Modelos Cart

Universidad Nacional de Colombia
Prof. Carlos Daniel Jiménez

2024

Contenido

- Conceptos Preliminares
- Gini y Entropía
- Regularización de los Hiperparámetros
- Regresión con Caret
- Random Forest y modelos robustos

Random Forest y modelos robustos

Random Forest

- Es un modelo basado en la construcción de múltiples árboles de decisión y la combinación de sus resultados para mejorar la precisión y robustez de las predicciones.

- Se basa en modelos de ensamble : es una técnica que combina múltiples modelos individuales para crear un modelo más poderoso y preciso.

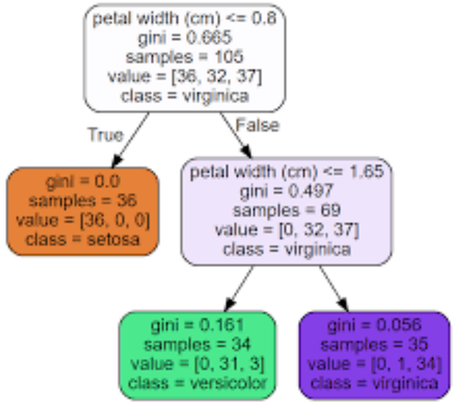
Conceptos Preliminares

Introducción a los Modelos CART

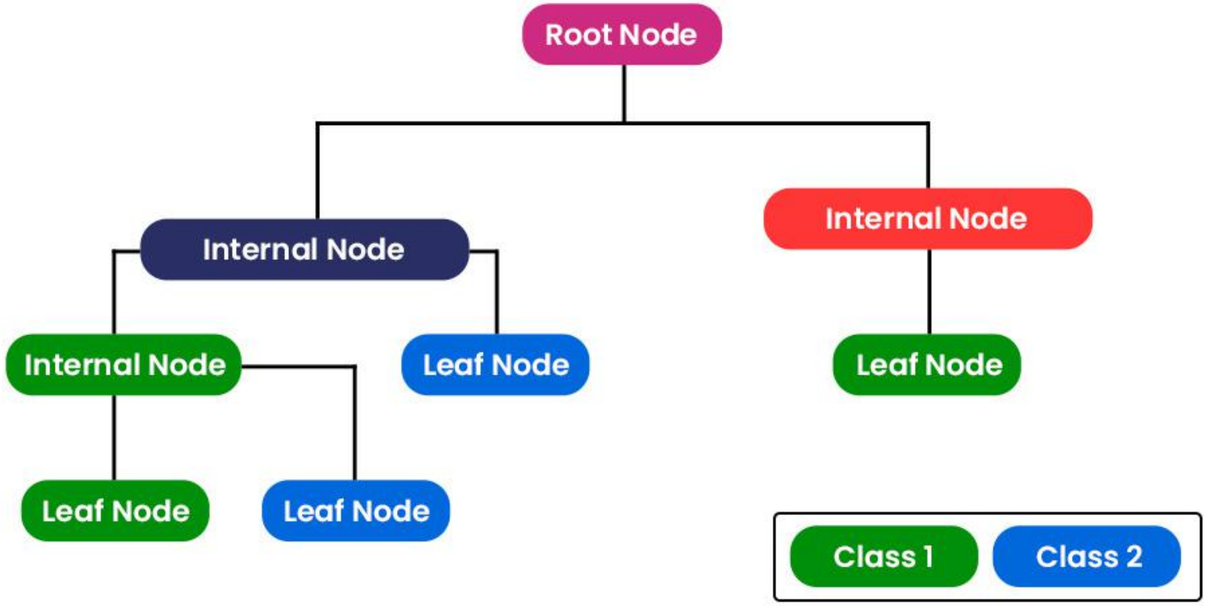
- **Definición:** CART (Classification and Regression Trees) es una técnica de aprendizaje automático utilizada tanto para tareas de clasificación como de regresión.
- **Propósito:** Se utiliza para construir modelos predictivos basados en la partición del espacio de características en regiones homogéneas.
- **Componentes:** Nodos, ramas y hojas.
- **Aplicación Ambiental:** Usado para predecir la calidad del aire, clasificación de tipos de suelo y análisis de deforestación.

Estructura de un Árbol CART

- **Nodos Internos:** Representan decisiones basadas en características de entrada.
- **Ramas:** Conectan nodos y representan posibles resultados de decisiones.
- **Hojas:** Representan las predicciones finales (clases en clasificación, valores en regresión).
- **Raíz del Árbol:** Nodo inicial que contiene todo el conjunto de datos.
- **Aplicación Ambiental:** Clasificación de áreas de riesgo ambiental basadas en características geográficas y meteorológicas.



Iris



Funcionalidad interna

Construcción de Árboles CART

- Proceso de Construcción:
  - **Selección de Característica:** Elegir la característica que mejor divide los datos.
  - **Criterio de División:** Usar métricas como Gini, Entropía (para clasificación) o MSE (para regresión).
  - **División Recursiva:** Repetir el proceso de división en los subconjuntos resultantes.
  - **Condición de Parada:** Determinar cuándo detener la división (profundidad del árbol, número mínimo de muestras en una hoja, etc.).

Construcción de Árboles CART

- **Métricas de Clasificación**
  - **Precisión:** Proporción de verdaderos positivos entre los positivos predichos.
  - **Recall (Sensibilidad):** Proporción de verdaderos positivos entre todos los casos reales positivos.
  - **F1-Score:** Media armónica de precisión y recall.

  Aplicación Ambiental: Predicción de la calidad del aire basada en características como emisiones de CO2, temperatura y humedad.

Ejemplos de Modelos CART

- **Árbol de Clasificación:**
  - **Clasificación de la Calidad del Aire:** Predecir si la calidad del aire es buena, moderada o mala.
  - **Detección de Tipos de Suelo:** Clasificar diferentes tipos de suelo basados en sus características químicas y físicas.

Estructura de un Árbol CART

- **Métricas de Evaluación**
  - **Error Cuadrático Medio (MSE):** Promedio de los errores al cuadrado. Penaliza más los errores grandes.
  - **Raíz del Error Cuadrático Medio (RMSE):** Raíz cuadrada del MSE. Facilita la interpretación ya que está en la misma escala que los valores de la variable objetivo.
  - **Error Absoluto Medio (MAE):** Promedio de los errores absolutos. Menos sensible a los valores atípicos.
  - **Coeficiente de Determinación (R^2):** Proporción de la varianza en la variable dependiente que es predecible a partir de las variables independientes.

    Aplicaciôn en temas ambientales : Modelado de la Contaminación: Medir el rendimiento de modelos que predicen niveles de contaminación basados en factores industriales y meteorológicos.

Estructura de un Árbol CART

- **Árbol de Regresión:**
  - **Predicción de Temperaturas Futuras:** Estimar las temperaturas futuras basadas en datos históricos y condiciones actuales.
  - **Modelado de la Deforestación: Predecir la cantidad de deforestación en una región específica basada en variables económicas y ambientales.**

Ventajas y Desventajas de los Árboles CART

Ventajas	Desventajas
Fácil de entender y visualizar.	Tienden a sobreajustar si no se poda adecuadamente.
Puede manejar tanto tareas de clasificación como de regresión.	Pequeños cambios en los datos pueden resultar en árboles diferentes.
Captura relaciones no lineales entre características	Pueden volverse ineficaces con muchas características y alta dimensionalidad.

Algos de Cart

- **Decision Tree Classifier:** Algoritmo de clasificación que divide los datos en subconjuntos basados en el valor de las características.
  - **Usos Ambientales:** Clasificación de la calidad del aire, detección de áreas con riesgo de incendios forestales.
- **Decision Tree Regressor:** Algoritmo de regresión que predice valores continuos dividiendo los datos en subconjuntos homogéneos
  - **Usos Ambientales:** Predicción de la temperatura, estimación de niveles de contaminación.

Algos Cart

- **Random Forest:** Conjunto de múltiples árboles de decisión que mejora la precisión y reduce el sobreajuste.
  - **Usos Ambientales:** Modelado de la biodiversidad, predicción de patrones de precipitación.
- **Gradient Boosting Trees:** Conjunto de árboles de decisión entrenados secuencialmente para corregir los errores de los árboles anteriores
  - **Usos Ambientales:** Predicción de eventos climáticos extremos, estimación de emisiones de gases de efecto invernadero.

Gini y Entropía

Gini y Entropía

- **Gini:** El índice de Gini mide la impureza o la dispersión de las clases en un nodo. Se calcula como la probabilidad de que un elemento seleccionado al azar sea clasificado incorrectamente si se asigna a una clase basada en la distribución de las clases en ese nodo.

- **Entropía :** La entropía mide la cantidad de desorden o incertidumbre en un nodo. Es una medida de la impredecibilidad de la información contenida en un nodo y se usa para decidir la mejor división en árboles de decisión.

Gini

$$GiniIndex = 1 - \sum_{j} p_j^2$$

Probabilidad en la clase j

Pureza de un nodo

Si gini = 0 -> todas las instancias de entrenamiento pertenecen a la misma clase

Si gini = 1 -> todas las instancias de entrenamiento pertenecen a diferentes clases

Entropía

$$Entropy = - \sum_{j} p_j \cdot log_2 \cdot p_j$$

Probabilidad en la clase j

Desorden de la información

Si Entropía = 0 -> Los nodos son puros

Si Entropía = 1 -> Los nodos tienen demasiadas clases

Gini

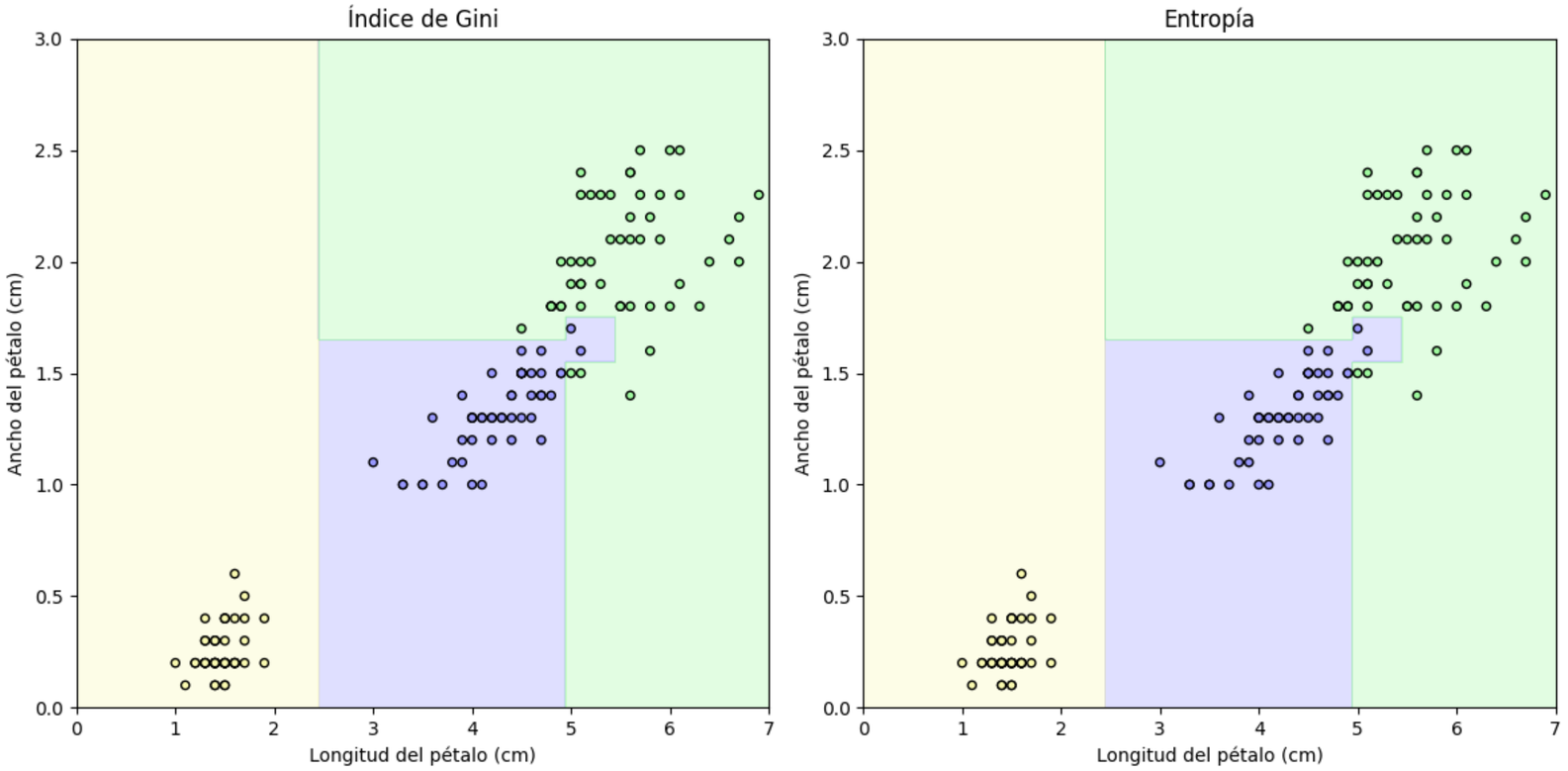
Utilizado cuando se requiere una rápida evaluación de la impureza de un nodo.

Una medida de impureza que sea simple y eficiente computacionalmente.

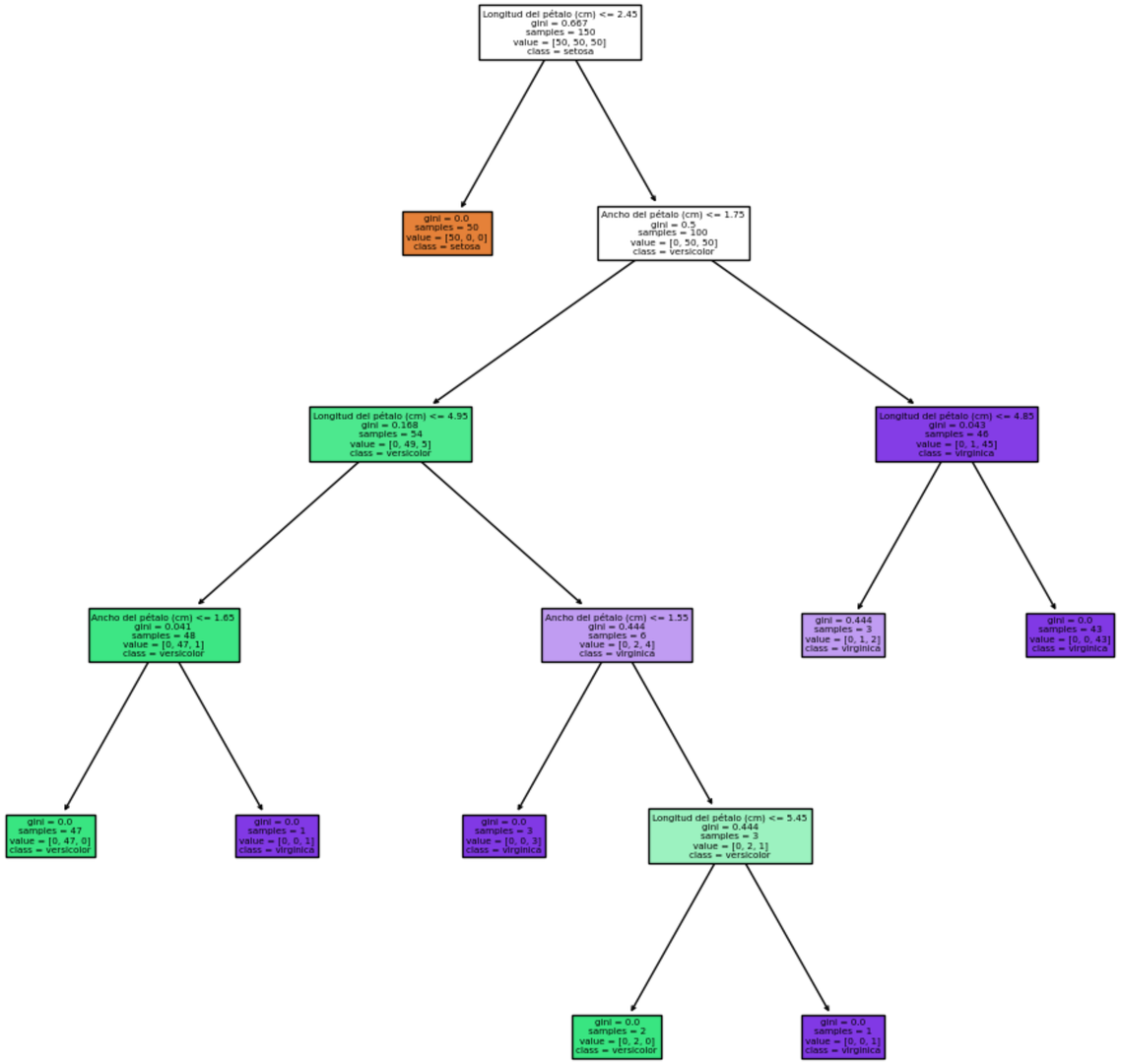
Entropía

Una medida que sea más sensible a los cambios en la distribución de clases

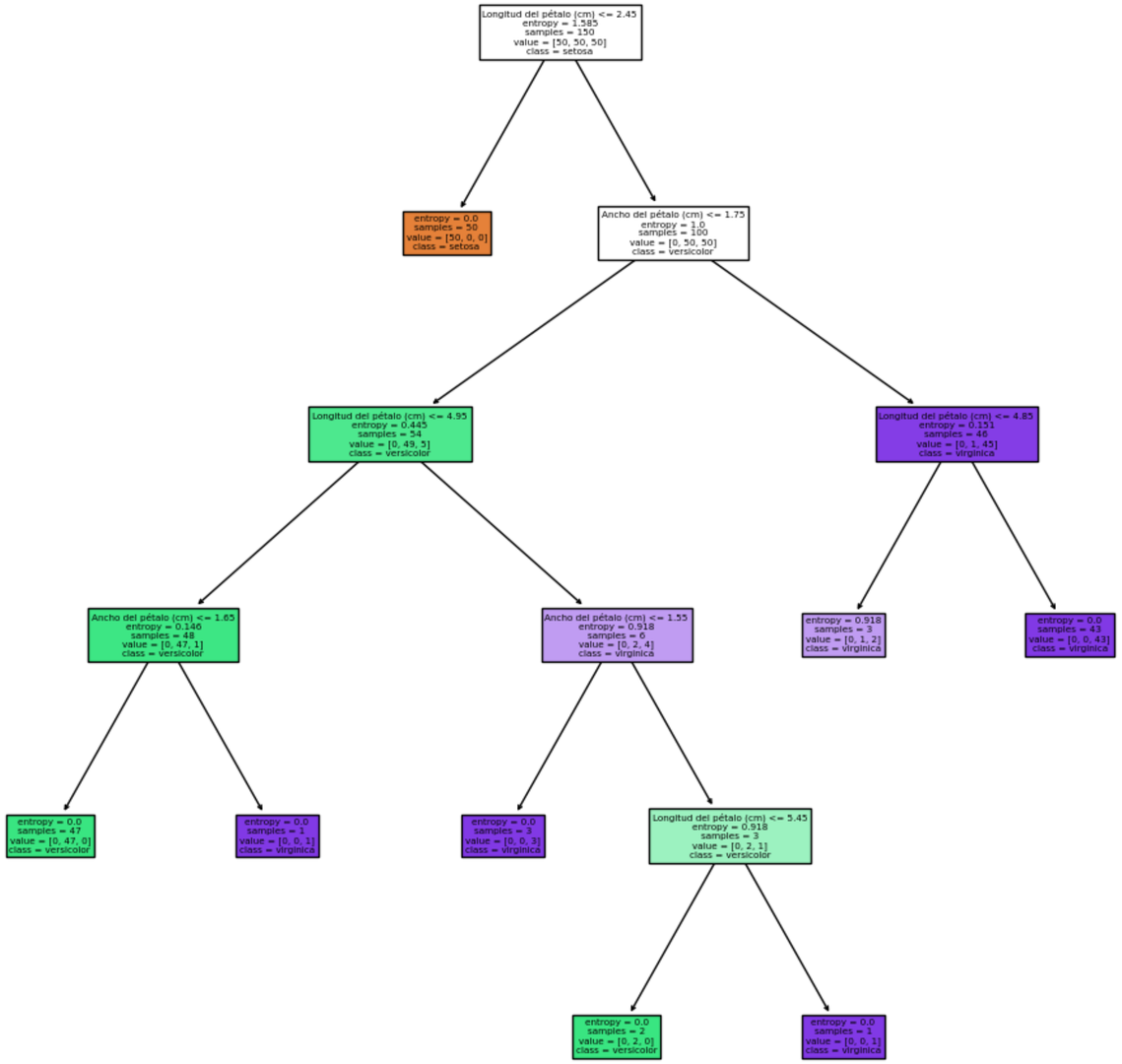
Una medida de impureza más rigurosa que capture mejor la incertidumbre de la distribución de clases


Índice de Gini
Entropía
Ancho del pétalo (cm)
Longitud del pétalo (cm)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
1
2
3
4
5
6
7

Árbol de Decisión con Índice de Gini



Árbol de Decisión con Entropía



```
gini_tree_clf = DecisionTreeClassifier(criterion='gini', random_state=42)
entropy_tree_clf = DecisionTreeClassifier(criterion='entropy', random_state=42)
```

Regularización de los hiperparametros

Regularización de los hiperparametros

- Max_depth: Ayuda a controlar el overfitting
- Max_featues: Máximo nivel de características usada en cada nodo
- Max_leaf_node: Máximo nivel de hojas por nodo
- Min_samples_split: Minimo número de muestras de un nodo después de split
- min_weigths_fraction_leaf: Fracción de instancias ponderadas

Max_depth

- Controla la profundidad máxima del árbol de decisión.

  i) **Para prevenir el overfitting**: Al limitar la profundidad del árbol, se *evita que el modelo se ajuste demasiado* a los datos de entrenamiento, lo que ayuda a mejorar la generalización en datos nuevos.

  ii) **Para mejorar la interpretabilidad**: Un árbol menos profundo es más fácil de entender y visualizar.

Max_features

- Número máximo de características a considerar para encontrar la mejor división en cada nodo.

  i) **Para reducir la variabilidad**: *Al limitar el número de características* consideradas, se puede reducir la variabilidad en los árboles de decisión individuales dentro de un ensemble, como en Random Forests

  ii) **Para mejorar la eficiencia computacional**: Considerar menos características puede reducir el tiempo de entrenamiento.

Max_leaf_nodes

- Número máximo de nodos hoja en el árbol

  i) **Para controlar el crecimiento del árbol**: Limitar el número de nodos hoja puede prevenir que el árbol se vuelva demasiado complejo y sobreajuste los datos

  ii) **Para mejorar la generalización**: Menos nodos hoja pueden ayudar al modelo a generalizar mejor en nuevos datos.

Min_samples_split

- Número mínimo de muestras que debe tener un nodo para poder dividirse.

  i) **Para evitar la división de nodos pequeños**: Esto previene que el modelo se ajuste a ruidos o peculiaridades en los datos.

  ii) **Para controlar la complejidad del árbol**: Aumentar este valor puede simplificar el árbol y mejorar la generalización.

Min_weight_fraction_leaf

- Fracción mínima del peso total (suma de los pesos de todas las instancias) que debe tener un nodo hoja.

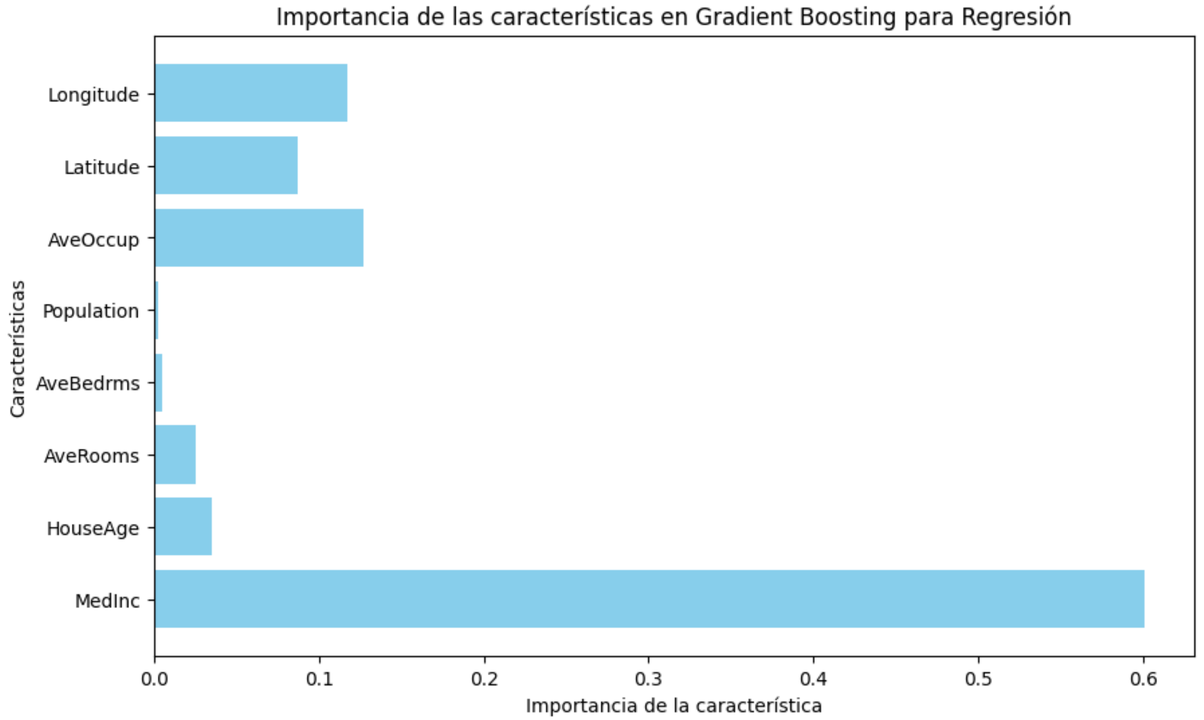
  i) **Para manejar datos desbalanceados**: *Esto es útil en conjuntos de datos donde algunas clases están sobrerrepresentadas.*

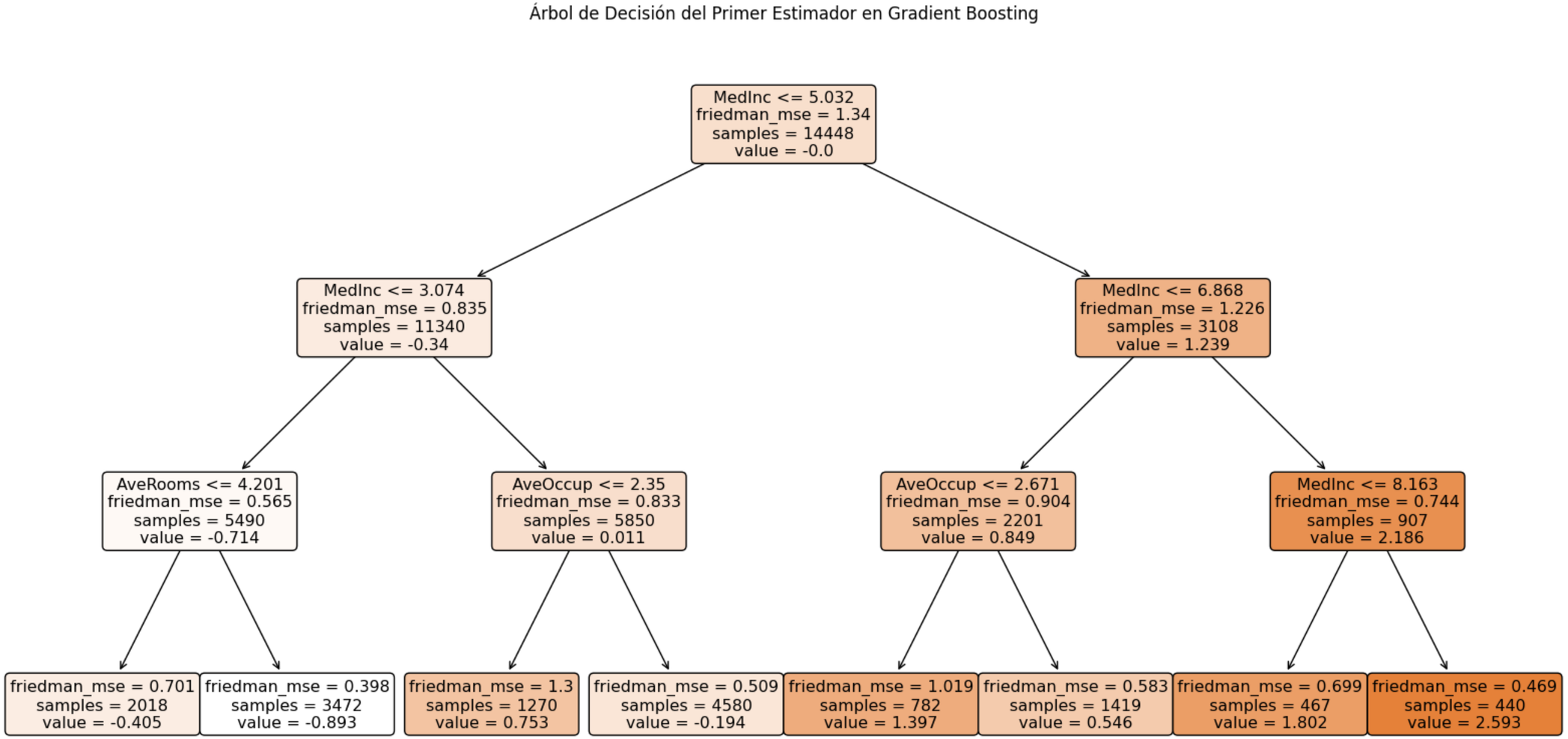
  ii) **Para mejorar la estabilidad del árbol**: Asegura que los nodos hoja contengan suficientes datos para hacer predicciones robustas.

Regresión con Caret

Regresión con Caret

- Una diferencia importante con los problemas de clasificación, que busca minimizar la impureza , la regresión lo que busca basado en problemas caret es dividir el conjunto de entrenamiento de manera que minimice el MSE




Árbol de Decisión del Primer Estimador en Gradient Boosting
MedInc <= 5.032
friedman_mse = 1.34
samples = 14448
value = -0.0
MedInc <= 3.074
friedman_mse = 0.835
samples = 11340
value = -0.34
MedInc <= 6.868
friedman_mse = 1.226
samples = 3108
value = 1.239
AveRooms <= 4.201
friedman_mse = 0.565
samples = 5490
value = -0.714
AveOccup <= 2.35
friedman_mse = 0.833
samples = 5850
value = 0.011
AveOccup <= 2.671
friedman_mse = 0.904
samples = 2201
value = 0.849
MedInc <= 8.163
friedman_mse = 0.744
samples = 907
value = 2.186
friedman_mse = 0.701
samples = 2018
value = -0.405
friedman_mse = 0.398
samples = 3472
value = -0.893
friedman_mse = 1.3
samples = 1270
value = 0.753
friedman_mse = 0.509
samples = 4580
value = -0.194
friedman_mse = 1.019
samples = 782
value = 1.397
friedman_mse = 0.583
samples = 1419
value = 0.546
friedman_mse = 0.699
samples = 467
value = 1.802
friedman_mse = 0.469
samples = 440
value = 2.593

Regresión con Caret

- Las decisiones o limites de decisión en los caret de regresión tienden a ser octogonales
  Divisiones Binarias: Cada nodo de un árbol de decisión divide el espacio de características en dos regiones basadas en una única característica.
  Divisiones Eje-alineadas: Estas divisiones son perpendiculares a uno de los ejes de las características. Por ejemplo, una división puede ser $X1 <= 5.5$ para una característica $X1$.

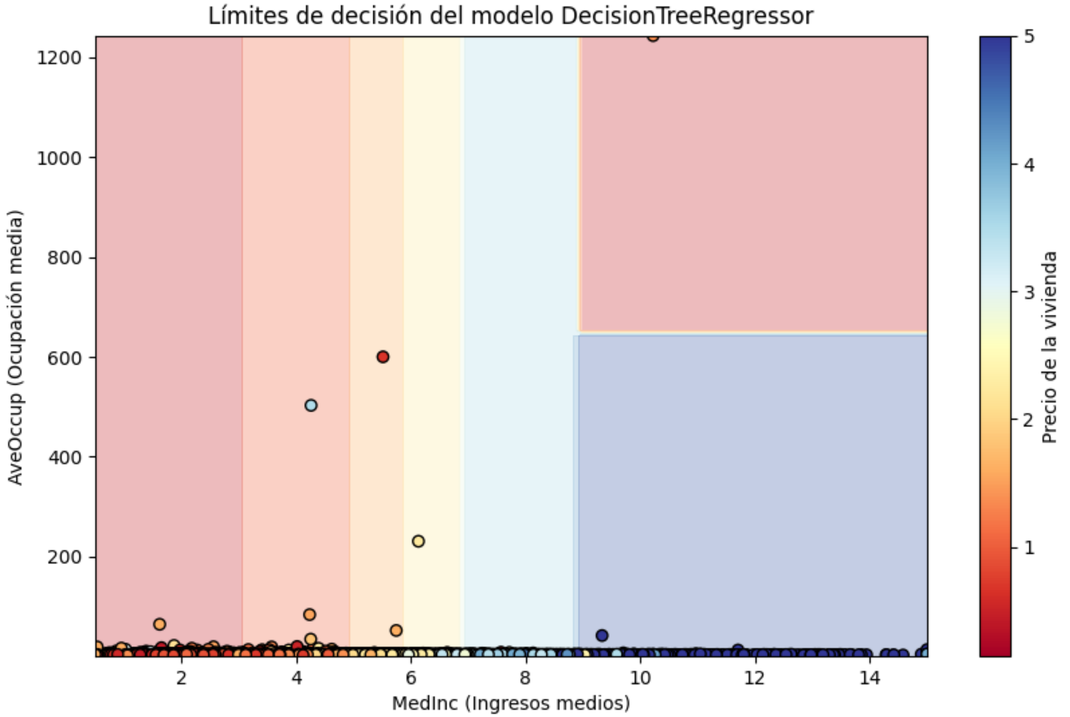
Regresión con Caret

- Los límites perpendiculares son fáciles de entender e interpretar. Podemos ver directamente cómo una característica específica afecta la clasificación.

- **Reglas de Decisión Claras**: Cada división del nodo representa una regla de decisión clara, como "Si `X1 <= 5.5`, entonces la clase es A; de lo contrario, es B".

- Debido a su estructura rígida, los árboles profundos pueden sobreajustarse a los datos de entrenamiento, capturando ruido en lugar de patrones verdaderos.

- **Balancear con Podado**: Para evitar el overfitting, se pueden usar técnicas como el podado (pruning) para simplificar el árbol.


Límites de decisión del modelo DecisionTreeRegressor
AveOccup (Ocupación media)
MedInc (Ingresos medios)
Precio de la vivienda
1200
1000
800
600
400
200
2
4
6
8
10
12
14
5
4
3
2
1

Regresión con Caret

- Los límites perpendiculares son fáciles de entender e interpretar. Podemos ver directamente cómo una característica específica afecta la clasificación.

- **Reglas de Decisión Claras**: Cada división del nodo representa una regla de decisión clara, como "Si `X1 <= 5.5`, entonces la clase es A; de lo contrario, es B".

- Debido a su estructura rígida, los árboles profundos pueden sobreajustarse a los datos de entrenamiento, capturando ruido en lugar de patrones verdaderos.

- **Balancear con Podado**: Para evitar el overfitting, se pueden usar técnicas como el podado (pruning) para simplificar el árbol.

Random Forest y modelos robustos

Random Forest

- Es un modelo basado en la construcción de múltiples árboles de decisión y la combinación de sus resultados para mejorar la precisión y robustez de las predicciones.

- Se basa en modelos de ensamble : es una técnica que combina múltiples modelos individuales para crear un modelo más poderoso y preciso.

Random Forest

- Bootstrap Aggregating (Bagging): Se crean múltiples subconjuntos del conjunto de datos original mediante muestreo con reemplazo (bootstrap).
- Cada subconjunto se utiliza para entrenar un árbol de decisión independiente.
- Para la clasificación, se toma la clase más votada (la moda) entre todos los árboles.
- Para la regresión, se promedia la predicción de todos los árboles.

Voting Classifier

- Un Voting Classifier es un modelo de ensamble que combina las predicciones de múltiples modelos base (de diferentes tipos) para mejorar la precisión y robustez de las predicciones.Cada subconjunto se utiliza para entrenar un árbol de decisión independiente.

- Hard Voting: Cada modelo base realiza una predicción y se toma la clase más votada como la predicción final.

- Soft Voting: Cada modelo base predice las probabilidades de cada clase y se promedian estas probabilidades.

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Cargar los datos
X, y = load_iris(return_X_y=True)

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Definir los modelos base
clf1 = LogisticRegression(random_state=42)
clf2 = RandomForestClassifier(n_estimators=100, random_state=42)
clf3 = SVC(probability=True, random_state=42)

# Crear el Voting Classifier con Soft Voting
voting_clf = VotingClassifier(estimators=[
    ('lr', clf1),
    ('rf', clf2),
    ('svc', clf3)],
    voting='soft')

# Entrenar el Voting Classifier
voting_clf.fit(X_train, y_train)

# Realizar predicciones
y_pred = voting_clf.predict(X_test)

# Calcular la precisión
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del Voting Classifier: {accuracy}")

# También se puede evaluar cada modelo base por separado
for clf, label in zip([clf1, clf2, clf3, voting_clf], ['Logistic Regression', 'Random Forest', 'SVM', 'Voting Classifier']):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"Precisión de {label}: {accuracy_score(y_test, y_pred)}")
```

```
Precisión del Voting Classifier: 1.0
Precisión de Logistic Regression: 1.0
Precisión de Random Forest: 1.0
Precisión de SVM: 1.0
Precisión de Voting Classifier: 1.0
```

Bagging

- Es una técnica o estrategia de ensamble que implica entrenar múltiples modelos en diferentes subconjuntos del conjunto de datos original generados mediante muestreo con reemplazo.
- **Muestreo con Reemplazo**: Se generan varios subconjuntos del conjunto de datos de entrenamiento seleccionando aleatoriamente ejemplos con reemplazo.
- **Entrenamiento de Modelos**: Se entrena un modelo separado en cada subconjunto.
- **Agregación de Predicciones**: Las predicciones de todos los modelos se combinan mediante votación (para clasificación) o promediado (para regresión).

Bagging

- Es una técnica o estrategia de ensamble que implica entrenar múltiples modelos en diferentes subconjuntos del conjunto de datos original generados mediante muestreo con reemplazo.
- **Muestreo con Reemplazo**: Se generan varios subconjuntos del conjunto de datos de entrenamiento seleccionando aleatoriamente ejemplos con reemplazo.
- **Entrenamiento de Modelos**: Se entrena un modelo separado en cada subconjunto.
- **Agregación de Predicciones**: Las predicciones de todos los modelos se combinan mediante votación (para clasificación) o promediado (para regresión).

Bagging

- Es una técnica o estrategia de ensamble que implica entrenar múltiples modelos en diferentes subconjuntos del conjunto de datos original generados mediante muestreo con reemplazo.
- **Muestreo con Reemplazo**: Se generan varios subconjuntos del conjunto de datos de entrenamiento seleccionando aleatoriamente ejemplos con reemplazo.
- **Entrenamiento de Modelos**: Se entrena un modelo separado en cada subconjunto.
- **Agregación de Predicciones**: Las predicciones de todos los modelos se combinan mediante votación (para clasificación) o promediado (para regresión).

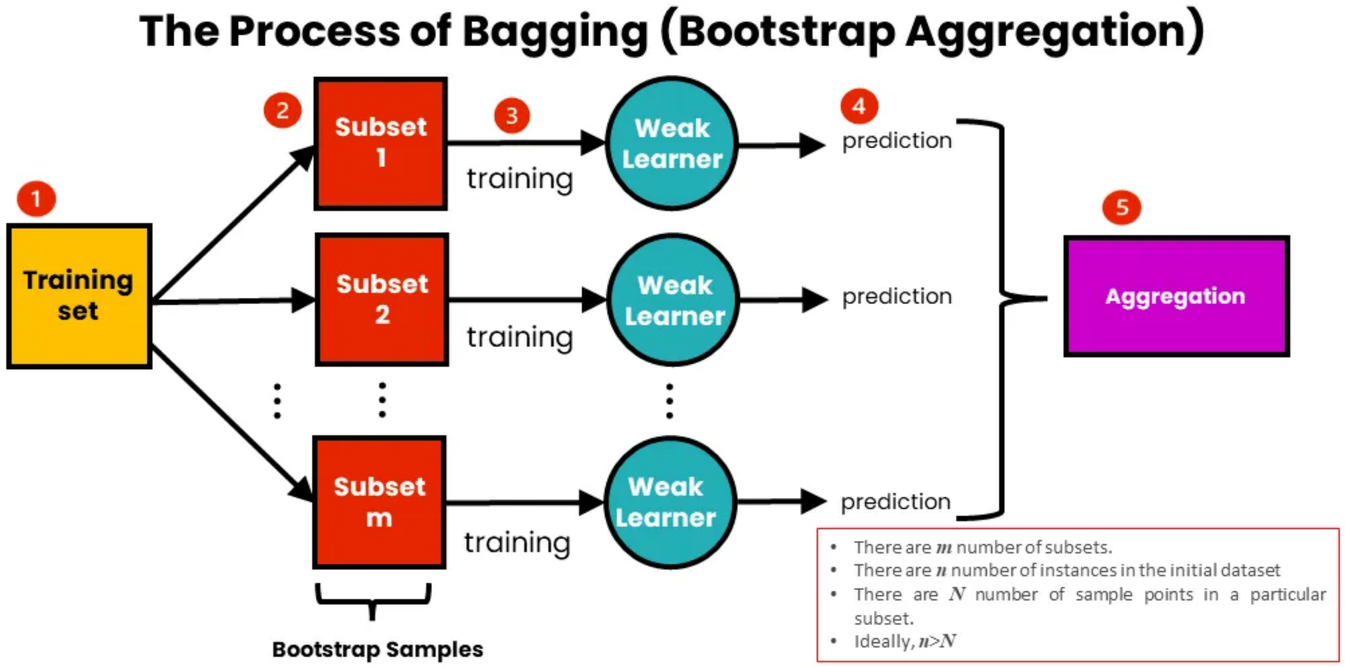
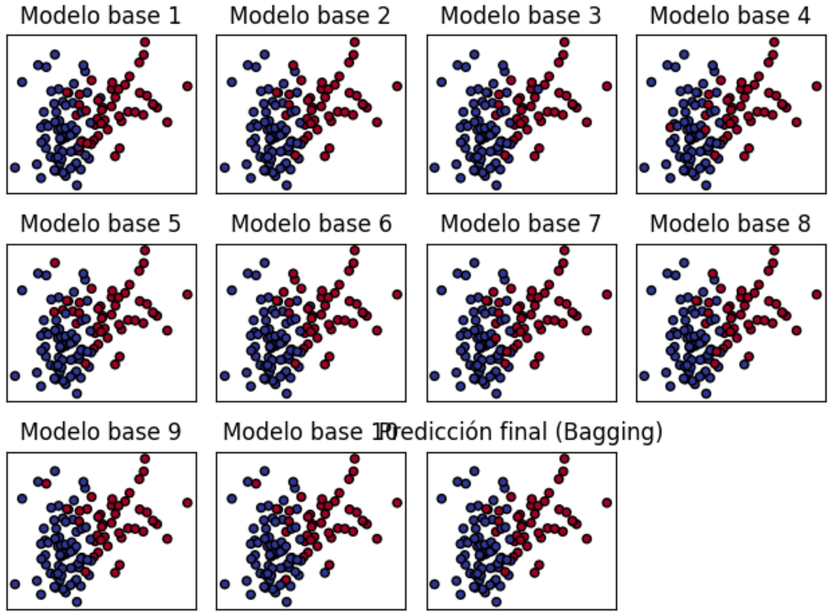


Imagen tomada de : https://medium.com/@brijesh_soni/boost-your-machine-learnir
models-with-bagging-a-powerful-ensemble-learning-technique-692bfc4d1a51

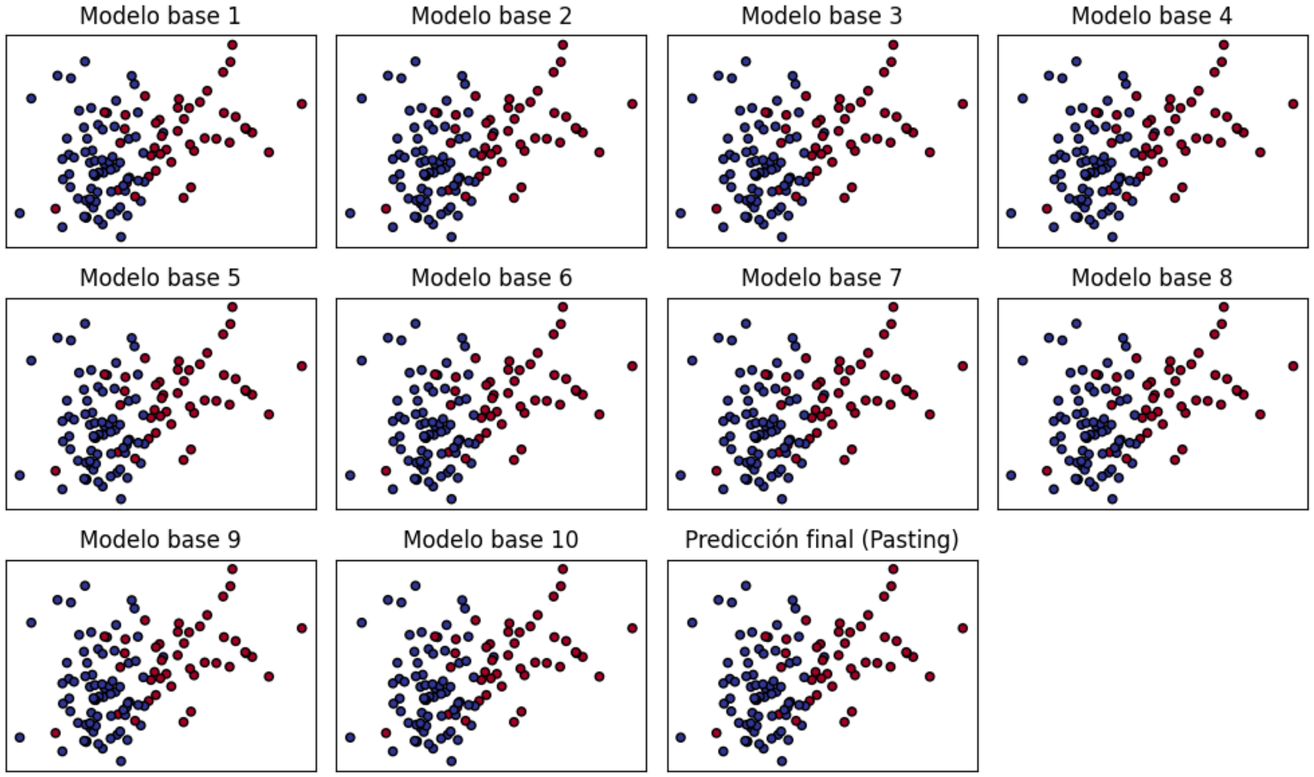
```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Base model selection: Decision Tree Classifier
base_model = DecisionTreeClassifier(random_state=42)

# Create a BaggingClassifier with 10 base models
bagging_model = BaggingClassifier(base_model, n_estimators=10, random_state=42)

# Fit the bagging model to the training data
bagging_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = bagging_model.predict(X_test)

# Calculate accuracy of the bagging model
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy of Bagging Classifier: {:.2f}".format(accuracy))
```



Pasting

- **Muestreo sin Reemplazo**: Cada subconjunto de datos de entrenamiento no contiene duplicados.
- **Aumenta la diversidad entre los modelos, lo que puede mejorar la precisión general del ensamble**

```
pasting_model = BaggingClassifier(base_model, n_estimators=10, bootstrap=False, random_state=42)

# Ajustar el modelo de pasting a los datos de entrenamiento
pasting_model.fit(X_train, y_train)

# Realizar predicciones en los datos de prueba
y_pred = pasting_model.predict(X_test)

# Calcular la precisión del modelo de pasting
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del Pasting Classifier: {:.2f}".format(accuracy))

# Visualización de las predicciones de cada modelo base y la predicción final
plt.figure(figsize=(10, 6))

# Predicciones de cada modelo base
for i, estimator in enumerate(pasting_model.estimators_):
    y_pred_base = estimator.predict(X_test)
    plt.subplot(3, 4, i + 1)
    plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred_base, cmap=plt.cm.RdYlBu, edgecolor='k', s=20)
    plt.title(f'Modelo base {i+1}')
    plt.xticks(())
    plt.yticks(())

# Predicción final del modelo de Pasting
plt.subplot(3, 4, 11)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap=plt.cm.RdYlBu, edgecolor='k', s=20)
plt.title('Predicción final (Pasting)')
plt.xticks(())
plt.yticks(())

plt.tight_layout()
plt.show()
```



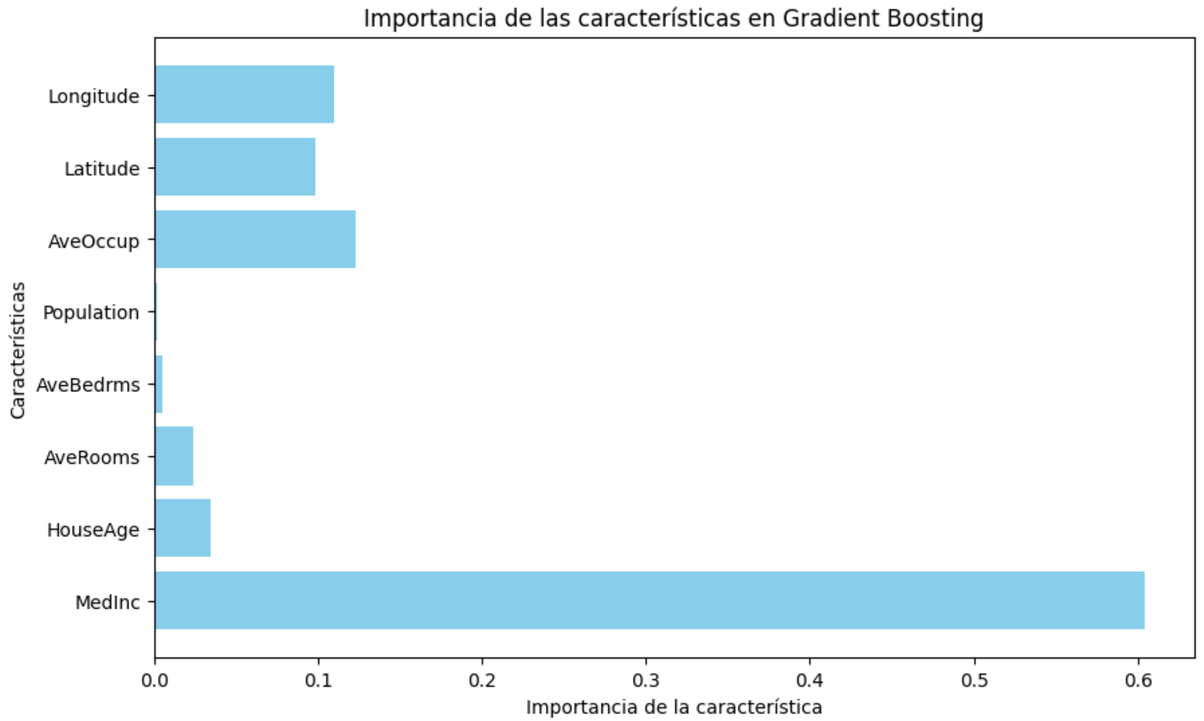
Histogram-based Gradient Boosting

- Una variante de Gradient Boosting que utiliza histogramas para acelerar el entrenamiento del modelo. -> curiosamente esto hace que pierda precisión
- Los datos se agrupan en histogramas en lugar de considerar todos los datos individuales en cada iteración, lo que reduce el costo computacional
- Maneja grandes conjuntos de datos de manera eficiente.

Gradient Boosting

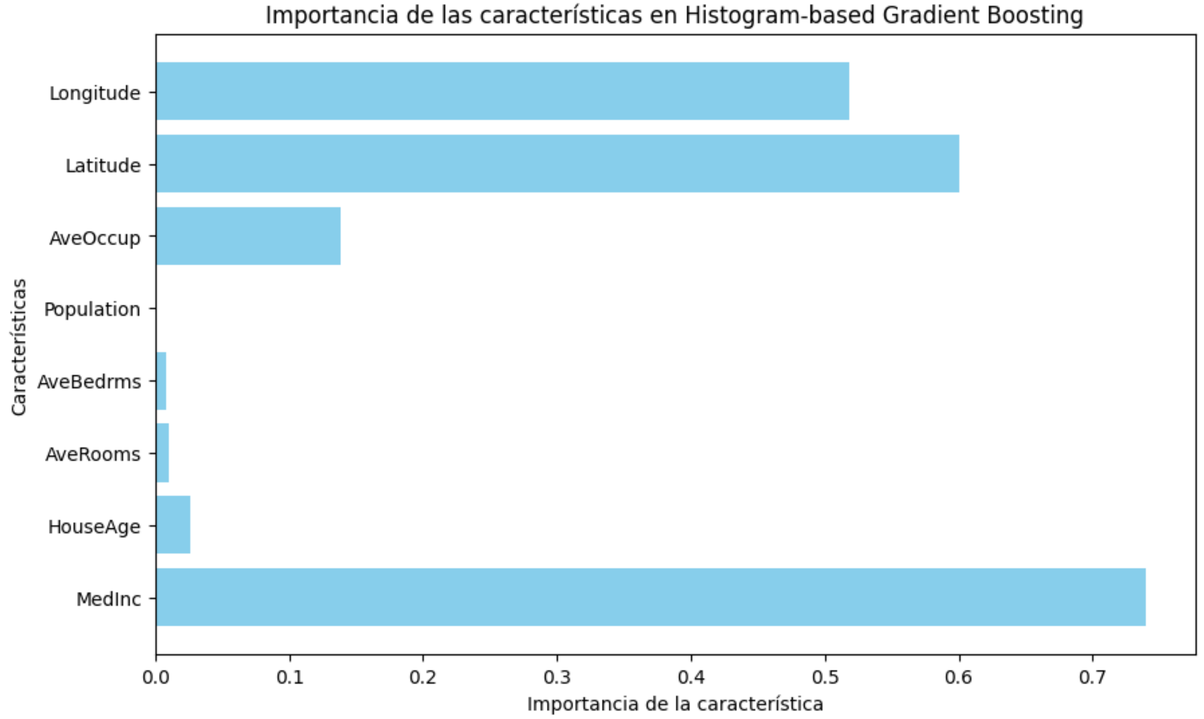
- Un método de ensamble que crea un modelo fuerte combinando predicciones de múltiples modelos débiles (normalmente árboles de decisión) en una secuencia iterativa
- Los modelos se entrenan secuencialmente, cada uno tratando de corregir los errores del modelo anterior.
- Utiliza la minimización de una función de pérdida para optimizar el modelo.
- Tiene Alta precisión y rendimiento.
- Lo malo : Requiere ajuste de hiperparámetros, lo que puede ser computacionalmente costoso.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

# Cargar los datos de California Housing
housing = fetch_california_housing()
X = housing.data
y = housing.target

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

# Definir y entrenar el modelo Gradient Boosting para regresión
gbrt = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42
gbrt.fit(X_train, y_train)

# Realizar predicciones y calcular el error cuadrático medio
y_pred = gbrt.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Error cuadrático medio: {mse}")

# Graficar la importancia de las características
importancias = gbrt.feature_importances_
caracteristicas = housing.feature_names

plt.figure(figsize=(10, 6))
plt.barh(caracteristicas, importancias, color='skyblue')
plt.xlabel("Importancia de la característica")
plt.ylabel("Características")
plt.title("Importancia de las características en Gradient Boosting")
plt.show()
```



Característica	Random Forest	Gradient Boosting	Histogram-based Gradient Boosting
Definición	Ensamble de árboles de decisión entrenados en subconjuntos del dataset con reemplazo.	Ensamble secuencial de modelos débiles (árboles de decisión) optimizando errores previos.	Variante de Gradient Boosting que utiliza histogramas para acelerar el entrenamiento.
Funcionamiento	- Cada árbol se entrena en un subconjunto diferente (bootstrap). - Predicciones mediante votación o promediado.	- Modelos entrenados secuencialmente. - Minimiza una función de pérdida en cada iteración.	- Agrupa los datos en histogramas. - Reduce el costo computacional. - Optimiza los errores previos.
Ventajas	- Alta robustez y precisión. - Reduce el overfitting. - Maneja datos faltantes y atípicos bien.	- Alta precisión. - Flexibilidad para diferentes tipos de datos. - Captura relaciones complejas.	- Más rápido que el Gradient Boosting tradicional. - Eficiente con grandes datasets. - Mantiene alta precisión.
Desventajas	- Menos interpretable que un solo árbol de decisión. - Puede ser computacionalmente costoso.	- Susceptible al overfitting si no se ajusta adecuadamente. - Requiere ajuste de muchos hiperparámetros.	- Requiere buena estrategia de discretización. - Puede perder precisión si no se ajusta bien.
Cuándo Usarlo	- Problemas donde la precisión y la robustez son cruciales. - Datasets con valores atípicos y faltantes.	- Problemas donde se necesita alta precisión. - Datasets con patrones complejos. - Capacidad computacional adecuada para ajustar hiperparámetros.	- Grandes conjuntos de datos. - Situaciones donde la velocidad es más importante que una pequeña pérdida de precisión.
Ejemplo de Implementación	```python import numpy as np from sklearn.datasets import load_breast_cancer from sklearn.ensemble import RandomForestClassifier	```python import numpy as np from sklearn.datasets import fetch_california_housing from sklearn.ensemble import GradientBoostingRegressor	```python import numpy as np from sklearn.datasets import fetch_california_housing from sklearn.ensemble import HistGradientBoostingRegressor

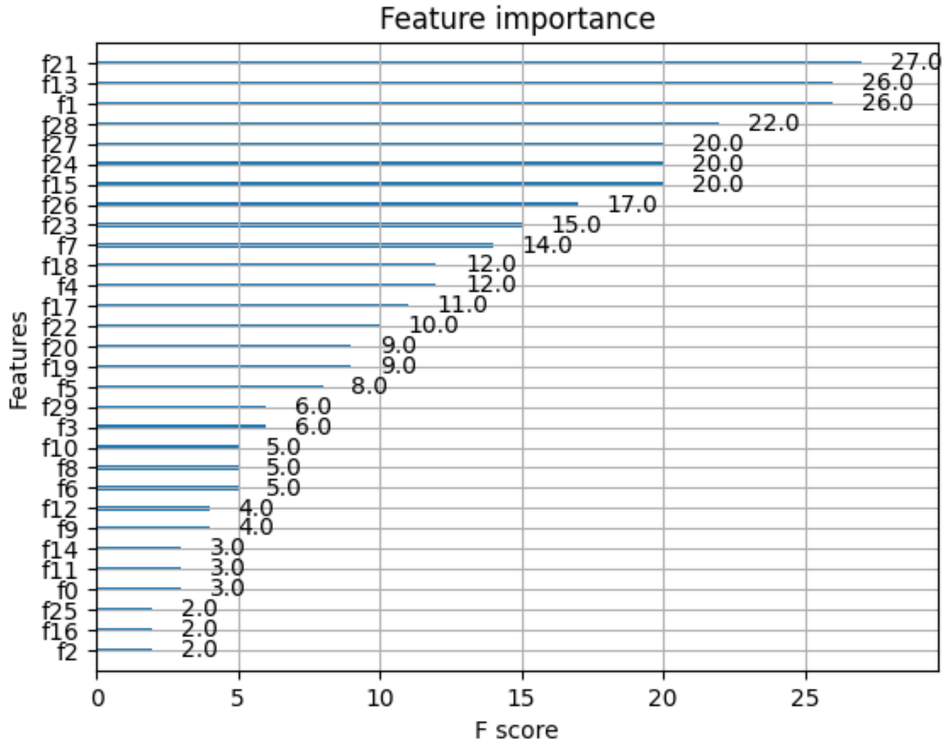
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor, HistGradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.inspection import permutation_importance

# Cargar los datos de California Housing
housing = fetch_california_housing()
X = housing.data
y = housing.target

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

# Definir y entrenar el modelo Gradient Boosting para regresión
gbrt = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
gbrt.fit(X_train, y_train)

# Realizar predicciones y calcular el error cuadrático medio
y_pred = gbrt.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Error cuadrático medio (Gradient Boosting): {mse}")

# Graficar la importancia de las características
importancias = gbrt.feature_importances_
caracteristicas = housing.feature_names

plt.figure(figsize=(10, 6))
plt.barh(caracteristicas, importancias, color='skyblue')
plt.xlabel("Importancia de la característica")
plt.ylabel("Características")
plt.title("Importancia de las características en Gradient Boosting")
plt.show()

# Definir y entrenar el modelo Histogram-based Gradient Boosting para regresión
hist_gbrt = HistGradientBoostingRegressor(max_iter=100, learning_rate=0.1, max_depth=3, random_state=
hist_gbrt.fit(X_train, y_train)

# Realizar predicciones y calcular el error cuadrático medio
y_pred_hist = hist_gbrt.predict(X_test)
mse_hist = mean_squared_error(y_test, y_pred_hist)
print(f"Error cuadrático medio (Histogram-based Gradient Boosting): {mse_hist}")

# Calcular la importancia de características usando permutation importance
result = permutation_importance(hist_gbrt, X_test, y_test, n_repeats=10, random_state=42, n_jobs=-1)

# Graficar la importancia de las características
importancias_hist = result.importances_mean

plt.figure(figsize=(10, 6))
plt.barh(caracteristicas, importancias_hist, color='skyblue')
plt.xlabel("Importancia de la característica")
plt.ylabel("Características")
plt.title("Importancia de las características en Histogram-based Gradient Boosting")
plt.show()
```



XGBoost

- Es una implementación optimizada y altamente eficiente del algoritmo de Gradient Boosting. Es conocida por su velocidad y rendimiento superior en tareas de clasificación y regresión.
- Utiliza técnicas de ensamble para combinar múltiples árboles de decisión, optimizando errores iterativamente.
- Emplea técnicas avanzadas como regularización para prevenir el overfitting y soporta paralelización para acelerar el entrenamiento.
- Alta precisión y rendimiento en competiciones de machine learning.
- Manejo eficiente de datos faltantes y soporte para diferentes tipos de pérdidas.
- Capacidad para realizar poda de ramas (pruning) que no mejoran la precisión del modelo.

```python
import numpy as np
import xgboost as xgb
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score

# Cargar los datos de cáncer de mama
data = load_breast_cancer()
X = data.data
y = data.target

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

# Crear el clasificador XGBoost
xgb_clf = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

# Definir los hiperparámetros para Grid Search
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.7, 0.8, 0.9]
}

# Configurar GridSearchCV
grid_search = GridSearchCV(estimator=xgb_clf, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1

# Entrenar el modelo con Grid Search
grid_search.fit(X_train, y_train)

# Obtener los mejores parámetros y el mejor modelo
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print(f"Mejores parámetros: {best_params}")

# Realizar predicciones con el mejor modelo
y_pred = best_model.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo XGBoost optimizado: {accuracy}")

# Gráfica de importancia de características
xgb.plot_importance(best_model)
plt.show()
```



Taller en clase (Examen #2)

Examen no sorpresa

- Vamos a analizar la siguiente base de datos (calidad del agua) https://www.kaggle.com/datasets/adityakadiwal/water-potability/data
- Vamos a usar dos modelos vistos en clase para predecir la calidad del agua
- Vamos a hacer un EDA para entender las variables que vamos a modelar
- Tienen el resto de la clase para terminar este examen en grupos