Asignatura	Datos del alumno	Fecha
Aprendizaje automático	Apellidos: Domínguez Espinoza Nombre: Edgar Uriel	16 de marzo de 2022

Árboles y random forest: House Sales Price Predictions

Librerías a utilizar

Para el tratamiento se han agregado principalmente cinco bibliotecas. En el caso de `sklearn`, se tuvo la necesidad de hacer importaciones parciales para que ciertas funciones y métodos sean detectados correctamente.

[1]:
```
from sklearn import metrics
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor,␣
 ↪RandomForestClassifier
from sklearn.model_selection import train_test_split, KFold,␣
 ↪cross_val_score
from sklearn.tree import DecisionTreeClassifier,␣
 ↪DecisionTreeRegressor, export_graphviz
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import sklearn as sl
```

Carga de datasets

Se usarán dos datasets (Ganesh, 2018) los cuales serán cargados en dos dataframes. Aquel identificado como `test` solo será usado al final para completar el dataset con el modelo.

[2]:
```
df_train = pd.read_csv("./ds/housing_train.csv")
df_test = pd.read_csv("./ds/housing_test.csv")
```

Resumen de datos

Las dimensiones del dataframe, filas y columnas, se obtiene con la propiedad `shape`, los valores de las cabeceras se obtienen con la propiedad `columns.values`.

[3]:
```
df_train.shape
```

[3]:
```
(1460, 81)
```

La función `describe()` devuelve el conteo de campos no nulos, media, desviación estándar y cuantiles para columnas númericas. En las columnas identificadas como objetos (categóricas) devolverá el conteo de campos no nulos, número de valores posibles, el valor más repetido y su frecuencia. Si se desea saber el tipo de datos que tienen las columnas se usa la propiedad `dtypes`.

[4]:
```
df_train.describe().transpose()
```

[4]:
```
                count           mean            std     min          25%␣
    ↪ \
Id             1460.0     730.500000     421.610009     1.0       365.75
MSSubClass     1460.0      56.897260      42.300571    20.0        20.00
LotFrontage    1201.0      70.049958      24.284752    21.0        59.00
LotArea        1460.0   10516.828082    9981.264932  1300.0      7553.50
OverallQual    1460.0       6.099315       1.382997     1.0         5.00
OverallCond    1460.0       5.575342       1.112799     1.0         5.00
YearBuilt      1460.0    1971.267808      30.202904  1872.0      1954.00
YearRemodAdd   1460.0    1984.865753      20.645407  1950.0      1967.00
MasVnrArea     1452.0     103.685262     181.066207     0.0         0.00
BsmtFinSF1     1460.0     443.639726     456.098091     0.0         0.00
BsmtFinSF2     1460.0      46.549315     161.319273     0.0         0.00
BsmtUnfSF      1460.0     567.240411     441.866955     0.0       223.00
TotalBsmtSF    1460.0    1057.429452     438.705324     0.0       795.75
1stFlrSF       1460.0    1162.626712     386.587738   334.0       882.00
2ndFlrSF       1460.0     346.992466     436.528436     0.0         0.00
LowQualFinSF   1460.0       5.844521      48.623081     0.0         0.00
GrLivArea      1460.0    1515.463699     525.480383   334.0      1129.50
BsmtFullBath   1460.0       0.425342       0.518911     0.0         0.00
BsmtHalfBath   1460.0       0.057534       0.238753     0.0         0.00
FullBath       1460.0       1.565068       0.550916     0.0         1.00
HalfBath       1460.0       0.382877       0.502885     0.0         0.00
BedroomAbvGr   1460.0       2.866438       0.815778     0.0         2.00
KitchenAbvGr   1460.0       1.046575       0.220338     0.0         1.00
TotRmsAbvGrd   1460.0       6.517808       1.625393     2.0         5.00
Fireplaces     1460.0       0.613014       0.644666     0.0         0.00
GarageYrBlt    1379.0    1978.506164      24.689725  1900.0      1961.00
GarageCars     1460.0       1.767123       0.747315     0.0         1.00
GarageArea     1460.0     472.980137     213.804841     0.0       334.50
WoodDeckSF     1460.0      94.244521     125.338794     0.0         0.00
OpenPorchSF    1460.0      46.660274      66.256028     0.0         0.00
EnclosedPorch  1460.0      21.954110      61.119149     0.0         0.00
3SsnPorch      1460.0       3.409589      29.317331     0.0         0.00
ScreenPorch    1460.0      15.060959      55.757415     0.0         0.00
PoolArea       1460.0       2.758904      40.177307     0.0         0.00
MiscVal        1460.0      43.489041     496.123024     0.0         0.00
MoSold         1460.0       6.321918       2.703626     1.0         5.00
YrSold         1460.0    2007.815753       1.328095  2006.0      2007.00
SalePrice      1460.0  180921.195890   79442.502883  34900.0    129975.00

                   50%          75%       max
Id               730.5      1095.25    1460.0
MSSubClass        50.0        70.00     190.0
LotFrontage       69.0        80.00     313.0
LotArea         9478.5     11601.50  215245.0
OverallQual        6.0         7.00      10.0
```

OverallCond	5.0	6.00	9.0
YearBuilt	1973.0	2000.00	2010.0
YearRemodAdd	1994.0	2004.00	2010.0
MasVnrArea	0.0	166.00	1600.0
BsmtFinSF1	383.5	712.25	5644.0
BsmtFinSF2	0.0	0.00	1474.0
BsmtUnfSF	477.5	808.00	2336.0
TotalBsmtSF	991.5	1298.25	6110.0
1stFlrSF	1087.0	1391.25	4692.0
2ndFlrSF	0.0	728.00	2065.0
LowQualFinSF	0.0	0.00	572.0
GrLivArea	1464.0	1776.75	5642.0
BsmtFullBath	0.0	1.00	3.0
BsmtHalfBath	0.0	0.00	2.0
FullBath	2.0	2.00	3.0
HalfBath	0.0	1.00	2.0
BedroomAbvGr	3.0	3.00	8.0
KitchenAbvGr	1.0	1.00	3.0
TotRmsAbvGrd	6.0	7.00	14.0
Fireplaces	1.0	1.00	3.0
GarageYrBlt	1980.0	2002.00	2010.0
GarageCars	2.0	2.00	4.0
GarageArea	480.0	576.00	1418.0
WoodDeckSF	0.0	168.00	857.0
OpenPorchSF	25.0	68.00	547.0
EnclosedPorch	0.0	0.00	552.0
3SsnPorch	0.0	0.00	508.0
ScreenPorch	0.0	0.00	480.0
PoolArea	0.0	0.00	738.0
MiscVal	0.0	0.00	15500.0
MoSold	6.0	8.00	12.0
YrSold	2008.0	2009.00	2010.0
SalePrice	163000.0	214000.00	755000.0

[5]:
```
df_train.describe(include='object').transpose()
```

[5]:

	count	unique	top	freq
MSZoning	1460	5	RL	1151
Street	1460	2	Pave	1454
Alley	91	2	Grvl	50
LotShape	1460	4	Reg	925
LandContour	1460	4	Lvl	1311
Utilities	1460	2	AllPub	1459
LotConfig	1460	5	Inside	1052
LandSlope	1460	3	Gtl	1382
Neighborhood	1460	25	NAmes	225
Condition1	1460	9	Norm	1260
Condition2	1460	8	Norm	1445

```
BldgType        1460      5     1Fam  1220
HouseStyle      1460      8   1Story   726
RoofStyle       1460      6    Gable  1141
RoofMatl        1460      8  CompShg  1434
Exterior1st     1460     15  VinylSd   515
Exterior2nd     1460     16  VinylSd   504
MasVnrType      1452      4     None   864
ExterQual       1460      4       TA   906
ExterCond       1460      5       TA  1282
Foundation      1460      6    PConc   647
BsmtQual        1423      4       TA   649
BsmtCond        1423      4       TA  1311
BsmtExposure    1422      4       No   953
BsmtFinType1    1423      6      Unf   430
BsmtFinType2    1422      6      Unf  1256
Heating         1460      6     GasA  1428
HeatingQC       1460      5       Ex   741
CentralAir      1460      2        Y  1365
Electrical      1459      5    SBrkr  1334
KitchenQual     1460      4       TA   735
Functional      1460      7      Typ  1360
FireplaceQu      770      5       Gd   380
GarageType      1379      6   Attchd   870
GarageFinish    1379      3      Unf   605
GarageQual      1379      5       TA  1311
GarageCond      1379      5       TA  1326
PavedDrive      1460      3        Y  1340
PoolQC             7      3       Gd     3
Fence            281      4    MnPrv   157
MiscFeature       54      4     Shed    49
SaleType        1460      9       WD  1267
SaleCondition   1460      6   Normal  1198
```

[6]:
```
df_train.dtypes
```

[6]:
```
Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
                  ...
MoSold            int64
YrSold            int64
SaleType         object
SaleCondition    object
SalePrice         int64
```

Para obtener detalles por columna podemos usar `df_train['Nombre de`

`columna'].describe()`, también es posible obtener por columna los posibles valores posibles y sus respectivas frecuencias, como en el siguiente ejemplo.

[7]:
```
df_train["SaleType"].value_counts()
```

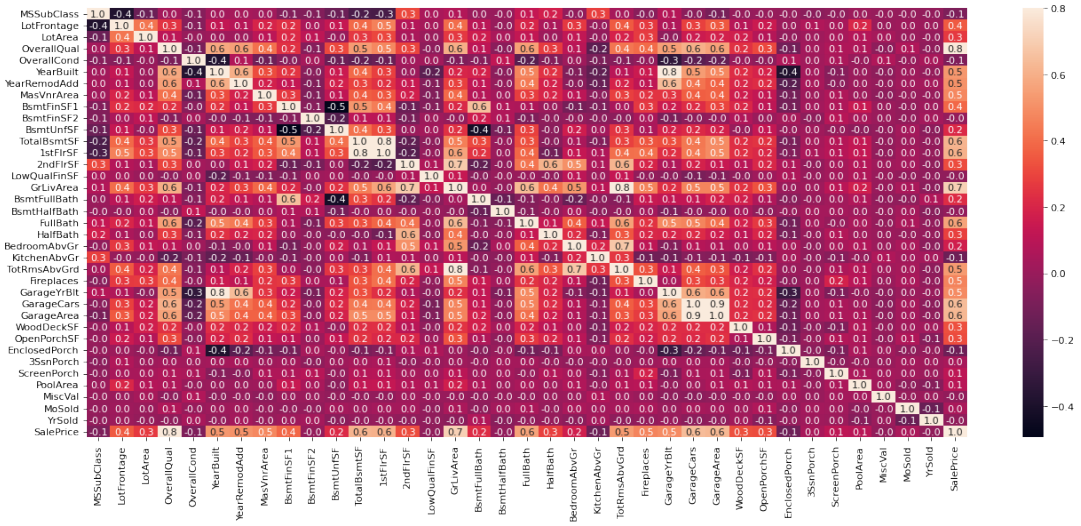
[7]:
```
WD        1267
New        122
COD         43
ConLD        9
ConLI        5
ConLw        5
CWD          4
Oth          3
Con          2
```

Matriz de correlación

La matriz de correlación indicará que tan fuerte o débil es la relación entre dos variables. Puede leerse por columnas o por filas. En la siguiente imagen se eliminó la columna `Id`, porque no será relevante para el análisis. La claridad de la celda es directamente proporcional a una mayor correlación.

[8]:
```
df_train = df_train.drop(columns = ['Id'])
plt.figure(figsize=(20,8),dpi=80)
corrmat = df_train.corr()
sns.heatmap(corrmat, vmax=.8, fmt='.1f', annot=True)
```

[8]:



De esta forma podemos saber que variables están más relacionadas con otras. en el caso de la variable `SalePrice` las diez variables más útiles serán aquellas con mayor índice de correlación, mismas que se usarán posteriormente para las predicciones.

[9]:
```
df_train.corr()['SalePrice'].sort_values(ascending=False)[1:11]
```

[9]:
```
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
YearBuilt       0.522897
YearRemodAdd    0.507101
```

Valores perdidos

Trabajar con los valores perdidos requiere primero su ubicación, posteriormente se seleccionará que debe ser borrado y luego que debe ser sustituido con un nuevo valor, por supuesto habrá que decidir cual será dicho valor nuevo.

Eliminar campos

Para ubicar si una celda tiene un valor vacío se usa la función `isnull()`, si se prefiere lógica inversa se usa `notnull`. Es posible obtener un vector de estos valores con la propiedad `values`, transformarlo a un array con la función `ravel()` y sumar los valores verdaderos con la función `sum()`. También es posible obtener una lista ordenada de las columnas con más valores vacíos.

[10]:
```
df_train.isnull().sum().sort_values(ascending=False)[0:19]
```

[10]:
```
PoolQC          1453
MiscFeature     1406
Alley           1369
Fence           1179
FireplaceQu      690
LotFrontage      259
GarageYrBlt       81
GarageCond        81
GarageType        81
GarageFinish      81
GarageQual        81
BsmtExposure      38
BsmtFinType2      38
BsmtCond          37
BsmtQual          37
BsmtFinType1      37
MasVnrArea         8
MasVnrType         8
Electrical         1
```

En el ejemplo de arriba, el valor es el número de valores vacíos, si usamos la función `notnull()`

sería el número de valores no vacíos, la suma de ambos debe ser el número total de filas obtenido anteriormente.

Hay dos razones para la falta de valores en los datasets: 1) Recolección de datos, no se consiguieron los datos y; 2) Extracción de datos, los datos están en la DB original pero no se extrajeron correctamente al dataset.

Se deben evitar datos vacíos para no tener problemas de manejo de información. Se tienen dos opciones: 1) Borrar las filas donde falten valores en alguna de las columnas y; Borrar las columnas donde no se tenga suficiente información.

En este ejercicio es posible observar que las columnas `MiscFeature, Fence, PoolQC, FirePlaceQu y Alley` tienen muy pocos valores proporcionados (menos del 55 por ciento)y no vale la pena conservarlas. Otro criterio para asegurar un buen curso de acción es revisar las correlaciones con la columna `SalePrice`.

Como el razonamiento es el correcto se procede al borrado de columnas.

[11]:
```
def toDel(df):
    for col in df.columns.values:
        nv = pd.isnull(df[col]).values.ravel().sum()
        if nv > df.shape[0] * 0.45:
            print("Deleting: "+col)
            del df[col]
    return df
df_train = toDel(df_train)
```

Llenar campos

Es necesario detectar nuevamente que columnas tienen valores vacíos. Esta vez se reemplazarán esos valores. Hay valores númericos y categóricos vacíos; los numéricos serán reemplazados por el promedio original de la columna, los categóricos serán remplazados por el valor no nulo más cercano puede ser el valor que va antes (`ffill`) o el que va después (`bfill`), en este análisis será el segundo.

Es necesario señalar que el procedimiento más preciso para las columnas categóricas sería colocar el valor de mayor frecuencia relacionado con el valor de la columna objetivo, por ejemplo: Si la columna `Y` del dataframe es la variable dependiente y `X` es una columna categórica con valores perdidos; dichos valores se llenarán por aquel de mayor frecuencia en `X` tomando en cuenta solo aquellos con los que coincidan en `Y`. Más adelante se retomará la justificación de porque no se ha hecho de esta forma.

[12]:
```
def DetectNull(df):
    candidates = []
    for col in df.columns.values:
        nv = pd.isnull(df[col]).values.ravel().sum()
        if nv > 0:
            candidates.append((col, df[col].dtype, nv))
    return candidates
```

[13]:
```
def FillNull(df, list):
    for col in list:
        if col[1] == 'float64':
            df[col[0]] = df[col[0]].fillna(df[col[0]].mean())
        else:
            df[col[0]] = df[col[0]].fillna(method="bfill")
    return df
```

[14]:
```
df_test = FillNull(df_test, DetectNull(df_test))
df_train = FillNull(df_train, DetectNull(df_train))
df_train.isnull().sum().sort_values(ascending=False)
```

[14]:
```
MSSubClass      0
GarageYrBlt     0
Fireplaces      0
Functional      0
TotRmsAbvGrd    0
               ..
MasVnrArea      0
MasVnrType      0
Exterior2nd     0
Exterior1st     0
SalePrice       0
```

Problema de regresión

Árboles de decisión

Primero serán creados los conjuntos de prueba y entrenamiento. Serán usados para el modelo solo aquellos campos que tengan un alto índice de correlación en la matriz de correlaciones mostrada anteriormente.

[15]:
```
train, test = train_test_split(df_train, test_size=0.2)
predictors = ['OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',␣
 ↪'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',␣
 ↪'YearRemodAdd']
target = ['SalePrice']
```

A continuación se entrena el árbol de regresión y se ingresan los datos para probar la predicción del mismo.

[16]:
```
dtr = DecisionTreeRegressor(max_depth=15, min_samples_split=20,␣
 ↪random_state=99)
dtr.fit(train[predictors], train[target])
prediction = dtr.predict(test[predictors])
```

Ahora se muestran los resultados: una comparación entre los datos originales y las predicciones, además, el árbol obtenido, mismo que fue guardado en la carpeta `out` del proyecto en formato graphviz y como imagen. (Galarnyk, 2021)

[17]:
```
test['preds'] = prediction
test[['SalePrice','preds']]
```

[17]:

	SalePrice	preds
520	106250	62509.090909
562	108000	130822.222222
265	175500	176128.571429
1228	367294	337085.071429
280	228500	199144.444444
…	…	…
670	173500	176352.631579
102	118964	136842.947368
1130	135000	145525.777778
1016	203000	263613.333333
12	144000	119778.571429

En la tabla se observa que muchos valores de predicción están repetidos, esto se debe a que entran en la misma lógica de predicción. Debe recordarse que el árbol funciona decidiendo con valores preestablecidos.

[18]:
```
with open('out/dtr.dot','w') as dotfile:
    export_graphviz(dtr, out_file=dotfile, feature_names=predictors)
    dotfile.close()
```

Para validar el modelo se usará un método de validación cruzada, un método estadístico para evaluar y comparar algoritmos de aprendizaje dividiendo datos en dos segmentos: entrenamiento y prueba. Típicamente, ambos conjuntos deben cruzarse en rondas sucesivas de modo que cada punto de datos tenga la posibilidad de ser validado. La forma básica es la validación cruzada k-fold. (Refaeilzadeh et al., 2018)

[20]:
```
dtr = DecisionTreeRegressor(max_depth=15, min_samples_split=20,␣
 ↪random_state=99)
dtr.fit(train[predictors], train[target])
cv = KFold(n_splits = 20, shuffle = True, random_state = 1)
score = np.mean(cross_val_score(dtr, train[predictors], train[target],␣
 ↪scoring = "neg_mean_squared_error", cv = cv, n_jobs = 1))
score
```

[20]:
```
-1695546565.4410434
```

El modelo es muy deficiente según el error cuadrático medio de pérdida. Este error es muy grande, debería ser cercano a cero. Sería mejor probar un modelo lineal, los árboles de regresión son útiles si es necesario estimar un modelo no lineal. Para confirmar se realizará este modelo bajo diferentes profundidades del árbol, de esta forma se podría encontrar un mejor conjunto de parámetros, no sucede en este caso.

[21]:
```
for i in range(1,21):
    dtr = DecisionTreeRegressor(max_depth=i, min_samples_split=20,␣
 ↪min_samples_leaf=5,random_state=99)
    dtr.fit(train[predictors], train[target])
    cv = KFold(n_splits = 20, shuffle = True, random_state = 1)
    score = np.mean(cross_val_score(dtr, train[predictors],␣
 ↪train[target], scoring = "neg_mean_squared_error", cv = cv, n_jobs =␣
 ↪1))
    print("Score para i=",i,": ",score)
```

```
      …                       …
Score para i= 10 :  -1551932650.1670842
Score para i= 11 :  -1551095985.7058103
Score para i= 12 :  -1551446872.7832217
Score para i= 13 :  -1551400950.2026584
Score para i= 14 :  -1549894543.9422781
Score para i= 15 :  -1550710493.3735516
Score para i= 16 :  -1552723780.9771752
Score para i= 17 :  -1552723780.9771752
```

Random Forest

Al igual que en la sección anterior, se entrena el modelo con los mismos conjuntos definidos anteriormente y se hace una predicción.

[22]:
```
rfr = RandomForestRegressor(n_jobs = 1, oob_score=True,␣
 ↪n_estimators=10000)
rfr.fit(train[predictors], train[target].values.ravel())
prediction = rfr.predict(test[predictors])
test['preds'] = prediction
test[['SalePrice','preds']]
```

[22]:
```
      SalePrice          preds
520      106250   96389.837800
562      108000  107334.079900
265      175500  178429.007700
1228     367294  332232.461100
280      228500  211677.532000
...         ...            ...
670      173500  176562.496908
102      118964  129663.313500
1130     135000  140604.314200
1016     203000  230900.547700
12       144000  113294.444000
```

Como puede verse, un bosque de diez mil árboles las estimaciones los valores se acercan notablemente. Esto puede confirmarse con la puntuación propia del bosque, la cual funciona como el coeficiente de determinación de un modelo de regresión.

[23]:
```
rfr.oob_score_
```

[23]:
```
0.8086789376427154
```

La conclusión es que sería mejor usar un modelo de regresión que un modelo de decisión porque pese a la mejora sustancial respecto al árbol anterior, el bosque no alcanza un 0.9 en la puntuación, condición que se le exigiría a un modelo lineal. Debido a que este es el mejor modelo obtenido, lo usaremos para `df_test`.

Problema de clasificación

Creación de categorías de SalesPrice

Ahora se procederá a crear categorías con la columna `SalePrice`. Para ello se ha escrito una función y una nueva columna dentro del dataframe.

[25]:
```
def SalePriceGroupValue(x):
    if x >= 500001:
        return 'G3'
    elif x <= 100000:
        return 'G1'
    return 'G2'
df_train["SalePriceGroup"] = df_train["SalePrice"].
 ↪apply(SalePriceGroupValue)
df_train["SalePriceGroup"].value_counts()
```

[25]:
```
G2    1328
G1     123
G3       9
```

Aquí es posible observar que la gran mayoria de los datos se encuentran en la categoría `G2`. Esto confirma que la opción antes seleccionada para llenar datos perdidos es buena debido a que binda una posibilidad de preservar datos las otras categorías.

Asignatura	Datos del alumno	Fecha
Aprendizaje automático	Apellidos: Domínguez Espinoza Nombre: Edgar Uriel	16 de marzo de 2022

Árboles de decisión

Se repetirá el procedimiento visto anteriormente, la diferencia es que ahora usará `DecisionTreeClassifier`. (Navlani, 2018)

[26]:
```
train, test = train_test_split(df_train, test_size=0.2)
```

[27]:
```
colnames = df_train.columns.values.tolist()
predictors = ['OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',␣
 ↪'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',␣
 ↪'YearRemodAdd']
target = colnames[75]
dtc = DecisionTreeClassifier(criterion="entropy", max_depth=3,␣
 ↪min_samples_split=20, random_state=99)
dtc.fit(train[predictors], train[target])
prediction = dtc.predict(test[predictors])
```

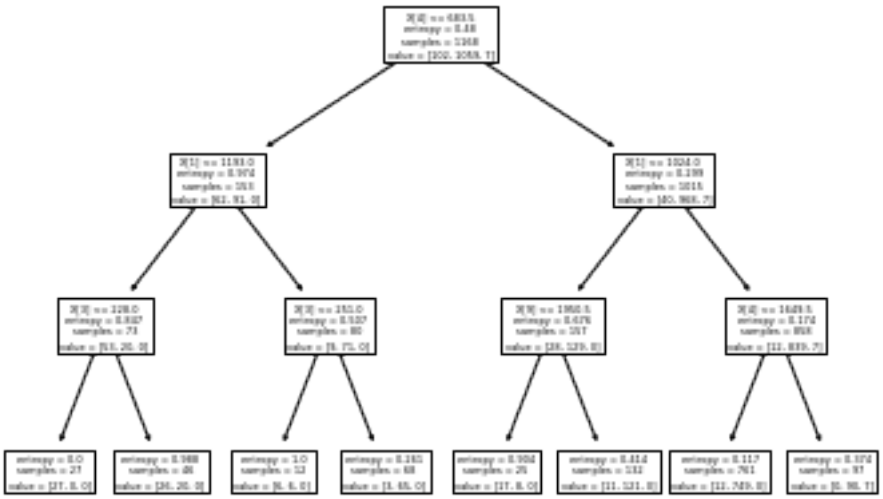
Se ha creado una tabla cruzada que logra visualizar los resultados, además es posible usar metricas simples para verificar la exactitud del árbol.

[28]:
```
pd.crosstab(test[target], prediction, colnames=["Predictions"],␣
 ↪rownames=["Real"])
```

[28]:
```
Predictions  G1   G2
Real
G1           10   11
G2           12  257
G3            0    2
```

[29]:
```
print("Accuracy: ", metrics.accuracy_score(prediction, test[target]))
```

```
Accuracy:  0.9143835616438356
```



Al usar nuevamente validación cruzada se observa que una buena clasificación esta entre `i=3` e `i=6`, lo que significa que si se deja crecer el árbol desde el nodo raíz con estas profundidades es posible obtener clasificaciones óptimas. También podemos ver que las variables de mayor importancia clasificatoria son `TotalBsmtSF` y `GrLivArea`.

[32]:
```
for i in range(1,10):
    dtc = DecisionTreeClassifier(criterion="entropy", max_depth=i,␣
 ↪min_samples_split=20, random_state=99)
    dtc.fit(train[predictors], train[target])
    cv = KFold(n_splits = 20, shuffle = True, random_state = 1)
    score = np.mean(cross_val_score(dtc, train[predictors],␣
 ↪train[target], scoring = "accuracy", cv = cv, n_jobs = 1))
    print("Score para i=",i,": ",score)
    print("Importancia de variables: \n\t",dtc.feature_importances_)
```

```
Score para i= 1 :  0.9066481589713618
Importancia de variables:
         [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
Score para i= 2 :  0.9323641145528929
Importancia de variables:
         [0.         0.46494326 0.         0.         0.53505674 0.
 0.         0.         0.         0.         ]
Score para i= 3 :  0.9332261835184102
Importancia de variables:
         [0.         0.33238498 0.         0.09651566 0.46853394 0.
 0.         0.         0.         0.10256542]
Score para i= 4 :  0.9366306253652835
Importancia de variables:
         [0.         0.31621815 0.         0.07775052 0.41561868 0.
 ↪02302104
 0.         0.         0.06193988 0.10545173]
Score para i= 5 :  0.9391729982466395
Importancia de variables:
         [0.02617778 0.31151851 0.         0.08319446 0.38373239 0.
 ↪02125486
 0.         0.         0.03762721 0.13649479]
Score para i= 6 :  0.939187609585038
Importancia de variables:
         [0.02517877 0.31133357 0.         0.08882965 0.37821928 0.
 ↪02044373
 0.         0.         0.03619127 0.13980373]
Score para i= 7 :  0.9383255406195208
Importancia de variables:
         [0.03296921 0.30531763 0.01780554 0.07541623 0.35646125 0.
 ↪01926765
 0.         0.         0.06100131 0.13176117]
```

```
Score para i= 8 :  0.9340590298071303
Importancia de variables:
         [0.03224009 0.28731886 0.01741177 0.07374839 0.35982465 0.
 ↪02625347
 0.         0.         0.06609772 0.13710506]
Score para i= 9 :  0.9331969608416131
Importancia de variables:
         [0.03174187 0.29167446 0.0171427  0.07260873 0.35426417 0.
 ↪02584777
 0.         0.         0.07173396 0.13498634]
```

Random Forest

En la implementación de este bosque se usa el mismo procedimiento visto anteriormente, es importante poner atención en los cambios de los argumentos de cada árbol, cada implementación dependerá del problema.

[33]:
```
rfc = RandomForestClassifier(n_jobs = 1, oob_score=True,␣
 ↪n_estimators=10000)
rfc.fit(train[predictors], train[target])
prediction = rfc.predict(test[predictors])
test['preds'] = prediction
test[['SalePriceGroup','preds']]
```

[33]:
```
     SalePriceGroup preds
1378             G1    G1
218              G2    G2
1328             G2    G2
194              G2    G2
919              G2    G2
…               …   …
415              G2    G2
1227             G2    G2
928              G2    G2
1354             G2    G2
362              G2    G2
```

En esta ocasión se ha aumentado la exactitud del árbol, es posible decir que se ha creado un modelo confible. Debido a que este es el mejor modelo obtenido, lo usaremos para `df_test`.

[34]:
```
rfc.oob_score_
```

[34]:
```
0.9417808219178082
```

Conclusión

En esta actividad se retomó el USA Housing Dataset, el cual presenta datos sobre la venta de casas en Estados Unidos. Se describieron los datos y se creó una matriz de corelaciones para obtener aquellos más importantes para el análisis. Como segundo paso se realizó una limpieza

de datos, se eliminaron columnas innecesarias y llenaron valores perdidos bajo criterios claros. Una vez realizadas dichas operaciones fue posible entrenar árboles de desición y random forest.

El dataset no tiene una descripción clara de los datos proporcionados, aunque es posible analizarlos e inferir algunos de los significados, una de los requisitos más importantes para el análisis siempre será una descripción inicial clara de los mismos, incluso la interpretación de los resultados depende de ello.

Para entrenar los modelos se usaron variables númericas, con esto se obtuvieron resultados positivos en lo general. Si se hubieran usado variables categóricas habría la necesidad de procesarlas y crear varibles separadas (variables dummy, con la función `pd.get_dummies`). Es importante decir que el árbol de regresión obtuvo un mal modelo, sin embargo, el bosque de regresión presentó resultados mucho mejores, aunque sería interesante compararlo con un modelo lineal en un trabajo futuro. Si el problema se convierte en categórico los resultados mejoran notablemente, la principal razón es que las hojas de los árboles siempre corresponden a una categoría, incluso en regresión, por lo tanto, el árbol de regresión será una opción si otros modelos de regresión no tuvieron resultados satisfactorios.

Otro punto a mencionar es que los modelos probados son ajustables mediante muchos parámetros, su ajuste dependerá del problema que se enfrenta y hasta cierto punto es un proceso de ajuste a prueba y error. Además el proceso de validación cruzada es de gran ayuda para evitar el crecimiento excesivo de los árboles.

Finalmente, decir que algunos de los resultados pueden inspeccionarse a mayor detalle en la carpeta `out` anexa a este documento, igualmente, pequeñas partes de código y sus respuestas someras han sido eliminadas de este reporte, son visibles en los archivos de código adjuntos a este documento o en línea, en el repositorio del proyecto.

Referencias

Galarnyk, M. (2021). Visualizing decision trees with python (scikit-learn, graphviz, matplotlib). *Towards Data Science*.

Ganesh (2018). Usa housing dataset. *USA Housing dataset*.

Navlani, A. (2018). Understanding random forests classifiers in python tutorial.

Refaeilzadeh, P., Tang, L., and Liu, H. (2018). *Cross-Validation*, page 677–684. Springer New York.