



## APRENDIZAJE AUTOMÁTICO Y MINERÍA DE DATOS LABORATORIO

Clasificación mediante árboles de decisión

Ejercitar un ejemplo completo para realizar tareas de clasificación  
mediante los algoritmos proporcionados en el paquete scikit-learn

Profesor: Javier Martínez Torres

Ernesto González Pradas  
ernesto.gonzalez023@comunidadunir.net

## Índice

### Contenido

Índice.....	1
Introducción.....	2
Estudio de la relevancia de los atributos y generación de los conjuntos de datos para entrenamiento, validación y test.....	3
Ejecución de algoritmos, descripción del método y ajuste de parámetros .....	6
Visualización de resultados y conclusiones .....	8
Se amplía el estudio a varias estaciones.....	14
Bibliografía .....	16

## Introducción


Con este laboratorio se pretende aprender a realizar tareas de clasificación mediante los algoritmos proporcionados en el paquete scikit-learn. Para ello discretizaremos los campos NO2 y PM10 y separaremos los datos por un lado para entrenamiento y por otro para test. Posteriormente ejecutaremos el algoritmo Decision Trees una o varias veces, analizaremos los resultados obtenidos comparándolos entre sí las dos predicciones y finalmente mostraremos y analizaremos el árbol obtenido, sus atributos relevantes, etc.

Para ello se utilizarán los datos proporcionados junto con las instrucciones de la actividad.

## Estudio de la relevancia de los atributos y generación de los conjuntos de datos para entrenamiento, validación y test

Para estudiar la relevancia de los atributos y su posterior generación de conjuntos de datos para poderlos entrenar, validar y testear, seleccionaremos dos variables objetivo o variables a predecir, que serán NO2 y PM10. El estudio se realizará por separado por lo que comenzaremos por PM10.

Lo primero que tenemos que realizar es un estudio por estaciones para ver, cuál de ellas tiene más datos, ya que cuantos más datos, más preciso será el estudio y la predicción.



Estacion:estacion_8							\
	SO2	CO	NO	NO2	PM2.5	PM10	
count	318.000000	326.000000	329.000000	329.000000	305.000000	304.000000	
mean	3.773585	0.278528	11.103343	35.088146	8.829508	18.648026	
std	1.902761	0.116219	16.282237	13.286299	5.379400	11.962171	
min	1.000000	0.100000	1.000000	11.000000	2.000000	4.000000	
25%	3.000000	0.200000	4.000000	26.000000	5.000000	11.000000	
50%	3.000000	0.300000	6.000000	33.000000	8.000000	16.000000	
75%	5.000000	0.300000	11.000000	42.000000	11.000000	22.000000	
max	15.000000	1.100000	156.000000	83.000000	37.000000	82.000000	
							\
	NOx	O3	TOL	BEN	EBE		
count	329.000000	320.000000	314.000000	306.000000	310.000000		
mean	52.136778	51.66250	1.977707	0.432680	0.503871		
std	35.843871	17.68122	1.142644	0.219636	0.290605		
min	14.000000	13.00000	0.100000	0.100000	0.100000		
25%	32.000000	38.00000	1.200000	0.300000	0.300000		
50%	43.000000	52.00000	1.700000	0.400000	0.450000		
75%	59.000000	64.25000	2.500000	0.600000	0.600000		
max	321.000000	104.00000	7.200000	1.300000	2.400000		

Observamos que la estación 8 tiene bastantes datos con los que podemos trabajar por lo que tanto para PM10 como para NO2 utilizaremos esta estación.

A continuación, realizaremos las transformaciones oportunas. En este paso podríamos haber realizado una normalización o mini-max, pero como tenemos muchos datos y pocos NaN, hemos optado por eliminar directamente las filas que contienen estos NaN y trabajar con los datos resultantes. Dichos datos para probar nos quedarían así:

```
Data columns (total 11 columns):
# Column Non-Null Count Dtype
0 SO2 318 non-null float64
1 CO 326 non-null float64
2 NO 329 non-null float64
3 NO2 329 non-null float64
4 PM2.5 305 non-null float64
5 PM10 304 non-null float64
6 NOx 329 non-null float64
7 O3 320 non-null float64
8 TOL 314 non-null float64
9 BEN 306 non-null float64
10 EBE 310 non-null float64
dtypes: float64(11)
```

```
Data columns (total 11 columns):
# Column Non-Null Count Dtype
0 SO2 266 non-null float64
1 CO 266 non-null float64
2 NO 266 non-null float64
3 NO2 266 non-null float64
4 PM2.5 266 non-null float64
5 PM10 266 non-null float64
6 NOx 266 non-null float64
7 O3 266 non-null float64
8 TOL 266 non-null float64
9 BEN 266 non-null float64
10 EBE 266 non-null float64
dtypes: float64(11)
```

Una vez que tenemos nuestros datos preparados para trabajar, lo que realizaremos será una discretización de la columna objetivo, en nuestro caso PM10. La discretización es una operación que nos permite separar en clases los datos disponibles de dicho compuesto. Esto nos permite simplificar la información agrupando los datos en objetos geográficos que presentan las mismas características en distintas clases. En el caso de PM10 discretizaremos la columna para un valor superior a 20, es decir, se generará un array con valores 0 y 1 (valor 0 para todos los datos que estén por debajo o igual a 20 y valor 1 para los que estén por encima) que serán las clases. La elección del rango [0,20] y no [0,50] (por ejemplo), es porque analizando los datos de PM10 hay muy pocos que estén por encima, por lo que el clasificador daría por bueno muchos datos catalogados en la clase 0 teniendo un error muy bajo y como consecuencia no estaría prediciendo bien.

Ahora debemos borrar la columna objetivo de nuestro dataframe, ya que no tendría mucho sentido predecir el campo PM10 teniendo el campo PM10 entre nuestros datos. También, tal y como dice el enunciado de la actividad, eliminaremos la columna PM2.5 ya que está muy correlacionada con PM10. Utilizamos la función “train\_test\_split” de sklearn y le pasamos el dataframe con las columnas eliminadas (X), la clase con encoded (y) el tamaño del test (test\_size=0.3) y random\_state=1 para que siempre nos salga el mismo resultado cuando ejecutemos el programa, es decir, hacemos que la distribución sea siempre la misma. Vemos tanto el conjunto de entrenamiento como de prueba:

```
87 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)
X_train.shape, X_test.shape
87 ((186, 9), (80, 9))
```

Seguimos los mismos pasos hasta aquí para la columna objetivo NO2. A la hora de discretizar esta columna hemos cambiado el rango con respecto a como lo hemos hecho con PM10, ya que al realizar el estudio hemos visto que con un rango de [0,20] la mayoría de los datos pertenecían a la clase 1 teniendo muy poca representación de la clase 0. Para hacerlo menos homogéneo subimos el rango hasta 35 ([0,35]):

[illegible]

A diferencia de con PM10, en este caso no vamos a eliminar las columnas PM10 ni PM2.5 porque sí que nos interesan para el estudio, sin embargo, sí que vamos a eliminar la columna NO2 ya que nuevamente, no tendría sentido predecir NO2 con NO2.

Por último, generamos el conjunto de entrenamiento y el de test a partir de los datos obtenidos hasta este punto, con la función “train\_test\_split” de sklearn, dataframe con la columna eliminada (Z), la clase (y), el tamaño del test (test\_size=0.3) y random\_state=1:

```
263 #Generamos un conjunto de entrenamiento y otro de test
    Z_train, Z_test, v_train, v_test = train_test_split(Z, y, test_size = 0.3, random_state = 1)
    Z_train.shape, Z_test.shape

263 ((186, 10), (80, 10))
```

Ahí podemos visualizar tanto el conjunto de entrenamiento como el de test.

## Ejecución de algoritmos, descripción del método y ajuste de parámetros

Una vez que ya tenemos nuestros conjuntos de entrenamiento tanto para PM10 como para NO2, lo siguiente que vamos a realizar es ejecutar el algoritmo de decisión. Declaramos el inicializador del objeto árbol sin parámetros y posteriormente lo entrenaríamos con la función `fit()`, pasándole el conjunto de entrenamiento y la clase obtenidos en el apartado anterior (`X_train`, `y_train`):

```
255 arbol_todas_columnas = tree.DecisionTreeClassifier()  
    arbol_todas_columnas = arbol_todas_columnas.fit(X_train, y_train)
```

El método `DecisionTreeClassifier()` tiene una serie de parámetros de entrada, que aunque no los hemos usado en el ejemplo, vamos a comentarlos (los más importantes):

- **Criterion:** la función para medir la calidad de un Split (o división), siendo sus valores permitidos “gini” para la impureza de Gini y “entropía” para la ganancia de información.
- **Splitter:** Es la estrategia utilizada para elegir la división en cada nodo, siendo sus valores permitidos “best” para elegir la mejor división y “random” para elegir la mejor división aleatoria.
- **Max-depth:** Delimita la profundidad máxima del árbol. Si no se pone nada, los nodos se expanden hasta que todas las hojas queden balanceadas. Este parámetro de entrada si lo vamos a usar más adelante.
- **Class\_weight:** Pesos asociados a las clases. Si está vacío, se supone que todas las clases tienen peso uno. Este es un parámetro muy interesante si queremos realizar pruebas con multisalida, incluyendo multietiqueta.

Como hemos visto en la descripción de los parámetros que acepta el método, tenemos “Max-depth” que nos delimita la profundidad. Como no queremos que genere un árbol con hojas hasta que quede balanceado, introducimos una profundidad máxima de 3:

```
arbol_3 = tree.DecisionTreeClassifier(max_depth=3)  
arbol_3 = arbol_3.fit(X_train, y_train)
```

Para NO2 realizaríamos lo mismo:

```
# Fijar una profundidad máxima = 3
arbol_5 = tree.DecisionTreeClassifier(max_depth=3)
arbol_5 = arbol_5.fit(Z_train, v_train)
```

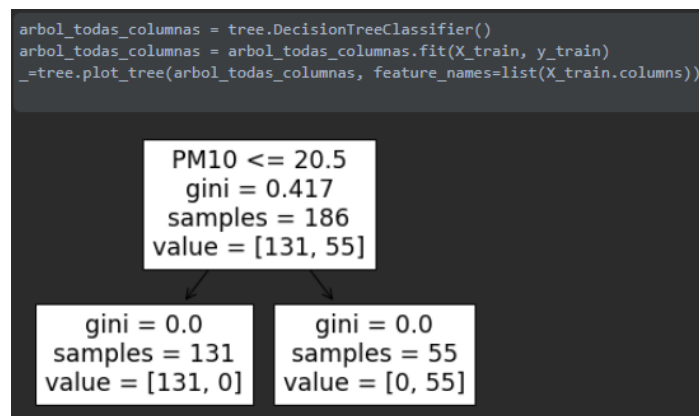
En este punto tendríamos nuestros algoritmos de árboles, ajustados y ejecutados. En el siguiente epígrafe vamos a describir los resultados que nos salen, entrenando de forma distinta a nuestros árboles, y como se visualizan cuando ajustamos los métodos con parámetros como max\_depth.



## Visualización de resultados y conclusiones

A continuación, vamos a describir y mostrar los resultados obtenidos con distintas pruebas realizadas. Primero lo veremos para el compuesto PM10.

Antes de nada, vamos a demostrar una afirmación que hacíamos más arriba. Y es que no tiene sentido predecir PM10 si dejamos la columna PM10 en el conjunto de entrenamiento. El resultado sería este:



Con el método `plot_tree()` pintamos el árbol. En este caso como podemos ver, ha encontrado el campo y además encuentra el sitio donde dividirlos exactamente (20.5). Encuentra el valor exacto donde la medida Gini disminuye hasta 0 y pasa de 0.417 a 0 en los nodos hojas. Con esto podemos concluir que el algoritmo funciona, pero no nos proporciona mucha información, ya que le estamos dando al clasificador el campo directamente.

La siguiente prueba que vamos a realizar es quitarle el campo PM10 y ver que ocurre:

```
borra_columna = ['PM10']
X = estaciones.drop(columns=borra_columna)
y = PM10_class_encoded
```

Podemos observar en nuestro cuaderno jupyter que, en el árbol generado, el componente PM2.5 tiene una gran correlación y obtenemos un árbol con una profundidad = 7. Es decir, el árbol se ha ido dividiendo hasta quedar totalmente balanceado o lo que sería equivalente a entropía = 0. Como podemos ver el algoritmo puede comprobar varias veces un mismo nodo utilizando distintas particiones (esto es

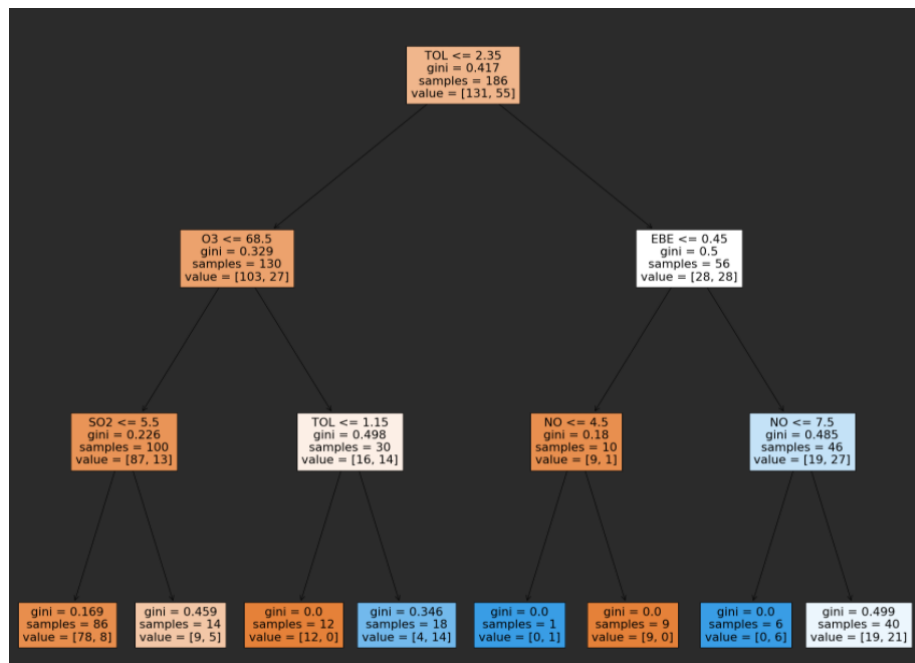
una característica que lo diferencia con el algoritmo ID3), es decir, no elimina los nodos.

También podemos observar con el método “feature\_importances\_” del árbol, cual es el atributo más correlacionado con el objetivo. Como hemos comentado más arriba es PM2.5:

```
print(X.columns)
print(arbol_2.feature_importances_)

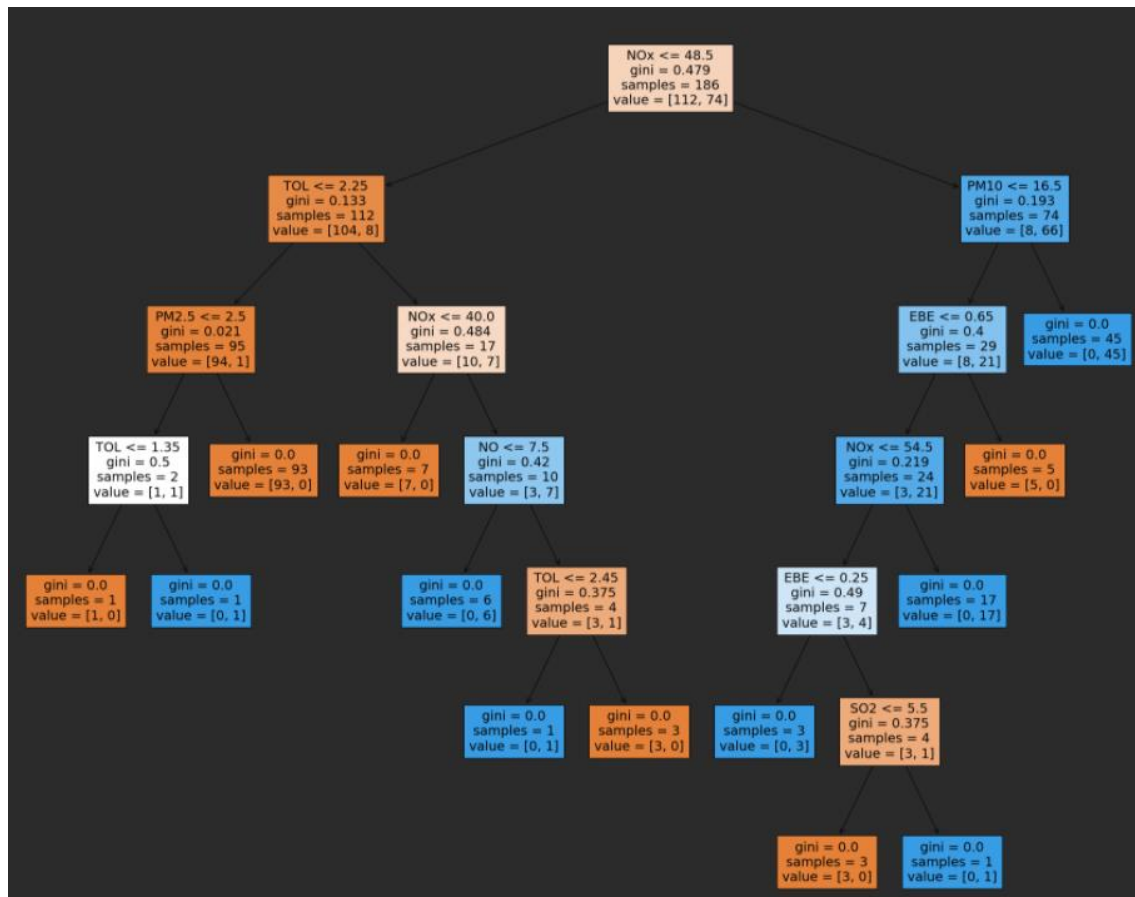
Index(['SO2', 'CO', 'NO', 'NO2', 'PM2.5', 'NOx', 'O3', 'TOL', 'BEN', 'EBE'], dtype='object')
[0.04127658 0.01932381 0.0842074 0.07945408 0.50019645 0.01966888
 0.13090711 0.01721027 0.07871308 0.02904233]
```

La última prueba que vamos a realizar es, como dice el enunciado de la actividad, borrar la columna PM2.5 y ver que sale. Además, le pondremos una restricción de máxima profundidad = 3 para que no tenga una entropía = 0. Este sería el resultado:



Podemos ver, que con el atributo filled, colorea los nodos de naranja si la clase es 0, de azul si es 1 y blanco si es mitad y mitad. Al limitarle la profundidad, vemos que los nodos hojas no quedan perfectamente balanceados salvo en cuatro casos en los que la medida Gini es 0. Si analizamos un poco más los resultados, vemos que por ejemplo en dichas hojas con el valor Gini = 0, de 12 ejemplos ha catalogado los 12 como clase 0.

Realizamos lo mismo con el compuesto NO2. Primero vamos a generar el árbol sin limitar la profundidad y ver que ocurre:

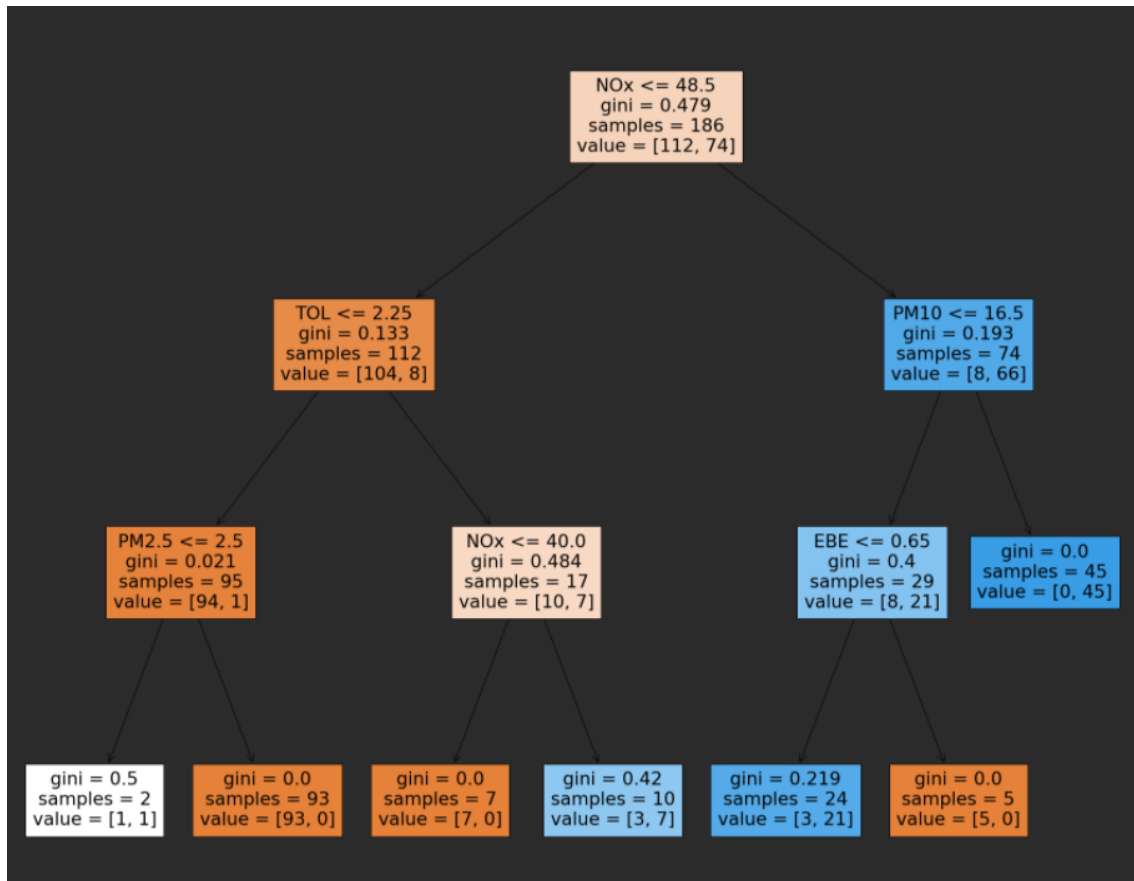


Vemos que el árbol tiene una profundidad de nivel 6 y que llega a una entropía = 0. En este caso, es igual que con PM10, los nodos naranjas pertenecen a la clase 0, los azules a la clase 1 y los blancos tienen un 50% de ambos. Tanto con el árbol generado, como si ejecutamos el método `feature_importances_` vemos que el compuesto más correlacionado con NO2 es NOx:

```
17 print(Z.columns)
   print(arbol_4.feature_importances_)

Index(['SO2', 'CO', 'NO', 'PM2.5', 'PM10', 'NOx', 'O3', 'TOL', 'BEN', 'EBE'], dtype='object')
[0.         0.         0.03029681 0.01098481 0.04694955 0.73887859
 0.         0.05209819 0.02164058 0.09915145]
```

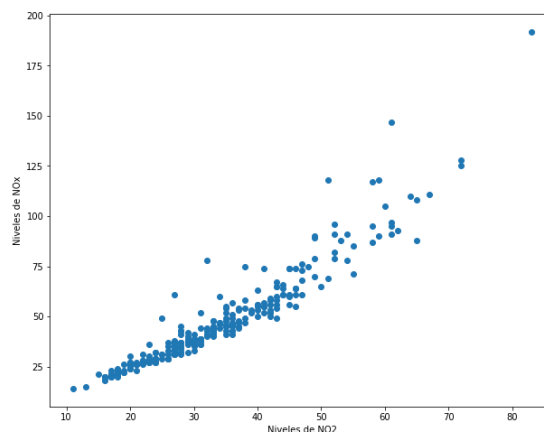
Realizaremos la misma prueba limitando la profundidad a tres. Este sería el árbol generado:



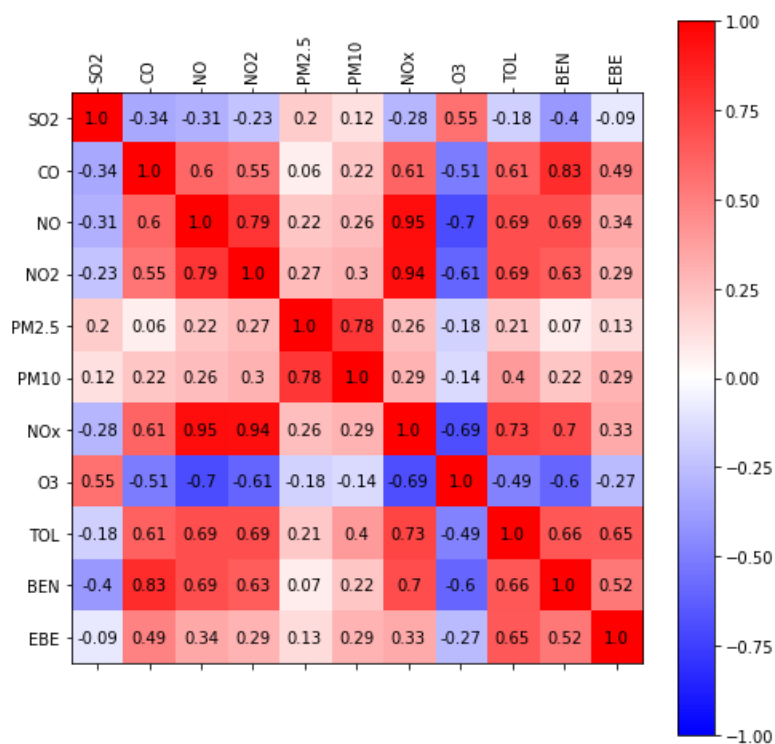
Aquí, ya vemos que encuentra más mezcla en los conjuntos de entrenamiento, pero aún así los clasifica con bastante acierto.

Una vez que tenemos los dos compuestos más correlacionados, vamos a estudiar su relación con la matriz de correlación, correlación lineal de Pearson y mapas de calor.

Esta sería la representación de la relación entre ambos compuestos:



Como podemos ver es una relación fuerte con pendiente positiva. Y esta sería la matriz de correlación con mapa de calor de todos los compuestos de la estación:



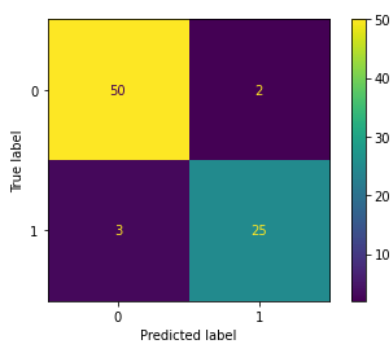
En este mapa de calor, nos llama la atención las celdas de color más rojo y azul. Vemos que tenemos correlación negativa con compuestos que salen en color azul.

Para finalizar este apartado de visualización de resultados y conclusiones, vamos a observar la matriz de confusión para el compuesto NO2, ya que el compuesto PM10 se visualizó en la clase de laboratorio.

```

TRAIN
Accuracy: 0.9623655913978495
F1: 0.954248366013072
TEST
Accuracy: 0.9375
F1: 0.9090909090909091
[[50  2]
 [ 3 25]]

```



Medimos cual es el resultado de aplicar el árbol a los datos que aún no hemos visto y a los que sí. Vemos en la matriz de confusión que la medida de éxito (Accuracy) es en el conjunto de entrenamiento es del 96.2% y para el conjunto de test 93.75%. Por otro lado, tenemos la métrica F1 para el conjunto de entrenamiento 95,4% (muy buen resultado) y para el conjunto test 90,9% (también un resultado muy bueno).

En la salida de la matriz de confusión, vemos que la salida verdadera está a la izquierda y la predicha a la derecha. Tendríamos que trasponer la matriz para que coincidiera con los apuntes del tema.

## Se amplía el estudio a varias estaciones

Para terminar con el estudio, vamos a realizar pruebas con dos estaciones para tener conjuntos de entrenamiento y test más grandes. Primero elegimos las estaciones que más datos pueden tener para nuestro estudio (este estudio con varias estaciones solo lo hemos realizado para el compuesto NO<sub>2</sub> ya que el PM<sub>10</sub> ya está visto en la clase de laboratorio), y vemos que son la estación 8 y 24:

```
#Elegimos las estaciones 8 y 24 por que son las que más datos tienen para el estudio
estaciones_8_24 = datos.loc[datos.index.get_level_values('ESTACION').isin([8,24])]
estaciones_8_24.head(20) estaciones_8_24: {DataFrame: (570, 9)}
```

Antes de seguir, vemos que las columnas SO<sub>2</sub> y CO, no tienen datos en la estación 24, por lo que procedemos a eliminar esas columnas de nuestro data frame.

Definimos un rango para discretizar la columna objetivo, en nuestro caso analizando los datos que tenemos de ambas estaciones, el rango elegido es [0,30]. Generamos la clase y a continuación borramos la columna objetivo (NO<sub>2</sub>).

Generamos el conjunto de entrenamiento:

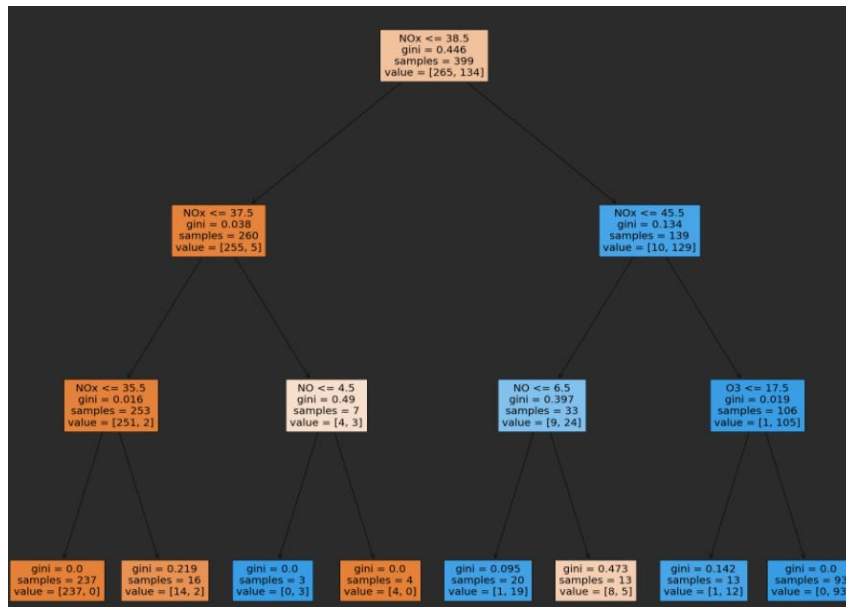
```
109 #Generamos el conjunto de entrenamiento
A_train, A_test, b_train, b_test = train_test_split(A, c, test_size = 0.3, random_state = 1)
A_train.shape, A_test.shape

109 ((399, 8), (171, 8))
```

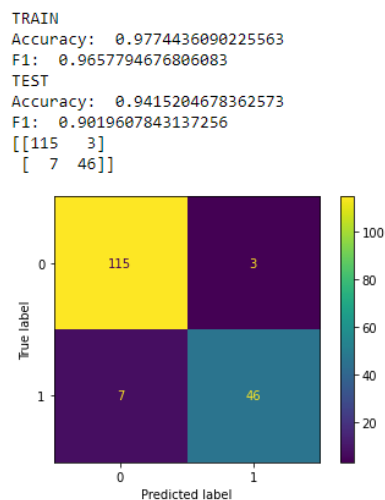
Y generamos el árbol con profundidad 3 (en el notebook de jupyter está generado también sin profundidad para ver el árbol resultante (tiene profundidad 6 hasta que las hojas quedan perfectamente clasificadas con su medida Gini =0.0)).

Como con los ejemplos anteriores, vemos que hay cierta impureza en algunos nodos hojas y nos llama la atención que ahora solo vemos dos compuestos correlacionados con NO<sub>2</sub>, que sería NO<sub>x</sub> y NO.

A continuación se muestra el árbol resultante donde se puede apreciar que para cada nodo, encuentra prácticamente la medida justa donde ir partiendo la clasificación (Nodo1 = 38.5, Nodo2=37.5, Nodo3= 45.5...)



Finalmente, visualizaremos los resultados y conclusiones observando la matriz de confusión para el compuesto NO2 con los datos de las estaciones 8 y 24.



Medimos cual es el resultado de aplicar el árbol a los datos que aún no hemos visto y a los que sí. Vemos en la matriz de confusión que la medida de éxito (Accuracy) en el conjunto de entrenamiento es del 97,74% y para el conjunto de test 94,15%. Por otro lado, tenemos la métrica F1 para el conjunto de entrenamiento 96,57% (muy buen resultado) y para el conjunto test 90,19% (también un resultado muy bueno).

En la salida de la matriz de confusión, vemos que la salida verdadera está a la izquierda (115) y la predicha a la derecha (46). Tendríamos que trasponer la matriz para que coincidiera con los apuntes del tema.



## Bibliografía

Dobilas, S. (31 de Enero de 2021). *towardsdatascience.com*. Obtenido de <https://towardsdatascience.com/cart-classification-and-regression-trees-for-clean-but-powerful-models-cc89e60b7a85>

Rovira, A. C. (2022). Tema 4. Árboles de decisión. Madrid.

Rovira, A. C. (2022). Tema 5. Evaluación de clasificadores. Madrid.