Implementación de Árboles de Decisión

Sesión 10 - Curso de Inteligencia Artificial

Objetivos:

- Aplicar árboles de decisión en diferentes datasets
- Visualizar y analizar la estructura de los árboles
- Optimizar hiperparámetros
- Entender limitaciones y extensiones

```
# %% [code]
# Importación de bibliotecas
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.datasets import load_iris, load_wine
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
import graphviz
from sklearn.tree import export_graphviz
```

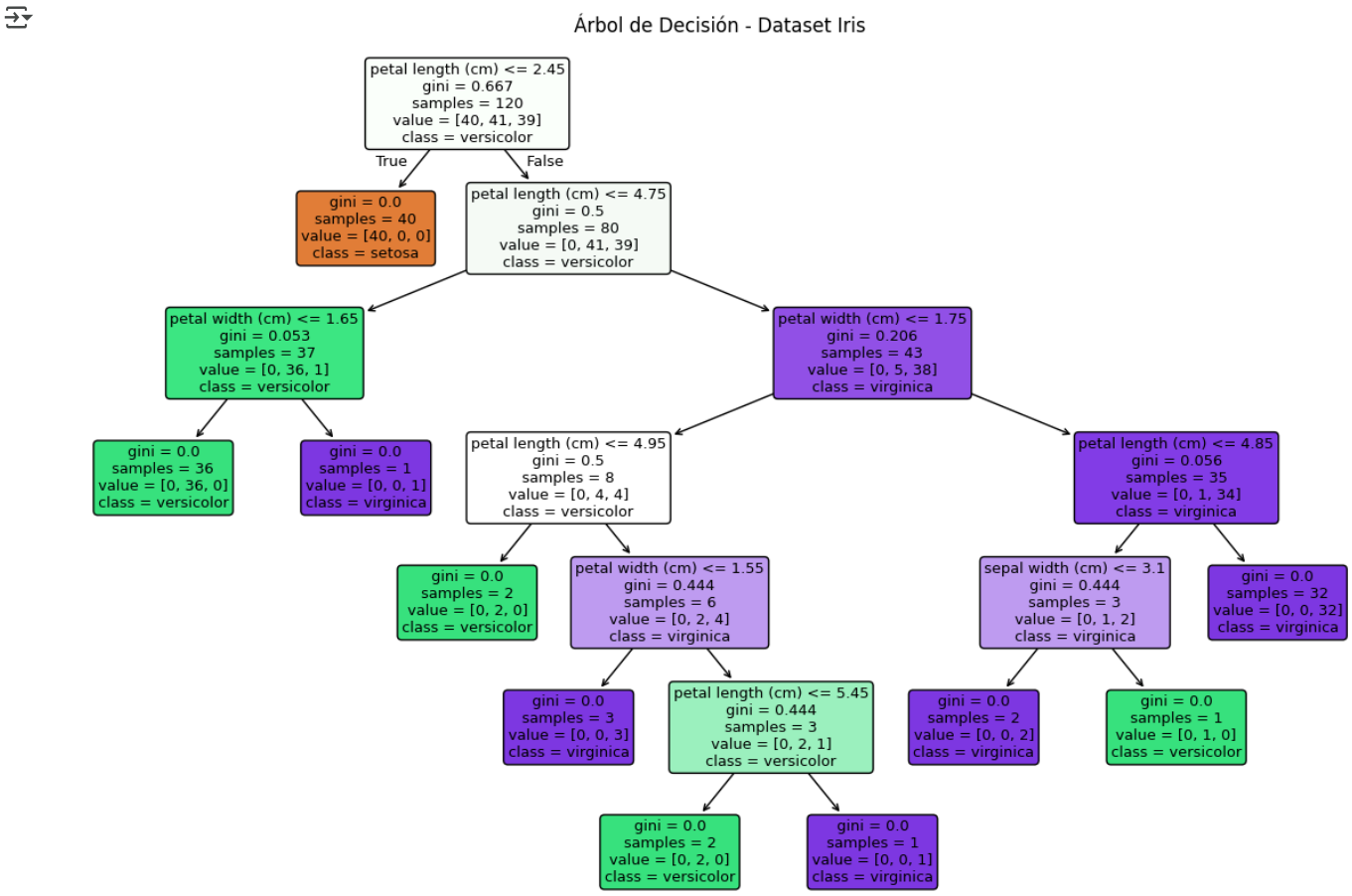
1. Dataset Iris (Clasificación Multiclase)

```
# %% [code]
# Cargar dataset
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
class_names = iris.target_names

# Dividir datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

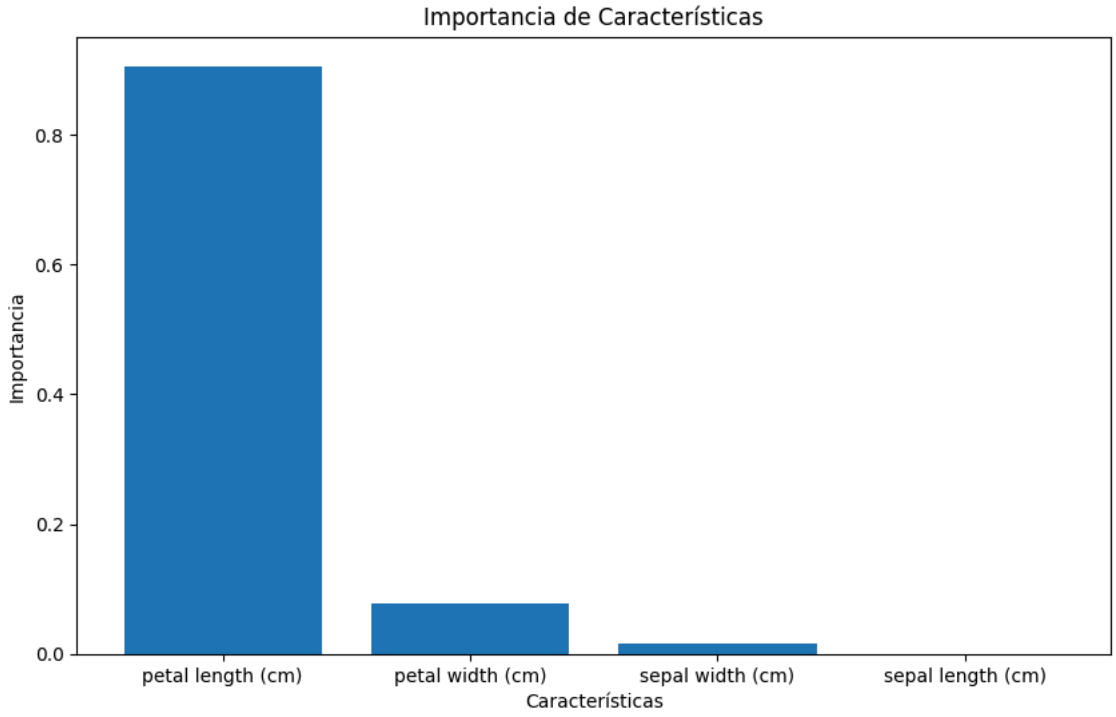
```
# %% [code]
# Modelo básico
modelo_basico = DecisionTreeClassifier(random_state=42)
modelo_basico.fit(X_train, y_train)

# Evaluación
y_pred = modelo_basico.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión modelo básico: {accuracy:.2f}")
```

```
Precisión modelo básico: 1.00
```

```
# %% [code]
# Visualización del árbol
plt.figure(figsize=(15,10))
plot_tree(modelo_basico,
          feature_names=feature_names,
          class_names=class_names,
          filled=True,
          rounded=True)
plt.title("Árbol de Decisión - Dataset Iris")
plt.show()
```




```
# %% [code]
# Análisis de importancia de características
importancias = modelo_basico.feature_importances_
indices = np.argsort(importancias)[::-1]

plt.figure(figsize=(10,6))
plt.title("Importancia de Características")
plt.bar(range(X.shape[1]), importancias[indices], align='center')
plt.xticks(range(X.shape[1]), [feature_names[i] for i in indices])
plt.xlabel("Características")
plt.ylabel("Importancia")
plt.show()
```



2. Dataset Titanic (Clasificación Binaria)

```
# %% [code]
# Cargar y preparar datos
titanic = pd.read_csv('https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv')
titanic = titanic[['Survived', 'Pclass', 'Sex', 'Age', 'Siblings/Spouses Aboard', 'Parents/Children Aboard', 'Fare']]
titanic['Age'].fillna(titanic['Age'].median(), inplace=True)

# Codificar variables categóricas
le = LabelEncoder()
titanic['Sex'] = le.fit_transform(titanic['Sex'])

# Dividir datos
X = titanic.drop('Survived', axis=1)
y = titanic['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

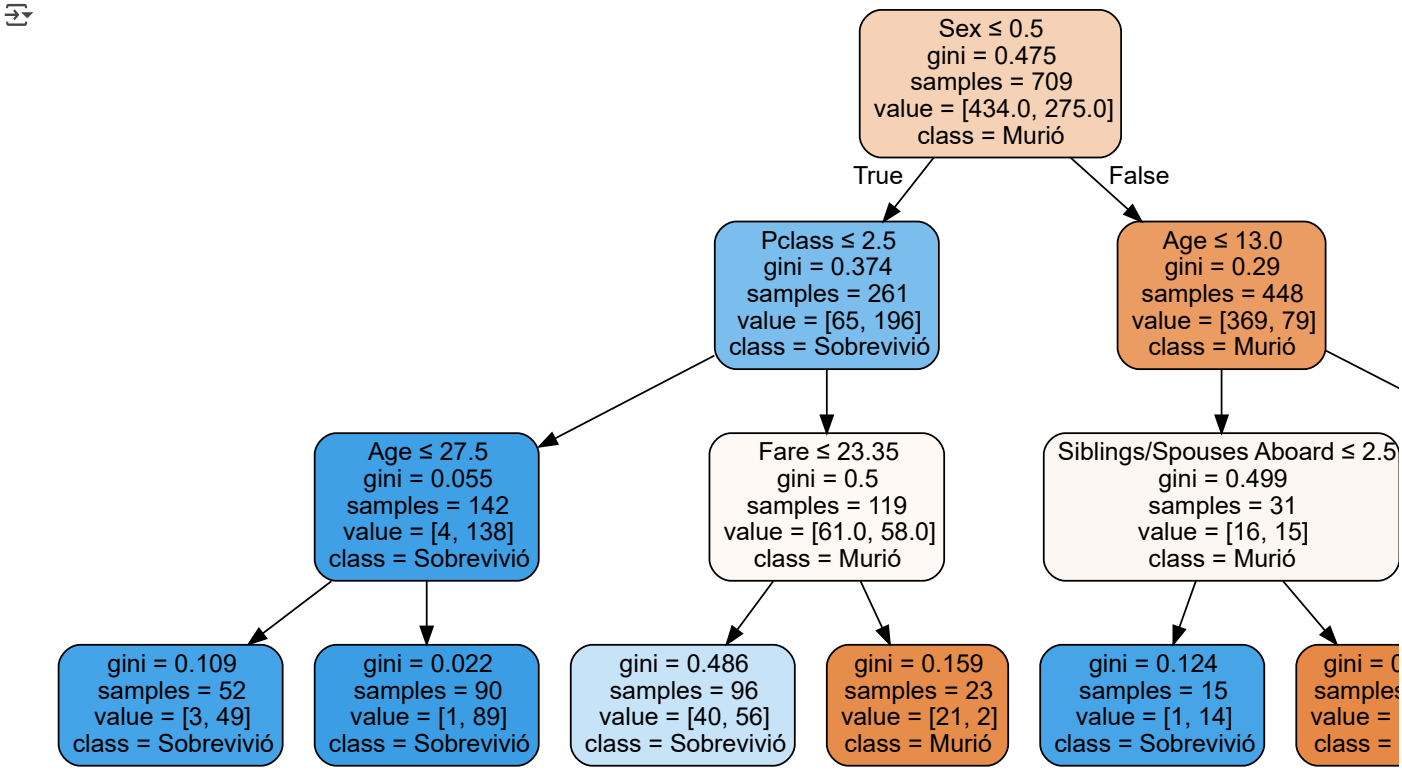
```
/tmp/ipython-input-3577344042.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col

  titanic['Age'].fillna(titanic['Age'].median(), inplace=True)
```

```
# %% [code]
# Modelo con hiperparámetros optimizados
modelo_titanic = DecisionTreeClassifier(
    max_depth=3,
    min_samples_split=20,
    min_samples_leaf=10,
    random_state=42
)
modelo_titanic.fit(X_train, y_train)

# Evaluación
y_pred = modelo_titanic.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión modelo Titanic: {accuracy:.2f}")
```

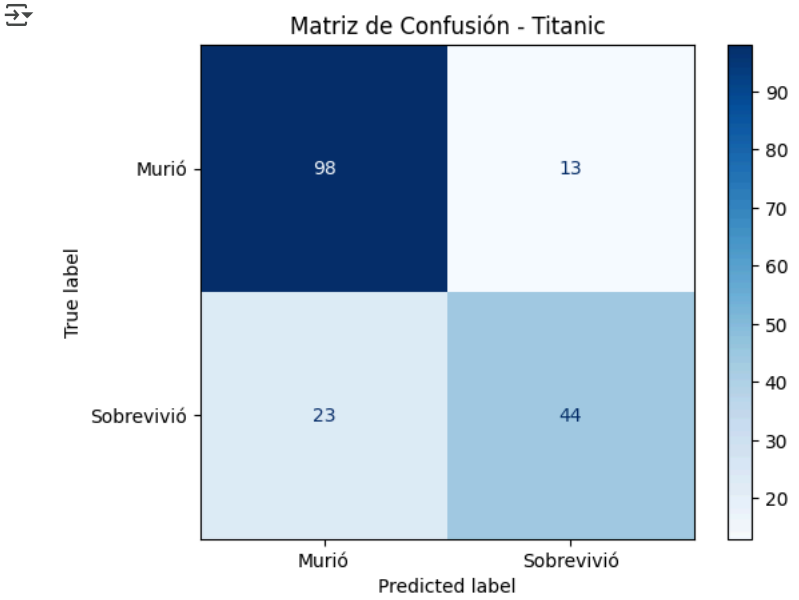
```
Precisión modelo Titanic: 0.80
```

```
# Visualización gráfica avanzada
dot_data = export_graphviz(
    modelo_titanic,
    out_file=None,
    feature_names=X.columns,
    class_names=['Murió', 'Sobrevivió'],
    filled=True,
```

```
    rounded=True,
    special_characters=True
)

graph = graphviz.Source(dot_data)
graph.render("titanic_decision_tree", format='png', cleanup=True)
graph
```




```
# %% [code]
# Matriz de confusión
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Murió', 'Sobrevivió'])
disp.plot(cmap='Blues')
plt.title('Matriz de Confusión - Titanic')
plt.show()
```



3. Dataset Wine (Clasificación Multiclase Compleja)

```
# %% [code]
# Cargar dataset
wine = load_wine()
```

```
X = wine.data
y = wine.target
feature_names = wine.feature_names
class_names = wine.target_names

# Dividir datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# %% [code]
# Optimización de hiperparámetros con GridSearch
param_grid = {
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

grid_search = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid,
    cv=5,
    scoring='accuracy'
)
grid_search.fit(X_train, y_train)

# Mejor modelo
best_tree = grid_search.best_estimator_
print(f"Mejores parámetros: {grid_search.best_params_}")
print(f"Precisión mejor modelo: {grid_search.best_score_:.2f}")
```

```
Mejores parámetros: {'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
Precisión mejor modelo: 0.94
```

```
# %% [code]
# Comparación con Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_accuracy = rf_model.score(X_test, y_test)
tree_accuracy = best_tree.score(X_test, y_test)

print(f"Precisión Árbol de Decisión: {tree_accuracy:.2f}")
print(f"Precisión Random Forest: {rf_accuracy:.2f}")
```

```
Precisión Árbol de Decisión: 0.96
Precisión Random Forest: 1.00
```

4. Caso de Estudio: Diagnóstico Médico

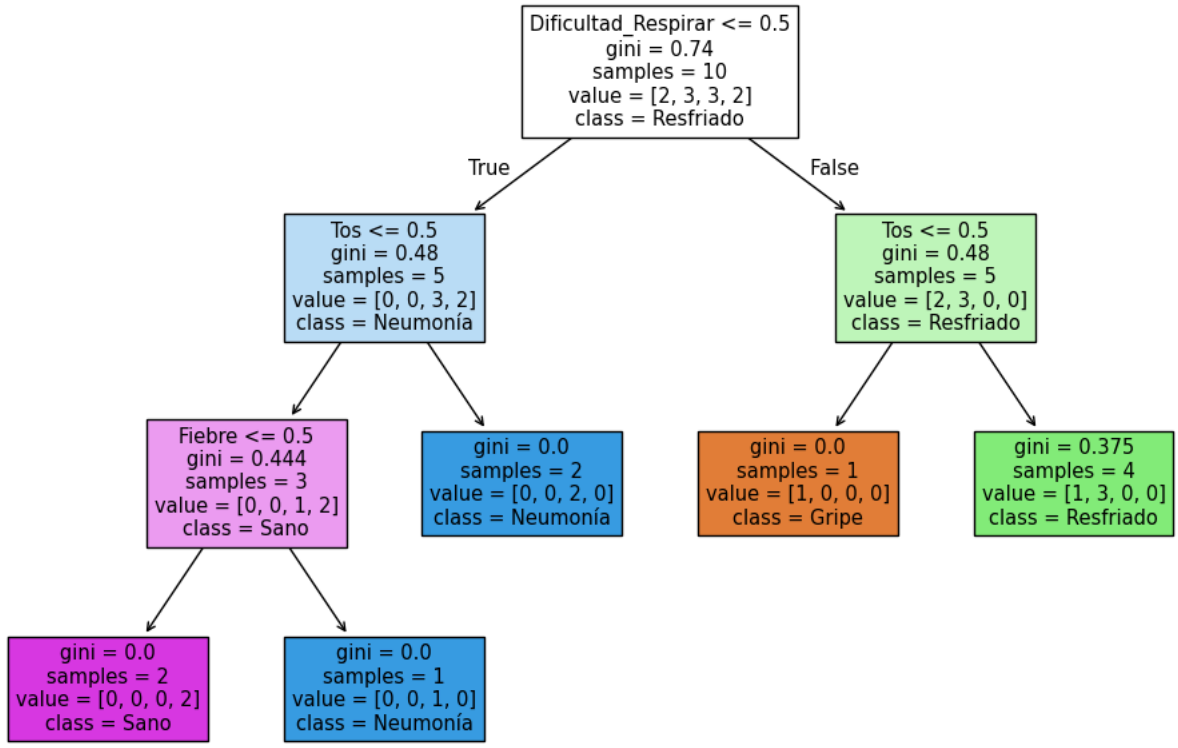
```
# Crear dataset sintético
data = {
    'Fiebre': [1, 1, 1, 0, 0, 1, 0, 1, 0, 1],
    'Tos': [1, 0, 1, 1, 0, 1, 0, 0, 1, 1],
    'Dificultad_Respirar': [1, 0, 1, 0, 0, 1, 0, 1, 0, 1],
    'Dolor_Cabeza': [0, 1, 0, 1, 1, 0, 1, 0, 1, 0],
    'Diagnóstico': ['Gripe', 'Resfriado', 'Neumonía', 'Resfriado', 'Sano', 'Neumonía', 'Sano', 'Gripe', 'Resfriado', 'Neumonía']
}

df_medical = pd.DataFrame(data)

# Preparar datos
X = df_medical.drop('Diagnóstico', axis=1)
y = df_medical['Diagnóstico']


# Entrenar modelo médico
medical_tree = DecisionTreeClassifier(max_depth=3, random_state=42)
medical_tree.fit(X, y)

# Reglas de decisión
tree_rules = export_text(medical_tree, feature_names=list(X.columns))
print("Reglas de diagnóstico médico:\n")
print(tree_rules)
```

```
Reglas de diagnóstico médico:

|--- Dificultad_Respirar <= 0.50
|   |--- Tos <= 0.50
|   |   |--- Fiebre <= 0.50
|   |   |   |--- class: Sano
|   |   |--- Fiebre >  0.50
```

```
|   |   |   |--- class: Resfriado
|   |--- Tos >  0.50
|   |   |--- class: Resfriado
|--- Dificultad_Respirar >  0.50
|   |--- Tos <= 0.50
|   |   |--- class: Gripe
|   |--- Tos >  0.50
|   |   |--- class: Neumonía
```

```
# Visualización interactiva (requiere ipywidgets)
from ipywidgets import interact

def plot_medical_tree(depth):
    model = DecisionTreeClassifier(max_depth=depth, random_state=42)
    model.fit(X, y)
    plt.figure(figsize=(12,8))
    plot_tree(model, feature_names=X.columns, class_names=y.unique(), filled=True)
    plt.show()

interact(plot_medical_tree, depth=(1, 5))
```

depth 3



plot_medical_tree
def plot_medical_tree(depth)

<no docstring>

5. Limitaciones y Soluciones

```
# Sobreajuste en árboles profundos
plt.figure(figsize=(10,6))

# Profundidad vs Precisión
depths = range(1, 15)
train_acc = []
test_acc = []

for depth in depths:
    tree = DecisionTreeClassifier(max_depth=depth, random_state=42)
    tree.fit(X_train, y_train)
    train_acc.append(tree.score(X_train, y_train))
    test_acc.append(tree.score(X_test, y_test))

plt.plot(depths, train_acc, 'bo-', label='Entrenamiento')
```

```
plt.plot(depths, test_acc, 'ro-', label='Prueba')
plt.xlabel('Profundidad del Árbol')
plt.ylabel('Precisión')
plt.title('Sobreajuste en Árboles de Decisión')
plt.legend()
plt.grid(True)
plt.show()
```