Guía de Laboratorio - Semana 5: Árboles de Decisión

Inteligencia Artificial (100000S14F)
Ingeniería de Software
Ciclo 2 Agosto 2025

13 de septiembre de 2025

1 Objetivos de la Sesión

Al finalizar esta sesión de laboratorio, el estudiante será capaz de:

- Comprender los fundamentos teóricos y matemáticos de los árboles de decisión
- Implementar un árbol de decisión utilizando scikit-learn con diversas configuraciones
- Realizar preprocesamiento de datos adecuado para modelos de árboles de decisión
- Evaluar el rendimiento del modelo utilizando múltiples métricas
- Visualizar e interpretar la estructura del árbol de decisión
- Optimizar hiperparámetros mediante validación cruzada
- Aplicar técnicas para evitar sobreajuste (overfitting)

2 Marco Teórico

Los árboles de decisión son estructuras jerárquicas utilizadas para clasificación y regresión. Constan de:

- **Nodos raíz e internos**: Representan pruebas sobre atributos
- **Ramas**: Resultados posibles de las pruebas
- **Nodos hoja**: Decisiones finales (clases o valores)

2.1 Algoritmos de Construcción

El algoritmo ID3 (Iterative Dichotomiser 3) utiliza la **ganancia de información** basada en la **entropía**:

$$\text{Entropía: } H(S) = -\sum_{i=1}^{c} p_i \log_2 p_i \tag{1}$$

$$\text{Ganancia de información: } IG(S, A) = H(S) - \sum_{t \in T} p(t)H(t) \tag{2}$$

El algoritmo C4.5 (extensión de ID3) utiliza la **ganancia de razón**:

$$\text{Ganancia de razón: } GR(S, A) = \frac{IG(S, A)}{H(A)} \tag{3}$$

El algoritmo CART utiliza el **índice Gini**:

$$\text{Índice Gini: } Gini(S) = 1 - \sum_{i=1}^{c} p_i^2 \tag{4}$$

3 Configuración del Entorno

3.1 Requisitos Previos

```
# Instalar las bibliotecas necesarias
pip install scikit-learn==1.2.2 pandas==2.0.1 matplotlib==3.7.1 numpy
    ==1.24.3 graphviz==0.20.1
```

3.2 Configuración de Entorno de Desarrollo

Recomendamos utilizar Jupyter Notebook o Google Colab para este laboratorio. Para configurar el entorno:

```
# Crear un entorno virtual (opcional pero recomendado)
python -m venv arboles_decision
source arboles_decision/bin/activate  # Linux/Mac
# o
.\arboles_decision\Scripts\activate  # Windows

# Instalar paquetes
pip install -r requirements.txt
```

4 Implementación Paso a Paso

4.1 Paso 1: Importar Bibliotecas

```
# Bibliotecas fundamentales
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image, display

# Scikit-learn para  rboles  de decisi n
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, export_graphviz,
    plot_tree
```

```
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import (accuracy_score, confusion_matrix, classification_report,
                             precision_score, recall_score, f1_score, roc_curve, auc)
from sklearn.preprocessing import label_binarize
from sklearn.utils import resample

# Graphviz para visualizaci n
import graphviz

# Configuraci n de estilo
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("husl")
%matplotlib inline
```

4.2 Paso 2: Cargar y Explorar Datos

```
# Cargar dataset
iris = load_iris()
X = iris.data
y = iris.target

# Crear DataFrame para mejor visualizaci n
df = pd.DataFrame(X, columns=iris.feature_names)
df['target'] = y
df['species'] = df['target'].apply(lambda x: iris.target_names[x])

print("=== INFORMACI N DEL DATASET IRIS ===")
print(f"Dimensiones: {X.shape}")
print(f"Caracter sticas: {iris.feature_names}")
print(f"Clases: {iris.target_names}")

print("\n=== PRIMERAS FILAS ===")
print(df.head())

print("\n=== ESTAD STICAS DESCRIPTIVAS ===")
print(df.describe())

print("\n=== DISTRIBUCI N DE CLASES ===")
print(df['species'].value_counts())

# Visualizaci n de distribuciones
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
for i, feature in enumerate(iris.feature_names):
    row, col = i // 2, i % 2
    for species in iris.target_names:
        species_data = df[df['species'] == species][feature]
        axes[row, col].hist(species_data, alpha=0.7, label=species)
    axes[row, col].set_title(f'Distribuci n de {feature}')
    axes[row, col].set_xlabel(feature)
    axes[row, col].set_ylabel('Frecuencia')
    axes[row, col].legend()

plt.tight_layout()
```

```
plt.savefig('distribuciones_caracteristicas.png', dpi=300, bbox_inches='tight')
plt.show()

# Matriz de correlaci n
plt.figure(figsize=(10, 8))
correlation_matrix = df[iris.feature_names].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Matriz de Correlaci n de Caracter sticas')
plt.savefig('matriz_correlacion.png', dpi=300, bbox_inches='tight')
plt.show()
```

4.3 Paso 3: Preprocesamiento de Datos

```
# Dividir en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y,  # Mantener proporci n de clases
    shuffle=True
)

print(f"Tama o conjunto de entrenamiento: {X_train.shape}")
print(f"Tama o conjunto de prueba: {X_test.shape}")

# Verificar distribuci n de clases en ambos conjuntos
print("\nDistribuci n en entrenamiento:")
unique_train, counts_train = np.unique(y_train, return_counts=True)
for cls, count in zip(unique_train, counts_train):
    print(f"Clase {iris.target_names[cls]}: {count} muestras ({count/len(y_train)*100:.2f}%)")

print("\nDistribuci n en prueba:")
unique_test, counts_test = np.unique(y_test, return_counts=True)
for cls, count in zip(unique_test, counts_test):
    print(f"Clase {iris.target_names[cls]}: {count} muestras ({count/len(y_test)*100:.2f}%)")
```

4.4 Paso 4: Crear y Entrenar el Modelo

```
# Configuraci n de hiperpar metros
parametros_arbol = {
    'criterion': 'entropy',        # Criterio de divisi n: 'gini' o 'entropy'
    'max_depth': 3,                # Profundidad m xima del  rbol
    'min_samples_split': 2,        # M nimo de muestras para dividir un nodo
    'min_samples_leaf': 1,         # M nimo de muestras en un nodo hoja
    'max_features': None,          # N mero de caracter sticas a considerar
    'random_state': 42,            # Semilla para reproducibilidad
    'ccp_alpha': 0.0,              # Par metro de podado cost-complexity
```

```
    'class_weight': None           # Pesos de clases ( til  para
        datasets desbalanceados)
}

# Crear el clasificador de  rbol  de decisi n
clf = DecisionTreeClassifier(**parametros_arbol)

# Entrenar el modelo
clf.fit(X_train, y_train)

# Hacer predicciones
y_pred = clf.predict(X_test)
y_pred_proba = clf.predict_proba(X_test)  # Probabilidades para cada
    clase

# Mostrar informaci n del  rbol  resultante
print("=== INFORMACI N DEL  RBOL  ===")
print(f"N mero de nodos: {clf.tree_.node_count}")
print(f"Profundidad del  rbol : {clf.get_depth()}")
print(f"N mero de hojas: {clf.get_n_leaves()}")
```

4.5 Paso 5: Evaluar el Modelo

```
# Calcular m tricas de evaluaci n
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("=== M TRICAS DE EVALUACI N ===")
print(f"Precisi n (Accuracy): {accuracy:.4f}")
print(f"Precisi n (Precision): {precision:.4f}")
print(f"Sensibilidad (Recall): {recall:.4f}")
print(f"Puntuaci n F1: {f1:.4f}")

# Matriz de confusi n
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=iris.target_names,
            yticklabels=iris.target_names)
plt.title('Matriz de Confusi n')
plt.ylabel('Etiqueta Real')
plt.xlabel('Etiqueta Predicha')
plt.savefig('matriz_confusion.png', dpi=300, bbox_inches='tight')
plt.show()

# Reporte de clasificaci n detallado
print("\n=== REPORTE DE CLASIFICACI N ===")
print(classification_report(y_test, y_pred, target_names=iris.
    target_names))

# Curvas ROC (para clasificaci n multiclase)
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]

```

```
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_proba[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(10, 8))
colors = ['aqua', 'darkorange', 'cornflowerblue']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(iris.target_names[i], roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Curvas ROC para Clasificaci n Multiclase')
plt.legend(loc="lower right")
plt.savefig('curvas_roc.png', dpi=300, bbox_inches='tight')
plt.show()
```

4.6 Paso 6: Visualizar el Árbol de Decisión

```
# Visualizaci n con Graphviz (alta calidad)
dot_data = export_graphviz(
    clf,
    out_file=None,
    feature_names=iris.feature_names,
    class_names=iris.target_names,
    filled=True,           # Nodos coloreados
    rounded=True,          # Bordes redondeados
    special_characters=True,
    proportion=True,       # Mostrar proporciones en lugar de counts
    impurity=True,         # Mostrar impureza
    node_ids=True          # Mostrar IDs de nodos
)

graph = graphviz.Source(dot_data)
graph.format = 'png'
graph.render('arbol_decision_iris', view=False, cleanup=True)

# Visualizaci n con matplotlib (m s simple pero interactiva)
plt.figure(figsize=(20, 12))
plot_tree(clf,
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          filled=True,
          rounded=True,
          fontsize=10)
plt.title(" rbol  de Decisi n - Dataset Iris")
```

```
plt.savefig('arbol_decision_matplotlib.png', dpi=300, bbox_inches='tight')
plt.show()

# Mostrar importancia de caracter sticas
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Importancia de las Caracter sticas")
plt.bar(range(X.shape[1]), importances[indices], align="center")
plt.xticks(range(X.shape[1]), [iris.feature_names[i] for i in indices])
plt.xlim([-1, X.shape[1]])
plt.tight_layout()
plt.savefig('importancia_caracteristicas.png', dpi=300, bbox_inches='tight')
plt.show()

print("Importancia de caracter sticas:")
for i, idx in enumerate(indices):
    print(f"{i+1}. {iris.feature_names[idx]}: {importances[idx]:.4f}")
```

5 Experimentación con Hiperparámetros

5.1 Análisis de Profundidad del Árbol

```
# Probar diferentes profundidades m ximas
profundidades = range(1, 21)
train_scores = []
test_scores = []
cv_scores = []

for profundidad in profundidades:
    # Crear y entrenar modelo
    clf_temp = DecisionTreeClassifier(max_depth=profundidad, random_state=42)
    clf_temp.fit(X_train, y_train)

    # Calcular precisi n en entrenamiento y prueba
    train_score = clf_temp.score(X_train, y_train)
    test_score = clf_temp.score(X_test, y_test)

    # Calcular validaci n cruzada
    cv_score = cross_val_score(clf_temp, X, y, cv=5).mean()

    train_scores.append(train_score)
    test_scores.append(test_score)
    cv_scores.append(cv_score)

    print(f"Profundidad: {profundidad:2d} | Train: {train_score:.4f} | Test: {test_score:.4f} | CV: {cv_score:.4f}")

# Graficar resultados
plt.figure(figsize=(12, 6))
```

```
plt.plot(profundidades, train_scores, 'o-', label='Precisi n en Entrenamiento')
plt.plot(profundidades, test_scores, 'o-', label='Precisi n en Prueba')
plt.plot(profundidades, cv_scores, 'o-', label='Validaci n Cruzada (5-fold)')
plt.xlabel('Profundidad M xima del  rbol ')
plt.ylabel('Precisi n')
plt.title('Efecto de la Profundidad del  rbol  en el Rendimiento')
plt.legend()
plt.grid(True)
plt.savefig('profundidad_vs_rendimiento.png', dpi=300, bbox_inches='tight')
plt.show()
```

5.2 Optimización de Hiperparámetros con Grid Search

```
# Definir espacio de b squeda de hiperpar metros
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': [None, 'sqrt', 'log2'],
    'ccp_alpha': [0.0, 0.01, 0.1]  # Par metro de podado
}

# Crear el modelo base
dt = DecisionTreeClassifier(random_state=42)

# Configurar la b squeda en grid
grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,            # Validaci n cruzada de 5 folds
    n_jobs=-1,       # Usar todos los cores disponibles
    verbose=1        # Mostrar progreso
)

# Ejecutar la b squeda en grid
grid_search.fit(X_train, y_train)

# Mejores par metros y resultados
print("Mejores par metros encontrados:")
print(grid_search.best_params_)
print(f"\nMejor precisi n en validaci n cruzada: {grid_search.best_score_:.4f}")

# Evaluar el mejor modelo en el conjunto de prueba
best_clf = grid_search.best_estimator_
y_pred_best = best_clf.predict(X_test)
best_accuracy = accuracy_score(y_test, y_pred_best)
print(f"Precisi n del mejor modelo en prueba: {best_accuracy:.4f}")

# Comparar con el modelo inicial
```

```
print(f"Mejora respecto al modelo inicial: {(best_accuracy - accuracy)
    :.4f}")

# Visualizar el mejor  rbol
plt.figure(figsize=(20, 12))
plot_tree(best_clf,
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          filled=True,
          rounded=True,
          fontsize=10)
plt.title("Mejor  rbol  de Decisi n despu s de la Optimizaci n")
plt.savefig('mejor_arbol_decision.png', dpi=300, bbox_inches='tight')
plt.show()
```

6 Ejercicios Prácticos

6.1 Ejercicio 1: Análisis de Importancia de Características

1. Extrae la importancia de cada característica del modelo entrenado
2. Visualiza la importancia de características en un gráfico de barras horizontal
3. Interpreta qué características son más relevantes para la clasificación
4. Compara los resultados con la matriz de correlación generada anteriormente

Solución:

```
# Obtener importancia de caracter sticas
importances = best_clf.feature_importances_
feature_names = iris.feature_names

# Crear DataFrame para mejor visualizaci n
importance_df = pd.DataFrame({
    'Caracter stica': feature_names,
    'Importancia': importances
}).sort_values('Importancia', ascending=True)

# Gr fico de barras horizontal
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Caracter stica'], importance_df['Importancia'
    ])
plt.xlabel('Importancia')
plt.title('Importancia de Caracter sticas en el Mejor Modelo')
for i, v in enumerate(importance_df['Importancia']):
    plt.text(v + 0.001, i, f'{v:.4f}', va='center')
plt.tight_layout()
plt.savefig('importancia_caracteristicas_mejor_modelo.png', dpi=300,
    bbox_inches='tight')
plt.show()

# An lisis
print("An lisis de importancia de caracter sticas:")
for i, row in importance_df.iterrows():
    print(f"- {row['Caracter stica']}: {row['Importancia']:.4f}")
```

6.2 Ejercicio 2: Validación Cruzada y Estabilidad del Modelo

1. Realiza validación cruzada con diferentes semillas aleatorias
2. Evalúa la estabilidad del modelo con diferentes divisiones de datos
3. Calcula intervalos de confianza para la precisión del modelo

Solución:

```
# Validaci n cruzada con diferentes semillas
n_iterations = 10
cv_scores = []

for i in range(n_iterations):
    # Crear modelo con diferentes semillas
    clf_cv = DecisionTreeClassifier(**grid_search.best_params_)
    clf_cv.random_state = i  # Cambiar semilla

    # Realizar validaci n cruzada
    scores = cross_val_score(clf_cv, X, y, cv=5, scoring='accuracy')
    cv_scores.append(scores.mean())
    print(f"Iteraci n {i+1}: Precisi n CV = {scores.mean():.4f}")

# Calcular estad sticas
mean_accuracy = np.mean(cv_scores)
std_accuracy = np.std(cv_scores)
confidence_interval = 1.96 * std_accuracy / np.sqrt(n_iterations)  #
    95% IC

print(f"\nPrecisi n media: {mean_accuracy:.4f}")
print(f"Desviaci n est ndar: {std_accuracy:.4f}")
print(f"Intervalo de confianza (95%): ({mean_accuracy -
    confidence_interval:.4f}, {mean_accuracy + confidence_interval:.4f})
    ")

# Visualizar distribuci n de precisiones
plt.figure(figsize=(10, 6))
plt.hist(cv_scores, bins=10, edgecolor='black', alpha=0.7)
plt.axvline(mean_accuracy, color='red', linestyle='--', label=f'Media:
    {mean_accuracy:.4f}')
plt.xlabel('Precisi n')
plt.ylabel('Frecuencia')
plt.title('Distribuci n de Precisiones en Validaci n Cruzada')
plt.legend()
plt.savefig('distribucion_precision_cv.png', dpi=300, bbox_inches='
    tight')
plt.show()
```

6.3 Ejercicio 3: Aplicación a un Dataset Diferente

1. Carga el dataset de cáncer de mama de scikit-learn
2. Aplica el mismo proceso de preprocesamiento y modelado
3. Compara los resultados con los obtenidos con el dataset Iris

Solución:

```python
from sklearn.datasets import load_breast_cancer

# Cargar dataset de c ncer de mama
cancer = load_breast_cancer()
X_cancer = cancer.data
y_cancer = cancer.target

print(f"Dataset de c ncer de mama: {X_cancer.shape}")
print(f"Caracter sticas: {cancer.feature_names}")
print(f"Clases: {cancer.target_names}")

# Preprocesamiento (escalado recomendado para este dataset)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_cancer_scaled = scaler.fit_transform(X_cancer)

# Divisi n en train/test
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(
    X_cancer_scaled, y_cancer, test_size=0.3, random_state=42, stratify
       =y_cancer
)

# Entrenar modelo con mejores par metros del ejercicio anterior
clf_cancer = DecisionTreeClassifier(**grid_search.best_params_)
clf_cancer.fit(X_train_c, y_train_c)

# Evaluar
y_pred_c = clf_cancer.predict(X_test_c)
accuracy_c = accuracy_score(y_test_c, y_pred_c)
print(f"Precisi n en dataset de c ncer de mama: {accuracy_c:.4f}")

# Comparar con modelo b sico
clf_basic = DecisionTreeClassifier(random_state=42)
clf_basic.fit(X_train_c, y_train_c)
y_pred_basic = clf_basic.predict(X_test_c)
accuracy_basic = accuracy_score(y_test_c, y_pred_basic)
print(f"Precisi n con modelo b sico: {accuracy_basic:.4f}")
print(f"Mejora con par metros optimizados: {accuracy_c -
   accuracy_basic:.4f}")
```

7 Recursos Adicionales

7.1 Bibliografía Recomendada

- Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. (Capítulo 3)
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer. (Capítulo 9)
- Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly. (Capítulo 6)

7.2 Enlaces Útiles

- Documentación de scikit-learn: `https://scikit-learn.org/stable/modules/tree.html`
- Tutorial interactivo de árboles de decisión: `https://www.datacamp.com/community/tutorials/decision-tree-classification-python`
- Visualización interactiva de árboles de decisión: `http://www.r2d3.us/visual-intro-to-machin`
- Repositorio de datasets para práctica: `https://archive.ics.uci.edu/ml/index.php`

7.3 Herramientas Alternativas

- WEKA: Herramienta gráfica para minería de datos
- RapidMiner: Plataforma de ciencia de datos con interfaz visual
- KNIME: Plataforma de analítica de datos de código abierto