houses-prices

October 16, 2024

1 3.3 Arbol de Desición

Predicción de precios a partir del conjunto de datos.

House Prices - Advanced Regression Techniques

[57]:
```
import pandas as pd
```

[58]:
```
home_data = pd.read_csv('train.csv')
home_data.columns
```

[58]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

Definimos la variable objetivo, en este ejercicio será 'SalePrice'

[59]:
```
y = home_data.SalePrice
```

Ahora creamos un DataFrame llamado X que contiene las siguientes caracteristicas predictivas: * LotArea * YearBuilt * 1stFlrSF * 2ndFlrSF * FullBath * BedroomAbvGr * TotRmsAbvGrd

[60]:
```
features = ['LotArea','YearBuilt','1stFlrSF','2ndFlrSF','FullBath',
                    'BedroomAbvGr','TotRmsAbvGrd']
```

```
X = home_data[features]
```

Revisamos los datos

[61]:
```
print(X)
print(y)
```

```
      LotArea  YearBuilt  1stFlrSF  2ndFlrSF  FullBath  BedroomAbvGr  \
0        8450       2003       856       854         2             3
1        9600       1976      1262         0         2             3
2       11250       2001       920       866         2             3
3        9550       1915       961       756         1             3
4       14260       2000      1145      1053         2             4
…         …          …         …         …         …             …
1455     7917       1999       953       694         2             3
1456    13175       1978      2073         0         2             3
1457     9042       1941      1188      1152         2             4
1458     9717       1950      1078         0         1             2
1459     9937       1965      1256         0         1             3

      TotRmsAbvGrd
0                8
1                6
2                6
3                7
4                9
…              …
1455             7
1456             7
1457             9
1458             5
1459             6

[1460 rows x 7 columns]
0       208500
1       181500
2       223500
3       140000
4       250000
         …
1455    175000
1456    210000
1457    266500
1458    142125
1459    147500
Name: SalePrice, Length: 1460, dtype: int64
```

[62]:
```
X.describe()
```

[62]:
```
             LotArea    YearBuilt     1stFlrSF     2ndFlrSF     FullBath  \
count    1460.000000  1460.000000  1460.000000  1460.000000  1460.000000
mean    10516.828082  1971.267808  1162.626712   346.992466     1.565068
std      9981.264932    30.202904   386.587738   436.528436     0.550916
min      1300.000000  1872.000000   334.000000     0.000000     0.000000
25%      7553.500000  1954.000000   882.000000     0.000000     1.000000
50%      9478.500000  1973.000000  1087.000000     0.000000     2.000000
75%     11601.500000  2000.000000  1391.250000   728.000000     2.000000
max    215245.000000  2010.000000  4692.000000  2065.000000     3.000000

       BedroomAbvGr  TotRmsAbvGrd
count   1460.000000   1460.000000
mean       2.866438      6.517808
std        0.815778      1.625393
min        0.000000      2.000000
25%        2.000000      5.000000
50%        3.000000      6.000000
75%        3.000000      7.000000
max        8.000000     14.000000
```

[63]:
```
X.columns
```

[63]:
```
Index(['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath',
       'BedroomAbvGr', 'TotRmsAbvGrd'],
      dtype='object')
```

Se crea un DecisionTreeRegressor que guardaremos en `iowa_model`.

Luego se ajusta el modelo que acabamos de crear utilizando los datos en X e y.

[64]:
```
from sklearn.tree import DecisionTreeRegressor
iowa_model = DecisionTreeRegressor(random_state=1)
iowa_model.fit(X,y)
```

[64]:
```
DecisionTreeRegressor(random_state=1)
```

[65]:
```
print(X.head())
prediccion = iowa_model.predict(X.head())
print("Predicciones: ",prediccion)
```

```
   LotArea  YearBuilt  1stFlrSF  2ndFlrSF  FullBath  BedroomAbvGr  \
0     8450       2003       856       854         2             3
1     9600       1976      1262         0         2             3
2    11250       2001       920       866         2             3
3     9550       1915       961       756         1             3
4    14260       2000      1145      1053         2             4

   TotRmsAbvGrd
```

```
0             8
1             6
2             6
3             7
4             9
Predicciones:  [208500. 181500. 223500. 140000. 250000.]
```

[66]:
```
print(home_data['SalePrice'].head())
```

```
0    208500
1    181500
2    223500
3    140000
4    250000
Name: SalePrice, dtype: int64
```

Los resultados de la predicción con los valores reales de las viviendas son exactamente los mismos.

2 3.5 Infraajuste y Sobreajuste

En este ejercicio, voy a crear un nuevo modelo de predicción utilizando Decision Tree Regressor entrenandolo y testeandolo con los grupos de datos que el metodo **train_test_split** genere y voy a calcular su precisión predictiva con el metodo **mean_absolute_error**.

Añadiré el parametro **max_leaf_nodes** y compararé su precisión predictiva contra un modelo que no lleva este parametro.

[67]:
```
home_data_train = pd.read_csv('train.csv')
```

[68]:
```
home_data_train.columns
```

[68]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
```

```
      dtype='object')
```

[69]:
```
features = ['LotArea','YearBuilt','1stFlrSF','2ndFlrSF','FullBath',
                    'BedroomAbvGr','TotRmsAbvGrd']
y = home_data_train.SalePrice
X = home_data_train[features]
```

[70]:
```
y.describe()
```

[70]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

[71]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error
```

[72]:
```
train_X,val_X,train_y,val_y = train_test_split(X,y,random_state = 0)
```

[73]:
```
model = DecisionTreeRegressor(random_state=0)
model.fit(train_X,train_y)
preds_val = model.predict(val_X)
mae = mean_absolute_error(val_y,preds_val)
print(mae)
```

```
32410.824657534245
```

[80]:
```
for i in [500,5000,10000,20000]:
  model = DecisionTreeRegressor(max_leaf_nodes = i,random_state=0)
  model.fit(train_X,train_y)
  preds_val = model.predict(val_X)
  mae = mean_absolute_error(val_y,preds_val)
  print(mae)
```

```
32685.401335072846
33404.21643835616
33404.21643835616
33404.21643835616
```