Diego Fernando Montaño Pérez A01282875

Aprendizaje Automático TC3020

Práctica 2

Dr. Jesús Guillermo Falcón Cardona

- Paso 1-4:

```
import sklearn
assert sklearn.__version__ >= "0.20"
import numpy as np
import os
import numpy as np
import pandas as pd
import seaborn as sns
import graphviz
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from graphviz import Source
from sklearn.tree import export_graphviz
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import datasets
from sklearn.tree import DecisionTreeRegressor

mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "decision_trees"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Estas es la parte del código en donde importe las librerías necesarias para llevar a cabo la practica y en la parte de abajo están las configuraciones para las imágenes que se crearon y guardaron al correr el código. Todo esto a excepción de las librerías fue copiado de forma exacta a como lo daba las instrucciones de la practicas, se agregaron librerías para el modelaje de los árboles de decisión.

- Paso 5-7:

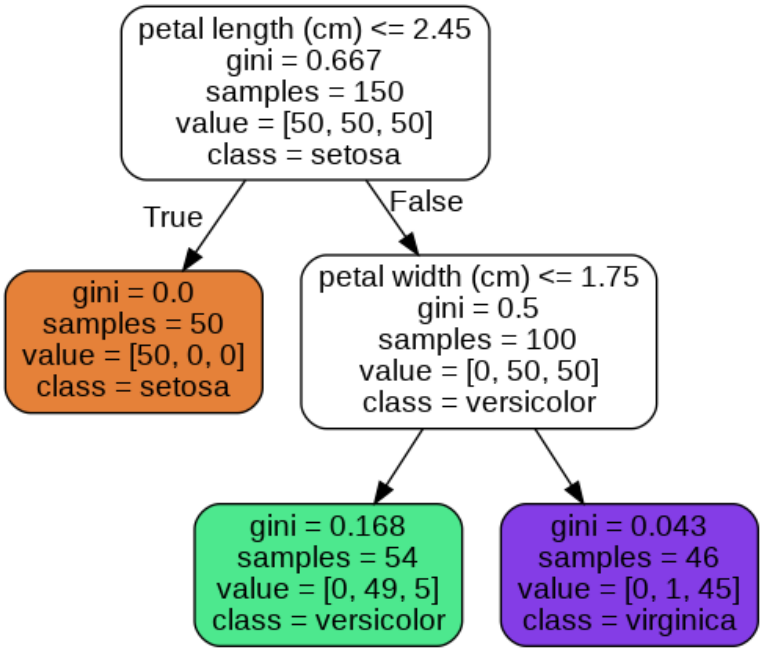
```
def IrisDS():
  iris = datasets.load_iris()
  #Modelando arbol de decision
  X = iris.data[:,2:]
  y = iris.target
  X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.2, random_state=15)
  tree_clf = tree.DecisionTreeClassifier(max_depth=2, random_state=30)
  tree_clf = tree_clf.fit(X_train, y_train)
  #Calculando precisión de modelo
  accuracy=tree_clf.score(X_test, y_test)
  print("Precision Modelo Iris:",accuracy)
  plot_decision_boundary(tree_clf,X,y)

  export_graphviz(
        tree_clf,
        out_file=os.path.join(IMAGES_PATH, "iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )

  Source.from_file(os.path.join(IMAGES_PATH, "iris_tree.dot"))
```

En estos pasos exporte el dataset Iris desde scikit-learn.org para hacer su árbol de decisión, para lograr hacer esto fue necesario crear el modelo primero, lo primero que se hizo fue darle a la variable **'x'** una matriz con toda la información de dataset y la variable **'y'** una serie que contenía los objetivos de clasificación. Con estos datos se creo el objeto **tree_clf** con ayuda de la función **DecisionTreeClassifier()** que nos dio el modelo final para dibujar el árbol.

En la parte de abajo esta el código usado para dibujar el modelo del árbol el cual se ve la siguiente manera:



(El código que se tiene creaba la imagen en versión .dot, por lo que desde terminal se tuvo que introducir el comando de `dot -Tpng iris_tree.dot -o iris_tree.png` para transformar la imagen a .png).

- Paso 8-9:

```
def IrisDS():
  iris = datasets.load_iris()
  #Modelando arbol de decision
  X = iris.data[:,2:]
  y = iris.target
  X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.2, random_state=15)
  tree_clf = tree.DecisionTreeClassifier(max_depth=2, random_state=30)
  tree_clf = tree_clf.fit(X_train, y_train)
  #Calculando precisión de modelo
  accuracy=tree_clf.score(X_test, y_test)
  print("Precision Modelo Iris:",accuracy)
```

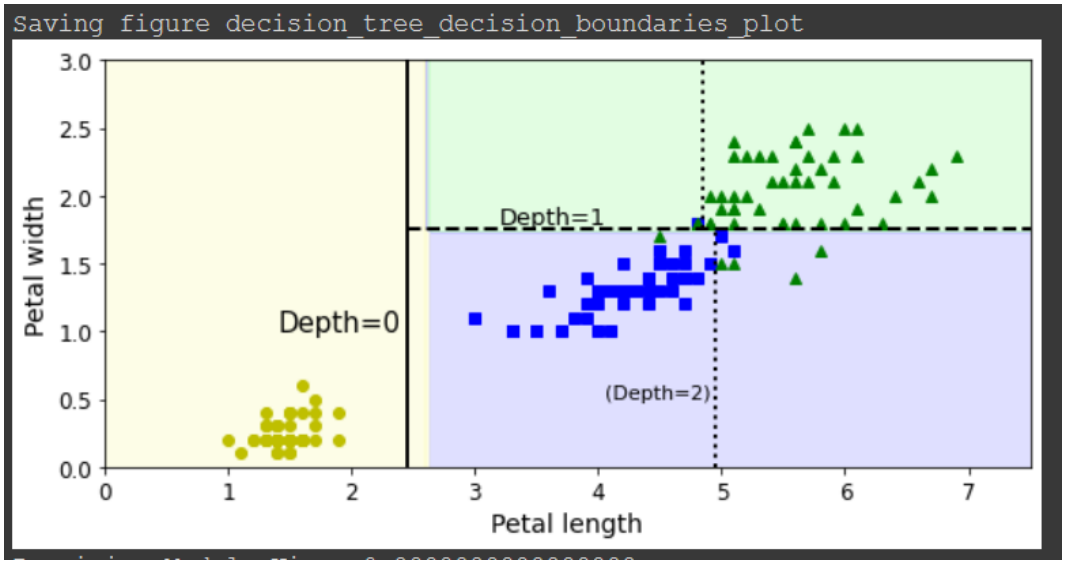
Una ves que ya tenía los datos y sabía cómo crear el modelo del árbol de clasificación, lo complete usando los conjuntos de entrenamiento y prueba. Estos conjuntos los obtuve desde la línea 3 a la 5 donde se uso el **80%** del dataset para el conjunto de entrenamiento y el resto para el de prueba, ya teniendo estos valores ajuste el árbol con el conjunto de entrenamiento y después saque la precisión del modelo usando los conjuntos de prueba. La precesión del modelo fue de **96%**, adjunto evidencia con resultado tomado desde la terminal después de correr el código:

```
Precision Modelo Iris: 0.9666666666666667
```

- Paso 10:

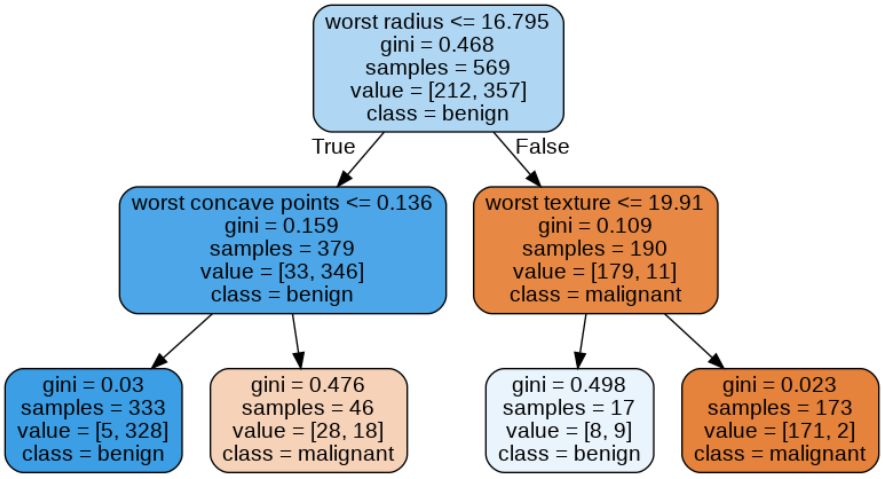
Este fue el paso final de modelo el cual servía para visualizar las particiones que generaba el árbol en cada separación de sus ramas. Para esto simplemente se uso el código dado por la practica y se le dio el modelo del árbol y las variables **X** y **Y** que se habían declarado al inicio de la función.

El resultado fue el siguiente:

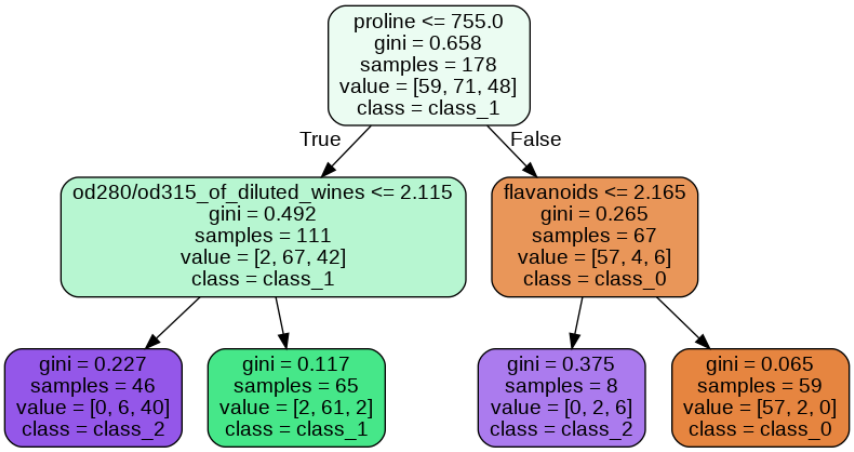


- Paso 11:

  Se aplico el mismo código a los datasets de wine y breast_cancer, se cambiaron nombres de variable para notas diferencias entre cada dataset y se tuvieron que cambiaron los nombres en el código que permitía guardar el dibujo del árbol de clasificación. Todo se dividió en funciones para tener un código limpio y que se pudieran encontrar las cosas fácilmente, en la parte de abajo se mostraran las imágenes de los arboles que se obtuvieron en los otros dos datasets:



Wine Dataset



Breast_cancer Dataset

Diego Fernando Montaño Pérez A01282875

```
def WineDS():
  wine = datasets.load_wine()
  #Modelando arbol de decision
  X = wine.data
  y = wine.target
  X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.2, random_state=15)
  tree_clf = tree.DecisionTreeClassifier(max_depth=2, random_state=30)
  tree_clf = tree_clf.fit(X_train, y_train)
  #Calculando precisión de modelo
  accuracy=tree_clf.score(X_test, y_test)
  print("Precision Modelo Wine:",accuracy)

  export_graphviz(
        tree_clf,
        out_file=os.path.join(IMAGES_PATH, "wine_tree.dot"),
        feature_names =wine.feature_names,
        class_names = wine.target_names,
        filled=True,
        rounded = True,
  )
  Source.from_file(os.path.join(IMAGES_PATH, "wine_tree.dot"))
```

```
def BreastCancerDS():
  breastCancer = datasets.load_breast_cancer()
  #Modelando arbol de decision
  X = breastCancer.data
  y = breastCancer.target
  X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.2, random_state=15)
  tree_clf = tree.DecisionTreeClassifier(max_depth=2, random_state=30)
  tree_clf = tree_clf.fit(X_train, y_train)
  #Calculando precisión de modelo
  accuracy=tree_clf.score(X_test, y_test)
  print("Precision Modelo Breast Cancer:",accuracy)

  export_graphviz(
        tree_clf,
        out_file=os.path.join(IMAGES_PATH, "breastCancer_tree.dot"),
        feature_names = breastCancer.feature_names,
        class_names = breastCancer.target_names,
        filled=True,
        rounded = True,
  )
  Source.from_file(os.path.join(IMAGES_PATH, "breastCancer_tree.dot"))
```

- Paso13:

Debido a lo forma en que programe mi practica 1 los datasets de iris y wine no funcionaban con mi método de regresión logística, solo el de breast_cancer funciono. Por otra parte, uso de la librería sciKit-learn de regresión logística si pudo procesar los tres datasets, para todas los datasets se usó un Alpha = 0.0005 y un threshold = 0.0001 para la regresión logística.
En las siguientes imágenes junte los resultados de la regresión lineal y árbol de decisión:

```
Precision Modelo Iris: 0.9666666666666667
Vector beta usando Scikit-Learn: Coeficiente = [[-2.74866104 -1.16890756]
 [ 0.08356447 -0.90803047]
 [ 2.66509657  2.07693804]] Intercepcion = [ 11.12767979   3.22717485 -14.35485463]
Taza de error usando Scikit-Learn: 0.9666666666666667
```

DataSet Iris: Como podemos observar en la imagen de arriba, la precisión de ambos modelos de predicción fue la misma, por lo tanto, ambas funcionan igual y son bastantes buenas.

```
Precision Modelo Wine: 0.8888888888888888
Vector beta usando Scikit-Learn: Coeficiente = [[-4.15379928e-02  2.16444999e-01  1.27971257e-01 -2.69114924e-01
  -2.98941106e-02  1.88828185e-01  4.42111931e-01 -2.43328547e-02
   8.27344089e-02  1.18093141e-02 -5.54601372e-03  3.15013420e-01
   8.42519774e-03]
 [ 3.78584720e-01 -7.08328572e-01 -1.33019774e-01  1.73749973e-01
   2.09538537e-02  2.51369762e-01  4.14692066e-01  2.39152350e-02
   2.82878799e-01 -1.17777133e+00  2.30169348e-01  3.91331137e-01
  -8.04013766e-03]
 [-3.37046728e-01  4.91883574e-01  5.04851782e-03  9.53649510e-02
   8.94025697e-03 -4.40197946e-01 -8.56803997e-01  4.17619765e-04
  -3.65613208e-01  1.16596202e+00 -2.24623334e-01 -7.06344556e-01
  -3.85060080e-04]] Intercepcion = [-0.03096874  0.0794164  -0.04844765]
Taza de error usando Scikit-Learn: 0.9662921348314607
```

DataSet Wine: En este dataset si hubo una diferencia en cuanto a la precisión de los modelos, el model de árbol obtuvo una precisión de 88% y la de regresión 96%. Esto nos quiere decir que para hacer predicciones para este dataset es mejor usar la regresión logística pues tiene un porcentaje menor de fallo.

```
Precision Modelo Breast Cancer: 0.9122807017543859
Vector beta usando Scikit-Learn: Coeficiente = [[ 0.94267563  0.45373706  0.28041415 -0.01624841 -0.03502947 -0.16508618
  -0.23114905 -0.09735151 -0.04852853 -0.00967746  0.04092519  0.37187903
   0.14377965 -0.10968995 -0.00316991 -0.03549209 -0.04942791 -0.01267718
  -0.01171584 -0.00329484  1.0020677  -0.5027342  -0.24872709 -0.0136379
  -0.06356793 -0.51486981 -0.64076162 -0.18733214 -0.15389115 -0.04946375]] Intercepcion = [0.17560913]
Taza de error usando Scikit-Learn: 0.9472759226713533
```

DatatSet Breast_Cancer: En este dataset se pudo también usar la regresión logística que yo programe, esta me lanzo una precisión de 94.89% las cual es muy cercana a la que me arrojo usando la versión de scikit-learn que es 94.72%, mientras que el árbol dio un porcentaje de 91% un poco menor a comparación del otro método. En este caso debido a la diferencia es mejor usar la regresión logística pero el árbol no es mala opción.

Diego Fernando Montaño Pérez A01282875

Reflexión

Tanto el modelo de árbol de decisión como el de regresión logística regresaron predicciones muy parecidas al igual que la taza de error, pero en la mayor parte de los datasets la regresión tuvo un porcentaje mayor, por lo tanto, para estos datatsets es mejor usar la regresión pero hablando mas en general todo depende del tipo de dataset que se maneje y que tanto datos tenga, una ves que se tenga un análisis de esto lo mejor sería probar ambos métodos y quedarse con el que deje un mejor resultado.