Práctica 8: Árbol de decisión

Profesor: Lauro Reyes Cocoletzi

Nancy Galicia, Daphne González y Diego Castro
{ngaliciac2100, , dgonzalezc2104 y dcastroe2100}@alumno.ipnx.mx

UPIIT: Unidad Profesional Interdisciplinaria en Ingeniería Campus Tlaxcala Instituto Politécnico Nacional, Tlaxcala, Tlaxcala, México 9000

Ingeniera en Inteligencia Artificial

20 de diciembre 2023

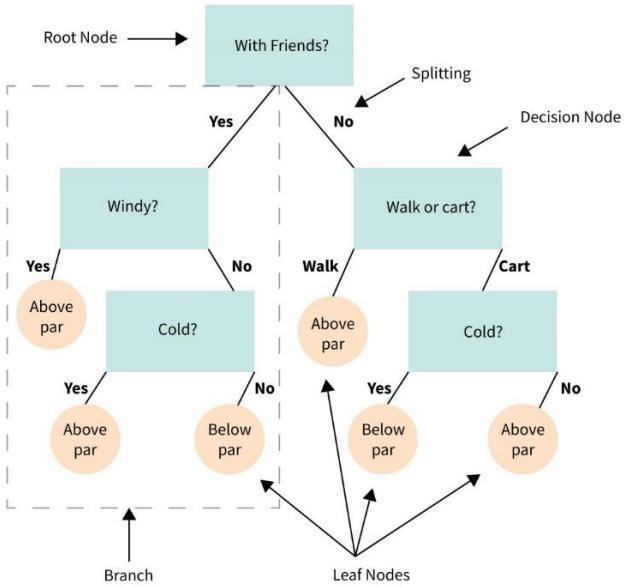
***Resumen*— Los árboles de decisión son modelos predictivos versátiles utilizados en aprendizaje automático. En la construcción de estos árboles, se busca maximizar la homogeneidad de los nodos mediante la reducción de la entropía (en clasificación) o el índice Gini. El algoritmo más comúnmente empleado es el CART (Classification and Regression Trees). Para problemas de clasificación, se utiliza el índice Gini como medida de impureza, buscando minimizar la probabilidad de error al asignar clases a instancias. En regresión, se emplean el Error Cuadrático Medio (MSE) y el Error Absoluto Medio (MAE) para evaluar la precisión de las predicciones numéricas.**

***Palabras clave* — Nodo, Gini, MSE, entropía, ramas**

I. MARCO TEORICO

A. ÁRBOLES DE DECISIÓN:

- *Definición*: Un árbol de decisión es un algoritmo de aprendizaje supervisado no paramétrico para tareas de clasificación y regresión. Tiene estructura jerárquica de nodos que representan decisiones y sus posibles consecuencias. Se utiliza para modelar decisiones y sucesos de manera lógica y visual.



- *Nodos*:
  - Nodo Raíz: El nodo principal del árbol desde el cual se ramifican las decisiones
  - Nodos Internos: Nodos que representan decisiones basadas en características específicas.
  - Nodos Hoja: Nodos terminales que representan la salida o el resultado final.
- *Ramas y Decisiones:* Las ramas conectan nodos y representan las posibles decisiones o resultados.
- *Características y Atributos:* Cada nodo interno se asocia con una característica o atributo y tiene ramas que representan los posibles valores de esa característica.
- *Criterios de División:* Se utilizan criterios para dividir los nodos internos, como la ganancia de información o la reducción de la impureza.
- *Entropía:*
  - La entropía es una medida de la impureza de un conjunto de datos en el contexto de los árboles de decisión. La fórmula para calcular la entropía se expresa de la siguiente manera:

$$Entropia(H) = -\sum_{i=1}^{c} p_i \cdot \log_2(p_i)$$

Donde:

- H es el conjunto de datos.
- c es el número de clases en el conjunto de datos.
- p_i es la proporción de instancias en la clase i en el conjunto de datos.

La entropía es máxima cuando las instancias están distribuidas de manera uniforme entre todas las clases, y es mínima cuando todas las instancias pertenecen a una sola clase.

B. CART (CLASSIFICATION AND REGRESSION TREES):

- *Clasificación*
  - Para problemas de *clasificación*, utiliza el índice Gini, el cual es otra medida de impureza utilizada en los árboles de decisión. Al igual que la entropía, el índice Gini mide la heterogeneidad o impureza de un conjunto de datos. La fórmula para calcular el índice Gini es la siguiente:

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

Donde:

- S es el conjunto de datos.
- c es el número de clases en el conjunto de datos.
- p_i es la proporción de instancias en la clase i en el conjunto de datos.

El índice Gini alcanza su valor mínimo de 0 cuando todas las instancias en el conjunto de datos pertenecen a una sola clase, es decir, cuando el conjunto es completamente puro. Por otro lado, el índice Gini es máximo (1) cuando las instancias están distribuidas de manera uniforme entre todas las clases, es decir, cuando el conjunto es completamente impuro.

- *Regresión*

Para problemas de *regresión*, en lugar de medir la impureza, utiliza el error cuadrático medio (MSE) o el error absoluto medio (MAE) para evaluar la calidad de las divisiones

- MSE

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$$

Donde:

- n es el número total de observaciones.
- y_i son los valores reales.
- $\widehat{y_i}$ son los valores predichos por el modelo.

El MSE calcula el promedio de los cuadrados de las diferencias entre los valores reales y los valores predichos. Cuanto menor sea el MSE, mejor será el rendimiento del modelo.

- MAE

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}|$$

Donde:

- n es el número total de observaciones.
- y_i son los valores reales.
- $\widehat{y_i}$ son los valores predichos por el modelo.

El MAE calcula el promedio de las diferencias absolutas entre los valores reales y los valores predichos. Al igual que el MSE, se busca minimizar el MAE.

Estas métricas son útiles para evaluar la precisión de modelos de regresión y comparar diferentes modelos. Ambas métricas penalizan los errores de predicción, pero lo hacen de manera diferente (el MSE penaliza más los errores grandes debido al cuadrado).

- *Ensemble Learning:*
  - Bosques Aleatorios (Random Forests): Extensión de CART que construye múltiples árboles y promedia sus predicciones para mejorar la generalización y reducir el sobreajuste.

C. VENTAJAS DE LOS ÁRBOLES DE DECISIÓN

1. Interpretabilidad: Los árboles de decisión son fáciles de entender y visualizar. Sus decisiones se pueden seguir de manera intuitiva.
2. Manejo de Datos Mixtos: Pueden manejar conjuntos de datos que contienen tanto variables numéricas como categóricas sin necesidad de preprocesamiento extenso.
3. No requieren escalamiento de características: No necesitan que las características se escalen o estandaricen, lo que simplifica el preprocesamiento de datos.
4. Identificación de Características Importantes: Los árboles de decisión pueden ayudar a identificar las características más importantes en el conjunto de datos, lo que puede ser valioso para comprender el problema.

D. DESVENTAJAS DE LOS ÁRBOLES DE DECISIÓN

1. Sensibilidad a Pequeñas Variaciones en los Datos: Pequeñas variaciones en los datos de entrenamiento pueden conducir a árboles de decisión diferentes. Esto los hace inestables.
2. Sobreajuste (Overfitting): Los árboles de decisión tienden a sobre ajustarse a los datos de entrenamiento si no se controla su crecimiento.
3. Incapacidad para Capturar Relaciones entre Características: Algunas relaciones más complejas entre características pueden no ser capturadas de manera efectiva por árboles de decisión, especialmente cuando las decisiones son independientes.
4. Baja Generalización: Los árboles de decisión tienden a crear modelos que no generalizan tan bien a datos no vistos como algunos otros modelos.

E. DATASET: BREAST CANCER

Scikit-learn proporciona un conjunto de datos estándar denominado "Breast Cancer" (Cáncer de Mama) que se utiliza comúnmente para problemas de clasificación. Este conjunto de datos está destinado a la tarea de clasificación binaria, donde el objetivo es predecir si un tumor es benigno (clase 0) o maligno (clase 1) en función de diversas características.

A continuación, se describen las características del conjunto de datos de cáncer de mama en scikit-learn:

- Número de Instancias: 569
- Número de Características: 30 características numéricas que describen diversas propiedades de los núcleos celulares presentes en las imágenes digitalizadas de biopsias de cáncer de mama.
- Variables de Entrada:
  - Radios, texturas y áreas de los núcleos celulares.
  - Suavidad, compacidad y concavidad de los contornos celulares.
  - Simetría y dimensión fractal.
- Variable de Salida:
  - 0 para tumores benignos.
  - 1 para tumores malignos.

II. DESARROLLO

A. ÁRBOLES DE DECISIÓN: ENTROPÍA

1. Cálculo de la entropía:

```
def entropy(y):
    classes, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    return -np.sum(probabilities * np.log2(probabilities + 1e-10))
```

1.1. En primer lugar, se calculan las clases únicas y la frecuencia de cada clase en el conjunto de etiquetas

1.2. Después se calcula las probabilidades de cada clase dividiendo la frecuencia de cada clase por el tamaño total del conjunto de etiquetas.

1.3. Por último, se calcula la entropía utilizando la fórmula de entropía de la información. La suma ponderada de las probabilidades logaritmizadas, con un pequeño término aditivo $1e^{-10}$ para evitar problemas cuando la probabilidad es cero.

2. División del conjunto de datos

```
def split(X, y, feature, threshold):
    left_mask = X[:, feature] <= threshold
    right_mask = ~left_mask
    return X[left_mask], y[left_mask], X[right_mask], y[right_mask]
```

La función *split* es responsable de dividir el conjunto de datos X e y en dos partes basándose en una característica específica y un umbral dado.

2.1. En primer lugar, se crea una máscara booleana para los datos que cumplen la condición (menor o igual al umbral)

2.2. La máscara complementaria, es decir, True para las muestras que no cumplen la condición, es decir, donde el valor de la característica es mayor al umbral.

2.3. Selecciona las muestras que cumplen la condición y sus correspondientes etiquetas y las asigna a X_left e y_left.

2.4. Selecciona las muestras que no cumplen la condición y sus correspondientes etiquetas y las asigna a X_right e y_right

3. Mejor división

```
def find_best_split(X, y):
    m, n = X.shape
    if m <= 1:
        return None, None

    parent_entropy = entropy(y)
    best_gain = 0
    best_feature = None
    best_threshold = None

    for feature in range(n):
        thresholds = np.unique(X[:, feature])
        for threshold in thresholds:
            X_left, y_left, X_right, y_right = split(X, y, feature,
threshold)

            if len(y_left) == 0 or len(y_right) == 0:
                continue

            left_weight = len(y_left) / m
            right_weight = len(y_right) / m
            gain = parent_entropy - (left_weight * entropy(y_left) +
right_weight * entropy(y_right))

            if gain > best_gain:
                best_gain = gain
                best_feature = feature
                best_threshold = threshold

    return best_feature, best_threshold
```

La función *find_best_split* se encarga de encontrar la mejor característica y umbral para dividir el conjunto de datos en dos partes, de modo que la ganancia de información sea maximizada.

3.1. $if\ m \leq 1: return\ None, None$: Verifica si el número de muestras es menor o igual a 1, en cuyo caso no es posible realizar una división y se devuelve $None, None$

3.2. $parent\_entropy = entropy(y)$: Calcula la entropía del conjunto de etiquetas original antes de la división.

3.3. $best\_gain = 0, best\_feature = None, best\_threshold = None$: Inicializa las variables que registrarán la mejor ganancia de información y las mejores características y umbral.

3.4. $thresholds = np.unique(X[:, feature])$: Obtiene los valores únicos de la característica actual para usarlos como posibles umbrales de división.

3.5. $X_{left}, y_{left}, X_{right}, y_{right} = split(X, y, feature, threshold)$: Divide el conjunto de datos en dos partes utilizando la función split.

3.6. $if\ len(y_{left}) == 0\ or\ len(y_{right}) == 0$: continue: Si una de las partes está vacía, no se considera esta división y se pasa a la siguiente iteración.

3.7. $left_{weight} = \frac{len(y_{left})}{m}, right_{weight} = \frac{len(y_{right})}{m}$: Calcula los pesos relativos de cada parte en relación con el conjunto completo.

3.8. $gain = parent\_entropy - \left(left_{weight} * entropy(y_{left}) + right_{weight} * entropy(y_{right})\right)$: Calcula la ganancia de información utilizando la entropía.

3.9. $if\ gain > best_{gain} \ldots$ Actualiza las mejores características y umbral si la ganancia de información es mayor que la mejor ganancia registrada hasta ahora.

4. Construcción del árbol recursivamente

```
def build_tree(X, y, depth=0, max_depth=None):
    if depth == max_depth or len(set(y)) == 1:
        return Node(value=np.argmax(np.bincount(y)))
    feature, threshold = find_best_split(X, y)
    if feature is None:
        return Node(value=np.argmax(np.bincount(y)))
    X_left, y_left, X_right, y_right = split(X, y, feature, threshold)
    left_subtree = build_tree(X_left, y_left, depth + 1, max_depth)
    right_subtree = build_tree(X_right, y_right, depth + 1, max_depth)

    return Node(feature=feature, threshold=threshold,
    left=left_subtree, right=right_subtree)
```

La función *build_tree* se encarga de construir el árbol de decisión recursivamente.

4.1. $if\ depth == max_{depth} or\ len(set(y)) == 1$: Esta es la condición de parada para la construcción del árbol. Si la profundidad actual alcanza la profundidad máxima o todos los ejemplos pertenecen a la misma clase, se crea un nodo hoja que contiene el valor de la clase mayoritaria utilizando $np.argmax(np.bincount(y))$.

4.2. $feature, threshold = find\_best\_split(X, y)$: Encuentra la mejor característica y umbral para dividir el conjunto de datos utilizando la función find_best_split.

4.3. $if\ feature\ is\ None: \ldots$ Si no se puede realizar más divisiones (por ejemplo, si todas las características tienen el mismo valor), se crea un nodo hoja con la clase mayoritaria.

4.4. $X_{left}, y_{left}, X_{right}, y_{right} = split(X, y, feature, threshold)$: Divide el conjunto de datos en dos partes utilizando la mejor característica y umbral.

4.5. $left\_subtree = build\_tree(X_{left}, y_{left}, depth + 1, max_{depth})$: Construye recursivamente el subárbol izquierdo llamando a la función build_tree con los datos izquierdos.

4.6. $right\_subtree = build\_tree(X_{right}, y_{right}, depth + 1, max_{depth})$: Construye recursivamente el subárbol derecho llamando a la función build_tree con los datos derechos.

4.7. $return\ Node(feature = feature, threshold = threshold, left = left_{subtree}, right = right_{subtree})$: Crea y devuelve un nodo que representa la división actual en el árbol. Este nodo tiene la información sobre la característica y el umbral utilizados para la división, así como los subárboles izquierdos y derechos construidos recursivamente.

B. ALGORITMO CART: CLASIFICACION

1. Predicción:

```
def predict(node, sample):
    if node.value is not None:
        return node.value  # Nodo hoja, devuelve el valor de predicción
    if sample[node.feature] <= node.threshold:
        return predict(node.left, sample)
    else:
        return predict(node.right, sample)
```

predict es una función que toma un nodo del árbol y una muestra, y devuelve la predicción para esa muestra según la estructura del árbol. Se realiza una búsqueda descendente a través del árbol hasta llegar a un nodo hoja, donde se devuelve el valor de predicción.

2. Construcción del arbol:

```
def build_tree(X, y):
    if len(set(y)) == 1:
        return DecisionNode(value=y[0])

    num_features = len(X[0])
    best_feature, best_threshold = None, None
    best_gini = float('inf') …
```

En *build_tree* es la función principal que construye el árbol de decisión. Utiliza el criterio de impureza de Gini para seleccionar la mejor característica y umbral en cada nodo. La construcción es recursiva y se detiene cuando todos los ejemplos pertenecen a la misma clase o se alcanza una condición de parada.

3. Cálculo de la impureza de Gini:

```
def gini_impurity(labels):
    if len(labels) == 0:
        return 0
    p_true = sum(label == 1 for label in labels) / len(labels)
    p_false = 1 - p_true
    return 1 - (p_true**2 + p_false**2)
```

En *gini_impurity* calcula la impureza de Gini para un conjunto de etiquetas. En el contexto del árbol de decisión, se utiliza para evaluar la impureza de las divisiones y seleccionar la que maximice la ganancia de información.

C. ALGORITMO CART: REGRESION

1. Cálculo del error cuadrático medio:

```
    def mse(y):
        if len(y) == 0:
            return 0
        mean = np.mean(y)
        return np.mean((y - mean)**2)
```

mse es una función que calcula el error cuadrático medio para un conjunto de etiquetas y. En el contexto del árbol de regresión, se utiliza para evaluar la calidad de las divisiones y seleccionar la que maximice la reducción en el MSE.

2. Mejor división

```
def find_best_split(X, y):
    m, n = X.shape
    if m <= 1:
        return None, None  # No se puede dividir

    parent_mse = mse(y)
    best_reduction = 0
    best_feature = None
    best_threshold = None

    for feature in range(n):
        thresholds = np.unique(X[:, feature])
        for threshold in thresholds:
            X_left, y_left, X_right, y_right = split(X, y, feature,
threshold)

            if len(y_left) == 0 or len(y_right) == 0:
                continue

            reduction = parent_mse - (len(y_left) / m * mse(y_left)
+ len(y_right) / m * mse(y_right))

            if reduction > best_reduction:
                best_reduction = reduction
                best_feature = feature
                best_threshold = threshold

    return best_feature, best_threshold
```

Del mismo modo que se calculó en la sección de clasificación con entropía, se realizaron los ajustes para que se utilice el criterio de reducción en el MSE para evaluar la calidad de las divisiones.

III. RESULTADOS

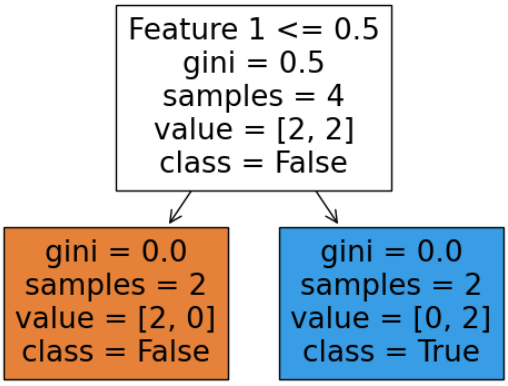
A. Ejemplo de Scikit Learn

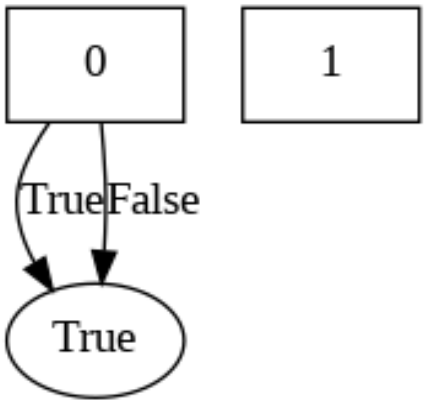
Para ejemplificar el funcionamiento del algoritmo CART, se generó primero un pequeño conjunto de datos con números binarios para su clasificación.

```
X = [[0, 0], [1, 1], [1, 0], [0, 1]]

y = [0, 1, 1, 0]  # Clases correspondientes: 0 para 'False', 1 para 'True'
```

En este pequeño ejemplo se hace una división sobre la característica 1 (feature 1) a la mitad. El conjunto de este primer nodo contiene 4 instancias De aquí se desprenden dos hojas, a las cuales se les calculo una impureza de 0 en ambos casos lo que significa que las instancias en ambos subgrupos se clasifican correctamente.

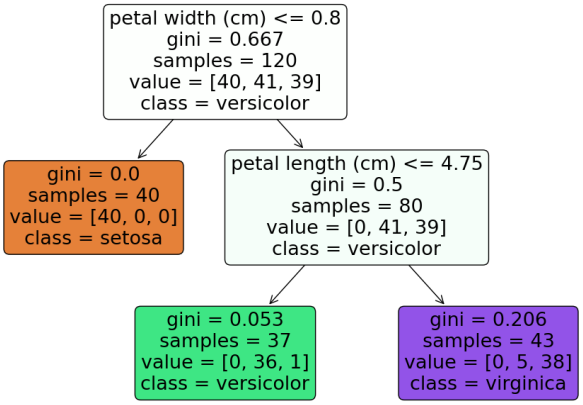




B. Dataset: *IRIS*

Inicialmente al probar el algoritmo con el dataset Iris, obtenemos un primer nodo que hace una primera partición en la característica petal_width en un valor de 0.8. En este punto se calcula el grado de impureza Gini de ambos lados de la división.

Recordemos que un índice Gini igual a 0 significa que todas las instancias en ese lado de la partición corresponden a la misma clasificación. Bajo esta lógica podemos deducir entones que todas las instancias que se encuentran en ese lado de la partición pertenecen a una sola clase que es setosa, tal como indica el árbol de la siguiente imagen.



Podemos considerar la parte antes mencionada como una hoja del árbol, sin embargo, aún es posible seguir haciendo particiones en el lado que aún tiene impurezas para hacer una mayor clasificación, entonces el otro lado de la partición inicial la Podemos considerar como un nuevo nodo.

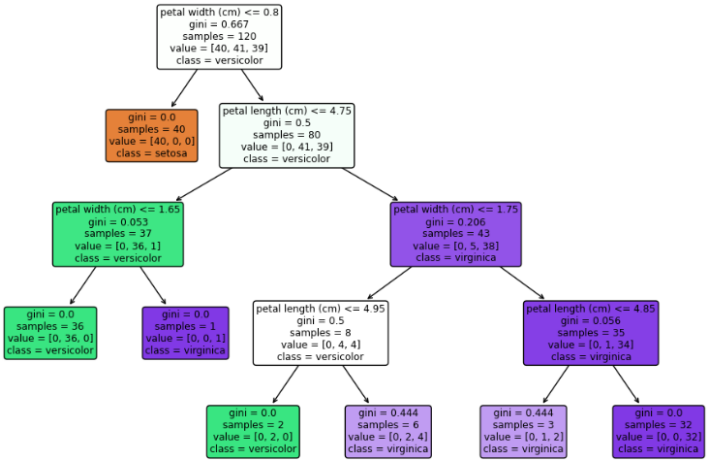
Al igual que en el primer nodo, a partir de aquí se hace una nueva partición, pero en la característica de petal_length en las que se calcula nuevamente el grado de impureza gini de ambas partes. En este caso se calculó un índice Gini de 0.053 del lado derecho que, aunque no es igual a 0 es un valor muy pequeño que lo cual indica poca impureza y se puede considerar a ese conjunto de la clase versicolor.

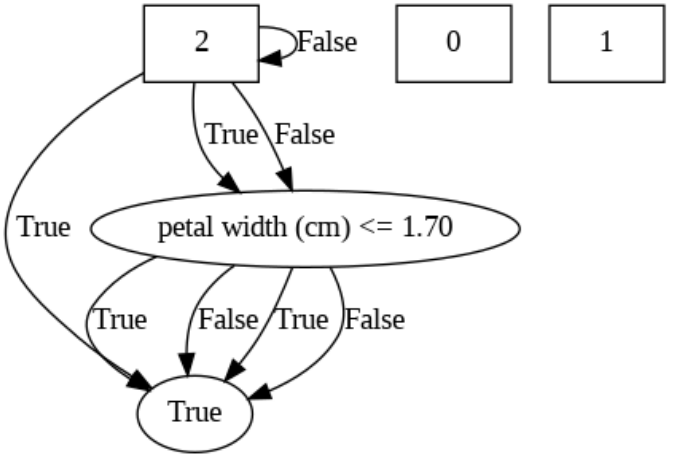
Por otro lado, en el lado de la partición restante se calculó un grado de impureza de 0.206. Un valor no tan pequeño pero la

mayoría de las instancias en ese grupo corresponden a la clase virginica.

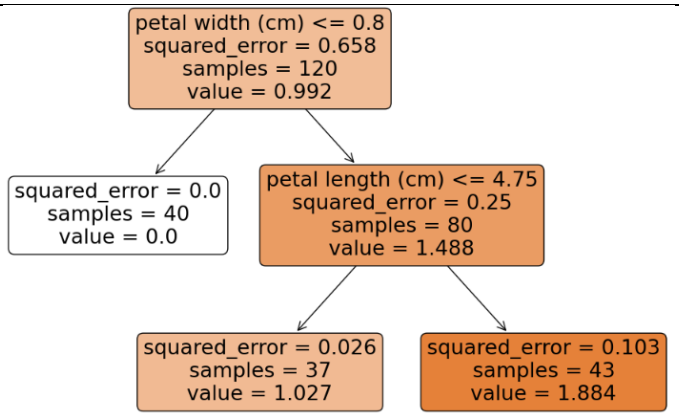
Cabe resaltar que el árbol de decisión anterior y sus resultados se calcularon para una profundidad de 2 niveles, por lo que aún es posible hacer más divisiones para obtener grados de impureza más pequeños y considerar aún más características del dataset.

Si consideramos una profundidad igual a 4 para el mismo dataset encontramos valores de Gini mucho más pequeños que en los algunos casos podrían traducirse en una mejor clasificación, sin embargo, al explorar a una mayor profundidad podemos caer en un problema de sobre ajuste.





Utilizando los árboles de regresión que nos proporciona sklearn, tenemos un valor para MSE de 0.0111845 y un arbol de decisión similar al obtenido anteriormente para una profundidad de 2.

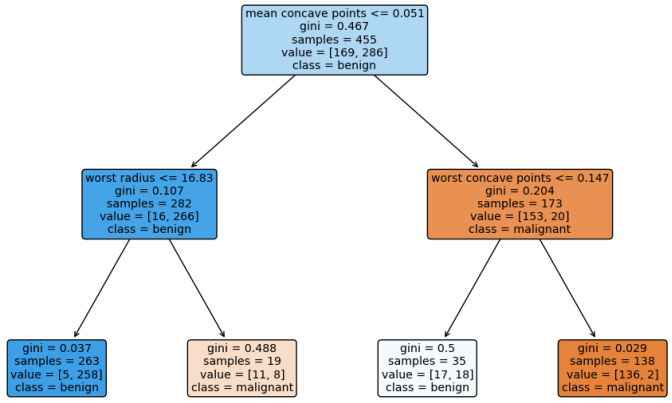


Sin utilizar la herramienta especializada de sklearn, calculamos un valor de 0.0311845 para MSE, un valor exactamente igual al calculado por sklearn.

C. Dataset: *BREAST_CANCER*

Probando con un caso más práctico, utilizamos el dataset breast_cancer de la librería scikitlearn para la detección de cáncer malignos y benignos. En este caso el código se ejecutó para calcular un árbol con profundidad igual a 2.

En este caso la primera partición se hizo sobre la característica mean_concave_points en un valor de 0.051. Hasta este momento la información obtenida no es suficiente como para tener una clasificación correcta (aunque clasifica hasta el momento a todas las instancias como benignas). Entonces se desprenden dos nuevos nodos de la primera partición. En el nodo de la izquierda se calculó una impureza de 0.107, lo suficiente como clasificar a ese grupo de instancias como Benignos, sin embargo, aún es posible hacer más particiones para obtener grupos más puros. En este segundo nodo se aplica una división cobre la característica worst_radius en el valor 16.83. Como resultado, en sus hojas tenemos un grupo con valores Gini de 0.037 y 0.488. en este caso la hoja que tiene un valor de impureza menor se considera como benigno y la otra hoja como maligno.

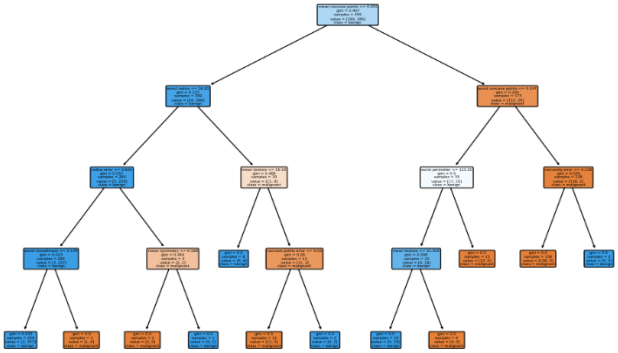


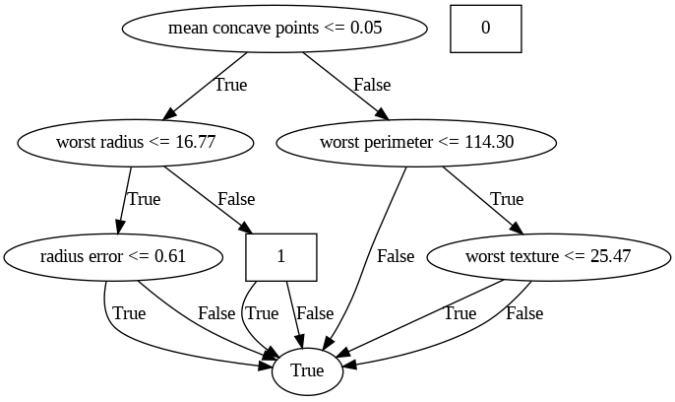
Por otro lado, en la hoja de la derecha derivada del primer nodo, las instancias se clasificaron como malignos. Al partir este conjunto con la característica worst_concave_points obtenemos dos hojas de valor Gini igual a 0.5 y 0.029, mismas que se

clasificaron como grupos de instancias benignas y malignas respectivamente.

Como se mencionó en el caso del dataset Iris, este proceso se puede iterar muchas veces cambiando el valor de la profundidad a la que queremos llegar con el árbol.

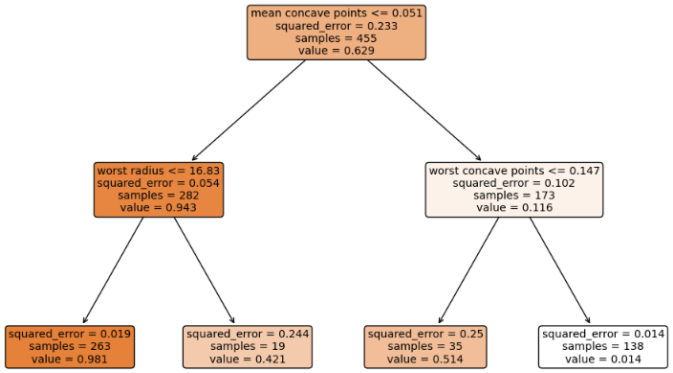
La siguiente imagen muestra el árbol del misma dataset pero calculando una profundidad de 4. Donde los valores de Gini en las hojas se reducen en su mayoría a 0, sin embargo, esto puede significar que esta cayendo en sobreajuste





Después pasamos a probar el dataset con los árboles de regresión. Recordemos que en este caso los nodos se dividen de tal forma que se minimice el error de predicción de cada nodo. En lugar de predecir clases, los usamos para predecir valores numéricos. Al final, el valor aproximado de MSE es 0.061060.

Comprobamos este valor con sklearn y obtenemos un árbol con estructura similar al CART que utiliza Gini y un resultado para MSE de 0.058993, un valor aproximado al calculado en el código.



IV. CONCLUSIONES

Recapitulando, se implementaron tres códigos para construir árboles de decisión utilizando diferentes criterios de división y aplicándolos a diferentes problemas.

En el primer código, se utilizó el criterio de entropía para construir un árbol de clasificación. La entropía se calculó para medir la impureza de los nodos, y se seleccionaron las divisiones que maximizaban la ganancia de información. Este enfoque es adecuado para problemas de clasificación y se demostró con éxito en un conjunto de datos de ejemplo.

En el segundo código, se implementó el algoritmo CART (Classification and Regression Trees) utilizando el criterio de Gini para medir la impureza de los nodos. Se buscó la división que minimizara la impureza ponderada en cada nodo, lo que resulta en un árbol de clasificación. Este método también se aplicó a un conjunto de datos de clasificación y demostró su capacidad para construir un árbol efectivo.

En el tercer código, se extendió el enfoque a problemas de regresión utilizando el criterio del error cuadrático medio (MSE). El árbol de regresión se construyó optimizando las divisiones que minimizaban la reducción en el MSE. Este enfoque se aplicó con éxito a un conjunto de datos de regresión, demostrando su capacidad para modelar relaciones no lineales y realizar predicciones precisas.

V. BIBLIOGRAFIA

[1] Sotaquirá M. “Regresión con Árboles de Decisión: el algoritmo CART”. codificandobit Accedido el 20 de diciembre de 2023. [En línea]. Disponible: https://www.codificandobits.com/blog/regresion-arboles-decision-algoritmo-cart/#otros-art%C3%ADculos-que-te-pueden-interesar

[2] Saini A. “Decision Tree Algorithm - A Complete Guide”. Analytics vidhya. Accedido el 20 de diciembre de 2023. [En línea]. Disponible: https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/