Housing Prices Competition for Kaggle Learn Users (pt. 1)

Pablo de la Asunción Cumbrera Conde

En el siguiente proyecto trataremos de resolver la competición de 'Kaggle' Housing Prices Competition for Kaggle Learn Users sobre la aplicación de Machine Learning para la predicción de precios de viviendas en Ames, Iowa. Procederemos a través de algoritmos relacionados con 'Decission Trees' y 'Random Forest' a través de cursos de formación.

La descripción e introducción de la competición reza así:

- Pídale a un comprador de vivienda que describa la casa de sus sueños, y probablemente no comenzará con la altura del techo del sótano o la proximidad a un ferrocarril que cruce todo el país. No obstante, este conjunto de datos para esta distendida competición demuestra que influye mucho más sobre el precio las negociaciones que el número de dormitorios o una valla blanca.

Objetivos para la competición:

- Con 79 variables explicativas que describen (casi) todos los aspectos de las viviendas residenciales en Ames, Iowa, esta competención desafía al usuario a predecir el precio final de cada casa.

Habilidades a poner en práctica:

- Ingeniería creativa de funciones
- Técnicas de regresión avanzadas como 'Random Forest' y 'Gradient Boosting'.

Contenidos

- Exploración básica del conjunto de datos
- Construcción del modelo 'Decission Tree'
- Validación del modelo 'Decission Tree'
- Construcción del modelo 'Random Forest'

Exploración básica del conjunto de datos

In [4]:
```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
```

In [8]:
```
#Importamos el set de datos
iowa_file_path = 'file:///C:/Users/pablo/OneDrive/Escritorio/Data/Data%20sets/housingprice_train.csv'
home_data = pd.read_csv(iowa_file_path)
home_data.head()
```

Out[8]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

In [9]:
```
home_data.describe()
```

Out[9]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	P
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	...	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	146
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	...	94.244521	46.660274	21.954110	3.409589	15.060959	
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	...	125.338794	66.256028	61.119149	29.317331	55.757415	4
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	...	0.000000	25.000000	0.000000	0.000000	0.000000	
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	...	168.000000	68.000000	0.000000	0.000000	0.000000	
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	...	857.000000	547.000000	552.000000	508.000000	480.000000	73

8 rows × 38 columns

Creamos 'X', que es el objetivo específico de predicción. Contiene los siguientes características, ya que no utilizaremos todas las columnas del set de datos.

In [10]:
```
# Create the list of features below
feature_names = ["LotArea","YearBuilt","1stFlrSF","2ndFlrSF","FullBath","BedroomAbvGr","TotRmsAbvGrd"]

# Select data corresponding to features in feature_names
X = home_data[feature_names]
```

Creamos 'Y', que asociamos con el precio de los inmuebles.

In [11]:
```
y = home_data.SalePrice
```

In [12]:
```
X.describe()
```

Out[12]:

	LotArea	YearBuilt	1stFlrSF	2ndFlrSF	FullBath	BedroomAbvGr	TotRmsAbvGrd
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	10516.828082	1971.267808	1162.626712	346.992466	1.565068	2.866438	6.517808
std	9981.264932	30.202904	386.587738	436.528436	0.550916	0.815778	1.625393
min	1300.000000	1872.000000	334.000000	0.000000	0.000000	0.000000	2.000000
25%	7553.500000	1954.000000	882.000000	0.000000	1.000000	2.000000	5.000000
50%	9478.500000	1973.000000	1087.000000	0.000000	2.000000	3.000000	6.000000
75%	11601.500000	2000.000000	1391.250000	728.000000	2.000000	3.000000	7.000000
max	215245.000000	2010.000000	4692.000000	2065.000000	3.000000	8.000000	14.000000

Construcción del modelo 'Decission Tree'

Comenzamos a construir nuestro modelo. Para su repodrucibilidad, asignamos un valor numérico a la función.

In [14]:
```
iowa_model = DecisionTreeRegressor(random_state=1)
iowa_model.fit (X, y)
```

Out[14]:
```
DecisionTreeRegressor(random_state=1)
```

In [15]:
```
predictions = iowa_model.predict(X)
```

In [16]:
```
y.head()
```

Out[16]:
```
0    208500
1    181500
2    223500
3    140000
4    250000
Name: SalePrice, dtype: int64
```

Nuestras primeras predicciones:

In [22]:
```
print("Primeras predicciones de nuestra muestra:", iowa_model.predict(X.head()))
print('Actuales valores objetivos para esos inmuebles:', y.head().tolist())
```

```
Primeras predicciones de nuestra muestra: [208500. 181500. 223500. 128000. 250000.]
Actuales valores objetivos para esos inmuebles: [208500, 181500, 223500, 140000, 250000]
```

Vamos a probar cuanto de válido es nuestro modelo

In [19]:
```
train_X, val_X, train_y, val_y = train_test_split (X, y,random_state=1)
iowa_model = DecisionTreeRegressor(random_state=1)
iowa_model.fit(train_X,train_y)
```

Out[19]:
```
DecisionTreeRegressor(random_state=1)
```

In [21]:
```
val_predictions = iowa_model.predict(val_X)
```

In [37]:
```
print (val_y.head(10),val_predictions [0:9])
```

```
258     231500
267     179500
288     122000
649      84500
1233    142000
167     325624
926     285000
831     151000
1237    195000
426     275000
Name: SalePrice, dtype: int64 [186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.]
```

Calculamos el Mean Absolute Error (MAE). La diferencia media entre las dos variables (predicha y observada).

In [41]:
```
val_mae = mean_absolute_error(val_predictions, val_y)
print("MAE:", val_mae)
```

```
MAE: 29652.931506849316
```

Desarrollamos una función para facilitar el cálculo del MAE

In [44]:
```
def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=0)
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)
```

Optimización del modelo

Dado que nuestras predicciones tienen aun mucho error, trataremos de mejorar y optimizar nuestro modelo. Primero, situaremos el número de nodos óptimo (tamaño de nuestro árbol).

In [45]:
```
candidate_max_leaf_nodes = [5, 25, 50, 100, 250, 500]
# Write loop to find the ideal tree size from candidate_max_leaf_nodes
for max_leaf_nodes in [5, 25, 50, 100, 250, 500]:
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print('Max leaf nodes: %d  \t\t Mean Absolute Error:  %d' %(max_leaf_nodes, my_mae))
# Store the best value of max_leaf_nodes (it will be either 5, 25, 50, 100, 250 or 500)
best_tree_size = 100
```

```
Max leaf nodes: 5               Mean Absolute Error:  35044
Max leaf nodes: 25              Mean Absolute Error:  29016
Max leaf nodes: 50              Mean Absolute Error:  27405
Max leaf nodes: 100             Mean Absolute Error:  27282
Max leaf nodes: 250             Mean Absolute Error:  27893
Max leaf nodes: 500             Mean Absolute Error:  29454
```

In [46]:
```
final_model = DecisionTreeRegressor(max_leaf_nodes=best_tree_size, random_state=1)
final_model.fit(X, y)
```

Out[46]:
```
DecisionTreeRegressor(max_leaf_nodes=100, random_state=1)
```

Construcción de un modelo 'Random Forest'

Para comprobar si podemos conseguir mejores predicciones a través de otro modelo, construiremos un modelo 'Random Forest'.

In [70]:
```
from sklearn.ensemble import RandomForestRegressor

# Definición del modelo. Ajustamos random_state a 1
rf_model = RandomForestRegressor(random_state=1)

# Ajustamos el modelo
rf_model.fit (train_X,train_y)

#Creamos un set de datos para X a partir de los datos 'test'
iowa_data_path_test= 'file:///C:/Users/pablo/OneDrive/Escritorio/Data/Data%20sets/housingprice_test.csv'
X_test_full = pd.read_csv(iowa_data_path_test, index_col='Id')
X_full= home_data
y = X_full.SalePrice
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = X_full[features].copy()
X_test = X_test_full[features].copy()

# Calculamos el MAE de nuestro modelo Random Forest para su validación.
rf_val_predictions = rf_model.predict(val_X)
rf_val_mae = mean_absolute_error(rf_val_predictions, val_y)

print("Validación MAE para modelo Random Forest: {}".format(rf_val_mae))
```

```
Validación MAE para modelo Random Forest: 21857.15912981083
```

In [77]:
```
X_train.head()
```

Out[77]:

	LotArea	YearBuilt	1stFlrSF	2ndFlrSF	FullBath	BedroomAbvGr	TotRmsAbvGrd
618	11694	2007	1828	0	2	3	9
870	6600	1962	894	0	1	2	5
92	13360	1921	964	0	1	2	5
817	13265	2002	1689	0	2	3	7
302	13704	2001	1541	0	2	3	6

In [71]:
```
# Extraemos set para validación del set de entrenamiento.
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.2,
                                                      random_state=0)
```

Creamos cinco modelos aleatorios de 'Random Forest' y definimos los modelos

In [91]:
```
model_1 = RandomForestRegressor(n_estimators=50, random_state=0)
model_2 = RandomForestRegressor(n_estimators=100, random_state=0)
model_3 = RandomForestRegressor(n_estimators=100, criterion='mae',random_state=0)
model_4 = RandomForestRegressor(n_estimators=200, min_samples_split=25, random_state=0)
model_5 = RandomForestRegressor(n_estimators=100, max_depth=7, random_state=0)

models = [model_1, model_2, model_3, model_4, model_5]
```

Calculamos el MAE de los diferentes modelos creados.

In [92]:
```
def score_model(model, X_t=X_train, X_v=X_valid, y_t=y_train, y_v=y_valid):
    model.fit(X_t, y_t)
    preds = model.predict(X_v)
    return mean_absolute_error(y_v, preds)

for i in range(0, len(models)):
    mae = score_model(models[i])
    print("Model %d MAE: %d" % (i+1, mae))
```

```
Model 1 MAE: 24015
Model 2 MAE: 23740
Model 3 MAE: 24116
Model 4 MAE: 23996
Model 5 MAE: 23706
```

Creamos un modelo acorde al 3º de los anteriores e iteramos hasta encontrar los mejores valores posibles.

In [98]:
```
my_model = RandomForestRegressor(n_estimators=100,criterion='mae', random_state=0, max_depth=13)
mae =score_model(my_model)
print(mae)
```

```
23209.20178082192
```

In [80]:
```
my_model.fit(X, y)
preds_test = my_model.predict(X_test)
print(preds_test)
```

```
[120759.33 157277.5  180615.51 ... 157581.52 130745.   231760.6 ]
```

Guardamos los Resultados y enviamos los resultados.

In [93]:
```
# Save predictions in format used for competition scoring
output = pd.DataFrame({'Id': X_test.index,
                       'SalePrice': preds_test})
output.to_csv('submission.csv', index=False)
```