TIC

Lección 2

Modelos basados en árboles de decisión

TIC

Contenido de la lección 2

1. Introducción a los Árboles de Decisión
2. Bosques aleatorios
3. Evaluación de modelos
4. Aplicación de modelos basados en árboles de decisión en Sklearn

TIC

1. Introducción a los Árboles de Decisión

ID3 (Iterative Dichotomiser 3):

1 **Contexto:** 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.

2 **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.

3 **Ventajas:** ID3 es simple y fácil de entender. Además, puede manejar tanto datos categóricos como numéricos.

4 **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.

TIC

C4.5:

1. **Contexto:** es una extensión del algoritmo ID3 desarrollado también por Ross Quinlan. Se mejoró para abordar algunas de las limitaciones de ID3.

2. **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.

3. **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.

4. **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.

CART (Classification and Regression Trees):

1. **Contexto:** 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.

2 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.

3 **Ventajas:** CART es eficiente y puede manejar tanto datos categóricos como numéricos. Además, es robusto frente al ruido y valores atípicos.

4 **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.

TIC

Implementación de un árbol de decisión Stilizando 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()
```

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 árbentrenamiento: se entrena un árbol de decisión utilizando un subconjunto aleatorio de características en cada división del árbol.oles de decisión: Para cada conjunto de datos de

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:**

1.**Número de árboles (n_estimators):** Determina la cantidad de árboles en el bosque.número máximo de características a considerar en cada división de un árbol.

2.Número máximo de características (max_features): Especifica el máxima de cada árbol en el bosque.

3.Profundidad máxima del árbol (max_depth): Controla la profundidad Establece el número mínimo de muestras requeridas para dividir un nodo interno.

4.Número mínimo de muestras para dividir un nodo (min_samples_split):

5.Número mínimo de muestras en cada hoja (min_samples_leaf): Define el número mínimo de muestras requeridas en cada hoja del árbol.

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())
```

TIC

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.

TIC

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)
```

TIC

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 load_boston
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Cargar el conjunto de datos Boston House Prices
boston = load_boston()
X, y = boston.data, boston.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)
```

3. Evaluación de Modelos

La actividad Evaluación de Modelos ofrece una comprensión profunda de cómo evaluar y interpretar el rendimiento de los modelos de inteligencia artificial. Comienza explorando métodos de evaluación como la validación cruzada y las curvas de aprendizaje para garantizar la generalización del modelo. Luego, se sumerge en la interpretación de métricas clave de evaluación, como precisión, recall y F1-score, que permiten comprender mejor la capacidad predictiva y el comportamiento del modelo en diferentes escenarios. Finalmente, la unidad proporciona ejercicios prácticos para evaluar modelos basados en árboles de decisión y Bosques Aleatorios.

Métodos de evaluación de modelos

Los métodos de evaluación de modelos son técnicas utilizadas para medir el rendimiento y la generalización de los modelos de aprendizaje automático. Esto incluye la validación cruzada, que consiste en dividir el conjunto de datos en múltiples subconjuntos para entrenar y evaluar el modelo repetidamente, y las curvas de aprendizaje, que muestran cómo cambia el rendimiento del modelo con respecto al tamaño del conjunto de datos de entrenamiento.

TIC

Interpretación de métricas de evaluación

Las métricas de evaluación son medidas utilizadas para interpretar el rendimiento del modelo. Entre las más comunes se encuentran:

La precisión

Mide la proporción de predicciones correctas.

El recall

Mide la proporción de casos positivos que fueron correctamente identificados.

El F1-score

Es una combinación de precisión y recall

Estas métricas son fundamentales para entender la capacidad predictiva y el comportamiento del modelo en diferentes situaciones.

Son importantes los ejercicios prácticos de evaluación de modelos, ya que estos permiten aplicar los conceptos aprendidos en situaciones reales. Esto incluye la implementación de métricas de evaluación como precisión, recall y F1-score para evaluar modelos basados en árboles de decisión y Bosques Aleatorios. Además, se pueden realizar análisis comparativos entre diferentes modelos y ajustar los parámetros para optimizar el rendimiento del modelo.

1 Precisión

La precisión es una métrica que mide la proporción de predicciones positivas correctas entre todas las predicciones positivas realizadas por el modelo. Se calcula como el cociente entre los verdaderos positivos (TP) y la suma de los verdaderos positivos y los falsos positivos (FP). La fórmula para calcular la precisión es:

Precisión: TP/(FP+TP)

2 Recall

El recall, también conocido como sensibilidad o tasa de verdaderos positivos (TPR), mide la proporción de casos positivos que fueron correctamente identificados por el modelo. Se calcula como el cociente entre los verdaderos positivos (TP) y la suma de los verdaderos positivos y los falsos negativos (FN). La fórmula para calcular el recall es:

Recall=TP/(TP+FN)

3 F1-score

El F1-score es una medida que combina la precisión y el recall en un solo valor. Se calcula como la media armónica de la precisión y el recall, lo que proporciona una métrica equilibrada entre ambas. La fórmula para calcular el F1-score es:

F1-score=2*Precisión*Recall/(Presición + Recall)

TIC

Implementación de Validación Cruzada

```
from sklearn.model_selection import cross_val_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import make_classification


# Generación de datos ficticios

X, y = make_classification(n_samples=1000, n_features=20,
random_state=42)


# Inicialización del modelo

clf = DecisionTreeClassifier()


# Validación cruzada con 5 folds

scores = cross_val_score(clf, X, y, cv=5)

print("Accuracy:", scores.mean())
```

Curvas de Aprendizaje

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import learning_curve

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import make_classification

# Generación de datos ficticios

X, y = make_classification(n_samples=1000, n_features=20,
random_state=42)

# Inicialización del modelo

clf = DecisionTreeClassifier()

# Cálculo de las curvas de aprendizaje

train_sizes, train_scores, test_scores = learning_curve(

      clf, X, y, train_sizes=np.linspace(0.1, 1.0, 10), cv=5)


# Visualización de las curvas de aprendizaje

plt.figure()

plt.title("Curvas de Aprendizaje")

plt.xlabel("Tamaño del Conjunto de Entrenamiento")

plt.ylabel("Puntuación")

plt.grid()
```

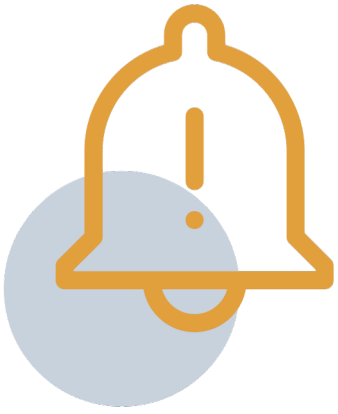
```
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)


plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1,
                 color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1,
color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
      label="Puntuación de Entrenamiento")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
      label="Puntuación de Validación")


plt.legend(loc="best")
plt.show()
```

TIC

Interpretación de Métricas de Evaluación

```
from sklearn.metrics import precision_score, recall_score, f1_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split



# Generación de datos ficticios

X, y = make_classification(n_samples=1000, n_features=20,

random_state=42)



# División de 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)



# Inicialización del modelo

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)
```

```
# Predicciones sobre el conjunto de prueba
y_pred = clf.predict(X_test)


# Cálculo de métricas de evaluación
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)


print("Precisión:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```



NOTA

Los ejercicios pueden ser similares a otros, usando árboles de decisión y Bosques Aleatorios, respectivamente. Se pueden usar los mismos códigos cambiando **DecisionTreeClassifier** por **RandomForestClassifier**.

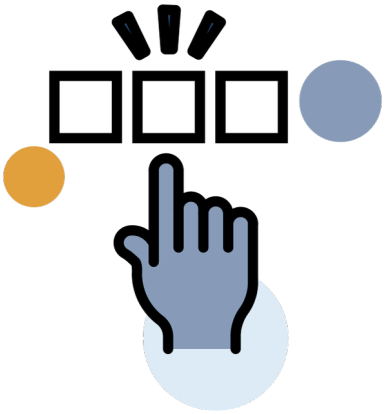
4. Aplicación de modelos basados en árboles de decisión en sklearn

Esta es una guía paso a paso para aplicar modelos basados en árboles de decisión en Scikit-learn a una base de datos tipo CSV descargada de plataformas como Kaggle o UCI Machine Learning Repository:

Descargar y explorar los datos:

Descarga el archivo CSV de la plataforma deseada y explora su contenido para comprender la estructura de los datos, las características disponibles y la variable objetivo que se desea predecir.

Preprocesamiento de datos:



- Limpia los datos eliminando filas con valores faltantes o duplicados.
- Realiza la ingeniería de características si es necesario, creando nuevas características o
- transformando las existentes. Divide los datos en características (X) y la variable objetivo (y).

División de datos:

Utiliza la función train_test_split de Scikit-learn para dividir los datos en conjuntos de entrenamiento y prueba. Esto permitirá evaluar el rendimiento del modelo en datos no vistos.

TIC

Selección de modelo:

- Elige un modelo basado en árboles de decisión adecuado para tu problema. Algunas opciones comunes incluyen DecisionTreeRegressor para problemas de regresión y DecisionTreeClassifier para
- problemas de clasificación. Importa el modelo correspondiente de Scikit-learn.

Entrenamiento del modelo:

- Ajusta el modelo a los datos de entrenamiento utilizando el método **fit**.
- Puedes ajustar diferentes hiperparámetros del modelo, como la profundidad máxima del árbol, el criterio de división y el número mínimo de muestras requeridas en cada hoja, utilizando la validación cruzada u otras técnicas de optimización de hiperparámetros.

Evaluación del modelo:

- Utiliza métricas de evaluación adecuadas para medir el rendimiento del modelo en los datos de prueba. Para problemas de clasificación, puedes utilizar métricas como precisión, recall y F1-score. Para problemas de regresión, puedes utilizar el error cuadrático medio (MSE) o el coeficiente de determinación (R^2).

- Visualiza los resultados del modelo utilizando gráficos como la matriz de confusión (para clasificación) o diagramas de dispersión de las predicciones vs. los valores reales (para regresión).

Interpretación del modelo:

- Analiza el árbol de decisión entrenado para comprender cómo se toman las decisiones en el modelo.
- Puedes visualizar el árbol utilizando herramientas como **plot_tree** de Scikit-learn. Identifica las características más importantes para las decisiones del modelo utilizando el atributo **feature_importances_** del modelo.

- Siguiendo estos pasos, un estudiante puede aplicar con éxito modelos basados en árboles de decisión en Scikit-learn a una base de datos tipo CSV descargada de plataformas de aprendizaje automático como Kaggle o UCI Machine Learning Repository.