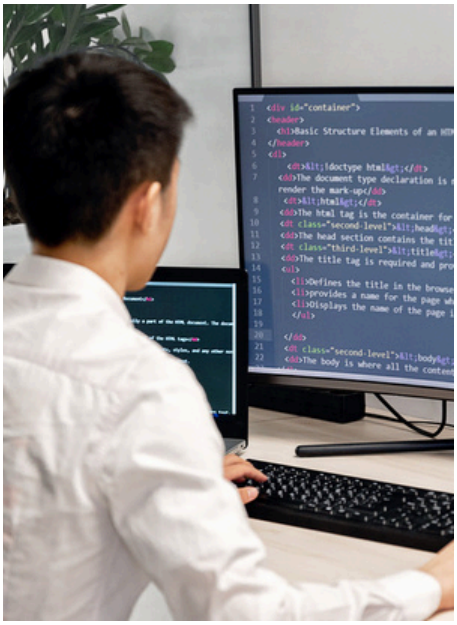
Árboles de Decisión

1 Árboles de Decisión

Los árboles de decisión son una técnica de modelado predictivo utilizada en el campo del aprendizaje automático y la inteligencia artificial. Se utilizan para modelar decisiones y sucesos mediante una estructura en forma de árbol. Esta estructura consta de nodos internos que representan pruebas sobre atributos (por ejemplo, "¿es el ingreso mayor a $50,000?") y ramas que representan los posibles resultados de estas pruebas. Los nodos hoja representan las decisiones finales o las clasificaciones que se derivan del árbol.



Algoritmos de Construcción de Árboles de Decisión

1. ID3 (Iterative Dichotomiser 3):

Descripción	ID3 es uno de los primeros algoritmos de árboles de decisión desarrollados por Ross Quinlan en la década de 1980. Su objetivo es construir un árbol de decisión que particione los datos en subconjuntos homogéneos basados en características específicas.
Funcionamiento	ID3 utiliza el concepto de entropía y ganancia de información para determinar la mejor característica en cada paso de construcción del árbol. La entropía se refiere a la medida de incertidumbre en un conjunto de datos, mientras que la ganancia de información se refiere a la reducción de entropía obtenida al dividir los datos en función de una característica específica.

Ventajas	ID3 es simple y fácil de entender. Además, puede manejar tanto datos categóricos como numéricos.
Limitaciones	ID3 tiende a sobreajustar los datos de entrenamiento debido a su enfoque de búsqueda exhaustiva para encontrar la mejor característica en cada paso. Además, no maneja bien los valores faltantes y los datos ruidosos.

2. C4.5:

Descripción	C4.5 es una extensión del algoritmo ID3 desarrollado también por Ross Quinlan. Se mejoró para abordar algunas de las limitaciones de ID3.
Funcionamiento	C4.5 utiliza la ganancia de información normalizada en lugar de la ganancia de información bruta utilizada por ID3. Esto ayuda a penalizar las características con una gran cantidad de valores únicos. Además, C4.5 puede manejar eficazmente valores faltantes y datos numéricos mediante técnicas de discretización.
Ventajas	C4.5 es más robusto que ID3 y puede manejar mejor los datos ruidosos y los valores faltantes. También puede manejar tanto datos categóricos como numéricos.
Limitaciones	A pesar de sus mejoras, C4.5 todavía puede tender al sobreajuste en conjuntos de datos pequeños o con características con muchos valores únicos.

3. CART (Classification and Regression Trees):

Descripción	CART es otro algoritmo popular de árboles de decisión que fue desarrollado por Leo Breiman en la década de 1980. A diferencia de ID3 y C4.5, CART puede construir tanto árboles de clasificación como árboles de regresión.
Funcionamiento	CART utiliza una estrategia de división recursiva binaria, lo que significa que cada nodo del árbol divide el conjunto de datos en dos subconjuntos utilizando una regla de decisión. La división se realiza seleccionando la característica y el punto de corte que maximiza la homogeneidad de los subconjuntos resultantes.
Ventajas	CART es eficiente y puede manejar tanto datos categóricos como numéricos. Además, es robusto frente al ruido y valores atípicos.
Limitaciones	CART tiende a producir árboles más profundos y complejos, lo que puede llevar al sobreajuste en conjuntos de datos pequeños. Además, CART no maneja bien los valores faltantes en los datos.

Parámetros Importantes en la Construcción de Árboles de Decisión

- Profundidad máxima del árbol.
- Criterios de división (por ejemplo, ganancia de información, índice Gini).
- Mínimo de muestras requeridas para dividir un nodo.
- Mínimo de muestras requeridas en una hoja.
- Discusión sobre cómo estos parámetros afectan la estructura y complejidad del árbol.

Ejemplos y Ejercicios Prácticos

- Implementación de algoritmos ID3, C4.5 y CART utilizando bibliotecas de Python como scikit-learn.
- Aplicación de los árboles de decisión a conjuntos de datos de ejemplo para clasificación y regresión.
- Resolución de ejercicios prácticos para construir y evaluar árboles de decisión en diferentes escenarios.

Implementación de un árbol de decisión utilizando scikit-learn

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

# Cargar el conjunto de datos Iris
iris = load_iris()
X, y = iris.data, iris.target

# Dividir el conjunto de 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)

# Crear un clasificador de árbol de decisión
clf = DecisionTreeClassifier()

# Entrenar el clasificador
clf.fit(X_train, y_train)
```

```
# Evaluar el rendimiento del clasificador
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del clasificador de árbol de decisión:", accuracy)
```

Ajuste de parámetros de un árbol de decisión

```
# Crear un clasificador de árbol de decisión con parámetros
personalizados
clf = DecisionTreeClassifier(max_depth=3, min_samples_split=5,
random_state=42)

# Entrenar el clasificador con los parámetros personalizados
clf.fit(X_train, y_train)

# Evaluar el rendimiento del clasificador con los parámetros
personalizados
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del clasificador de árbol de decisión (con parámetros
personalizados):", accuracy)
```

Visualización del árbol de decisión

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

# Visualizar el árbol de decisión
plt.figure(figsize=(10, 7))
plot_tree(clf, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)
plt.show()
```

Comparación entre diferentes algoritmos de árboles de decisión

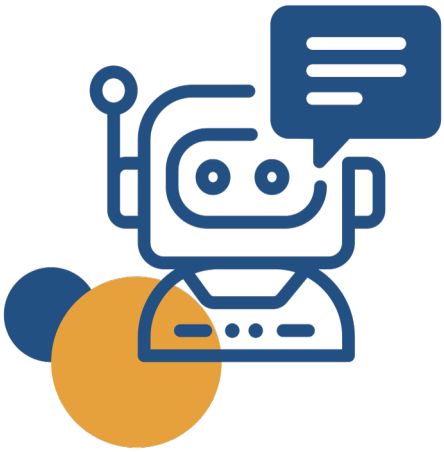
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

# Crear clasificadores con diferentes algoritmos de árboles de decisión
clf_id3 = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_c45 = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_cart = DecisionTreeClassifier(criterion='gini', random_state=42)

# Evaluar los clasificadores utilizando validación cruzada
scores_id3 = cross_val_score(clf_id3, X, y, cv=5)
scores_c45 = cross_val_score(clf_c45, X, y, cv=5)
scores_cart = cross_val_score(clf_cart, X, y, cv=5)

# Imprimir los resultados de la validación cruzada
print("Precisión media de ID3:", scores_id3.mean())
print("Precisión media de C4.5:", scores_c45.mean())
print("Precisión media de CART:", scores_cart.mean())
```

2 Bosques aleatorios

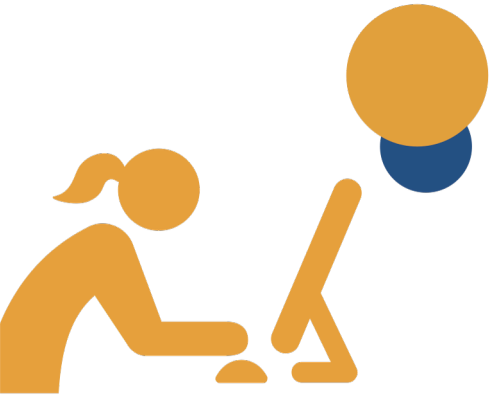
Los Bosques Aleatorios son un método de aprendizaje automático que combina múltiples árboles de decisión para obtener una predicción más precisa y robusta. La motivación detrás de los Bosques Aleatorios radica en su capacidad para reducir el sobreajuste y mejorar la generalización al promediar múltiples modelos débiles.

Construcción y entrenamiento de Bosques Aleatorios

1. **Muestreo Bootstrap:** Se construyen múltiples conjuntos de datos de entrenamiento mediante muestreo con reemplazo de los datos originales.
2. **Construcción de árboles de decisión:** Para cada conjunto de datos de entrenamiento, se entrena un árbol de decisión utilizando un subconjunto aleatorio de características en cada división del árbol.
3. **Combinación de árboles:** Se combinan los resultados de todos los árboles de decisión para obtener una predicción final, ya sea por votación (en clasificación) o por promedio (en regresión).



Parámetros importantes en los Bosques Aleatorios



- **Número de árboles (n_estimators):** Determina la cantidad de árboles en el bosque.
- **Número máximo de características (max_features):** Especifica el número máximo de características a considerar en cada división de un árbol.
- **Profundidad máxima del árbol (max_depth):** Controla la profundidad máxima de cada árbol en el bosque.

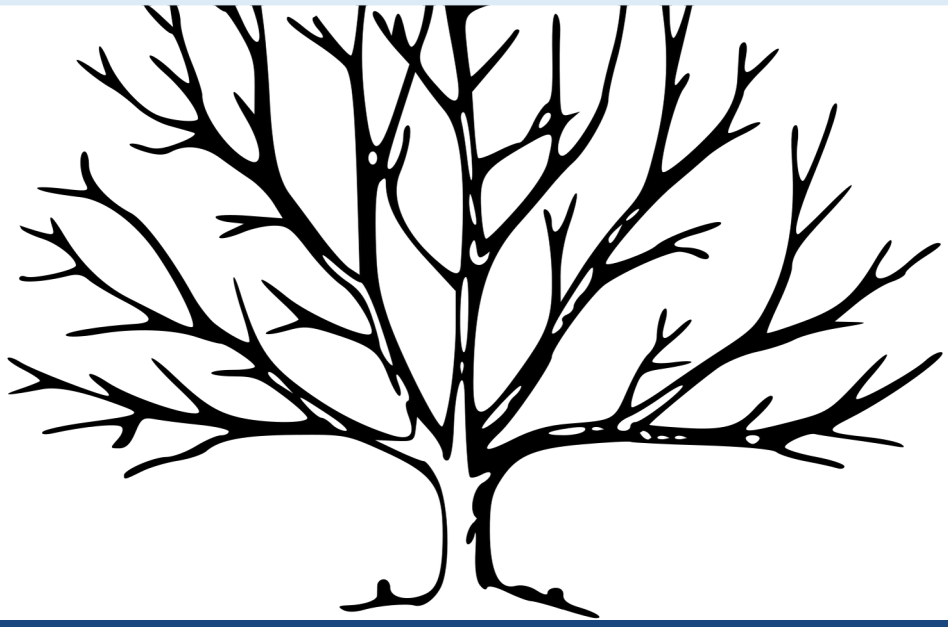
- **Número mínimo de muestras para dividir un nodo (min_samples_split):** Establece el número mínimo de muestras requeridas para dividir un nodo interno.
- **Número mínimo de muestras en cada hoja (min_samples_leaf):** Define el número mínimo de muestras requeridas en cada hoja del árbol.

Comparación con los árboles de decisión tradicionales

Los Bosques Aleatorios tienden a tener una mayor precisión y generalización en comparación con un solo árbol de decisión, especialmente en conjuntos de datos grandes y complejos.

Son menos propensos al sobreajuste debido a la diversidad introducida por el entrenamiento de múltiples árboles con diferentes subconjuntos de datos y características.

Sin embargo, los Bosques Aleatorios pueden ser computacionalmente más costosos y difíciles de interpretar en comparación con los árboles de decisión individuales.



Ejemplos y Ejercicios Prácticos

- Clasificación de especies de flores Iris utilizando un conjunto de datos de Iris.
- Predicción del precio de las viviendas utilizando un conjunto de datos de precios de viviendas.
- Ejercicio práctico: Implementación de un Bosque Aleatorio en Python utilizando la biblioteca scikit-learn y evaluación de su rendimiento en un conjunto de datos de clasificación o regresión.

Implementación básica de Bosques Aleatorios

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Cargar el conjunto de datos Iris
iris = load_iris()
X, y = iris.data, iris.target

# Dividir el conjunto de 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)

# Crear y entrenar un clasificador de Bosques Aleatorios
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Evaluar el rendimiento del clasificador
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del clasificador de Bosques Aleatorios:", accuracy)
```

Ajuste de parámetros de Bosques Aleatorios

```
# Crear y entrenar un clasificador de Bosques Aleatorios con parámetros
personalizados
clf = RandomForestClassifier(n_estimators=100, max_depth=5,
min_samples_split=2, random_state=42)
clf.fit(X_train, y_train)

# Evaluar el rendimiento del clasificador con los parámetros
personalizados
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del clasificador de Bosques Aleatorios (con parámetros
personalizados):", accuracy)
```

Comparación con árboles de decisión tradicionales

```
from sklearn.tree import DecisionTreeClassifier

# Crear y entrenar un clasificador de árbol de decisión
dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train, y_train)

# Evaluar el rendimiento del clasificador de árbol de decisión
y_pred_dt = dt_clf.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Precisión del clasificador de árbol de decisión:", accuracy_dt)
```

Ejemplo práctico de regresión con Bosques Aleatorios

```
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

```
# Cargar el conjunto de datos california House Prices
california = fetch_california_housing()
X, y = california.data, california.target

# Dividir el conjunto de 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)

# Crear y entrenar un regresor de Bosques Aleatorios
reg = RandomForestRegressor(n_estimators=100, random_state=42)
reg.fit(X_train, y_train)

# Evaluar el rendimiento del regresor
y_pred = reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Error cuadrático medio del regresor de Bosques Aleatorios:", mse)
```