el rellenado de valores nan en las columnas se lo hizo con 0.

- *Referencias sobre mares o sus derivados:* se buscó aquellas propiedades que tengan alguna referencia en la descripcion sobre algún tema relacionado con el mar, rio, lagunas cercas, etc.

- *Lujos en las propiedades*: en la columna de descripción se buscaron aquellas características que definen a una propiedad por su exclusividad, que van desde palabras claves como lujos,lujosa, jacuzzi, etc.

- *Palabras con referencia a espacios verdes:* sobre las descripciones se buscaron palabras claves como jardín, espacio verde, patio, vergel, etc.

- *Espacios recreativos:* desde la columna con descripciones se buscaron palabras claves que se relacionen con espacios de recreacion, desde una plaza o hasta un parque.

- *Seguridad:* como en las features anteriores, en este caso se busca en las descripciones palabras que aporten cierta certidumbre sobre seguridad, sinónimos o palabras claves referidas a éste tópico.

- *Turismo:* en las descripciones se buscaban tópicos que relacionen a la propiedad con alguna atracción de turismo.

- *Posición de la propiedad en alguna avenida:* sobre la columna que tenía la dirección, se indagó sobre si la propiedad estaba ubicada sobre alguna avenida que le de más peso al valor de la propiedad.

- *Dolar:* feature que fue agregada con un set de dato externo extraído desde https://es.investing.com/currencies/usd-mxn-historical-data que contiene todo tipo de información sobre divisas extranjeras, se buscó el precio del dólar desde la primer venta de propiedades en nuestro set de

datos, y se obtuvo la varianza de la misma (es decir, la devaluacion de la moneda)

Observacion: el set de datos propocionado por la web no contempla aquellos valores dónde la bolsa estuvo cerrada (fin de semanas y feriados), entonces la téctina tomada para rellenar esos datos faltantes fue la de extraer el dato del día anterior más cercano con datos.

- *Fecha:* como esta era una features originales, se desglosó en varias features que van desde el año, el mes, día del mes, día del año, quarter, dia de la semana y si es fin de semana.
- *Habitables:* se dividieron las propiedades en dos categorías, aquellas dónde las personas pueden residir y en las que no.
- *Agrupaciones por ciudad, provincia y tipo de propiedad:* sobre cada una de estas columnas se agrupo y se sacó el promedio de todas aquellas columnas númericas nativas del set, es decir, aquellas que ya venían con el set de datos original (piscina, garages, etc).
- *Tiempo desde la última compra del mismo tipo en días.*
- *La latitud y la longitud estandarizadas.*

Encoding utilizados:

- *One hot encoding :* por lo general esta fue la de mejor resultados, aunque su principal desventaja es la exploción de columnas en aquellas features que tienen muchas diversidad de categóricos.
- *Label encoding:* dando peores resultados pero muchisimo más eficiente para entrenar y correr modelos.

Scripts utilizados:

- *filter_feature_string.py :* su función es la de limpiar la columna descripción de todos los errores de tipeos o errores de formatos.
- *encodeador.py:* este script tiene dos métodos, uno para realizar one hot encoding y el otro es para hacer un label encoding.

- f*eaturizerV2.py:* tiene todo lo explicado anteriormente, maneja desde limpieza de datos hasta la inyección de features al set de datos original.

- *getTopicos.py:* dados un dataframe, una lista de tópicos a buscar, la columna dónde buscar dichos tópicos y el nombre de la columna resultante de buscar dichos tópicos.

  Extremadamente útil para extraer información en las descripciones y las direcciones.

Metodología de trabajo:

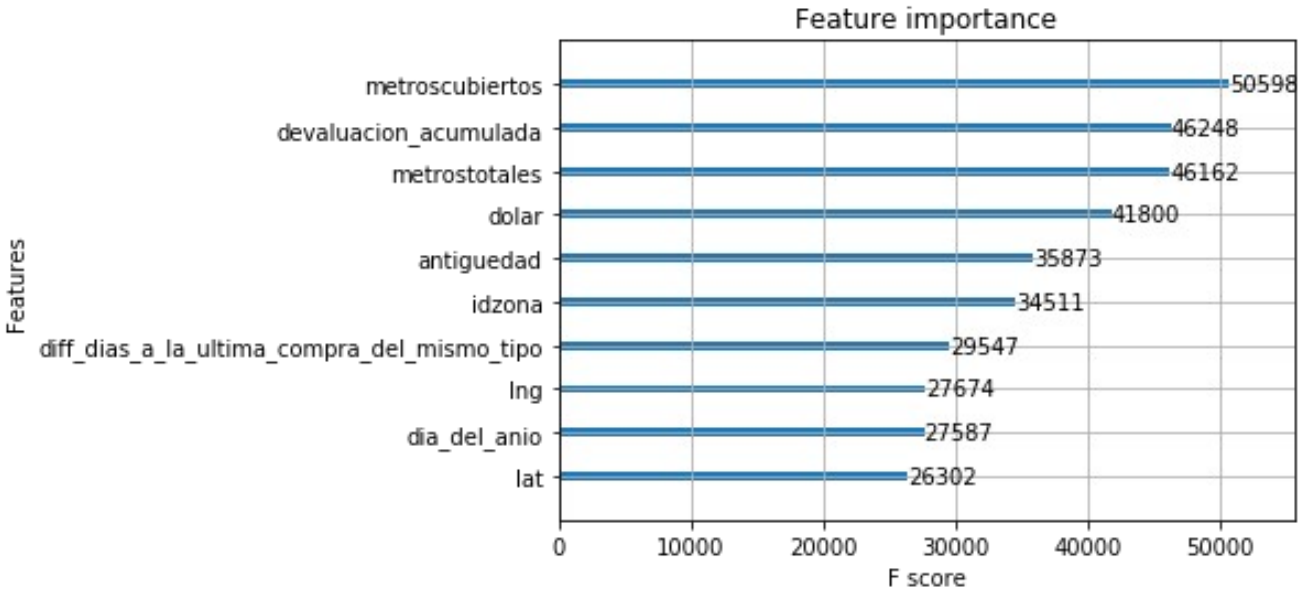
Fue mejorando de manera incremental, aquellos primeros experimentos sirvieron como piso para luego ir agregando complejidad paso a paso.

Como resultado final, las predicciones se hacen mediante una serie de pasos que se van a nombrar a continuación.

1. Lectura de csvs (train, test y otra instancia de test para el submit)
2. Se aplica sobre train y test el script featurizer.py
3. Se eliminaban aquellas columnas que no tenían un interés en particular.
4. Se separa el dataset de train en dos, uno que contiene las features y en el otro, el valor a predecir.
5. Se tratan los valores categóricos (one hot encoding o label encoding).
6. Se separ la data con algún train test split o time series split.
7. Se busca con un array de posibles hiperparametros, aquellos hiperparametros óptimos mediante un GridSearchCV o RandomSearchCV. (Este paso es opcional, debido a que cada ejecución de GridSearchCV o RandomSearchCV lleva mucho tiempo de procesamiento, por ende, se suple el paso tomando aquellos hiperparametros ya encontrados que son relativamente buenos).
8. Se entrena el modelo con los hiperparametros seleccionados, y se hace la predicción.
9. Se calcula con la métrica MAE (Mean Absolute Error) entre el valor predecido y el test que te genera los spliteos de train test split o time series split un score que posteriormente es analizado.
10. De ser una métrica medianamente buena, asumimos que no hay underfitting.
11. Una vez llegado a este paso, se prosige con el modelo entrenado, hacer una predicción sobre el set de datos de testing entregado por la cátedra.

12. Se une el resultado con el id correspondiente y se lo guarda como archivo csv.
13. Se sube el resultado de la predicción a la competencia.

Importancia de los features (según xgboost):



Conclusiones:

El modelo que mejor predicción dió es un XGBoost con :

Hiperparametros:

```
hyperOpt = {
        'objective' :'reg:squarederror',
        'colsample_bytree' : 0.5,
        'learning_rate' : 0.1,
        'max_depth' : 15,
        'alpha' : 10,
        'n_estimators' : 300,
        'eval_metric' : 'mae',
        'subsample' : 1,
        'min_child_weight' : 25,
        'gamma' : 5
}
```

Tiempo de entrenamiento : `19min 23s`

MAE local: `529910.7930725098`

MAE en kaggle : 564454.71518