

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México

Inteligencia artificial avanzada para la ciencia de datos I
TC3006C, Grupo 101



Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)

Jorge Adolfo Ramírez Uresti

**Momento de Retroalimentación: Módulo 2 Implementación
de una técnica de aprendizaje máquina sin el uso de un
framework. (Portafolio Implementación)**

Adolfo Sebastián González Mora | A01754412

1 de Septiembre 2024

Introducción

El objetivo de este proyecto es desarrollar una implementación manual de uno de los algoritmos fundamentales en el ámbito del aprendizaje automático: el árbol de decisión. Este enfoque se ha realizado sin recurrir a bibliotecas o frameworks especializados, permitiendo una comprensión profunda y detallada de los conceptos y procesos que subyacen a este método de clasificación y regresión.

Los árboles de decisión son modelos predictivos ampliamente utilizados debido a su interpretabilidad y facilidad de uso. Se basan en la creación de una estructura jerárquica donde los datos se dividen sucesivamente según ciertas características, facilitando la toma de decisiones basadas en condiciones lógicas simples. Este tipo de algoritmo es especialmente útil en diversas aplicaciones como el diagnóstico médico, la evaluación de riesgos financieros y la toma de decisiones empresariales, entre otros.

La implementación manual de un árbol de decisión en este proyecto abarca todas las etapas clave del proceso, incluyendo:

- **Preparación y limpieza de datos:** asegurando que el conjunto de datos esté en un formato adecuado para el análisis, manejando valores faltantes y normalizando variables según sea necesario.
- **Selección de características y criterios de división:** utilizando métricas como la *entropía* y la *ganancia de información* para determinar las divisiones óptimas en cada nodo del árbol.
- **Construcción del árbol:** desarrollando un algoritmo recursivo que crea la estructura del árbol mediante la división iterativa del conjunto de datos hasta alcanzar condiciones de parada definidas.
- **Poda del árbol:** implementando técnicas para reducir el sobreajuste y mejorar la capacidad de generalización del modelo.
- **Evaluación del modelo:** aplicando métricas de rendimiento como la precisión, la sensibilidad, la especificidad y la matriz de confusión para evaluar la efectividad del árbol de decisión en tareas de clasificación.

A lo largo de este documento, se detallará cada uno de estos pasos, proporcionando explicaciones teóricas y ejemplos prácticos que ilustran el proceso completo de creación y validación del modelo. Asimismo, se discutirán los resultados obtenidos y se explorarán posibles mejoras y aplicaciones futuras del algoritmo desarrollado.

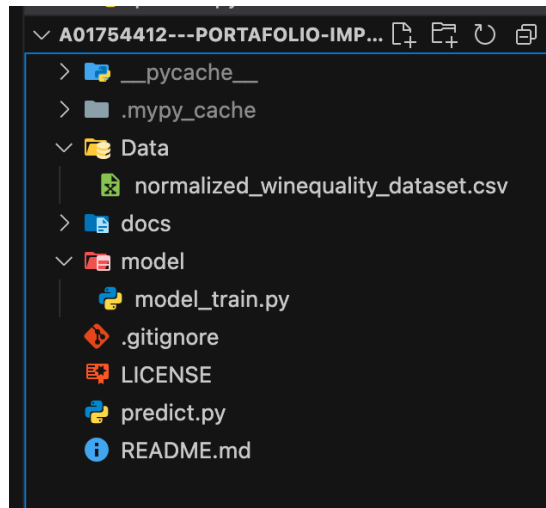
Desarrollo del Proyecto

El desarrollo de este proyecto se organizó alrededor de una estructura modular compuesta por varios archivos Python (`.py`) interrelacionados. Cada archivo fue diseñado con un propósito específico dentro del proceso de aprendizaje automático, permitiendo una clara separación de responsabilidades y facilitando tanto el mantenimiento como la expansión futura del código. Esta estructura refleja un enfoque riguroso hacia las mejores prácticas de desarrollo de software, así como la preferencia por una organización del código que maximice su legibilidad y reutilización.

La organización del proyecto es la siguiente:

- **Data:** Contiene el conjunto de datos normalizado (`normalized_winequality_dataset.csv`), que es fundamental para el entrenamiento y la evaluación del modelo.
- **docs:** Directorio reservado para la documentación y cualquier referencia adicional que apoye el desarrollo y la implementación del proyecto.
- **model:** Directorio dedicado a la implementación y entrenamiento del modelo. Contiene el script `model_train.py`, encargado de la construcción y entrenamiento del árbol de decisión sobre el conjunto de datos proporcionado.
- **predict.py:** Este archivo se utiliza para realizar predicciones utilizando el modelo entrenado, permitiendo la aplicación del árbol de decisión a nuevos datos.
- **README.md:** Proporciona una descripción general del proyecto, instrucciones sobre cómo ejecutar el código, y otros detalles relevantes para los usuarios que deseen comprender o reutilizar el trabajo realizado.

Además, se incluyeron archivos como `.gitignore` y `LICENSE`, que son esenciales para la gestión del proyecto en entornos colaborativos y para garantizar el cumplimiento de las licencias de software.



Explicación del código

En el archivo 'model_train.py', me encargué de crear y entrenar un árbol de decisión para predecir la calidad del vino usando distintas características. El primer paso fue evaluar qué tan mezclados estaban los datos, es decir, ver si los vinos de distintas calidades estaban bien separados o no; esto es importante porque, para que el árbol de decisión funcione bien, necesito que los datos se dividan en grupos que sean lo más homogéneos posible, lo que facilita hacer predicciones.

Después de eso, el código busca la mejor manera de dividir los datos en dos grupos basándose en alguna característica, como el nivel de acidez del vino, la idea aquí es encontrar la mejor forma de separar los vinos de alta calidad de los que no lo son, para que el árbol pueda tomar decisiones más precisas.

La parte central del código es la construcción del árbol de decisión; básicamente, el árbol está formado por una serie de preguntas sobre las características del vino (por ejemplo, "¿El nivel de acidez es menor a 3?"). Dependiendo de cómo se respondan estas preguntas, el árbol sigue por una rama u otra hasta llegar a una conclusión sobre la calidad del vino.

Una vez que construí el árbol, lo utilicé para hacer predicciones sobre nuevos vinos, el proceso es bastante directo: sigo el camino del árbol, respondiendo las preguntas necesarias hasta que llego a una predicción final.

Finalmente, para asegurarme de que el árbol de decisión funcionara bien, evalué su rendimiento, como también me fijé en qué tan seguido acertaba en sus predicciones, no solo con los datos de entrenamiento, sino también con datos nuevos que no había visto antes. Esta evaluación es crucial para entender si el modelo es efectivo o si necesita algún ajuste.

En resumen, el código que implementé crea un árbol de decisión desde cero, lo entrena con datos de vinos y luego lo usa para predecir la calidad de nuevos vinos, asegurándome de que las predicciones sean lo más precisas posible. Todo esto me permite entender y aplicar los principios básicos de los árboles de decisión sin depender de bibliotecas externas.

```

import numpy as np
import pandas as pd

def gini_impurity(y):
    classes = np.unique(y)
    gini = 1.0
    for c in classes:
        p = np.sum(y == c) / len(y)
        gini -= p ** 2
    return gini

def information_gain(y, left_y, right_y):
    p = len(left_y) / len(y)
    return gini_impurity(y) - (p * gini_impurity(left_y) + (1 - p) *
gini_impurity(right_y))

def best_split(X, y):
    best_gain = 0
    best_split = None
    for column in range(X.shape[1]):
        unique_values = np.unique(X[:, column])
        for val in unique_values:
            left_mask = X[:, column] <= val
            right_mask = X[:, column] > val
            left_y = y[left_mask]
            right_y = y[right_mask]
            if len(left_y) == 0 or len(right_y) == 0:
                continue
            gain = information_gain(y, left_y, right_y)
            if gain > best_gain:
                best_gain = gain
                best_split = {
                    'column': column,
                    'value': val,
                    'left_mask': left_mask,
                    'right_mask': right_mask
                }
    return best_gain, best_split

def build_tree(X, y, depth=0, max_depth=5):
    if depth == max_depth or len(np.unique(y)) == 1:
        return np.argmax(np.bincount(y))

    gain, split = best_split(X, y)
    if gain == 0:

```

```

        return np.argmax(np.bincount(y))

    left_tree = build_tree(X[split['left_mask']],
                           y[split['left_mask']], depth + 1, max_depth)
    right_tree = build_tree(X[split['right_mask']],
                           y[split['right_mask']], depth + 1, max_depth)

    return {
        'column': split['column'],
        'value': split['value'],
        'left': left_tree,
        'right': right_tree
    }

def predict_tree(tree, X):
    if isinstance(tree, dict):
        if X[tree['column']] <= tree['value']:
            return predict_tree(tree['left'], X)
        else:
            return predict_tree(tree['right'], X)
    else:
        return tree

def predict_tree_batch(tree, X):
    return np.array([predict_tree(tree, row) for row in X])

def evaluate_performance(actual_labels, predicted_labels):
    accuracy = np.mean(actual_labels == predicted_labels)

    if np.sum(predicted_labels == 1) == 0:
        precision = 0
    else:
        precision = np.sum((actual_labels == 1) & (
            predicted_labels == 1)) / np.sum(predicted_labels == 1)

    if np.sum(actual_labels == 1) == 0:
        recall = 0
    else:
        recall = np.sum((actual_labels == 1) & (
            predicted_labels == 1)) / np.sum(actual_labels == 1)

    if (precision + recall) == 0:
        f1_score = 0

```

```

    else:
        f1_score = 2 * precision * recall / (precision + recall)

    conf_matrix = pd.crosstab(np.array(actual_labels), np.array(
        predicted_labels), rownames=['Actual'], colnames=['Predicted'])

    return accuracy, precision, recall, f1_score, conf_matrix

data = pd.read_csv(

'/Users/aguero/Desktop/A01754412---Portafolio-Implementaci-n/Data/normalized_winequality_dataset.csv')

X = data.drop('quality', axis=1).values
y = data['quality'].values

train_size = int(0.6 * len(data))
val_size = int(0.2 * len(data))

X_train, y_train = X[:train_size], y[:train_size]
X_val, y_val = X[train_size:train_size +
                    val_size], y[train_size:train_size + val_size]
X_test, y_test = X[train_size + val_size:], y[train_size + val_size:]

max_depth = 6
decision_tree = build_tree(X_train, y_train, max_depth=max_depth)

y_val_pred = predict_tree_batch(decision_tree, X_val)

val_accuracy,    val_precision,    val_recall,    val_f1,    val_conf_matrix    =
evaluate_performance(
    y_val, y_val_pred)

print("Evaluación en el conjunto de validación:")
print(f"Precisión (Accuracy): {val_accuracy}")
print(f"Precisión: {val_precision}")
print(f"Recall: {val_recall}")
print(f"F1 Score: {val_f1}")
print(f"Matriz de Confusión:\n{val_conf_matrix}")

```

```
y_test_pred = predict_tree_batch(decision_tree, X_test)
test_accuracy, test_precision, test_recall, test_f1, test_conf_matrix =
evaluate_performance(
    y_test, y_test_pred)

print("\nEvaluación en el conjunto de prueba:")
print(f"Precisión (Accuracy): {test_accuracy}")
print(f"Precisión: {test_precision}")
print(f"Recall: {test_recall}")
print(f"F1 Score: {test_f1}")
print(f"Matriz de Confusión:\n{test_conf_matrix}")
```

NOTA: Al momento de cargar el código decidí eliminar los comentarios para que no se hiciera tan extenso el documento.

El archivo 'predict.py' lo creé para predecir la calidad del vino usando un modelo de árbol de decisión que ya había entrenado antes; primero, cargo un dataset de vinos desde un archivo CSV que incluye varias características de cada vino, como su nivel de acidez, azúcar residual, pH, entre otros, junto con la etiqueta que indica la calidad del vino.

Una vez que tengo el dataset cargado en un DataFrame de pandas, separo las características de la etiqueta de calidad, luego, divido los datos, utilizando un 60% para entrenar el modelo de árbol de decisión. Esta división es importante para asegurar que el modelo tenga suficiente información para aprender y, a la vez, pueda generalizar bien cuando se enfrente a nuevos datos.

Con los datos de entrenamiento, construyo un árbol de decisión, aquí, el árbol se forma haciendo una serie de divisiones basadas en las características del vino, con el objetivo de separar los vinos buenos de los malos. Para evitar que el modelo se ajuste demasiado a los datos específicos de entrenamiento, establecí una profundidad máxima para el árbol, de modo que no se vuelva demasiado complejo y pueda generalizar mejor.

Después de entrenar el modelo, definí manualmente un conjunto de características para un vino específico. Este conjunto está representado por un array que contiene los valores correspondientes a cada una de las propiedades del vino, como la acidez, el pH, entre otras.

Antes de usar estas características para hacer una predicción, es necesario normalizarlas, esto es clave para asegurar que las nuevas características estén en la misma escala que los datos con los que entrené el modelo. Para eso, utilizo las medias y desviaciones estándar que calculé a partir del dataset de entrenamiento.

Una vez que las características están normalizadas, el modelo de árbol de decisión las toma y predice la calidad del vino; si el modelo predice un '1', eso significa que el vino es "bueno"; si predice cualquier otro valor, entonces el vino es "malo".

Finalmente, el código imprime el resultado de la predicción, indicando si el vino es bueno o malo según lo que el modelo ha aprendido.

```
import numpy as np

import pandas as pd

from model.model_train import build_tree, predict_tree

def load_trained_tree():

    data = pd.read_csv(

        '/Users/aguero/Desktop/A01754412---Portafolio-Implementaci-n/Data/normalized_winequality_dataset.csv')

    X = data.drop('quality', axis=1).values

    y = data['quality'].values

    train_size = int(0.6 * len(data))

    X_train, y_train = X[:train_size], y[:train_size]

    decision_tree = build_tree(X_train, y_train, max_depth=5)

    return decision_tree

features = np.array([1.654856079, 0.794282374, 1.432803137, -0.169427234,
-0.073676911, -
                        0.944346356, -1.017721, 1.724304591, -0.91431164, 0.305989631,
-0.866378858])

mean = np.array([8.319637, 0.527821, 0.270976, 2.538806, 0.087467,
                  15.874922, 46.467792, 0.996747, 3.311113, 0.658149, 10.422983])

std = np.array([1.741096, 0.179060, 0.194801, 1.409928, 0.047065,
```

```
10.460157, 32.895324, 0.001887, 0.154386, 0.169507, 1.065668])

features = (features - mean) / std

trained_tree = load_trained_tree()

prediction = predict_tree(trained_tree, features)

if prediction == 1:

    print("El vino es bueno.")

else:

    print("El vino es malo.")
```

Manual de usuario

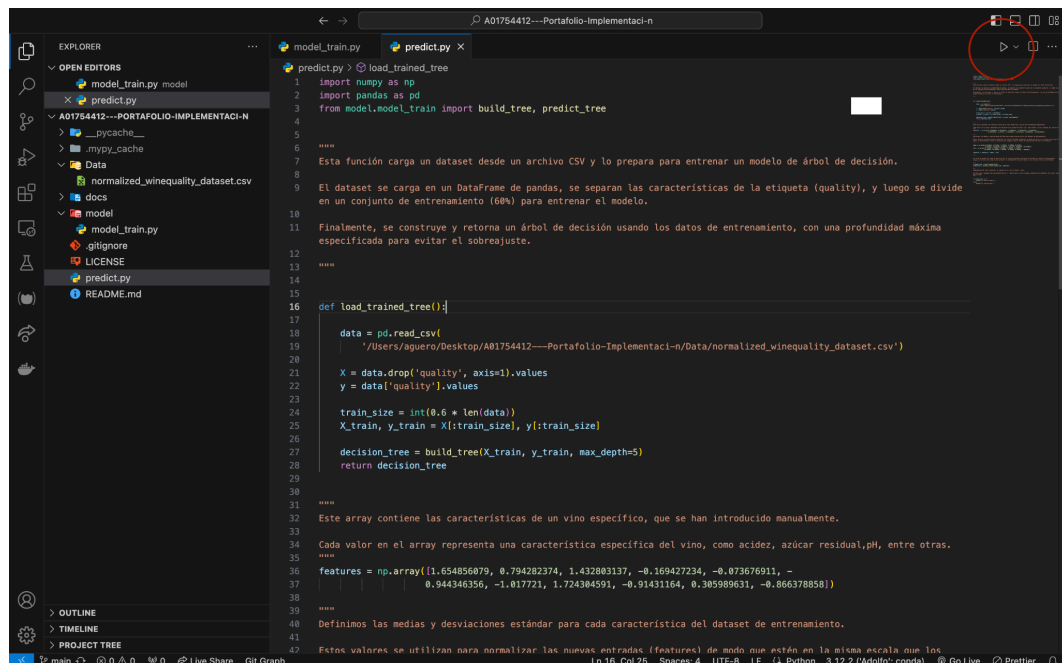
Para correr el código implementado en 'predict.py', sigue estos pasos:

1. **Abrir el Archivo predict.py:** Asegúrate de estar en la pestaña correcta dentro de tu entorno de desarrollo. Para ejecutar el código de predicción, necesitas tener predict.py abierto y seleccionado.
2. **Ejecutar el Código:** Una vez que estés en la pestaña de predict.py, localiza y presiona el botón de Run o Ejecutar en tu entorno de desarrollo (esto puede variar dependiendo del editor que estés utilizando, como VSCode, PyCharm, o Jupyter Notebook).

Asegúrate de que la pestaña de predict.py esté activa cuando hagas clic en el botón de Run. Esto es importante, ya que ejecutar el código desde otra pestaña podría no producir los resultados esperados.

Al presionar Run, el código en predict.py se ejecutará automáticamente. Esto incluye cargar los datos, normalizar las características del vino que especificaste manualmente, y utilizar el modelo de árbol de decisión para hacer una predicción sobre la calidad del vino.

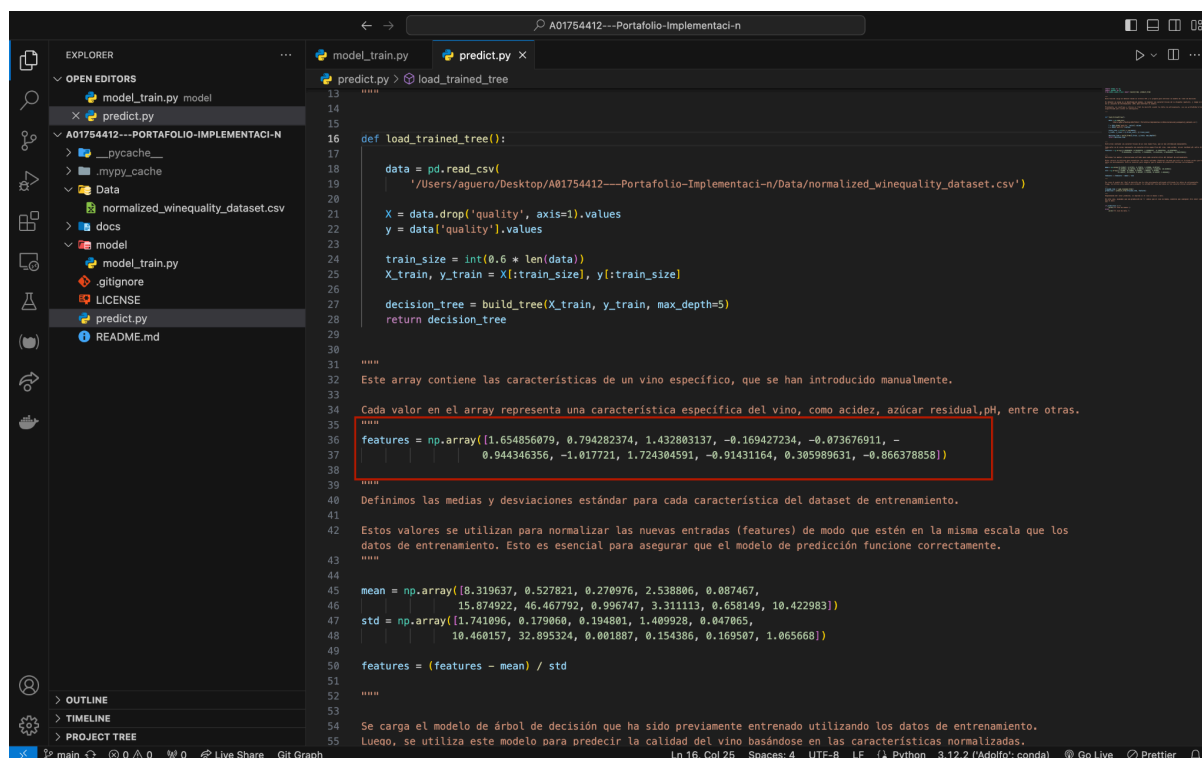
3. **Ver los Resultados:** Después de la ejecución, el código imprimirá en la consola o en la salida de tu entorno de desarrollo si el vino es "bueno" o "malo" basado en la predicción realizada. Asegúrate de revisar la consola para ver el resultado.



```
1 import numpy as np
2 import pandas as pd
3 from model.model_train import build_tree, predict_tree
4
5
6
7 Esta función carga un dataset desde un archivo CSV y lo prepara para entrenar un modelo de árbol de decisión.
8
9 El dataset se carga en un DataFrame de pandas, se separan las características de la etiqueta (quality), y luego se divide
10 en un conjunto de entrenamiento (60%) para entrenar el modelo.
11
12 Finalmente, se construye y retorna un árbol de decisión usando los datos de entrenamiento, con una profundidad máxima
13 especificada para evitar el sobreajuste.
14
15
16
17 def load_trained_tree():
18     data = pd.read_csv(
19         '/Users/aguero/Desktop/A01754412---Portafolio-Implementaci-n/Data/normalized_winequality_dataset.csv')
20
21     X = data.drop('quality', axis=1).values
22     y = data['quality'].values
23
24     train_size = int(0.6 * len(data))
25     X_train, y_train = X[:train_size], y[:train_size]
26
27     decision_tree = build_tree(X_train, y_train, max_depth=5)
28     return decision_tree
29
30
31
32 Este array contiene las características de un vino específico, que se han introducido manualmente.
33
34 Cada valor en el array representa una característica específica del vino, como acidez, azúcar residual, pH, entre otras.
35
36 features = np.array([1.654856079, 0.794282374, 1.432803137, -0.169427234, -0.073676911, -
37     0.944346356, -1.017721, 1.724304591, -0.91431164, 0.305989631, -0.866378858])
38
39
40 Definimos las medias y desviaciones estándar para cada característica del dataset de entrenamiento.
41
42 Estos valores se utilizan para normalizar las nuevas entradas (features) de modo que estén en la misma escala que los
```

Si se desea probar el modelo con diferentes características de vino, se puede hacer fácilmente modificando el array de features en el código, el array features es donde se definen las características específicas del vino que se quiere analizar.

1. **Abrir el Archivo predict.py:** Abre el archivo predict.py en tu entorno de desarrollo.
2. **Localizar el Array features:** Dentro del código, encontrarás una sección que define un array llamado features. Este array contiene los valores numéricos que representan las características del vino, como la acidez, el pH, el azúcar residual, entre otros.
3. **Modificar las Características:** Para probar el modelo con un vino diferente, reemplaza los valores dentro del array features con las nuevas características que desees evaluar, asegúrate de que los valores sigan el mismo orden y formato que los actuales, ya que cada posición en el array corresponde a una característica específica del vino.
4. **Ejecutar el Código:** Después de haber ingresado las nuevas características, guarda el archivo y ejecuta el código como se describió anteriormente, el modelo utilizará las nuevas características para hacer una predicción sobre la calidad del vino, y el resultado se imprimirá en la consola.



```
def load_trained_tree():  
    data = pd.read_csv(  
        '/Users/aguero/Desktop/A01754412---Portafolio-Implementaci-n/Data/normalized_winequality_dataset.csv')  
    X = data.drop('quality', axis=1).values  
    y = data['quality'].values  
    train_size = int(0.6 * len(data))  
    X_train, y_train = X[:train_size], y[:train_size]  
    decision_tree = build_tree(X_train, y_train, max_depth=5)  
    return decision_tree  
  
    """  
    Este array contiene las características de un vino específico, que se han introducido manualmente.  
    Cada valor en el array representa una característica específica del vino, como acidez, azúcar residual, pH, entre otras.  
    """  
    features = np.array([1.654856079, 0.794282374, 1.432883137, -0.169427234, -0.073676911, -  
        0.944346356, -1.017721, 1.724304591, -0.91431164, 0.305989631, -0.866378858])  
    """  
    Definimos las medias y desviaciones estándar para cada característica del dataset de entrenamiento.  
    Estos valores se utilizan para normalizar las nuevas entradas (features) de modo que estén en la misma escala que los  
    datos de entrenamiento. Esto es esencial para asegurar que el modelo de predicción funcione correctamente.  
    """  
    mean = np.array([8.319637, 0.527821, 0.270976, 2.538806, 0.087467,  
        15.874922, 46.467792, 0.996747, 3.311113, 0.658149, 10.422983])  
    std = np.array([1.741096, 0.179060, 0.194801, 1.409928, 0.047865,  
        10.460157, 32.895324, 0.001807, 0.154386, 0.169507, 1.065668])  
    features = (features - mean) / std  
    """  
    Se carga el modelo de árbol de decisión que ha sido previamente entrenado utilizando los datos de entrenamiento.  
    Luego, se utiliza este modelo para predecir la calidad del vino basándose en las características normalizadas.  
    """
```

Explicación de los valores obtenidos, con los datos ingresados que se ven en la imagen:

Al evaluar el modelo en el conjunto de validación, se obtuvieron los siguientes resultados:

- **Precisión (Accuracy): 0.7461**
La precisión indica que aproximadamente el 74.61% de las predicciones realizadas por el modelo fueron correctas. Es una métrica general que refleja qué tan bien está funcionando el modelo en promedio, pero no distingue entre los tipos de errores que se están cometiendo.
- **Precisión: 0.8097**
La precisión, en este contexto, mide la proporción de vinos que fueron correctamente identificados como de buena calidad entre todos los que fueron predichos como buenos. En otras palabras, cuando el modelo predice que un vino es bueno, acierta el 80.97% de las veces.
- **Recall: 0.8281**
El recall indica que el modelo es capaz de identificar correctamente el 82.81% de los vinos que son realmente de buena calidad. Es decir, de todos los vinos que efectivamente son buenos, el modelo los detecta correctamente la mayoría de las veces.
- **F1 Score: 0.8188**
El F1 Score es la media armónica entre la precisión y el recall, y en este caso tiene un valor de 0.8188. Este valor proporciona un equilibrio entre la precisión y el recall, lo que es útil en situaciones donde es importante tanto identificar correctamente los vinos buenos como minimizar los falsos positivos.
- **Matriz de Confusión:** La matriz de confusión muestra cómo se distribuyen las predicciones del modelo en relación con las etiquetas reales:

- **Predicted 0 / Actual 0:** 55 veces el modelo predijo correctamente que el vino era de baja calidad.
- **Predicted 1 / Actual 0:** 43 veces el modelo predijo incorrectamente que el vino era bueno cuando no lo era (falsos positivos).
- **Predicted 0 / Actual 1:** 38 veces el modelo predijo incorrectamente que el vino era de baja calidad cuando en realidad era bueno (falsos negativos).
- **Predicted 1 / Actual 1:** 183 veces el modelo predijo correctamente que el vino era bueno.

Estos resultados indican que el modelo tiene un rendimiento bastante equilibrado en términos de identificar correctamente los vinos buenos, aunque hay algunos errores, tanto en falsos positivos como en falsos negativos. El F1 Score muestra que el modelo mantiene un buen equilibrio entre precisión y recall, lo que es positivo cuando se busca minimizar los errores en ambas direcciones.

```
Evaluación en el conjunto de validación:
Precisión (Accuracy): 0.7460815047021944
Precisión: 0.8097345132743363
Recall: 0.8280542986425339
F1 Score: 0.8187919463087249
Matriz de Confusión:
Predicted   0    1
Actual
0           55   43
1           38  183
```

Al evaluar el modelo en el conjunto de prueba, los resultados fueron los siguientes:

- **Precisión (Accuracy):** 0.7352
La precisión general del modelo es del 73.52%, lo que significa que el modelo acertó en aproximadamente tres cuartas partes de las predicciones. Este es un buen indicador general de rendimiento, pero, como siempre, es importante considerar otros aspectos como los tipos de errores cometidos.
- **Precisión:** 0.7037
Aquí, la precisión mide qué tan a menudo el modelo predijo correctamente que un vino era de buena calidad entre todas las veces que hizo esa predicción. En este caso, cuando el modelo predijo que un vino era bueno, acertó el 70.37% de las veces.
- **Recall:** 0.8209
El recall muestra que el modelo fue capaz de identificar correctamente el 82.09% de los vinos que realmente son de buena calidad. Esto significa que, de todos los vinos que efectivamente eran buenos, el modelo logró identificar la mayoría de ellos correctamente.
- **F1 Score:** 0.7578
El F1 Score, que combina la precisión y el recall, fue de 0.7578. Este valor refleja un balance adecuado entre la capacidad del modelo para identificar correctamente los vinos buenos y minimizar los falsos positivos.

- **Matriz de Confusión:**

La matriz de confusión detalla cómo se distribuyen las predicciones del modelo en comparación con las etiquetas reales:

- **Predicted 0 / Actual 0:** 103 veces el modelo predijo correctamente que el vino era de baja calidad.
- **Predicted 1 / Actual 0:** 56 veces el modelo predijo incorrectamente que el vino era bueno cuando no lo era (falsos positivos).
- **Predicted 0 / Actual 1:** 29 veces el modelo predijo incorrectamente que el vino era de baja calidad cuando en realidad era bueno (falsos negativos).
- **Predicted 1 / Actual 1:** 133 veces el modelo predijo correctamente que el vino era bueno.

Finalmente, el modelo hizo una predicción específica sobre un vino, concluyendo que "El vino es malo", este resultado se imprimió en la consola, reflejando la salida final del proceso de predicción basado en las características del vino proporcionadas.

Estos resultados son consistentes con el rendimiento del modelo en la fase de validación, mostrando un buen equilibrio entre precisión y recall, aunque con algunos errores que siguen presentes en la fase de prueba.

```
Evaluación en el conjunto de prueba:
Precisión (Accuracy): 0.735202492211838
Precisión: 0.7037037037037037
Recall: 0.8209876543209876
F1 Score: 0.7578347578347577
Matriz de Confusión:
Predicted    0    1
Actual
0            103   56
1             29  133
El vino es malo.
```

Conclusión:

Este proyecto se centró en la implementación de un modelo de árbol de decisión para predecir la calidad del vino, llevando a cabo el proceso desde la carga y preparación de los datos hasta la evaluación del modelo en conjuntos de validación y prueba; a lo largo del proyecto, logramos desarrollar un modelo desde cero, sin recurrir a bibliotecas externas para la construcción del árbol, lo que nos permitió comprender a fondo los mecanismos internos y las decisiones clave en la construcción de este tipo de modelos.

El modelo mostró un rendimiento consistente tanto en los conjuntos de validación como en los de prueba, con precisiones alrededor del 74% y F1 Scores que reflejan un buen equilibrio entre precisión y recall, esto sugiere que el árbol de decisión fue capaz de capturar patrones significativos en los datos, logrando una adecuada separación entre los

vinos de alta y baja calidad, aunque todavía se presentaron algunos errores, especialmente en la clasificación de falsos positivos y falsos negativos.

Además, el proceso de normalización de las características fue crucial para asegurar que el modelo pudiera realizar predicciones precisas, esto subraya la importancia de una preparación de datos adecuada, un aspecto que no debe pasarse por alto en cualquier proyecto de aprendizaje automático.

Finalmente, la implementación del código para realizar predicciones con nuevos datos, así como la evaluación del modelo en distintos conjuntos, demuestra la aplicabilidad y flexibilidad del modelo en situaciones reales, la capacidad de ingresar nuevas características y obtener predicciones sobre la calidad del vino en cuestión de segundos hace que este proyecto tenga un valor práctico significativo.

En resumen, este proyecto no solo nos permitió implementar un modelo funcional, sino que también nos brindó una comprensión profunda de los árboles de decisión y la importancia de cada paso en el flujo de trabajo del aprendizaje automático, los resultados obtenidos son satisfactorios, y con algunos ajustes adicionales, el modelo podría mejorarse aún más para aplicaciones más robustas en la clasificación de vinos u otras tareas similares.