Universidad Francisco Marroquín
Christian Medina
Elements of Machine Learning

Used vehicles price prediction model

Boris Rendón No. 20180497
Cuarto Año, Computer Science
18/05/2021

Introducción

El propósito de este informe es hacer un modelo de regresión para poder predecir el **precio** de los vehículos usados. Para luego hacer una comparación del resultado obtenido y el resultado esperado.

El dataset con el que trabajé cuenta con 10 variables y 108,540 observaciones.

Las variables son las siguientes :

- Model
- Year
- Price (variable objetivo)
- Transmission
- Mileage
- FuelType
- Tax
- Mpg
- EngineSize
- Make

Para este proyecto yo utilicé 3 modelos de regresión distintos y la manera para comparar su desempeño es utilizando la métrica de MAE (mean absolute error), esta métrica junto a los modelos aplicados los voy a estar explicando más adelante.

Data

Como primer paso, se hizo una exploración de datos a nivel general, donde revisé la cantidad de valores nulos en las columnas. Esto lo hago para poder saber qué hacer en caso de obtener una cantidad de valores nulos, en este caso con este código :

```
samples = vehicle_prices.shape[0]
keep_cols = []
drop_cols = []

for col in vehicle_prices.columns:
    nas = vehicle_prices[col].isna().sum()
    percent = round(100*nas/samples,2)
    if nas > 0:
        print('{0:<20} {1:<20} {2}%'.format(col,nas,percent))
    if percent < 30:
        keep_cols.append(col)
    else:
        drop_cols.append(col)
```

```
tax                  9353                 8.62%
mpg                  9353                 8.62%
```

Obtuvimos que de los 10 features, solamente 2 tienen valores nulos, pero como son menores al 20% vamos a hacer una imputación para llenar esos valores nulos.

Luego , revisé los tipos de variables que tenemos en el dataset, porque debemos de trabajar de manera distinta los valores categóricos y los numéricos. Con este análisis obtuvimos que los features numéricos son:

- Year
- Price
- Mileage
- Tax
- Mpg
- engineSize

Y los valores categóricos son :

- Model
- Transmission
- fuelType
- Make

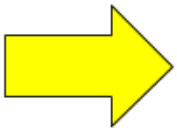
Data Categórica

Ahora que ya dividimos los datos de acuerdo a su tipo, vamos a proceder a hacer la limpieza necesaria para los valores categóricos. Como no obtuvimos valores nulos en este tipo de dato lo único que realicé es quitar la variable **model** ya que cuenta con demasiados valores y podía afectar el desempeño de nuestro modelo.

Una vez nos quedamos con las variables que vamos a utilizar, procedimos a realizar un OneHotEncoding. Esto nos ayuda para poder procesar este tipo de datos (categóricos) y ayuda para que los modelos tengan un mejor desempeño.

Este es un ejemplo sencillo de OneHotEncoding:

Color
Red
Red
Yellow
Green
Yellow



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

En lugar de tener la palabra en la fila, la tenemos en la columna y un valor 1 o 0 dependiendo si cumple el criterio.

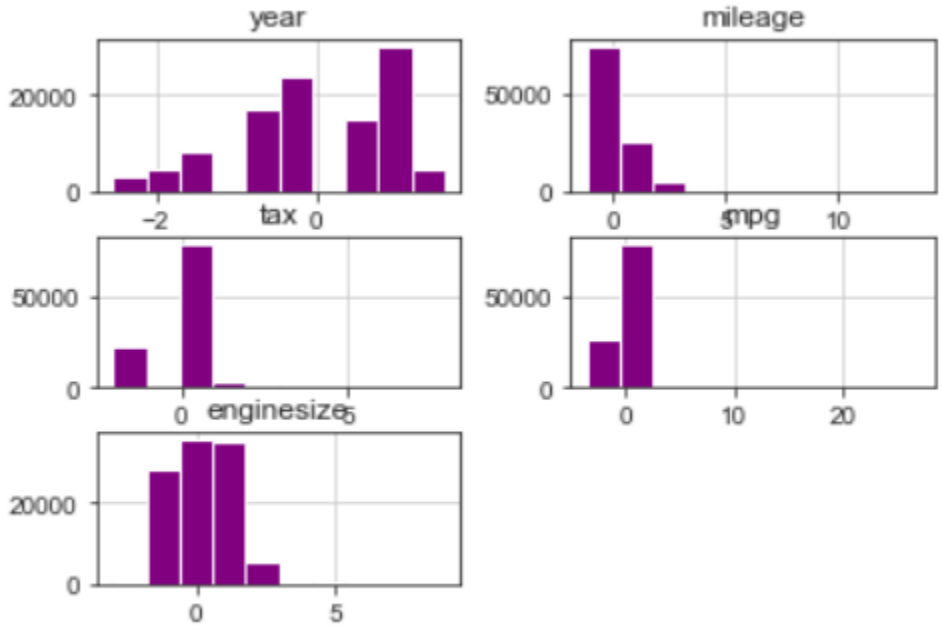
Data numérica

Con la data numérica podemos hacer más tipos de limpieza de datos. Como mencionamos arriba, la variable mpg y tax contaban con valores nulos por lo cual hicimos una imputación por media :

```
tax = vehicle_prices['tax'].mean()
tax =math.floor(tax)
mpg = vehicle_prices['mpg'].mean()
mpg = math.floor(mpg)
print("Media de tax" , tax , "y la media de media de mpg", mpg)
```

```
Media de tax 120 y la media de media de mpg 55
```

También hicimos un análisis de distribución de la data y nos dimos cuenta que no estaba estandarizada la data, para hacerlo aplicamos la técnica de StandardScaler y este fue el resultado que obtuvimos:



*** podemos referirnos al notebook para ver más a detalle el proceso de estandarización de la data.**

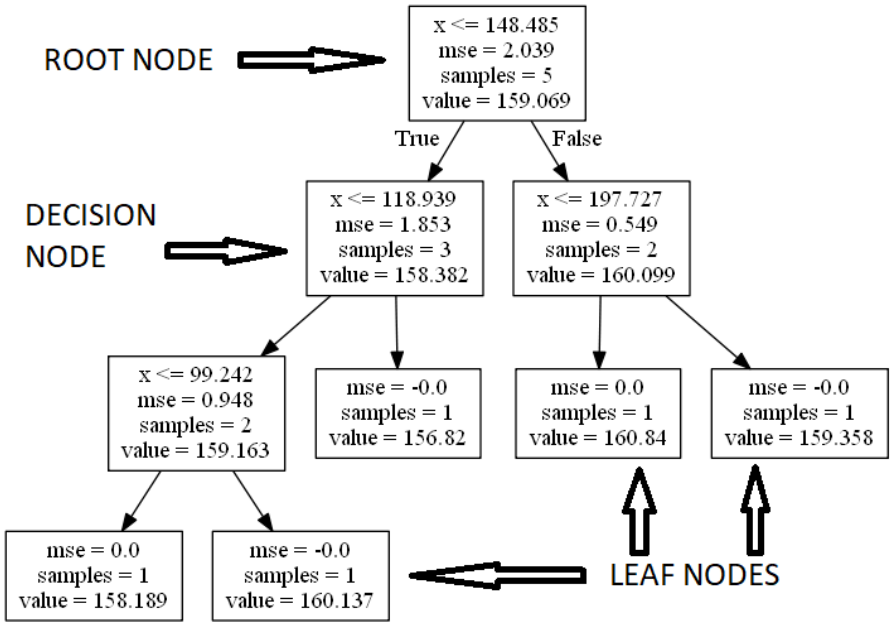
Una vez ya tenemos nuestra data categórica y numérica lista para trabajar, procedemos a combinar los dos datasets en uno solo para dividir la data en entrenamiento y testing.

Para este proyecto, dediqué el 30% de la data para test y el 70% para hacer training, también hice shuffle a la data para evitar overfitting.

Modelos

Decision Tree

Este modelo puede llegar a ser bastante útil para poder predecir un dataset como este. El hiperparámetro con el que jugué para obtener mejores resultados fue el **max_depth** que representa la profundidad del árbol.



https://medium.com/@dhiraj8899

=

Este es un ejemplo de cómo se ve un decision tree. El parámetro que utilizamos para medir la pureza de un nodo es GINI.

Para este modelo hicimos 4 pruebas con diferentes hiper parámetros y el mejor resultado fue con una profundidad de 9.

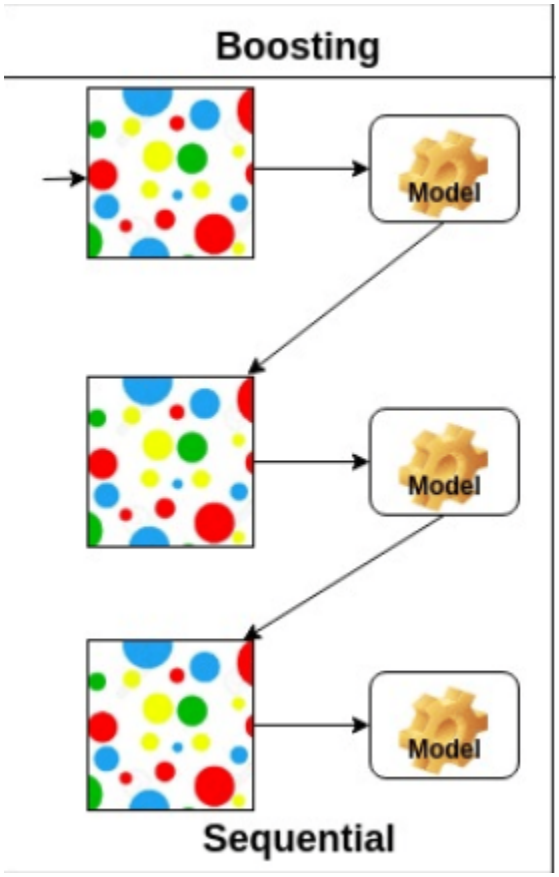
EL mejor MAE obtenido fue de 2675 lo cual nos dice que está alejado en 2675 dólares del valor esperado.

- La documentación de Decision tree la podemos encontrar en : *https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html*

AdaBoost

Para entender el concepto de AdaBoost debemos de saber que a diferencia de un decision tree o un Gradient Boosting regressor, no podemos modificar el depth del árbol, ya que aquí solo se hace un árbol con 1 root node y 2 leaves nodes.

Estos árboles se llaman **Stumps** los cuales son predictores débiles y de tantos predictores débiles que realiza, hace un predictor fuerte, este algoritmo le pone más peso a las predictores que no clasifica tan bien para poder mejorar el desempeño.



Este ejemplo nos muestra como los stumps van avanzado de manera secuencial aprendiendo del anterior.

La mejor regresión que logré tener es con los hiper parámetros de 100 estimadores, un learning rate de 1 y loss = ‘linear’ y obtuve un MAE de 8764 , lo cual es bastante alejado de lo esperado.

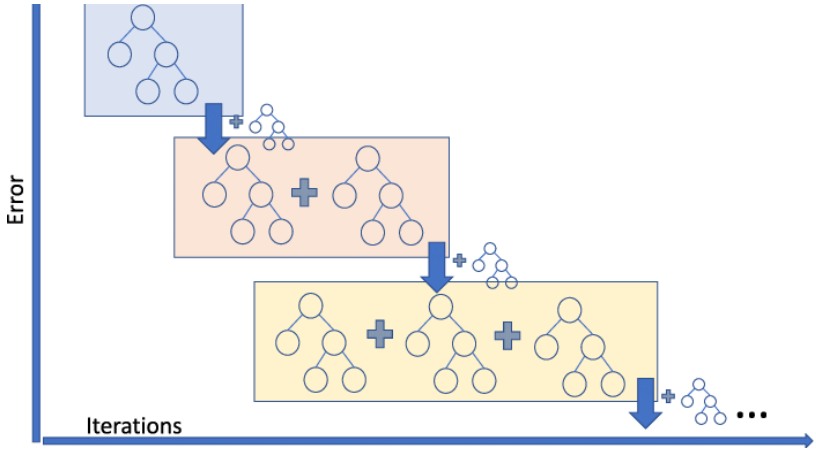
- La documentación de AdaBoost regression la podemos encontrar en: *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html*

Gradient Boosting Regressor

Este algoritmo fue el que mejor resultados me dió y tiene sentido porque es el algoritmo más robusto de los 3 que probé y a diferencia de AdaBoost, aquí sí podemos cambiar el depth del árbol, parámetro el cual mejora el desempeño .

Gradient boosting tiene tres elementos principales:

- Función de pérdida para optimizar
- Weak learners para hacer predicciones
- Manera iterativa para agregar árboles.



En este ejemplo podemos ver cómo se van agregando árboles de manera iterativa al modelo bajando así cada vez más su error.

Para el Gradient Boosting regresor el mejor MAE que pudimos obtener es de 1320, lo cual es un resultado mucho mejor a los demás. Los parámetros para obtener este resultado fueron learning rate de 0.1, número de estimadores 450 , random state de 42 y un max depth de 5.

- La documentacion de Gradient Boosting Regressor la podemos revisar *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html*

Explicando MAE

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|$$

Mae es el promedio de la magnitud de los errores de un set de predicciones sin considerar su dirección. Es el promedio de la muestra del test de todas las diferencias absolutas entre la predicción y la observación.

En otras palabras es ver que tanto se alejó el valor en términos absolutos el valor predicho contra el valor real.

Conclusiones:

Era de esperarse que el modelo con mejor performance de estos 3 haya sido el GradientBoosting Regressor ya que como mencione anteriormente , es el más robusto de los 3 que utilicé.