

from sklearn.tree import DecisionTreeRegressor DecisionTreeRegressor() como uso esto?

`DecisionTreeRegressor` de `sklearn.tree` es un modelo de regresión basado en árboles de decisión. Se utiliza para predecir valores continuos. Aquí tienes un ejemplo de cómo usarlo paso a paso:

Ejemplo de uso de `DecisionTreeRegressor`

Python

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# ① Generamos datos de ejemplo
X = np.arange(1, 11).reshape(-1, 1) # Variables independientes
y = np.array([3, 7, 4, 9, 11, 13, 15, 18, 20, 25]) # Valores a predecir

# ② Dividimos los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# ③ Creamos el modelo
modelo = DecisionTreeRegressor(max_depth=3) # Limitamos la profundidad del árbol para evitar sobreajuste

# ④ Entrenamos el modelo
modelo.fit(X_train, y_train)

# ⑤ Hacemos predicciones
y_pred = modelo.predict(X_test)

# ⑥ Evaluamos el modelo
error = mean_squared_error(y_test, y_pred)
print(f"Error cuadrático medio: {error:.2f}")
```

```
# 7 Visualizamos el ajuste del modelo
X_grid = np.linspace(1, 10, 100).reshape(-1, 1)
y_grid_pred = modelo.predict(X_grid)

plt.scatter(X, y, color="blue", label="Datos reales")
plt.plot(X_grid, y_grid_pred, color="red", linestyle="dashed",
label="Predicción del modelo")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.title("Regresión con Árbol de Decisión")
plt.show()
```

Parámetros clave de **DecisionTreeRegressor**

- **max_depth**: Controla la profundidad del árbol para evitar sobreajuste.
- **min_samples_split**: Mínimo número de muestras necesarias para dividir un nodo.
- **min_samples_leaf**: Mínimo número de muestras en una hoja.
- **random_state**: Para reproducibilidad de los resultados.

♦ **Resumen:** Se entrena el modelo con `fit(X_train, y_train)`, se hacen predicciones con `predict(X_test)`, y se evalúa con métricas como `mean_squared_error()`.

¿Quieres un ejemplo con datos reales o alguna personalización específica? 

vale los datos de la UCI de calidad del vino, quieor hacer una regresion con el dts

¡Genial! Vamos a usar el dataset de calidad del vino de la **UCI Machine Learning Repository** y entrenar un modelo de regresión con **DecisionTreeRegressor**.

Pasos a seguir:

1. **Cargar los datos** desde un archivo CSV.

2. **Preprocesar** los datos (manejo de valores nulos, normalización, etc.).
 3. **Dividir en conjunto de entrenamiento y prueba.**
 4. **Entrenar un `DecisionTreeRegressor`.**
 5. **Evaluar el modelo con métricas adecuadas.**
 6. **Visualizar los resultados.**
-

📄 Código completo

Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# ❶ Cargar los datos (Dataset de calidad del vino de UCI)
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/win
e-quality/winequality-red.csv"
data = pd.read_csv(url, sep=";")

# ❷ Ver información general
print(data.info()) # Revisar si hay valores nulos
print(data.describe()) # Estadísticas básicas

# ❸ Separar variables predictoras (X) y variable objetivo (y)
X = data.drop(columns=["quality"]) # Todas las columnas menos
la calidad
y = data["quality"] # La calidad del vino es la variable a
predecir

# ❹ Dividir en conjunto de entrenamiento y prueba (80% - 20%)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```

# 5 Crear el modelo y entrenarlo
modelo = DecisionTreeRegressor(max_depth=5, random_state=42)
# Limitar profundidad para evitar sobreajuste
modelo.fit(X_train, y_train)

# 6 Hacer predicciones
y_pred = modelo.predict(X_test)

# 7 Evaluar el modelo
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Error cuadrático medio (MSE): {mse:.2f}")
print(f"Coeficiente de determinación (R²): {r2:.2f}")

# 8 Visualizar predicciones vs valores reales
plt.scatter(y_test, y_pred, color="blue", alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test),
max(y_test)], color="red", linestyle="dashed") # Línea de
ajuste perfecto
plt.xlabel("Valores reales")
plt.ylabel("Predicciones")
plt.title("Regresión con Árbol de Decisión - Calidad del
Vino")
plt.show()

```

Explicación del código

- ✓ **Cargamos el dataset** directamente desde la web de UCI.
- ✓ **Preprocesamos los datos**, eliminando la columna `quality` de `X` y usándola como `y`.
- ✓ **Dividimos los datos** en 80% entrenamiento y 20% prueba.
- ✓ **Entrenamos un `DecisionTreeRegressor`** con `max_depth=5` para evitar sobreajuste.
- ✓ **Evaluamos con métricas** como el **Error Cuadrático Medio (MSE)** y R^2 .
- ✓ **Visualizamos la comparación** entre valores reales y predicciones en un gráfico de dispersión.

🔥 ¿Qué más podríamos hacer?

- ♦ **Optimizar hiperparámetros** con `GridSearchCV` o `RandomizedSearchCV`.
- ♦ **Comparar con otros modelos** como `RandomForestRegressor`.
- ♦ **Probar diferentes profundidades** (`max_depth`) para ver su impacto en el rendimiento.

¿Quieres que probemos otra configuración o añadamos más análisis? 🚀

quiero hacer un PCA y un escalado de datos

Buena idea. **PCA** (Análisis de Componentes Principales) nos ayuda a reducir la dimensionalidad del dataset, mientras que **el escalado** normaliza los datos para mejorar el rendimiento del modelo.

Voy a incluir estos pasos en el código:

1. **Escalar los datos** con `StandardScaler` (media = 0, desviación estándar = 1).
2. **Aplicar PCA** para reducir la dimensionalidad a 2 componentes.
3. **Entrenar el `DecisionTreeRegressor` con los datos transformados.**
4. **Evaluar el modelo.**
5. **Visualizar los datos después de PCA.**

📄 Código con PCA y escalado

Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score

# ① Cargar los datos
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
```

```

data = pd.read_csv(url, sep=";")

# ② Separar variables predictoras (X) y variable objetivo (y)
X = data.drop(columns=["quality"]) # Todas las columnas menos
la calidad
y = data["quality"]

# ③ Escalar los datos con StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ④ Aplicar PCA para reducir a 2 componentes principales
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# ⑤ Dividir en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
test_size=0.2, random_state=42)

# ⑥ Entrenar el modelo
modelo = DecisionTreeRegressor(max_depth=5, random_state=42)
modelo.fit(X_train, y_train)

# ⑦ Hacer predicciones
y_pred = modelo.predict(X_test)

# ⑧ Evaluar el modelo
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Error cuadrático medio (MSE): {mse:.2f}")
print(f"Coeficiente de determinación (R²): {r2:.2f}")

# ⑨ Visualizar los datos después del PCA
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap="coolwarm",
alpha=0.6)
plt.colorbar(label="Calidad del vino")
plt.xlabel("Componente Principal 1")

```

```
plt.ylabel("Componente Principal 2")
plt.title("Visualización de los datos después de PCA")
plt.show()
```

Explicación del código

✓ **Estandarizamos los datos** con `StandardScaler` para que todas las variables tengan la misma escala.

✓ **Aplicamos PCA** para reducir la dimensionalidad a 2 componentes principales.

✓ **Entrenamos el `DecisionTreeRegressor`** usando los datos transformados.

✓ **Evaluamos el modelo** con **MSE** y **R²**.

✓ **Visualizamos los datos después del PCA** en un gráfico de dispersión con colores según la calidad del vino.

♦ ¿Quieres probar con más componentes en el PCA o algún otro ajuste? 🚀

quiero hacer un script que use el mejor numero de capas del arbol de `DecisionTreeRegressor` de sklearn

¡Buena idea! Podemos usar **validación cruzada** y búsqueda de hiperparámetros para encontrar el mejor número de capas (`max_depth`) en `DecisionTreeRegressor`.

Pasos que seguiremos:

1. **Cargar y preprocesar los datos** (escalado y PCA opcional).
2. **Probar diferentes valores de `max_depth`.**
3. **Evaluar cada modelo con validación cruzada.**
4. **Seleccionar el mejor `max_depth` y reentrenar el modelo final.**
5. **Mostrar el rendimiento del modelo óptimo.**

Código:

```
Python
import pandas as pd
import numpy as np
```

```

import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split,
cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

# ❶ Cargar los datos
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/wine-  
e-quality/winequality-red.csv"
data = pd.read_csv(url, sep=";")

# ❷ Separar variables predictoras (X) y variable objetivo (y)
X = data.drop(columns=["quality"])
y = data["quality"]

# ❸ Escalar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ❹ Dividir en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
y, test_size=0.2, random_state=42)

# ❺ Probar diferentes valores de max_depth
max_depths = range(1, 21) # Probamos de 1 a 20 niveles de
profundidad
mse_scores = []

for depth in max_depths:
    model = DecisionTreeRegressor(max_depth=depth,
random_state=42)
    mse = -np.mean(cross_val_score(model, X_train, y_train,
cv=5, scoring='neg_mean_squared_error'))
    mse_scores.append(mse)

# ❻ Encontrar el mejor max_depth
best_depth = max_depths[np.argmin(mse_scores)]

```



```

print(f"Mejor profundidad del árbol: {best_depth}")

# 7 Reentrenar con la mejor profundidad
best_model = DecisionTreeRegressor(max_depth=best_depth,
    random_state=42)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

# 8 Evaluar el modelo final
mse_final = mean_squared_error(y_test, y_pred)
print(f"Error cuadrático medio del mejor modelo:
    {mse_final:.2f}")

# 9 Visualizar la relación entre max_depth y MSE
plt.figure(figsize=(8, 5))
plt.plot(max_depths, mse_scores, marker='o',
    linestyle='dashed', color='blue')
plt.axvline(best_depth, color='red', linestyle='dashed',
    label=f"Mejor profundidad ({best_depth})")
plt.xlabel("Profundidad del árbol (max_depth)")
plt.ylabel("Error Cuadrático Medio (MSE)")
plt.title("Selección del mejor número de capas en
    DecisionTreeRegressor")
plt.legend()
plt.show()

```

Explicación del código

- ✓ Prueba múltiples valores de **max_depth** entre 1 y 20.
- ✓ Usa validación cruzada (**cross_val_score**) para obtener el MSE promedio de cada modelo.
- ✓ Selecciona el **max_depth** que tenga el menor MSE.
- ✓ Entrena el modelo final con la mejor profundidad y evalúa su rendimiento.
- ✓ Grafica la relación entre **max_depth** y el error **MSE** para visualizar el mejor punto.

 ¿Qué más podríamos hacer?

- ♦ Probar más hiperparámetros como `min_samples_split`, `min_samples_leaf`, etc.
- ♦ Usar **GridSearchCV** o **RandomizedSearchCV** para buscar más combinaciones automáticamente.
- ♦ Comparar con otros modelos como `RandomForestRegressor`.
- ♦ ¿Quieres agregar más optimización o probar con otro dataset? 🚀

hay ciertos parametros que podemos añadir verdad? De entropia algo asi al decision tree

Efectivamente, se pueden ajustar muchos parámetros en los árboles de decisión para mejorar su rendimiento y controlar el sobreajuste. Sin embargo, es importante notar que el parámetro `criterion` en el modelo de regresión (**DecisionTreeRegressor**) no utiliza la entropía, ya que ésta se utiliza en problemas de clasificación.

Parámetros Comunes en DecisionTreeRegressor

Algunos de los parámetros que puedes ajustar en **DecisionTreeRegressor** son:

- **criterion:**
 - Por defecto es `"squared_error"`, que mide el error cuadrático medio.
 - Otras opciones incluyen `"friedman_mse"` (una mejora en el cálculo del error) o `"absolute_error"` (que usa la mediana absoluta del error).
Nota: La entropía no es un criterio utilizado en regresión; se usa en clasificación con **DecisionTreeClassifier**.
- **max_depth:**
 - Limita la profundidad del árbol para evitar sobreajuste.
- **min_samples_split:**
 - El número mínimo de muestras necesarias para dividir un nodo.
- **min_samples_leaf:**
 - El número mínimo de muestras que debe tener una hoja.
- **max_features:**
 - El número de características a considerar al buscar la mejor división.
- **splitter:**

- Define la estrategia usada para elegir la división en cada nodo ("best" o "random").

Parámetro de Entropía en Árboles de Clasificación

Si estuvieras trabajando en un problema de clasificación, podrías utilizar **DecisionTreeClassifier** y ajustar el parámetro `criterion` a "entropy" o "gini". Por ejemplo:

```
Python
from sklearn.tree import DecisionTreeClassifier

# Usando entropía como criterio en un árbol de decisión para
# clasificación
clf = DecisionTreeClassifier(criterion="entropy", max_depth=5,
random_state=42)
clf.fit(X_train, y_train)
```

En cambio, para regresión, podrías ajustar parámetros como:

```
Python
from sklearn.tree import DecisionTreeRegressor

# Usando squared_error o absolute_error en un árbol de
# decisión para regresión
reg = DecisionTreeRegressor(criterion="squared_error",
max_depth=5, random_state=42)
reg.fit(X_train, y_train)
```

Resumen

- **Regresión:** Utiliza criterios como "squared_error", "friedman_mse" o "absolute_error".
- **Clasificación:** Puedes utilizar criterios como "entropy" o "gini" en **DecisionTreeClassifier**.

Ajustar estos parámetros mediante técnicas de búsqueda de hiperparámetros (por ejemplo, **GridSearchCV** o **RandomizedSearchCV**) te permitirá encontrar la configuración óptima

para tu modelo. ¿Te gustaría ver un ejemplo de cómo implementar una búsqueda de hiperparámetros para estos parámetros?

en que afecta usar un criterio u otro? es bueno combinarlo con GridSearchCV??

¡Buena pregunta! Elegir el **criterio (criterion)** adecuado en un **árbol de decisión** afecta directamente cómo el modelo **divide los datos** y, por lo tanto, su rendimiento. Además, combinar esto con **GridSearchCV** ayuda a encontrar la mejor configuración para el modelo.

1. Diferencias entre criterios (**criterion**) en DecisionTreeRegressor

En **DecisionTreeRegressor**, hay tres opciones principales:

squared_error (por defecto)

- ♦ Calcula el **error cuadrático medio** (MSE).
- ♦ Tiende a favorecer divisiones que minimicen la varianza dentro de cada nodo.
- ♦ Funciona bien cuando los **datos tienen valores extremos o ruidosos**.
- ♦ **Bueno en la mayoría de los casos.**

absolute_error

- ♦ Utiliza el **error absoluto medio (MAE)** en lugar de MSE.
- ♦ Es más **robusto a valores atípicos (outliers)** que **squared_error**.
- ♦ Puede ser más lento, ya que requiere cálculos iterativos en cada nodo.
- ♦ **Útil cuando los datos tienen valores extremos importantes.**

friedman_mse

- ♦ Similar a **squared_error**, pero optimizado para **minimizar el sesgo en modelos de boosting**.
- ♦ Se usa en modelos como **GradientBoostingRegressor**.
- ♦ **Útil cuando planeas usar boosting después del árbol.**

En resumen:

- **squared_error** es una buena opción general.
- **absolute_error** es mejor si tienes muchos valores atípicos.

- `friedman_mse` es útil para boosting.

2. ¿Es bueno usar `GridSearchCV` para encontrar el mejor criterio?

Sí, combinar `GridSearchCV` con `criterion` y otros hiperparámetros ayuda a encontrar la mejor configuración. `GridSearchCV` evalúa todas las combinaciones de parámetros posibles mediante validación cruzada y devuelve la mejor opción.

Ejemplo de `GridSearchCV` con diferentes criterios

Python

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

# ① Cargar los datos
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/win
e-quality/winequality-red.csv"
data = pd.read_csv(url, sep=";")

# ② Separar variables predictoras (X) y objetivo (y)
X = data.drop(columns=["quality"])
y = data["quality"]

# ③ Escalar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ④ Dividir en conjunto de entrenamiento y prueba
```

```

X_train, X_test, y_train, y_test = train_test_split(X_scaled,
y, test_size=0.2, random_state=42)

# 5 Definir el modelo y los hiperparámetros a probar
param_grid = {
    "criterion": ["squared_error", "absolute_error",
"friedman_mse"], # Diferentes criterios
    "max_depth": [3, 5, 10, None], # Diferentes profundidades
del árbol
    "min_samples_split": [2, 5, 10], # Tamaño mínimo para
dividir un nodo
    "min_samples_leaf": [1, 2, 5] # Tamaño mínimo de las
hojas
}

# 6 Usar GridSearchCV para encontrar la mejor combinación de
hiperparámetros
grid_search =
GridSearchCV(DecisionTreeRegressor(random_state=42),
param_grid, cv=5, scoring="neg_mean_squared_error", n_jobs=-1)
grid_search.fit(X_train, y_train)

# 7 Obtener los mejores parámetros
best_params = grid_search.best_params_
print(f"Mejores parámetros encontrados: {best_params}")

# 8 Entrenar el mejor modelo
best_model = DecisionTreeRegressor(**best_params,
random_state=42)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

# 9 Evaluar el modelo final
mse_final = mean_squared_error(y_test, y_pred)
print(f"Error cuadrático medio del mejor modelo:
{mse_final:.2f}")

```

Conclusiones

- 1 Diferentes criterios afectan la forma en que el árbol divide los datos, lo que puede influir en el rendimiento del modelo.
- 2 **GridSearchCV** es una excelente herramienta para encontrar la mejor combinación de **criterion**, **max_depth**, y otros hiperparámetros.
- 3 No hay un criterio universalmente mejor; la mejor elección depende de la estructura de los datos.

🚀 ¿Quieres probar con más hiperparámetros o usar **RandomizedSearchCV** para una búsqueda más rápida?

Que quiere decir cuando de criterion usas entropy o gini??

Cuando usas **criterion="entropy"** o **criterion="gini"**, estás trabajando con un árbol de decisión para clasificación (**DecisionTreeClassifier**) en lugar de regresión (**DecisionTreeRegressor**).

Estos criterios determinan cómo el árbol divide los datos en cada nodo durante el entrenamiento.

🔍 1. Diferencias entre **gini** y **entropy**

Ambos criterios miden la pureza de los nodos en un árbol de clasificación:

- **gini** (Índice de Gini) → Mide la impureza de un nodo.
- **entropy** (Entropía de Shannon) → Mide el grado de desorden en un nodo.

💡 Un nodo es "puro" cuando contiene solo una clase (por ejemplo, si en una hoja del árbol solo hay vinos de calidad 7).

📋 2. Fórmulas

📌 Gini (Impureza de Gini)

Fórmula:

$$\text{Gini} = 1 - \sum p_i^2$$

Donde p_i es la proporción de elementos de la clase i en el nodo.

- Si el nodo tiene ejemplos de solo una clase → **Gini = 0 (puro)**.

- Si el nodo tiene una **mezcla equilibrada de clases** → **Gini es más alto**.

Entropía (Entropy de Shannon)

Fórmula:

$$\text{Entropy} = -\sum p_i \log_2(p_i) \quad \text{Entropy} = - \sum p_i \log_2(p_i)$$

- Si el nodo tiene **ejemplos de solo una clase** → **Entropía = 0 (puro)**.
- Si las clases están **uniformemente distribuidas** → **Entropía es más alta**.

3. Comparación entre **gini** y **entropy**

| Criterio | ¿Cómo funciona? | Velocidad | ¿Cuándo usar? |
|-----------------|--|----------------------------------|---|
| Gini | Prefiere nodos con una clase dominante | Más rápido | Cuando necesitas eficiencia |
| Entropía | Tiende a hacer divisiones más equilibradas | Más lento (cálculo de logaritmo) | Cuando necesitas precisión en árboles más grandes |

✓ **gini es más rápido** porque solo usa sumas y multiplicaciones.

✓ **entropy tiende a hacer mejores divisiones** en algunos casos porque usa logaritmos, pero es más costoso computacionalmente.

4. Ejemplo en Python

Python

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# ① Cargar el dataset Iris (clasificación de flores)
iris = load_iris()
X, y = iris.data, iris.target
```



```

# ② Dividir en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# ③ Entrenar dos modelos: uno con Gini y otro con Entropía
modelo_gini = DecisionTreeClassifier(criterion="gini",
max_depth=3, random_state=42)
modelo_entropy = DecisionTreeClassifier(criterion="entropy",
max_depth=3, random_state=42)

modelo_gini.fit(X_train, y_train)
modelo_entropy.fit(X_train, y_train)

# ④ Hacer predicciones
y_pred_gini = modelo_gini.predict(X_test)
y_pred_entropy = modelo_entropy.predict(X_test)

# ⑤ Evaluar los modelos
accuracy_gini = accuracy_score(y_test, y_pred_gini)
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)

print(f"Precisión con Gini: {accuracy_gini:.2f}")
print(f"Precisión con Entropía: {accuracy_entropy:.2f}")

```

Conclusión

- Si buscas velocidad y eficiencia → Usa **gini**.
- Si buscas divisiones más equilibradas (aunque sea más lento) → Usa **entropy**.
- En la práctica, ambos suelen dar resultados similares, así que prueba ambos con **GridSearchCV** para ver cuál funciona mejor.

 ¿Te gustaría ver un ejemplo con **GridSearchCV** para elegir entre **gini** y **entropy**?

gera un script que use los mejores criterios para hacer una regresion con decisiontree

Aquí tienes un **script automatizado** que encuentra los mejores hiperparámetros para un **árbol de regresión (DecisionTreeRegressor)**, incluyendo el **mejor criterio** y otros parámetros clave. Utilizaremos **GridSearchCV** para optimizar:

- ♦ `criterion` → "squared_error", "friedman_mse", "absolute_error"
 - ♦ `max_depth` → Controla la profundidad del árbol
 - ♦ `min_samples_split` → Controla cuántas muestras se necesitan para dividir un nodo
 - ♦ `min_samples_leaf` → Controla el tamaño mínimo de las hojas
-

▼ Código del script

Python

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# ❶ Cargar los datos
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
data = pd.read_csv(url, sep=";")

# ❷ Separar variables predictoras (X) y variable objetivo (y)
X = data.drop(columns=["quality"])
y = data["quality"]

# ❸ Escalar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ❹ Dividir en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
y, test_size=0.2, random_state=42)

# ❺ Definir los hiperparámetros a probar
```

```

param_grid = {
    "criterion": ["squared_error", "friedman_mse",
"absolute_error"],
    "max_depth": [3, 5, 10, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 5]
}

# 6 Usar GridSearchCV para encontrar la mejor combinación de
hiperparámetros
grid_search =
GridSearchCV(DecisionTreeRegressor(random_state=42),
param_grid, cv=5, scoring="neg_mean_squared_error", n_jobs=-1)
grid_search.fit(X_train, y_train)

# 7 Obtener los mejores parámetros encontrados
best_params = grid_search.best_params_
print(f"✅ Mejores parámetros encontrados: {best_params}")

# 8 Entrenar el modelo con los mejores parámetros
best_model = DecisionTreeRegressor(**best_params,
random_state=42)
best_model.fit(X_train, y_train)

# 9 Hacer predicciones
y_pred = best_model.predict(X_test)

# 10 Evaluar el modelo final
mse_final = mean_squared_error(y_test, y_pred)
r2_final = r2_score(y_test, y_pred)

print(f"📊 Error cuadrático medio (MSE): {mse_final:.2f}")
print(f"📊 Coeficiente de determinación (R²): {r2_final:.2f}")

```

Explicación del código

- 1 **Carga los datos** del dataset de calidad del vino.
- 2 **Escala las variables** con `StandardScaler` para mejorar la estabilidad del modelo.

- 3 Divide los datos en entrenamiento y prueba (80%-20%).
 - 4 Define una cuadrícula de hiperparámetros (`criterion`, `max_depth`, `min_samples_split`, `min_samples_leaf`).
 - 5 Utiliza `GridSearchCV` para encontrar la mejor combinación de parámetros usando validación cruzada de 5 folds.
 - 6 Entrena un `DecisionTreeRegressor` con los mejores parámetros encontrados.
 - 7 Evalúa el modelo usando **Error Cuadrático Medio (MSE)** y **Coeficiente de Determinación (R^2)**.
-

¿Qué más podrías hacer?

- ✓ Probar `RandomizedSearchCV` para una búsqueda más rápida.
- ✓ Visualizar la estructura del árbol con `plot_tree()`.
- ✓ Comparar con otros modelos como `RandomForestRegressor`.

♦ ¿Quieres agregar visualización o comparar con otro modelo? 

No, `entropy` y `gini` no se pueden usar en `DecisionTreeRegressor` porque estos criterios son específicos para **clasificación**, no para regresión.

¿Por qué?

- `DecisionTreeRegressor` trabaja con valores numéricos y usa criterios como el **error cuadrático medio (`squared_error`)** para minimizar la diferencia entre los valores predichos y reales.
 - `entropy` y `gini` se usan en `DecisionTreeClassifier` porque miden la **pureza de las clases en cada nodo**.
-

✓ ¿Qué criterios puedes usar en regresión (`DecisionTreeRegressor`)?

En la `param_grid`, puedes incluir estos criterios válidos:

| Criterio | Descripción |
|---|---|
| <code>"squared_error"</code> (default) | Minimiza el error cuadrático medio (MSE) . |
| <code>"friedman_mse"</code> | Similar a <code>squared_error</code> , pero optimizado para boosting . |

"absolute_error" Minimiza el **error absoluto medio (MAE)**, más robusto a outliers.

♦ Si quisieras usar **entropy** o **gini**, tendrías que cambiar a **DecisionTreeClassifier**, que es para clasificación.

📌 Ejemplo correcto de **param_grid** para regresión

```
Python
param_grid = {
    "criterion": ["squared_error", "friedman_mse",
    "absolute_error"], # ❌ NO entropy ni gini
    "max_depth": [3, 5, 10, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 5]
}
```

🚀 ¿Te gustaría probar **DecisionTreeClassifier** en lugar de regresión para ver cómo funcionan **entropy** y **gini**?

Como puede ser que me salga una R^2 tanm baja con este dataset de wine quality de la Universidad de California?? No solo es bajo en si, sino que ademas en comparacion a un kNN regresor tambien es mas bajo

El **bajo R^2** en tu modelo de **DecisionTreeRegressor** para el dataset de **Wine Quality** puede deberse a varios factores. Vamos a analizarlo y ver cómo podrías mejorarlo.

🔍 Posibles razones del bajo R^2

1 El dataset es difícil para regresión con árboles de decisión

El **Wine Quality Dataset** tiene **datos continuos**, pero la variable objetivo (**quality**) es **casi categórica** (toma valores discretos entre 3 y 9).

♦ Los árboles de decisión funcionan mejor con patrones estructurados, y en este dataset, la relación entre las características y la calidad del vino es compleja y ruidosa.

♦ El kNN podría funcionar mejor en este caso porque se basa en vecinos cercanos y no en divisiones rígidas como los árboles de decisión.

2 Overfitting o Underfitting

Si la **profundidad máxima** (**max_depth=6**) es **demasiado baja**, el modelo podría no estar capturando bien las relaciones.

Si es **demasiado alta**, el modelo podría estar sobreajustando.

- ♦ Prueba ajustar **max_depth** más grande (10-20) y usar **pruning (ccp_alpha)** para evitar sobreajuste.

3 Características mal escaladas o irrelevantes

Si bien ya has escalado los datos con **StandardScaler**, podrías probar **PCA** o **selección de características** para eliminar ruido en los datos.

- ♦ Prueba eliminar características menos relevantes usando **SelectKBest** o **mutual_info_regression**.

4 Evaluación con GridSearchCV con pocos parámetros

Tu **GridSearchCV** solo prueba unos pocos hiperparámetros. Podrías ampliar la búsqueda con más valores:

```
Python
param_grid = {
    "criterion": ["squared_error", "friedman_mse",
    "absolute_error"],
    "max_depth": [3, 5, 10, 15, 20, None],
    "min_samples_split": [2, 5, 10, 20],
    "min_samples_leaf": [1, 2, 5, 10],
    "ccp_alpha": [0.0, 0.01, 0.1, 0.5] # Pruning para evitar
    overfitting
}
```

- ♦ Usar **ccp_alpha** puede ayudar a encontrar una poda óptima para evitar sobreajuste.

5 Otras técnicas que podrían mejorar R^2

- ✓ Prueba **RandomForestRegressor** (puede generalizar mejor que un solo árbol).
 - ✓ Prueba **GradientBoostingRegressor** o **XGBoost**, que suelen funcionar bien en datasets como este.
 - ✓ Haz **Feature Engineering**, combinando características para crear nuevas variables.
-

Próximo paso

- ♦ ¿Quieres que haga un script con un modelo de **RandomForestRegressor** para comparar el rendimiento? 

hay alguna manera visual en sklearn de ver que arbol se esta usando ??


Sí, **Scikit-learn** tiene herramientas para **visualizar el árbol de decisión** que se está usando en **DecisionTreeRegressor**. Aquí te muestro cómo hacerlo:

1. Usando **plot_tree()** de Scikit-learn

Esta es la forma más fácil de visualizar el árbol:

```
Python
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Entrenar el modelo con los mejores parámetros encontrados
best_model = DecisionTreeRegressor(criterion="squared_error",
max_depth=6, min_samples_leaf=5, min_samples_split=2,
random_state=42)
best_model.fit(X_train, y_train)

#  Dibujar el árbol
plt.figure(figsize=(20,10)) # Ajusta el tamaño de la imagen
plot_tree(best_model, feature_names=data.columns[:-1],
filled=True, rounded=True)
plt.show()
```

- ♦ **feature_names=data.columns[:-1]** → Muestra los nombres de las características en los nodos.
- ♦ **filled=True** → Colorea los nodos según los valores.

2. Guardar el árbol en formato gráfico con `export_graphviz()`

Otra opción es usar **Graphviz** para generar una imagen más detallada.

```
Python
from sklearn.tree import export_graphviz
import graphviz

# Exportar el árbol en formato DOT
dot_data = export_graphviz(best_model, out_file=None,
                           feature_names=data.columns[:-1], filled=True, rounded=True)

# Mostrar el árbol con Graphviz
graph = graphviz.Source(dot_data)
graph.view()
```

- ♦ Esto genera un archivo **.pdf** o **.png** con el árbol completo.

¿Quieres que probemos con un modelo **RandomForestRegressor** o **XGBoost** para ver si mejora?

el objeto data que se supone que es?

No puedo generar un PDF directamente, pero puedes seguir estos pasos para hacerlo fácilmente:

- 1 **Abre el chat completo** y desplázate hacia arriba para cargar todos los mensajes.
- 2 **Presiona `Ctrl + P`** (o `Cmd + P` en Mac) para abrir la opción de impresión.
- 3 En el destino de la impresora, selecciona **"Guardar como PDF"**.
- 4 Haz clic en **"Guardar"** y elige una ubicación en tu computadora.

 Si necesitas que te estructure las conversaciones en un formato más limpio antes de guardarlas, dime y te lo organizo.