Árboles de decisión

EQUIPO 5

Integrantes:
Martínez Loera Mario Vladimir 1688132
Quiroz García Víctor Antonio 1741667
Rodríguez Charles Karla Alejandra 1801994

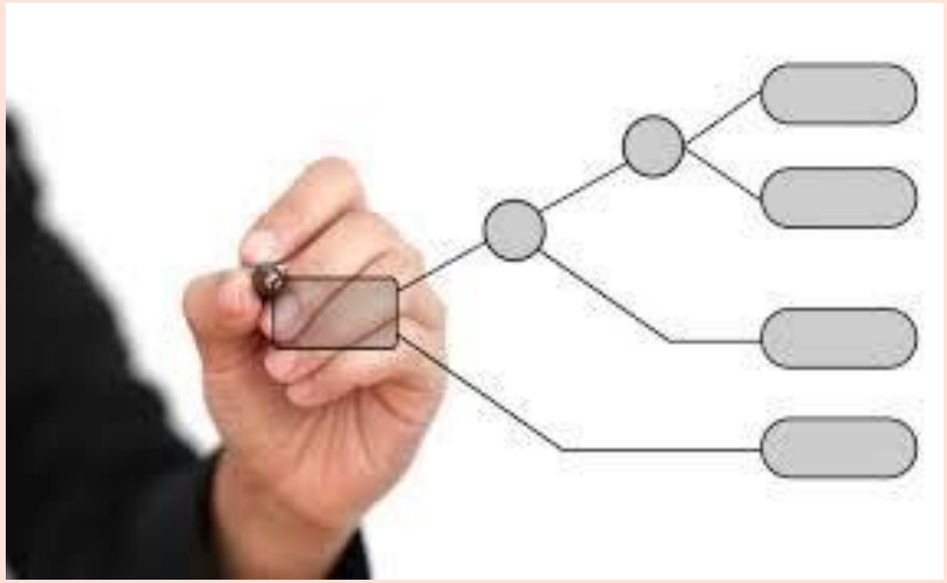
Árboles de decisión

Los árboles de decisión son modelos predictivos formados por reglas binarias (si/no) con las que se consigue repartir las observaciones en función de sus atributos y predecir así el valor de la variable respuesta.

Son un tipo de algoritmo que clasifica la información de forma que, como resultado, se genere un modelo en forma de árbol. Se trata de un modelo esquematizado de la información que representa las diferentes alternativas junto con los posibles resultados para cada alternativa elegida.

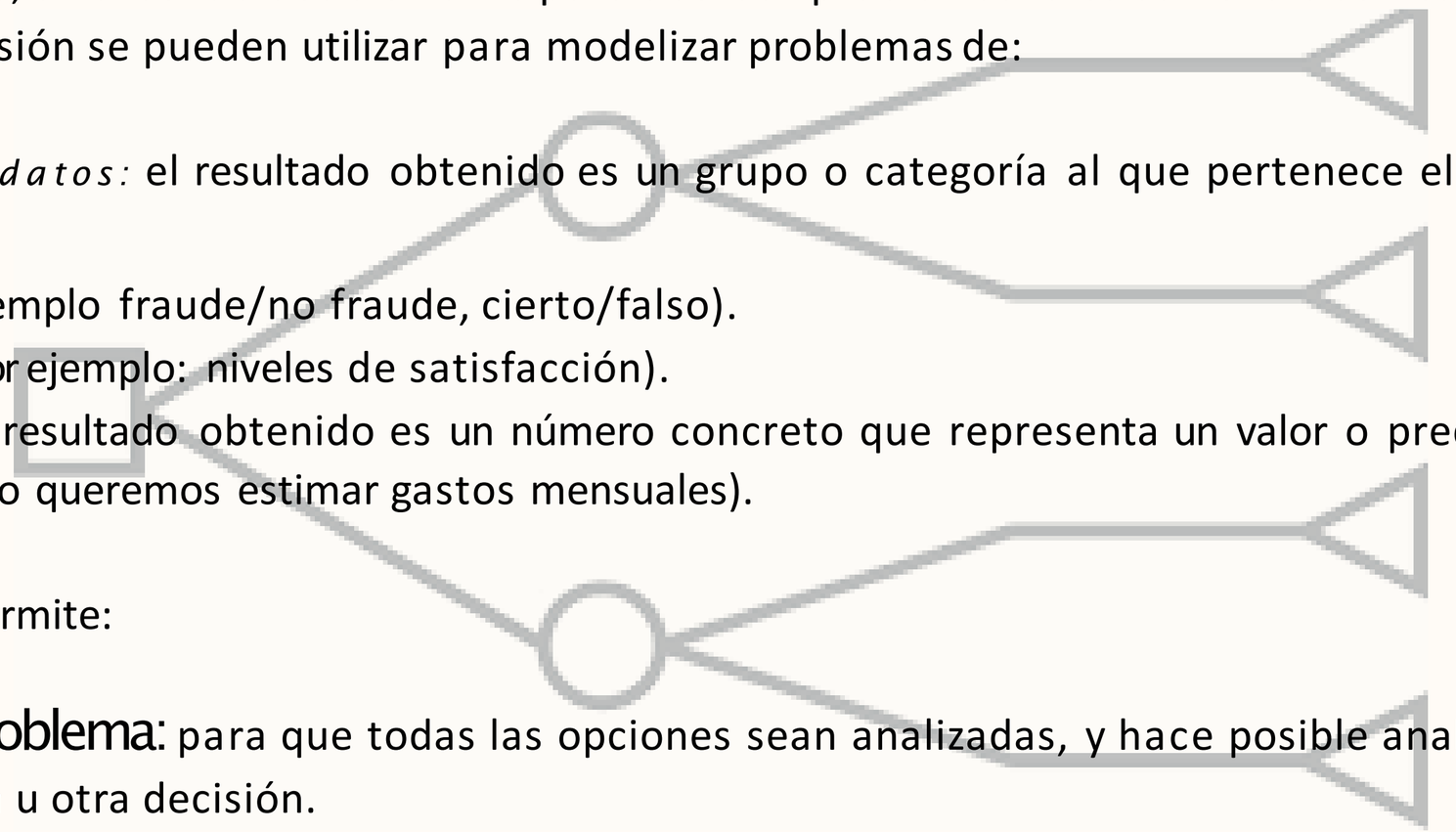
Un árbol de decisión indica las acciones a realizar (=valor que toma una variable dependiente o de respuesta: la que queremos predecir) en función del valor de una o varias variables explicativas o independientes, realiza construcciones lógicas que establecen que establecen reglas que nos permitan clasificar las observaciones en función de la variable de respuesta y sus relaciones con las variables independientes.

El objetivo es siempre llegar a una conclusión (= decisión elegida).



Usos y aplicaciones de los árboles de decisiones.

En minería de datos, un árbol de decisión sirve para abordar problemas tales como la clasificación, la predicción
Los árboles de decisión se pueden utilizar para modelizar problemas de:



- *Clasificación de datos:* el resultado obtenido es un grupo o categoría al que pertenece el elemento a clasificar, la cual puede ser :
  - Binario:(por ejemplo fraude/no fraude, cierto/falso).
  - Multiclase:(por ejemplo: niveles de satisfacción).
  - Regresión: el resultado obtenido es un número concreto que representa un valor o predicción a futuros. (por ejemplo, cuando queremos estimar gastos mensuales).

Esta técnica nos permite:

1. Plantear el problema: para que todas las opciones sean analizadas, y hace posible analizar las consecuencias de adoptar una u otra decisión.
2. Cuantificar su coste: las probabilidades de ocurrencia de cada decisión.

Ventajas

1. Los árboles son fáciles de interpretar aun cuando las relaciones entre predictores son complejas.
2. Los modelos basados en un solo árbol (no es el caso de random forest, boosting) se pueden representar gráficamente aun cuando el número de predictores es mayor de 3.
3. Los árboles pueden, en teoría, manejar tanto predictores numéricos como categóricos sin tener que crear variables dummy o one-hot-encoding. En la práctica, esto depende de la implementación del algoritmo que tenga cada librería.
4. Al tratarse de métodos no paramétricos, no es necesario que se cumpla ningún tipo de distribución específica.
5. No se ven muy influenciados por outliers.
6. Son muy útiles en la exploración de datos, permiten identificar de forma rápida y eficiente las variables (predictores) más importantes.
7. Son capaces de seleccionar predictores de forma automática.
8. Pueden aplicarse a problemas de regresión y clasificación.

Desventajas

1. La capacidad predictiva de los modelos basados en un único árbol es bastante inferior a la conseguida con otros modelos. Esto es debido a su tendencia al overfitting y alta varianza. Sin embargo, existen técnicas más complejas que, haciendo uso de la combinación de múltiples árboles
2. Son sensibles a datos de entrenamiento desbalanceados (una de las clases domina sobre las demás).
3. Cuando tratan con predictores continuos, pierden parte de su información al categorizarlos en el momento de la división de los nodos.
4. No son capaces de extrapolar fuera del rango de los predictores observado en los datos de entrenamiento.

Tipo y composición

Tipo.

Existe 2 tipos de árboles de predicción (en base al tipo de variable que queremos predecir):

·Si queremos predecir variables categóricas (clases): se emplean los árboles de clasificación

·Si queremos predecir variables continuas (valor real): de emplean los árboles de regresión

Composición:

Un árbol de decisión está formado por 2 elementos

1. Un conjunto de nodos, que puede ser de 2 tipos:
   - **Nodos de decisión (o nodos internos):** están asociados a uno de los atributos y tienen 2 o más ramas que salen de ellos, cada uno de ellas representando los posibles valores que puede tomar el atributo asociado.
   - **Nodos de respuesta (u hojas):** está asociado a la clasificación que se quiere proporcionar, y nos devuelve la decisión del árbol con respecto al ejemplo de entrada.
2. Una serie de ramas que se bifurcan en función de los valores tomados por los atributos en base a la pregunta del nodo de decisión.

Pasos para diseñar el diagrama del árbol de decisiones.

A la hora de diseñar el árbol, se deben seguir una serie de pasos:

1. Definición del problema
2. Dibujo del árbol
3. Asignación de las probabilidades a los eventos
4. Estimación de los resultados para las combinaciones de las diferentes alternativas posibles
5. Elección de la solución más óptima reflejada en forma de ruta

Existen varios tipos de criterios a la hora de la toma de decisiones bajo incertidumbre:

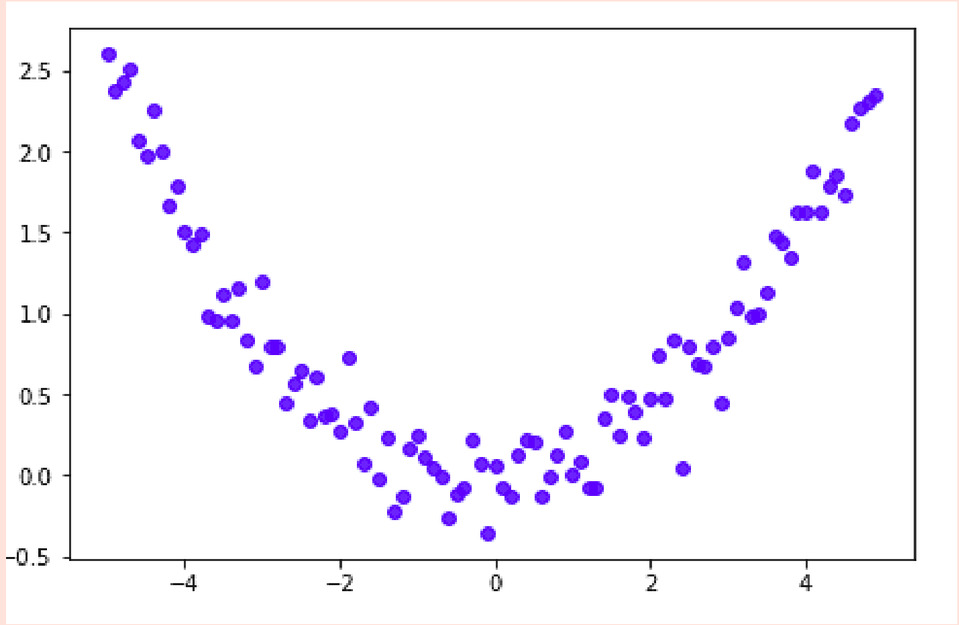
- ·Criterio MAXIMAX o criterio optimista: se opta por la estrategia que maximice el mejor de los resultados posibles.
- ·Criterio MAXIMIN o criterio pesimista: se opta por la estrategia que maximice el peor de los resultados posibles.
- ·Criterio de frustración mínima: se opta por ordenar las estrategias y diferenciar entre el resultado obtenido y el mayor posible para cada situación, eligiendo la estrategia que minimice el resultado.

Árboles de decisión en Python

EQUIPO 5

Árboles de decisión en Python

La principal implementación de árboles de decisión en Python está disponible en la librería scikit-learn a través de las clases DecisionTreeClassifier y DecisionTreeRegressor. Una característica importante para aquellos que han utilizado otras implementaciones es que, en scikit-learn, es necesario convertir las variables categóricas en variables dummy (one-hot-encoding).

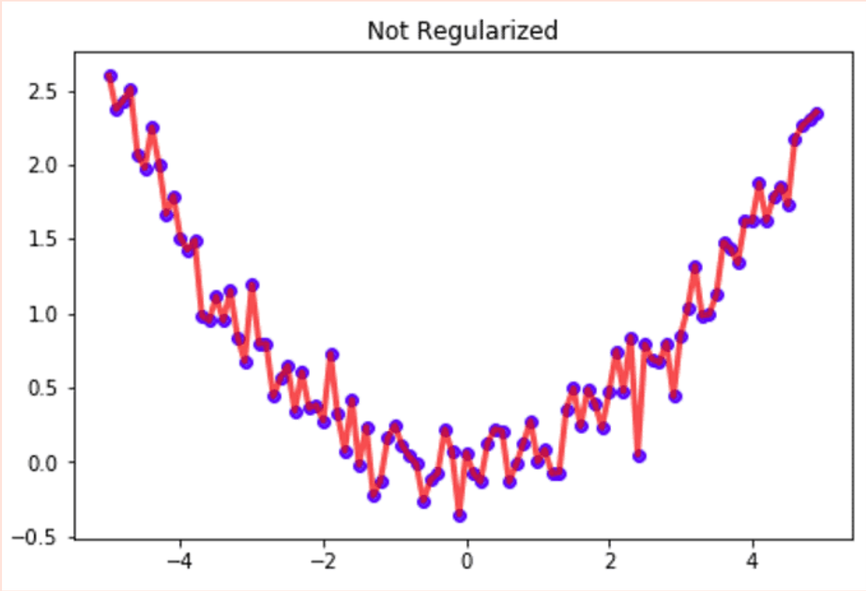


Árboles de regresión

Los árboles de regresión son el subtipo de árboles de predicción que se aplica cuando la variable respuesta es continua. En términos generales, en el entrenamiento de un árbol de regresión, las observaciones se van distribuyendo por bifurcaciones (nodos) generando la estructura del árbol hasta alcanzar un nodo terminal. Cuando se quiere predecir una nueva observación, se recorre el árbol acorde al valor de sus predictores hasta alcanzar uno de los nodos terminales. La predicción del árbol es la media de la variable respuesta de las observaciones de entrenamiento que están en ese mismo nodo terminal.

Predicción del árbol

Tras la creación de un árbol, las observaciones de entrenamiento quedan agrupadas en los nodos terminales. Para predecir una nueva observación, se recorre el árbol en función de los valores que tienen sus predictores hasta llegar a uno de los nodos terminales. En el caso de regresión, el valor predicho suele ser la media de la variable respuesta de las observaciones de entrenamiento que están en ese mismo nodo. Si bien la media es valor más empleado, se puede utilizar cualquier otro (mediana, cuantil...).



Árboles de clasificación

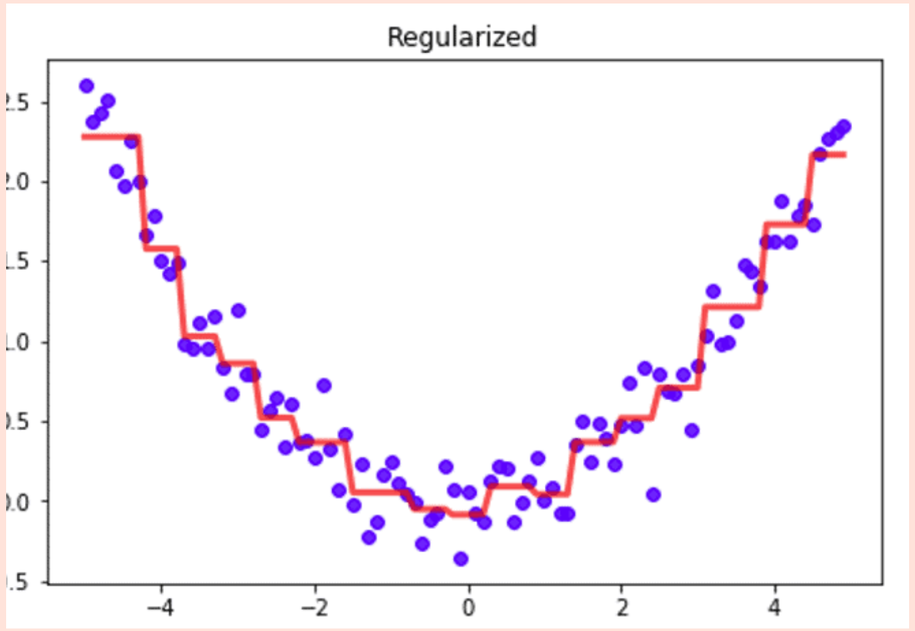
Los árboles de regresión son el subtipo de árboles de predicción que se aplica cuando la variable respuesta es categórica.

Construcción del árbol

Para construir un árbol de clasificación, se emplea el mismo método recursive binary splitting descrito en los árboles de regresión. Sin embargo, como la variable respuesta en cualitativa, no se puede emplear el RSS como criterio de selección de las divisiones óptimas. Existen varias alternativas, todas ellas con el objetivo de encontrar nodos lo más puros/ homogéneos posible.

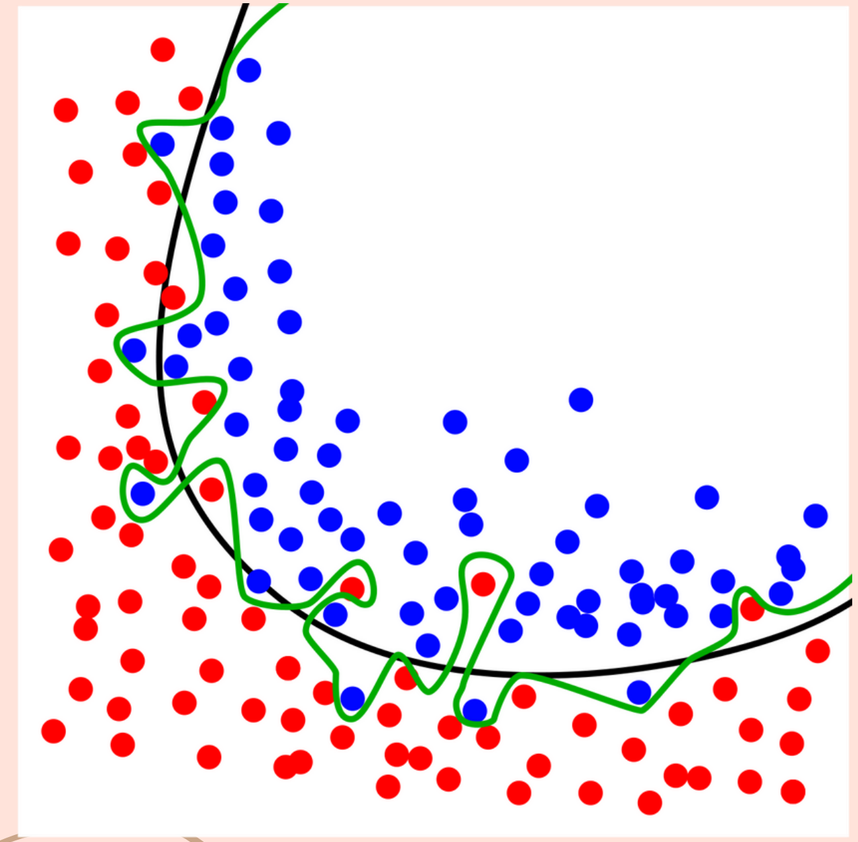
Predicción del árbol

Tras la creación de un árbol, las observaciones de entrenamiento quedan agrupadas en los nodos terminales. Para predecir una nueva observación, se recorre el árbol en función del valor de sus predictores hasta llegar a uno de los nodos terminales. En el caso de clasificación, suele emplearse la moda de la variable respuesta como valor de predicción, es decir, la clase más frecuente del nodo. Además, puede acompañarse con el porcentaje de cada clase en el nodo terminal, lo que aporta información sobre la confianza de la predicción.



Evitar el overfitting

El proceso de construcción de árboles descrito en las secciones anteriores tiende a reducir rápidamente el error de entrenamiento, es decir, el modelo se ajusta muy bien a las observaciones empleadas como entrenamiento. Como consecuencia, se genera un overfitting que reduce su capacidad predictiva al aplicarlo a nuevos datos. La razón de este comportamiento radica en la facilidad con la que los árboles se ramifican adquiriendo estructuras complejas. De hecho, si no se limitan las divisiones, todo árbol termina ajustándose perfectamente a las observaciones de entrenamiento creando un nodo terminal por observación. Existen dos estrategias para prevenir el problema de overfitting de los árboles: limitar el tamaño del árbol (parada temprana) y el proceso de podado (pruning).



Arboles de decision python (cienciadedatos.net)
Minería de Datos: Árboles de decisión | AnálisisDeDatos.net (analisisdedatos.net)
Arboles de decisiones en la minería de datos - Conecta Software

Ejemplo de arboles de decisión de regresión

Se hace el llamado a las librerías y con sklearn se obtiene la base de datos de vivienda en Boston. Útil para este ejemplo

In [1]:
```
# Tratamiento de datos
# ------------------------------------------------------------------------------
import numpy as np
import pandas as pd

# Gráficos
# ------------------------------------------------------------------------------
import matplotlib.pyplot as plt

# Preprocesado y modelado
# ------------------------------------------------------------------------------
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
from sklearn.tree import export_graphviz
from sklearn.tree import export_text
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Configuración warnings
# ------------------------------------------------------------------------------
import warnings
warnings.filterwarnings('once')
```

Visualizamos los datos a tratar dentro de la base de datos. Con el propósito de que se pretende ajustar un modelo de regresión que permita predecir el precio medio de una vivienda (MEDV) en función de las variables disponibles.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

Como en todo estudio de regresión, no solo es importante ajustar el modelo, sino también cuantificar su capacidad para predecir nuevas observaciones. Para poder hacer la posterior evaluación, se dividen los datos en dos grupos, uno de entrenamiento y otro de test.

In [44]:
```
# División de los datos en train y test
# ------------------------------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
                                        datos.drop(columns = "MEDV"),
                                        datos['MEDV'],
                                        random_state = 123
                                    )
# Creación del modelo
# ------------------------------------------------------------------------------
modelo = DecisionTreeRegressor(
            max_depth         = 3,
            random_state      = 123
          )

# Entrenamiento del modelo
# ------------------------------------------------------------------------------
modelo.fit(X_train, y_train)
```

Out[44]:
```
DecisionTreeRegressor(max_depth=3, random_state=123)
```

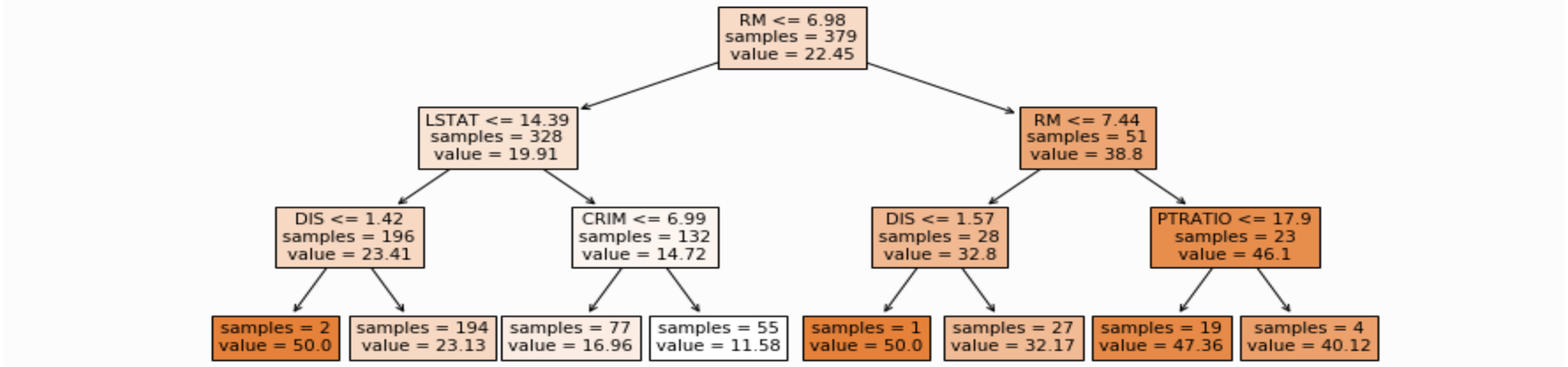
Una vez entrenado el árbol, se puede representar mediante la combinación de las funciones plot_tree() y export_text(). La función plot_tree() dibuja la estructura del árbol y muestra el número de observaciones y valor medio de la variable respuesta en cada nodo. La función export_text() representa esta misma información en formato texto

```
# Estructura del árbol creado
# ------------------------------------------------------------------------------
fig, ax = plt.subplots(figsize=(12, 5))

print(f"Profundidad del árbol: {modelo.get_depth()}")
print(f"Número de nodos terminales: {modelo.get_n_leaves()}")

plot = plot_tree(
            decision_tree = modelo,
            feature_names = datos.drop(columns = "MEDV").columns,
            class_names   = 'MEDV',
            filled        = True,
            impurity      = False,
            fontsize      = 10,
            precision     = 2,
            ax            = ax
       )
```

```
Profundidad del árbol: 3
Número de nodos terminales: 8
```



```
texto_modelo = export_text(
                    decision_tree = modelo,
                    feature_names = list(datos.drop(columns = "MEDV").columns)
                )
print(texto_modelo)
```

```
|--- RM <= 6.98
|   |--- LSTAT <= 14.39
|   |   |--- DIS <= 1.42
|   |   |   |--- value: [50.00]
|   |   |--- DIS >  1.42
|   |   |   |--- value: [23.13]
|   |--- LSTAT >  14.39
|   |   |--- CRIM <= 6.99
|   |   |   |--- value: [16.96]
|   |   |--- CRIM >  6.99
|   |   |   |--- value: [11.58]
|--- RM >  6.98
|   |--- RM <= 7.44
|   |   |--- DIS <= 1.57
|   |   |   |--- value: [50.00]
|   |   |--- DIS >  1.57
|   |   |   |--- value: [32.17]
|   |--- RM >  7.44
|   |   |--- PTRATIO <= 17.90
|   |   |   |--- value: [47.36]
|   |   |--- PTRATIO >  17.90
|   |   |   |--- value: [40.12]
```

Siguiendo la rama más a la izquierda del árbol, puede verse que el modelo predice un precio promedio de 50000 dólares para viviendas que están en una zona con un RM <= 6.98, un LSTAT <= 14.39 y un DIS <= 1.42.

La importancia de cada predictor en modelo se calcula como la reducción total (normalizada) en el criterio de división, en este caso el mse, que consigue el predictor en las divisiones en las que participa. Si un predictor no ha sido seleccionado en ninguna división, no se ha incluido en el modelo y por lo tanto su importancia es 0.

```
importancia_predictores = pd.DataFrame(
                            {'predictor': datos.drop(columns = "MEDV").columns,
                             'importancia': modelo.feature_importances_}
                            )
print("Importancia de los predictores en el modelo")
print("-------------------------------------------")
importancia_predictores.sort_values('importancia', ascending=False)
```

```
Importancia de los predictores en el modelo
-------------------------------------------
```

	predictor	importancia
5	RM	0.671680
12	LSTAT	0.222326
7	DIS	0.064816
0	CRIM	0.034714
10	PTRATIO	0.006465
1	ZN	0.000000
2	INDUS	0.000000
3	CHAS	0.000000
4	NOX	0.000000
6	AGE	0.000000
8	RAD	0.000000
9	TAX	0.000000
11	B	0.000000

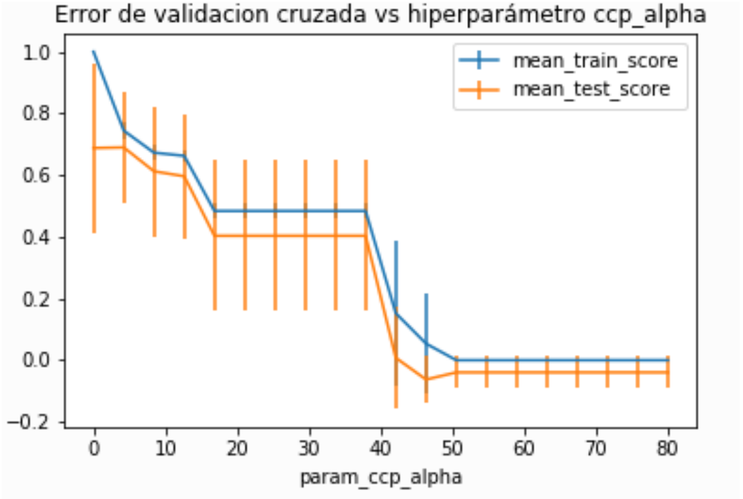
Para aplicar el proceso de pruning es necesario indicar el argumento ccp_alpha que determina el grado de penalización por complejidad. Cuanto mayor es este valor, más agresivo el podado y menor el tamaño del árbol resultante.

```
# Pruning (const complexity pruning) por validación cruzada
# ------------------------------------------------------------------------------
# Valores de ccp_alpha evaluados
param_grid = {'ccp_alpha':np.linspace(0, 80, 20)}

# Búsqueda por validación cruzada
grid = GridSearchCV(
        # El árbol se crece al máximo posible para luego aplicar el pruning
        estimator = DecisionTreeRegressor(
                            max_depth         = None,
                            min_samples_split = 2,
                            min_samples_leaf  = 1,
                            random_state      = 123
                        ),
        param_grid = param_grid,
        cv         = 10,
        refit      = True,
        return_train_score = True
       )

grid.fit(X_train, y_train)

fig, ax = plt.subplots(figsize=(6, 3.84))
scores = pd.DataFrame(grid.cv_results_)
scores.plot(x='param_ccp_alpha', y='mean_train_score', yerr='std_train_score', ax=ax)
scores.plot(x='param_ccp_alpha', y='mean_test_score', yerr='std_test_score', ax=ax)
ax.set_title("Error de validacion cruzada vs hiperparámetro ccp_alpha");
```



```
# Mejor valor ccp_alpha encontrado
# ------------------------------------------------------------------------------
grid.best_params_
```

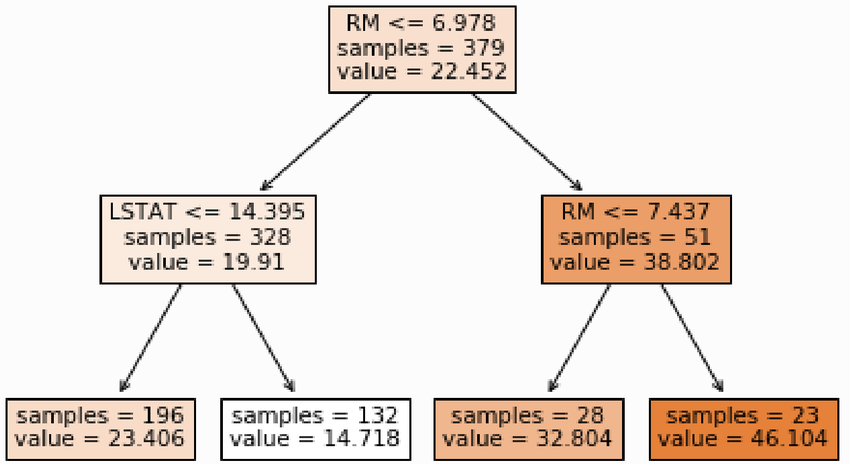
```
{'ccp_alpha': 4.2105263157894735}
```

Una vez identificado el valor óptimo de ccp_alpha, se reentrena el árbol indicando este valor en sus argumentos. Si en el GridSearchCV() se indica refit=True, este reentrenamiento se hace automáticamente y el modelo resultante se encuentra almacenado en .best_estimator_.

```python
# Estructura del árbol final
# ------------------------------------------------------------------------------
modelo_final = grid.best_estimator_
print(f"Profundidad del árbol: {modelo_final.get_depth()}")
print(f"Número de nodos terminales: {modelo_final.get_n_leaves()}")

fig, ax = plt.subplots(figsize=(7, 5))
plot = plot_tree(
            decision_tree = modelo_final,
            feature_names = datos.drop(columns = "MEDV").columns,
            class_names   = 'MEDV',
            filled        = True,
            impurity      = False,
            ax            = ax
       )
```

```
Profundidad del árbol: 2
Número de nodos terminales: 4
```



Ejemplo de arboles de decisión de clasificación

Se llaman a las librerías a utilizar. Cargamos un archivo csv para importar sus datos. Al igual que anteriormente visualizamos los datos a tratar

```
# Cargue las bibliotecas
import  pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image
from sklearn import tree
import pydotplus
from sklearn.preprocessing import LabelEncoder
```

```
#cargamos el archivo con el cual trabajaremos
dataset = pd.read_csv('diabetes_data_upload.csv')
```

```
dataset.head(5)
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity	c
0	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Po
1	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	No	Yes	No	Po
2	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Po
3	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Po
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Po

Se cambian los nombres de las columnas.

Importamos el label enconder para permitirnos cambiar los datos de las columnas, remplazándolos por 0 o 1.

```
: dataset.columns = ['Age', 'Gender','polyuria', 'Polydipsia', 'sudden', 'weakness', 'Polyphagia', 'Genital', 'visual',
                     'Itching', 'Irritability', 'delayed', 'partial', 'muscle', 'Alopecia', 'Obesity', 'clase']
  dataset.columns
```

```
: Index(['Age', 'Gender', 'polyuria', 'Polydipsia', 'sudden', 'weakness',
         'Polyphagia', 'Genital', 'visual', 'Itching', 'Irritability', 'delayed',
         'partial', 'muscle', 'Alopecia', 'Obesity', 'clase'],
        dtype='object')
```

```
: from sklearn.preprocessing import LabelEncoder
```

```
: le = LabelEncoder()
```

```
: dataset.Gender = le.fit_transform(dataset.Gender)
  dataset.polyuria = le.fit_transform(dataset.polyuria)
  dataset.Polydipsia = le.fit_transform(dataset.Polydipsia)
  dataset.sudden = le.fit_transform(dataset.sudden)
  dataset.weakness = le.fit_transform(dataset.weakness)
  dataset.Polyphagia = le.fit_transform(dataset.Polyphagia)
  dataset.Genital = le.fit_transform(dataset.Genital)
  dataset.visual = le.fit_transform(dataset.visual)
  dataset.Itching = le.fit_transform(dataset.Itching)
  dataset.Irritability = le.fit_transform(dataset.Irritability)
  dataset.delayed = le.fit_transform(dataset.delayed)
  dataset.partial = le.fit_transform(dataset.partial)
  dataset.muscle = le.fit_transform(dataset.muscle)
  dataset.Alopecia = le.fit_transform(dataset.Alopecia)
  dataset.Obesity = le.fit_transform(dataset.Obesity)
  dataset.clase = le.fit_transform(dataset.clase)
```

Visualizamos los datos cambiados junto con el nombre de las columnas

```
dataset
```

	Age	Gender	polyuria	Polydipsia	sudden	weakness	Polyphagia	Genital	visual	Itching	Irritability	delayed	partial	muscle	Alopecia	Obesity	clase
0	40	1	0	1	0	1	0	0	0	1	0	1	0	1	1	1	1
1	58	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	1
2	41	1	1	0	0	1	1	0	0	1	0	1	0	1	1	0	1
3	45	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	1
4	60	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
515	39	0	1	1	1	0	1	0	0	1	0	1	1	0	0	0	1
516	48	0	1	1	1	1	1	0	0	1	1	1	1	0	0	0	1
517	58	0	1	1	1	1	1	0	1	0	0	0	1	1	0	1	1
518	32	0	0	0	0	1	0	0	1	1	0	1	0	0	1	0	0
519	42	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tomamos a x como los datos de entrada y ‘y’ como el dato de salida

Se importa train explit que nos permite separar los datos en dos casos (xtrain datos de entrenamientos) (x_test datos de prueba) (test_size cuanto se separaran los datos).

Importamos en este caso decisión tree classifier

El criterio entropy trabaja con probabilidades, trabajando con datos ordenados. Max es la profundidad del arbol, entre mas profundo es mas preciso. Random state permite que sea fijo la variable escogida.

```
x = dataset.iloc[:,0:16].values
y = dataset.iloc[:, 16].values
```

```
#conjunto de datos en 75% (Entrenamiento) y 25% (Test) utilizando la función train_test_split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)
```

```
#entrenamiento de el arbol de decicion
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', max_depth = 4, random_state = 0)
classifier.fit(x_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

Se crea una variable de predicción a través de las variables de prueba (x_test) comparada con la otra variable de decisión. Para no estar comparando dato por dato podemos usar la matriz de comparación

```
y_pred = classifier.predict(x_test)
```

```
y_pred
```

```
array([0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0])
```

```
y_test
```

```
array([1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0])
```

```
#matriz de confucion
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[51,  5],
       [ 4, 70]], dtype=int64)
```

El accuracy nos permite ver la efectividad de nuestro arbol generado.

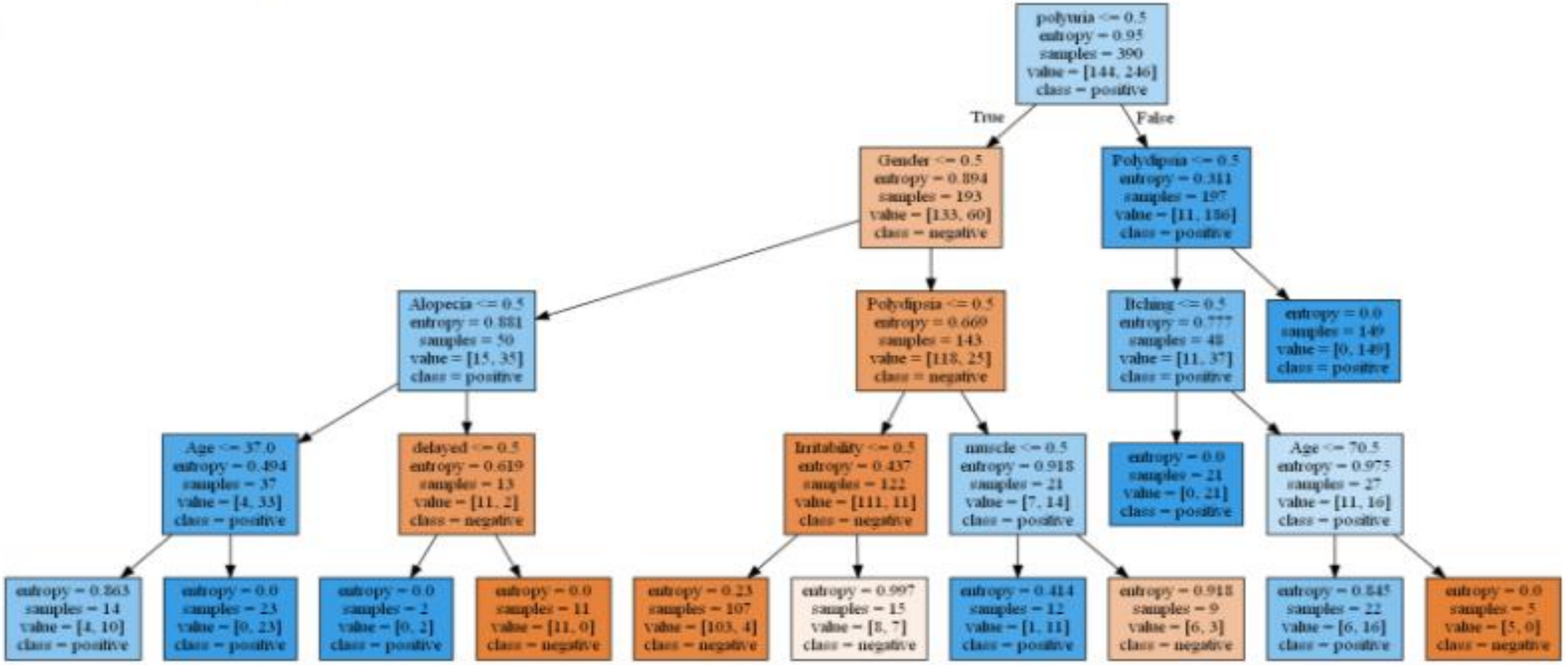
```
#accuracy
from sklearn import metrics
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
print("F1 Score: ", metrics.f1_score(y_test, y_pred, average = 'weighted'))
print("ROC: ", metrics.roc_auc_score(y_test, y_pred))
print("Recall: ", metrics.recall_score(y_test, y_pred, average = 'weighted'))
```

```
Accuracy:  0.9307692307692308
F1 Score:  0.9306896984749333
ROC:  0.9283301158301158
Recall:  0.9307692307692308
```

Al final generamos el arbol, el class name en los datos objetivos si son positivos o negativos

El feature_names es para la recuperación del nombre de las variables logrando una mayor comprensión del arbol y por ultimo el filled true que le otorga el color al arbol

```
dot_data = tree.export_graphviz(classifier,
                                out_file = None,
                                class_names = ['negative', 'positive'],
                                feature_names = list(dataset.drop(['clase'], axis=1)),
                                filled= True
                               )
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```



Enlace de los ejemplos de Python:
Ejercicio de Clasificación: https://github.com/VictorQuirozGarcia/miner-a-de-datos-grupo-02/blob/main/Ejemplo%20clasificacion.ipynb
Ejercicio de Regresión: https://github.com/VictorQuirozGarcia/miner-a-de-datos-grupo-02/blob/main/Ejemplo%20regresion.ipynb

Enlace de las calificaciones del Kahoot
Calificaciones (miner-a-de-datos-grupo-02/Calificacion_Arboles-de-Decision_Equipo-5.pdf at main · VictorQuirozGarcia/miner-a-de-datos-grupo-02 (github.com)