Arboles de decision

June 13, 2023

1 Árboles de decisión (DT)

1.1 Paquetes numpy y pandas

[1]:
```
import numpy as np
import pandas as pd
```

1.1.1 Para fines de estética en la salida, se desactivan las advertencias que pueda informar el intérprete Python

[2]:
```
import warnings
warnings.filterwarnings("ignore")
```

1.2 Paquetes para la construcción del gráfico

[4]:
```
# Paquetes para los gráficos
import matplotlib.pyplot as plt
import graphviz
```

1.3 Importación método para creación del conjunto de entrenamiento desde paquete *sklearn*

[5]:
```
from sklearn.model_selection import train_test_split
```

1.4 Paquete sklearn que contiene los métodos para árboles de decisión

[6]:
```
# Métodos para árboles de decisión desde sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn import tree
```

1.5 Lectura de los datos desde el archivo *datosAB.txt*

[7]:
```
# Esta es la opción para Jupyter Lab/Notebook
datos = pd.read_table("datosAB.txt", sep='\t')
```

1.6 Creación de conjunto de datos

[8]:
```
# Conjunto de datos
X = datos.iloc[:,:-1]
y = datos.iloc[:,2]
```

1.7 Creación de subconjutos CP y CE

[9]:
```
# Se elige una semilla para la selección pseudo-aleatoria
semilla = 123456
```

[10]:
```
X_ce, X_cp, y_ce, y_cp = train_test_split(X, y, test_size=0.3,␣
↪random_state=semilla)
```

1.8 Creación y ajuste del clasificador DT

[11]:
```
# Entrenamiento y ajuste
clasificador = tree.DecisionTreeClassifier()
clasificador = clasificador.fit(X_ce, y_ce)
```

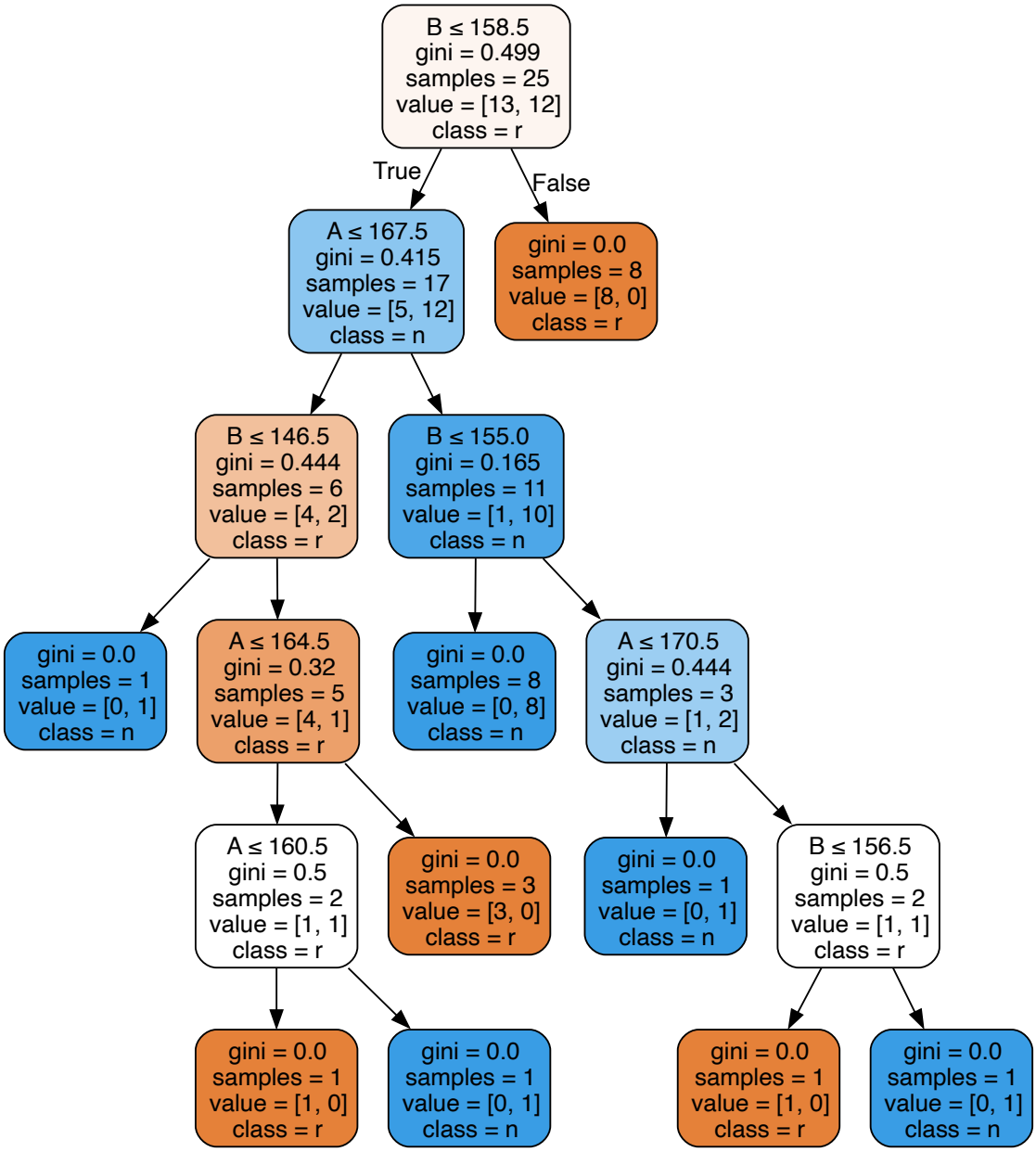
2 Creación del gráfico del árbol de decisión

[12]:
```
# Configuración del gráfico
dot_data = tree.export_graphviz(clasificador, out_file=None,
    feature_names=['A','B'],
    class_names=['r','n'],
    filled=True, rounded=True,
    special_characters=True)
```

2.1 Generación del gráfico

[13]:
```
graph = graphviz.Source(dot_data)
graph.render("datosAB")
graph
```

[13]:



2.2 Predicción

[14]:
```
y_pred = clasificador.predict(X_cp)
```

2.3 Creación de los resultados estadísticos de la clasificación

2.3.1 Importación de método para la matriz de confusión desde paquete *sklearn*

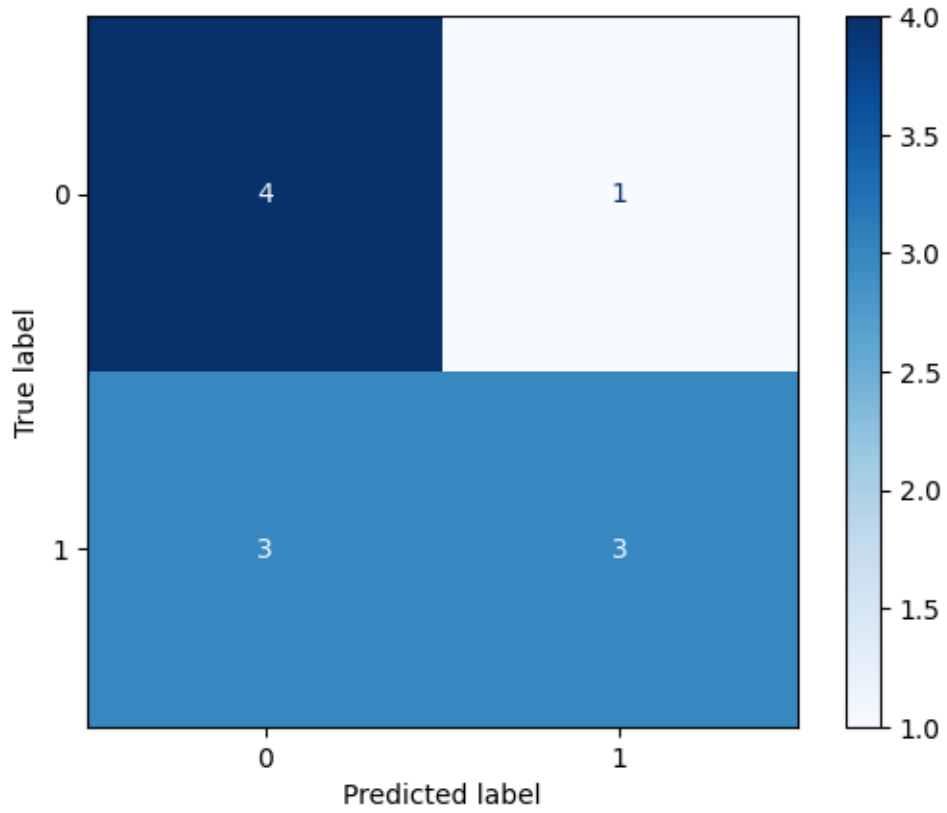
[15]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

2.3.2 Cálculo de la matriz de confusión

[16]:
```
mconf = confusion_matrix(y_cp, y_pred)
```

2.3.3 Impresión de la matriz de confusión

[23]:
```
mconfg = ConfusionMatrixDisplay(mconf).plot(cmap='Blues')
```



2.3.4 Importación de método para la puntuación de precisión desde paquete *sklearn*

[24]:
```
from sklearn.metrics import accuracy_score
```

2.3.5 Cálculo de la puntuación de precisión

[25]:
```
cc = accuracy_score(y_cp, y_pred)
```

2.3.6 Impresión de la puntuación

[26]:
```
print(f'Accuracy Score = {cc}')
```

```
Accuracy Score = 0.6363636363636364
```

2.4 Importación de métodos para el gráfico

[27]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

2.5 Ajuste del etiquetado de la variable y

[28]:
```
# Importación del etiquetador
from sklearn.preprocessing import LabelEncoder
# Creación del etiquetador
labelencoder_y = LabelEncoder()
# Etiquetado y ajuste
y_ce = labelencoder_y.fit_transform(y_ce)
```

2.5.1 Nota: Es necesario realizar el ajuste de nuevo dado que cambió la variable y debido al proceso de etiquetado

[29]:
```
clasificador.fit(X_ce, y_ce)
```

[29]:
```
DecisionTreeClassifier()
```

3 Se grafica todo el conjunto de datos empleando el clasificador DT para cada dato

[30]:
```
# Etiquetado y ajuste del conjunto de datos original
X_set, y_set = X, labelencoder_y.fit_transform(y)
```

3.1 Creación de la malla (plano cartesiano)

[31]:
```
X1, X2 = np.meshgrid(
    np.arange(start = X_set.iloc[:,0].min()-1, stop = X_set.iloc[:,0].max()+1,␣
 ↪step=0.1),
    np.arange(start = X_set.iloc[:,1].min()-1, stop = X_set.iloc[:,1].max()+1,␣
 ↪step=0.1)
)
```

3.2 Creación del gráfico

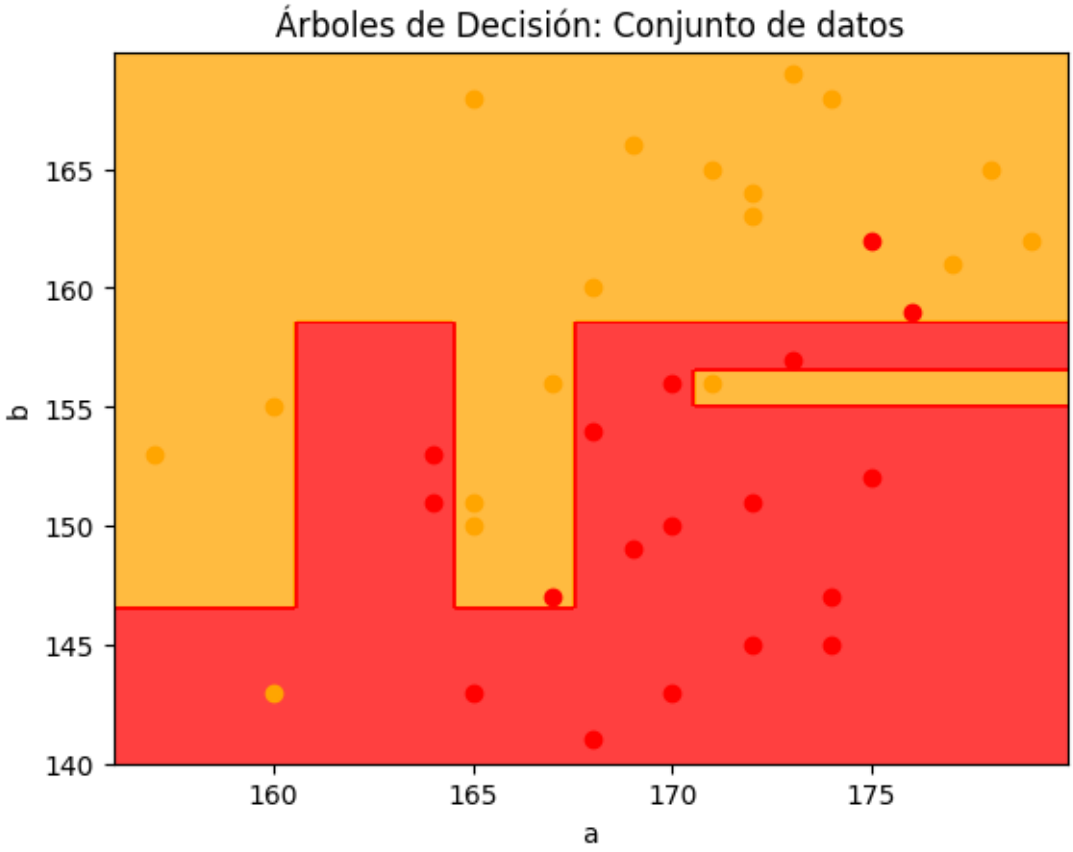
[32]:
```
# Al construir la malla, se colorea la región de naranja o rojo
# de acuerdo al clasificador DT obtenido
plt.contourf(X1, X2,
    clasificador.predict(
        np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
        alpha = 0.75, cmap = ListedColormap(('orange', 'red'))
)

# Se establecen los límites de los ejes x,y en el gráfico
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# Se grafica cada dato en el plano cartesiano, la clase de cada dato determina␣
 ↪el color.
# Debido al proceso de etiquetado, 'n' fue sustituido por 0 y 'r' sustituido␣
 ↪por 1
# 0 -> Naranja
# 1 -> Rojo
j=0
for i in y_set:
    if i==0:
        color = "orange"
    else:
        color = "red"
    plt.scatter(
        X_set.iloc[j,0],
        X_set.iloc[j,1],
        c = color,
        label = i
    )
    j=j+1

# Etiqueta del gráfico y sus ejes
plt.title('Árboles de Decisión: Conjunto de datos')
plt.xlabel('a')
plt.ylabel('b')

# Creación del gráfico
plt.show()
```



4 Clasificar nuevos datos con DT

4.1 Se clasifica un dato con el clasificador construido con DT

dato = (160, 145)

[33]:
```
# Predicción del dato = (160, 145)
x = clasificador.predict([[160, 145]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
rojo
```

4.2 Se clasifica otro dato con el clasificador construido con DT

dato = (160, 165)

[34]:
```
# Predicción del dato = (160, 165)
x = clasificador.predict([[160, 165]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
naranja
```

4.3 Ahora, a manera de prueba, se clasifica el promedio de los datos

[35]:
```
# X_set es un DataFrame de pandas
X_set.mean(0)
```

[35]:
```
a    169.694444
b    155.000000
dtype: float64
```

[36]:
```
# Predicción del dato promedio = (169.6944, 155)
x = clasificador.predict([[169.6944, 155]])
if x==0:
    print('naranja')
else:
    print('rojo')
```

```
rojo
```