Classification of diabetes and drugs by Decision Tree model

September 29, 2024

1 Modelo de árbol de decisión (Decision Tree)

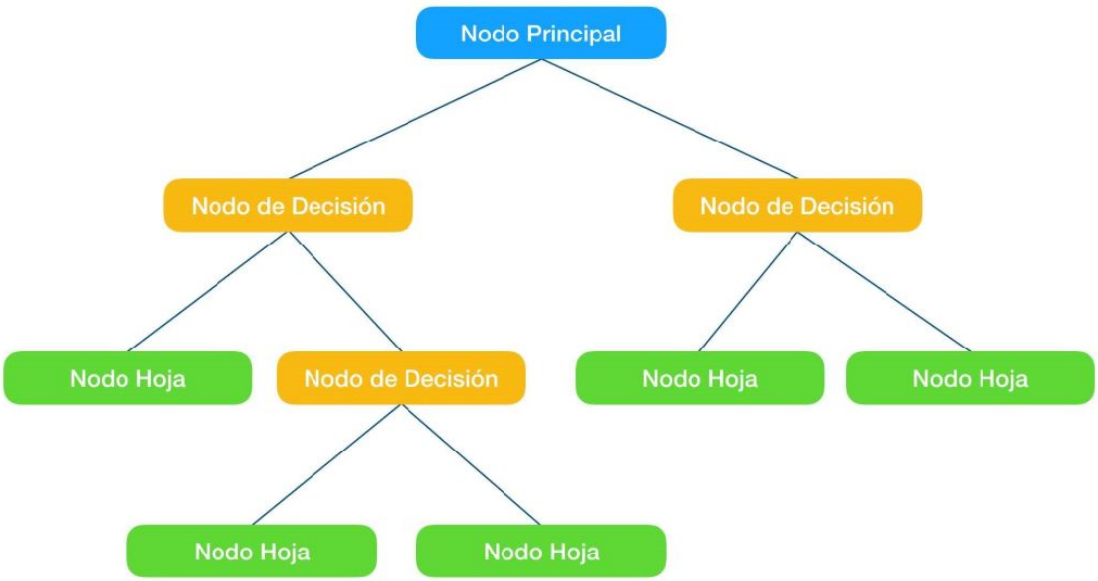
Elaborado por: **Gabriel Armando Landín Alvarado**

1.1 ¿Qué es un árbol de decisión?

Un árbol de decisión en machine learning es un modelo predictivo que se utiliza para tomar decisiones basadas en datos. Se estructura como un árbol, donde cada nodo interno representa una prueba o condición sobre un atributo, cada rama representa el resultado de la prueba, y cada nodo hoja representa una etiqueta de clase (en clasificación) o un valor continuo (en regresión). El objetivo es dividir el conjunto de datos en subconjuntos homogéneos en términos de la variable objetivo. El algoritmo más comúnmente utilizado para construir árboles de decisión es el CART (Classification and Regression Trees) 1. Su estructura es similar a un diagrama de flujo, donde:

- **Nodo raíz**: Es el punto de inicio del árbol.
- **Nodos internos**: Representan decisiones basadas en características o atributos de los datos.
- **Ramas**: Indican las posibles decisiones o caminos a seguir.
- **Nodos hoja**: Representan los resultados finales o las predicciones.

El árbol se construye dividiendo recursivamente el conjunto de datos en subconjuntos más pequeños utilizando criterios como la ganancia de información o la impureza. Este proceso continúa hasta que los nodos hoja contienen datos homogéneos o hasta que se alcanza un criterio de parada 2 3.



Algunos conceptos clave en el uso de este algoritmo es la Entropía e Impureza de Gini, podría decirse que son una medida de la cantidad de desorden o incertidumbre en un conjunto de datos. Se utilizan para determinar qué atributo dividirá mejor a los datos en subconjuntos más homogéneos. En términos simples, se puede decir que la entropía y la impureza de Gini miden la pureza de un nodo.

Definiciones más detalladas de las anteriores serían las siguientes:

Entropía: Es una medida de la aleatoriedad o desorden en los datos. En el contexto de los árboles de decisión, se calcula para cada nodo y se utiliza para decidir qué atributo dividirá mejor los datos. La fórmula de la entropía para un conjunto de datos (S) es:

$$\text{Entropía}(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Donde: - (p_i) es la proporción de elementos en la clase (i). Un nodo puro tendrá una entropía de 0.

Impureza de Gini, medida que se usa para medir la pureza de un nodo, se calcula de la siguiente manera:

$$\text{Gini} = 1 - \sum (P_i)^2$$

Donde: - (P_i) es la proporción de cada clase en el nodo. Un valor cercano a 0 indica que el nodo es puro, es decir, contiene principalmente elementos de una sola clase.

Otro concepto muy importante es la **Ganancia de Información**(Information Gain), una definición es la siguiente:

La **ganancia de información** es una métrica utilizada en algoritmos de aprendizaje automático, especialmente en la construcción de árboles de decisión. Su propósito es medir la reducción de la incertidumbre o impureza en un conjunto de datos después de dividirlo en subconjuntos basados en un atributo específico. Se utiliza para seleccionar el atributo que mejor separa los datos en subconjuntos homogéneos 4 5. La formula de la Ganancia de Información (GI por sus siglas en inglés) es la siguiente:

$$\text{IG} = H(S) - \sum_{j=1}^{k} \frac{|S_j|}{|S|} H(S_j)$$

Donde: - (S_j) es el subconjunto (j)-ésimo, ($|S_j|$) es el número de ejemplos en el subconjunto(S_j), y ($|S1$) es el número total de ejemplos en el conjunto de datos original.

Un ejemplo de lo anterior sería el siguiente:

Supongamos que tenemos un conjunto de datos con 14 ejemplos, de los cuales 10 son de una clase y 4 de otra. La entropía del nodo padre sería:

$$H(S) = -(\frac{10}{14} \log_2 \frac{10}{14} + \frac{4}{14} \log_2 \frac{4}{14})$$

Si dividimos este conjunto en dos subconjuntos con entropías calculadas, la ganancia de información se obtiene restando la entropía ponderada de estos subconjuntos de la entropía del nodo padre 6 7 .

Finalmente, podemos entender la importancia de la entropía y la impureza de Gini para construir árboles de decisión eficientes, ya que ayuda a identificar las divisiones que proporcionan la mayor cantidad de información sobre la clasificación de los datos.

1.2 Ejemplo 1: Clasificación de casos positivos y negativos para diabetes.

1.2.1 Conociendo los datos

Para el siguiente ejemplo en la aplicación de un árbol de decisión se construirá un modelo que permita realizar la predicción de diabetes en nuevos pacientes, con base en en las decisiones generadas a partir de las características o columnas de los pacientes en el conjunto de datos.

Entre las características de los pacientes que se incluyen en los datos están las siguientes:

- **Pregnancies** (embarazos)
- **Glucose** (glucosa)
- **BloodPressure** (presión arterial)
- **SkinThickness** (grosor de la piel)
- **Insulin** (insulina)
- **BMI** (Índice de Masa Corporal (IMC))
- **Pedigree** (genealogía)
- **Age** (edad)

Finalmente, la columna de resultado u objetivo:

- **Outcome**

[1]:
```
# importar las bibliotecas necesarias para explorar los datos
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data = pd.read_csv('../Data/diabetes.csv')
data.head()
```

[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
   Pedigree  Age  Outcome
0     0.627   50        1
1     0.351   31        0
2     0.672   32        1
3     0.167   21        0
4     2.288   33        1
```

[3]:
```
# información general de los datos
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Pregnancies    768 non-null    int64  
 1   Glucose        768 non-null    int64  
 2   BloodPressure  768 non-null    int64  
 3   SkinThickness  768 non-null    int64  
 4   Insulin        768 non-null    int64  
 5   BMI            768 non-null    float64
 6   Pedigree       768 non-null    float64
 7   Age            768 non-null    int64  
 8   Outcome        768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[4]:
```
# número de filas y columnas
data.shape
```

[4]:
```
(768, 9)
```

[5]:
```
# estadisticas descriptivas de las columnas
data.describe()
```

[5]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000   
mean      3.845052  120.894531      69.105469      20.536458   79.799479   
std       3.369578   31.972618      19.355807      15.952218  115.244002   
min       0.000000    0.000000       0.000000       0.000000    0.000000   
25%       1.000000   99.000000      62.000000       0.000000    0.000000   
50%       3.000000  117.000000      72.000000      23.000000   30.500000   
75%       6.000000  140.250000      80.000000      32.000000  127.250000   
max      17.000000  199.000000     122.000000      99.000000  846.000000   

              BMI    Pedigree         Age     Outcome  
count  768.000000  768.000000  768.000000  768.000000  
mean    31.992578    0.471876   33.240885    0.348958  
```

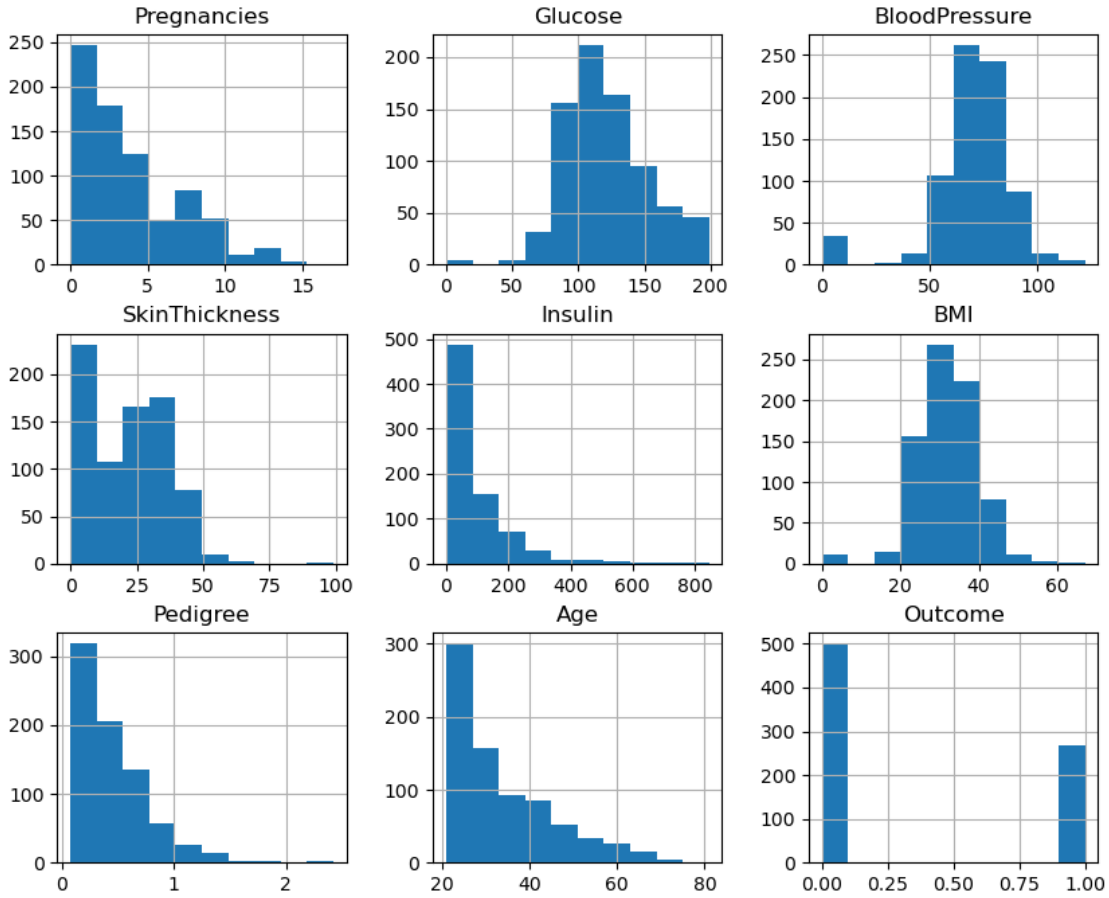
```
std       7.884160    0.331329   11.760232    0.476951
min       0.000000    0.078000   21.000000    0.000000
25%      27.300000    0.243750   24.000000    0.000000
50%      32.000000    0.372500   29.000000    0.000000
75%      36.600000    0.626250   41.000000    1.000000
max      67.100000    2.420000   81.000000    1.000000
```

[6]:
```
# ver la distribución de los valores de la variable objetivo
data['Outcome'].value_counts()
```

[6]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

Observamos que existen 268 casos positivos a diabetes y 500 negativos, lo ideal, es que las clases estén equilibradas, pero para este ejemplo se trabajará así.

[7]:
```
# veamos la distribución de los datos mediante histogramas
data.hist(figsize=(10, 8))
plt.show()
```



1.2.2 División de los datos

Dado que todas las variables son númericas, pues son con las que trabaja el modelo procedemos a realizar la división de los datos.

[8]:
```
# importar la biblioteca para dividir los datos
from sklearn.model_selection import train_test_split
```

Creamos los conjuntos de entrenamientio (train) y prueba (test) para las variable "X" que son el conjunto de variables independientes e "y" que es la variable dependiente.

[9]:
```
# creamos el conjunto de las variables independientes (X) solo eliminando la␣
 ↪variable objetivo
X = data.drop(['Outcome'], axis=1)
# variable "y" o variable objetivo
y = data['Outcome']
```

[10]:
```
# ahora dividimos las variables X e y en los conjuntos de entrenamiento y␣
 ↪prueba con una proporción de 20% para prueba,
# además aplicamos el parámetro random_state para una posible replica de la␣
 ↪práctica
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[11]:
```
# imprimimos las formasd de los datos
print("Número de filas y columnas del conjunto de entrenamiento:", X_train.
 ↪shape, y_train.shape)

print("Número de filas y columnas del conjunto de prueba:", X_test.shape,␣
 ↪y_test.shape)
```

```
Número de filas y columnas del conjunto de entrenamiento: (614, 8) (614,)
Número de filas y columnas del conjunto de prueba: (154, 8) (154,)
```

1.2.3 Implementación del modelo

Con los datos dividos, procedemos a implementar el modelo, lo primero es importar la bibliotecas necesarias.

[12]:
```
from sklearn.tree import DecisionTreeClassifier
```

[13]:
```
# instanciamos el modelo con el criterio de entropía y máxima profundidad en 4
DecTreeClass = DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

[14]:
```
DecTreeClass
```

[14]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

[15]:
```
# momento de entrenar el modelo
DecTreeClass.fit(X_train, y_train)
```

[15]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

Lo siguiente será realizar la predicción con el modelo ya entrenado.

[16]:
```
y_pred_tree = DecTreeClass.predict(X_test)
```

Podemos hacer una impresión comparativa de los valores predichos vs los valores reales.

[17]:
```
print("Valores predichos:", y_pred_tree[0:15])
print("-----------------------------------------------------------------")
print("Valores reales:", list(y_test[0:15]))
```

```
Valores predichos: [1 0 0 0 0 1 0 1 1 1 1 1 1 1 0]
-----------------------------------------------------------------
Valores reales: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0]
```

1.2.4 Evaluación del modelo.

De acuerdo con lo anterior, no parece que nuestro modelo arroje muy buenos resultados, pero para corroborar evaluemos el mismo con algunas métricas.

[18]:
```
# importarmos las métricas
from sklearn import metrics
```

[19]:
```
print("Precisión del modelo:", round(metrics.accuracy_score(y_test,␣
↪y_pred_tree), 3))
```

```
Precisión del modelo: 0.714
```

Tenemos una precisión del 71.4%. Veamos otras métricas de evaluación, mismas que se explican enseguida.

[20]:
```
print("Matriz de confusión:\n", metrics.confusion_matrix(y_test, y_pred_tree))
```

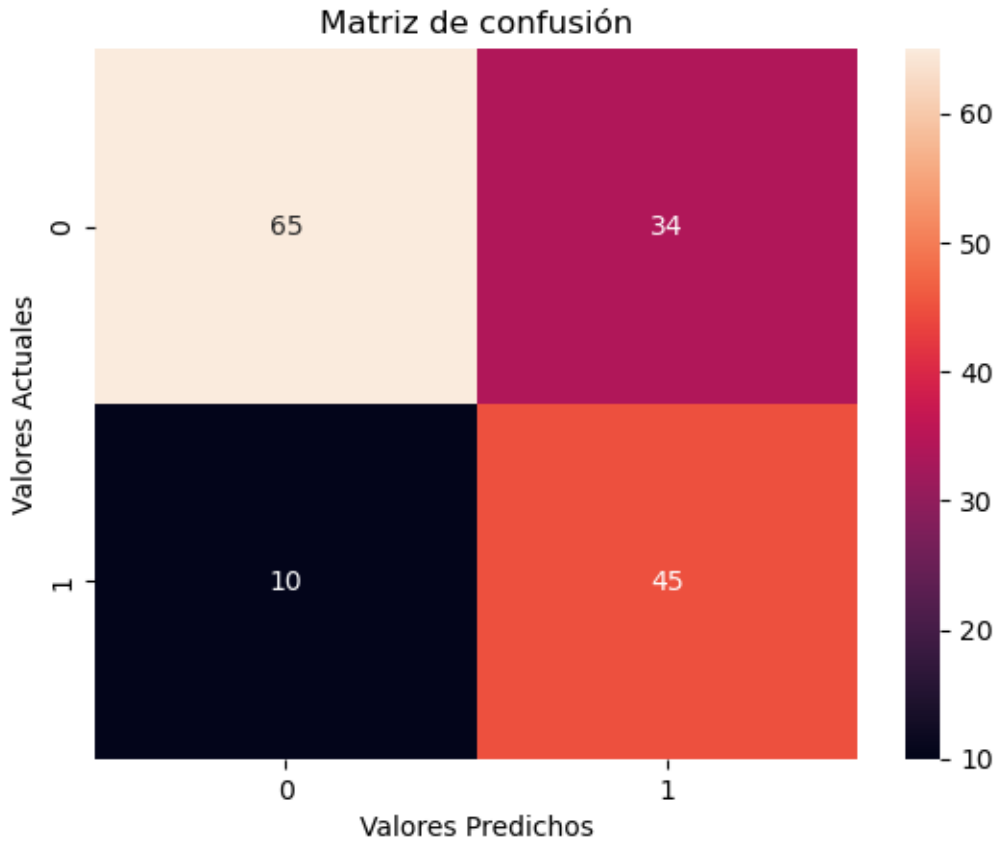
```
Matriz de confusión:
 [[65 34]
 [10 45]]
```

[21]:
```
print("Reporte de clasificación:\n\n", metrics.classification_report(y_test,␣
↪y_pred_tree))
```

```
Reporte de clasificación:

              precision    recall  f1-score   support

           0       0.87      0.66      0.75        99
           1       0.57      0.82      0.67        55
```

```
    accuracy                           0.71       154
   macro avg       0.72      0.74      0.71       154
weighted avg       0.76      0.71      0.72       154
```

[22]:
```
# realizamos un gráfico con seaborn de la matriz de confusión
sns.heatmap(metrics.confusion_matrix(y_test, y_pred_tree), annot=True,␣
 ↪cbar=True)
plt.xlabel("Valores Predichos")
plt.ylabel("Valores Actuales")
plt.title("Matriz de confusión")

plt.show()
```



La **matriz de confusión** es una herramienta fundamental para evaluar el rendimiento de un modelo de clasificación. Es una tabla que permite visualizar las predicciones correctas e incorrectas realizadas por el modelo, desglosadas por cada clase.

En una matriz de confusión de 2x2 para un problema de clasificación binaria, las filas representan las clases reales y las columnas las clases predichas. Los elementos de la matriz son:

- **True Positive (TP) o Verdaderos Positivos**: Predicciones correctas de la clase positiva.
- **True Negative (TN) o Verdaderos Negativos**: Predicciones correctas de la clase negativa.
- **False Positive (FP) o Falsos Positivos**: Predicciones incorrectas de la clase positiva (error de tipo I).
- **False Negative (FN) o Falsos Negativos**: Predicciones incorrectas de la clase negativa (error de tipo II).

Reporte de clasificación:

Las métricas que nos retorna el reporte de clasificación se explican a continuación:

- **Accuracy (Exactitud):** Proporción de predicciones correctas.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision (Precisión):** Proporción de verdaderos positivos entre todas las predicciones positivas.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall (Sensibilidad) o Tasa de Verdaderos Positivos:** Proporción de verdaderos positivos entre todas las instancias reales positivas.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **Specificity (Especificidad):** Proporción de verdaderos negativos entre todas las instancias reales negativas.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

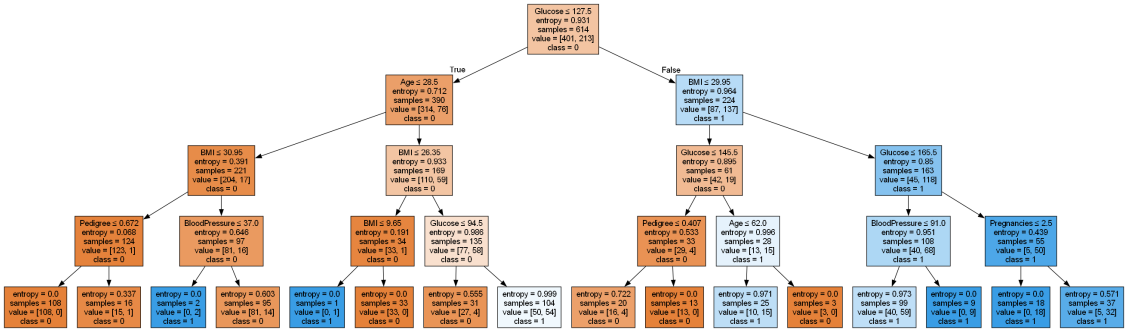
1.2.5 Generación gráfica del árbol.

[23]:
```
from sklearn.tree import export_graphviz
```

[24]:
```
import pydotplus
from IPython.display import Image
import graphviz
```

[25]:
```
export_graphviz(DecTreeClass,
                out_file='../Images/tree.dot',
                filled=True,
                feature_names=['Pregnancies', 'Glucose', 'BloodPressure',␣
 ↪'SkinThickness', 'Insulin', 'BMI', 'Pedigree', 'Age'],
                class_names=['0', '1'],
                special_characters=True
               )
!dot -Tpng tree.dot -o dec_tree.png
```

Interpretación del árbol:



Caso Positivo a Diabetes, revisión de la última rama derecha del árbol:

1. **Glucose > 127.5:** Si el nivel de glucosa es mayor a 127.5 seguimos por la derecha.
2. **BMI > 29.85:** Si el índice de masa corporal (BMI) es mayor a 29.95, continuamos por la derecha. Se revisa nuevamente la glucosa.
3. **Glucose > 165.5:** Si el nivel de glucosa es mayor a 165.5 seguimos por la derecha.
4. **Pregnancies > 2.5:** Si el valor del número de embarazos es mayor a 2.5, seguimos por la derecha.
5. **Resultado:** En este caso llegamos a una rama con hojas azules que indica que cualquier valor es de clase 1, es decir, positivo para diabetes.

Caso Negativo a Diabetes, revisión de la última rama izquierda del árbol:

1. **Glucose <= 127.5:** Si el nivel de glucosa es menor o igual a 127.5 seguimos por la izquierda.
2. **Age <= 28.5:** Si la edad es menor o igual a 28.5, continuamos por la izquierda.
3. **BMI <= 30.95:** Si el índice de masa corporal es menor o igual a 30.95 seguimos por la izquierda.
4. **Pedigree <= 0.672:** Si el valor de genealogía es menor o igual a 0.672, seguimos por la izquierda.
5. **Resultado:** En este caso llegamos a una rama con hojas naranja que indica que cualquier valor es de clase 0, es decir, negativo para diabetes.

1.3 Ejemplo 2: Clasificación para la asignación de medicamentos o drogas.

El siguiente conjunto de datos se conforma de diversas características de 200 pacientes, entre ellas están la edad, sexo, presión arterial, etc., asimismo, una columna objetivo, la cual es el tipo de medicamento o droga asignado, el modelo consiste en asignar un médicamento de acuerdo con estas características a nuevos pacientes.

A contiuación, se carga el archivo en un dataframe de pandas y se describe a detalle cada columna del mismo:

[144]:
```
data2 = pd.read_csv('../Data/drug200.csv')
# primeros 5 registros
data2.head()
```

[144]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

[145]:
```
# últimos 5 registros
data2.tail()
```

[145]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

Columnas del dataframe:

1. **Age**: La edad del paciente.
2. **Sex**: El sexo del paciente, generalmente codificado como 'M' para masculino (male) y 'F' para femenino (female).
3. **BP**: La presión arterial del paciente, que puede estar clasificada en categorías como baja, normal o alta (low, normal, high).
4. **Cholesterol**: Los niveles de colesterol del paciente, que para este caso están clasificados en dos categorías, normal y alto (normal, high).
5. **Na_to_K**: La proporción de sodio (Na) a potasio (K) en la sangre del paciente.
6. **Drug**: El tipo de medicamento o droga asignado al paciente, que es la variable objetivo en este conjunto de datos.

1.3.1 Conociendo los datos

[146]:
```
# información general de los datos
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

[147]:
```
# conteo de algunos valores de las variables categoricas
data2['Sex'].value_counts()
```

[147]:
```
Sex
M    104
F     96
Name: count, dtype: int64
```

[148]:
```
data2['BP'].value_counts()
```

[148]:
```
BP
HIGH      77
LOW       64
NORMAL    59
Name: count, dtype: int64
```

[149]:
```
data2['Cholesterol'].value_counts()
```

[149]:
```
Cholesterol
HIGH      103
NORMAL     97
Name: count, dtype: int64
```

[150]:
```
data2['Drug'].value_counts()
```

[150]:
```
Drug
drugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: count, dtype: int64
```

En las líneas de código anteriores, podemos observar la distribución o conteo de las variables categóricas del dataframe, desafortunadamente, el árbol de decisión de sklearn no maneja adecuadamente este tipo de variables, lo que implica cambiar los valores a números, para ello nos apoyaremos del método **LabelEncoder()** del modulo **preprocessing** de sklearn. Para esto, primero se dividen los datos en 'X' con las variables independientes e 'y' con la variable dependiente.

[151]:
```
X = data2[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values
X[:5]
```

[151]:
```
array([[23, 'F', 'HIGH', 'HIGH', 25.355],
       [47, 'M', 'LOW', 'HIGH', 13.093],
       [47, 'M', 'LOW', 'HIGH', 10.114],
       [28, 'F', 'NORMAL', 'HIGH', 7.798],
       [61, 'F', 'LOW', 'HIGH', 18.043]], dtype=object)
```

[152]:
```
y = data2['Drug'].values
y[:5]
```

[152]:
```
array(['drugY', 'drugC', 'drugC', 'drugX', 'drugY'], dtype=object)
```

[153]:
```
# importanmos el módulo
from sklearn import preprocessing
```

[154]:
```
# codificar la columna de sexo
le_sex = preprocessing.LabelEncoder() # instanciar el objeto
le_sex.fit(['F', 'M']) # mapeo de las etiquetas
X[:,1] = le_sex.transform(X[:,1]) # transformar las etiquetas

# codificar la columna de presión arterial
le_bp = preprocessing.LabelEncoder()
le_bp.fit(['LOW', 'NORMAL', 'HIGH'])
X[:,2] = le_bp.transform(X[:,2])

# codificar la columna de colesterol
le_chol = preprocessing.LabelEncoder()
le_chol.fit(['NORMAL', 'HIGH'])
X[:,3] = le_chol.transform(X[:,3])

X[:5]
```

[154]:
```
array([[23, 0, 0, 0, 25.355],
       [47, 1, 1, 0, 13.093],
       [47, 1, 1, 0, 10.114],
       [28, 0, 2, 0, 7.798],
       [61, 0, 1, 0, 18.043]], dtype=object)
```

1.3.2 División de los datos

[156]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[157]:
```
print("Número de filas y columnas del conjunto de entrenamiento:", X_train.
↪shape, y_train.shape)
print('\n')
print("Número de filas y columnas del conjunto de prueba:", X_test.shape,␣
↪y_test.shape)
```

```
Número de filas y columnas del conjunto de entrenamiento: (160, 5) (160,)


Número de filas y columnas del conjunto de prueba: (40, 5) (40,)
```

1.3.3 Implementación del modelo

[158]:
```
# para este ejemplo se usa el criterio de Gini
DecTreeClass2 = DecisionTreeClassifier(criterion='gini', max_depth=4)
DecTreeClass2
```

[158]:
```
DecisionTreeClassifier(max_depth=4)
```

[159]:
```
# ajustar o entrenar el modelo
DecTreeClass2.fit(X_train, y_train)
```

[159]:
```
DecisionTreeClassifier(max_depth=4)
```

[160]:
```
# predicción
y_pred_tree2 = DecTreeClass2.predict(X_test)
```

[161]:
```
# comparar algunos valores predichos vs los reales
print("Valores predichos:", y_pred_tree2[:10])
print("Valores actuales:", y_test[:10])
```

```
Valores predichos: ['drugX' 'drugY' 'drugX' 'drugC' 'drugY' 'drugY' 'drugY'
'drugX' 'drugA'
 'drugX']
Valores actuales: ['drugX' 'drugY' 'drugX' 'drugC' 'drugY' 'drugY' 'drugY'
'drugX' 'drugA'
 'drugX']
```

1.3.4 Evaluación

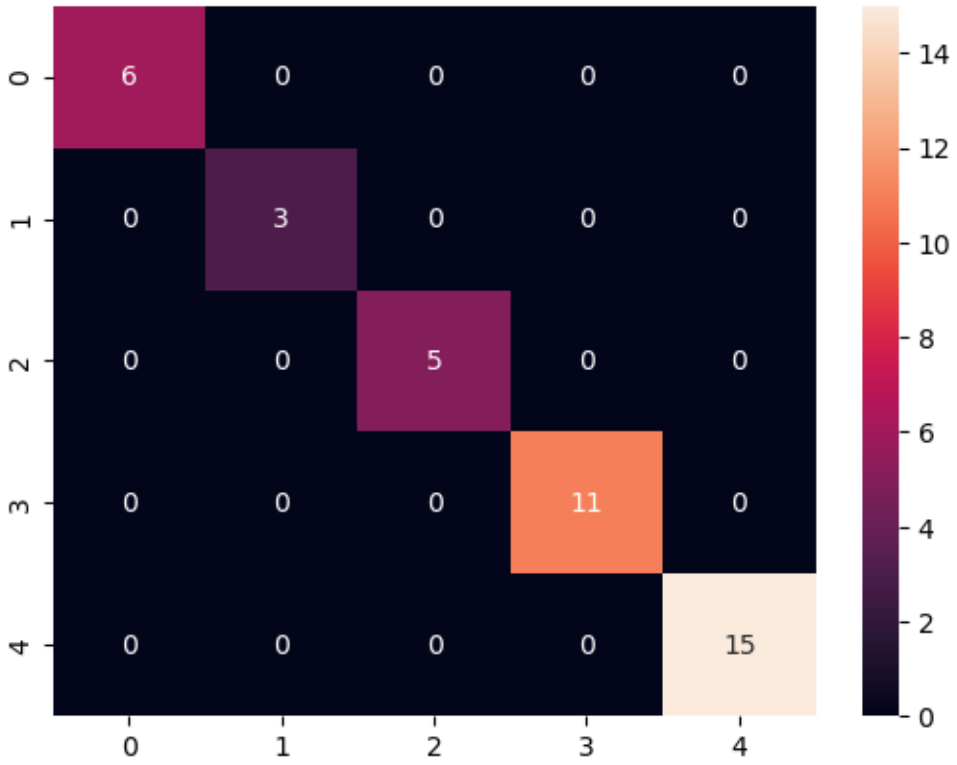
[162]:
```
print("Precisión del modelo:", metrics.accuracy_score(y_test, y_pred_tree2))
```

```
Precisión del modelo: 1.0
```

[163]:
```
print("Matriz de confusión:\n", metrics.confusion_matrix(y_test, y_pred_tree2))
```

```
Matriz de confusión:
 [[ 6  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  5  0  0]
 [ 0  0  0 11  0]
 [ 0  0  0  0 15]]
```

[164]:
```
sns.heatmap(metrics.confusion_matrix(y_test, y_pred_tree2), annot=True,␣
↪cbar=True)
plt.show()
```



[165]:
```
print("Reporte de clasificación:\n", metrics.classification_report(y_test,␣
↪y_pred_tree2))
```

```
Reporte de clasificación:
              precision    recall  f1-score   support

       drugA       1.00      1.00      1.00         6
       drugB       1.00      1.00      1.00         3
       drugC       1.00      1.00      1.00         5
       drugX       1.00      1.00      1.00        11
       drugY       1.00      1.00      1.00        15

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

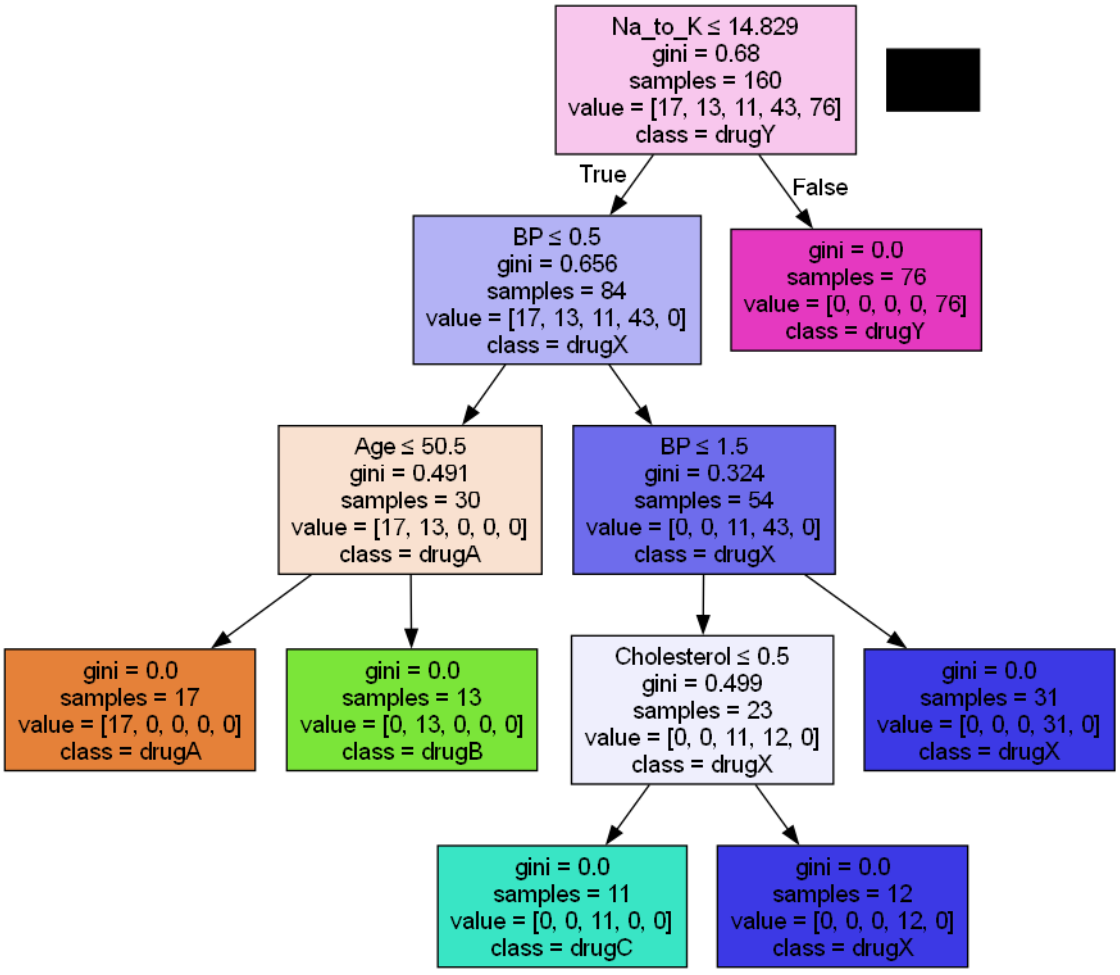
De acuerdo con las métricas anteriores, podemos observar que nuestro modelo clasifica muy bien, esto puede deberse a que son pocas características, o bien, pocos registros. Aunque en la realidad, en pocas ocasiones encontraremos situaciones como esta.

Lo siguiente será generar el gráfico o imagen del árbol.

1.3.5 Generar imagen del árbol

[170]:
```
# Exportar el árbol de decisión a formato dot
dot_data2 = export_graphviz(DecTreeClass2,
                            out_file=None,
                            feature_names=['Age', 'Sex', 'BP', 'Cholesterol',␣
 ↪'Na_to_K'],
                            class_names=['drugA', 'drugB', 'drugC', 'drugX',␣
 ↪'drugY'],
                            filled=True,
                            special_characters=True
                            )

# Visualizar el árbol
graph = pydotplus.graph_from_dot_data(dot_data2)
graph.write_png("../Images/DecTreeClass2.png")
Image(graph.create_png())
```

[170]:



En el gráfico podemos observar que para asignar el medicamento Y (drugY) basta que la proporción de sodio a potasio en la sangre del paciente sea mayor a 14.829. Otro eejmplo es la clasificación para el medicamento B (drugB), donde la proporción de sodio a potasio en sangre debe ser menor o igual a 14.829, después se evalua la presión arterial (BP), esta debe ser menor o igual a 0.5, y finalmente, se evalua la edad (Age), esta debe ser mayor a 50.5.

Elaborado por: **Gabriel Armando Landín Alvarado**

Descarga del código y datos aquí.