

# Curso: Inteligencia Artificial

## Guía de Laboratorio: Aprendizaje de Reglas en IA

Docente: Carlos R. P. Tovar

### 1. Introducción

El aprendizaje de reglas es una técnica fundamental en inteligencia artificial que permite extraer conocimiento en forma de reglas "SI-ENTONCES.<sup>a</sup> a partir de datos. Estas reglas son interpretables y se asemejan al razonamiento humano, lo que las hace valiosas para sistemas expertos, diagnóstico médico y toma de decisiones. En esta guía integrada, exploraremos tres enfoques diferentes para el aprendizaje de reglas: algoritmos básicos (OneR), árboles de decisión y programación lógica con Prolog.

### 2. Objetivos del Laboratorio

Al completar este laboratorio, serás capaz de:

- Comprender los fundamentos del aprendizaje de reglas
- Implementar algoritmos básicos de aprendizaje de reglas (OneR)
- Utilizar árboles de decisión para generar conjuntos de reglas
- Representar y consultar reglas en un lenguaje declarativo (Prolog)
- Evaluar la calidad de conjuntos de reglas aprendidas
- Comparar enfoques simbólicos y estadísticos en el aprendizaje de reglas
- Aplicar estas técnicas en problemas de clasificación

### 3. Requisitos

- Python 3.6 o superior
- Bibliotecas Python: pandas, scikit-learn, matplotlib
- SWI-Prolog (para los ejercicios de programación lógica)
- Conocimientos básicos de pandas y scikit-learn
- Datasets: Titanic, Iris (disponibles en línea)

## 4. Parte 1: Algoritmos Básicos de Aprendizaje de Reglas

### 4.1. Preparación de Datos

#### 1. Importación de bibliotecas:

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4 from sklearn.tree import DecisionTreeClassifier
5 from sklearn.metrics import accuracy_score, confusion_matrix
6 import matplotlib.pyplot as plt
7
```

#### 2. Carga y preprocesamiento de datos:

```
1 # Cargar datos
2 df = pd.read_csv('titanic.csv')
3
4 # Preprocesamiento básico
5 df['Age'].fillna(df['Age'].mean(), inplace=True)
6 df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
7 df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)
8
9 # Seleccionar características relevantes
10 features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked_Q', 'Embarked_S']
11 X = df[features]
12 y = df['Survived']
13
```

### 4.2. Implementación de Algoritmo OneR

#### 1. Implementación de OneR:

```
1 def one_rule_algorithm(X, y, feature_name):
2     # Encontrar el mejor umbral para la característica
3     best_accuracy = 0
4     best_threshold = 0
5
6     for threshold in np.linspace(X[feature_name].min(), X[feature_name].max(), 100):
7         predictions = (X[feature_name] > threshold).astype(int)
8         accuracy = accuracy_score(y, predictions)
9
10        if accuracy > best_accuracy:
11            best_accuracy = accuracy
12            best_threshold = threshold
13
14        return best_threshold, best_accuracy
15
16 # Probar con diferentes características
17 for feature in ['Age', 'Fare', 'Pclass']:
18     threshold, accuracy = one_rule_algorithm(X, y, feature)
```

```

19     print(f"Característica: {feature}, Umbral: {threshold:.2f},
20           Precisión: {accuracy:.2f}")

```

## 5. Parte 2: Árboles de Decisión para Generación de Reglas

### 5.1. Árboles de Decisión con scikit-learn

#### 1. Entrenar árbol de decisión con profundidad limitada:

```

1 # Entrenar árbol de decisión con una sola regla
2 model = DecisionTreeClassifier(max_depth=1)
3 model.fit(X, y)
4
5 # Evaluar el modelo
6 y_pred = model.predict(X)
7 accuracy = accuracy_score(y, y_pred)
8 print(f"Precisión del árbol de una regla: {accuracy:.2f}")
9

```

#### 2. Visualizar la regla:

```

1 from sklearn.tree import plot_tree
2
3 plt.figure(figsize=(10, 8))
4 plot_tree(model, feature_names=features, class_names=['No
5             Sobrevivió', 'Sobrevivió'], filled=True)
6 plt.title("Árbol de Decisión de Una Regla")
7 plt.show()
8

```

### 5.2. Extracción de Reglas de Árboles de Decisión

#### 1. Uso del dataset Iris:

```

1 from sklearn.datasets import load_iris
2 from sklearn.tree import export_text
3
4 # Cargar dataset Iris
5 X_iris, y_iris = load_iris(return_X_y=True)
6 feature_names = load_iris().feature_names
7 target_names = load_iris().target_names
8
9 # Entrenar árbol con profundidad 3
10 clf = DecisionTreeClassifier(max_depth=3, random_state=42)
11 clf.fit(X_iris, y_iris)
12
13 # Extraer reglas
14 rules = export_text(clf, feature_names=feature_names)
15 print("Reglas del árbol de decisión:")
16 print(rules)
17

```

## 6. Parte 3: Implementación Manual de Sistemas de Reglas

### 6.1. Reglas Simples

#### 1. Clasificación por edades:

```
1 def clasificar_edad(edad):
2     if edad < 12:
3         return "Niño"
4     elif edad < 18:
5         return "Joven"
6     elif edad < 60:
7         return "Adulto"
8     else:
9         return "Anciano"
10
11 print(clasificar_edad(25)) # Adulto
12
```

### 6.2. Reglas con Múltiples Condiciones

#### 1. Clasificador de frutas:

```
1 def clasificar_fruta(color, tamaño, sabor):
2     if color == "rojo" and sabor == "dulce":
3         return "Manzana"
4     elif color == "naranja" and tamaño == "mediano":
5         return "Naranja"
6     elif color == "amarillo" and tamaño == "largo":
7         return "Plátano"
8     else:
9         return "Desconocido"
10
11 print(clasificar_fruta("rojo", "pequeño", "dulce"))
12
```

### 6.3. Sistema de Reglas con Prioridad

#### 1. Sistema de aprobación de créditos:

```
1 def evaluar_credito(ingreso, deuda, historial):
2     # Regla 1: Prioridad alta
3     if ingreso > 5000 and deuda < 2000 and historial == "bueno":
4         return "Aprobado"
5     # Regla 2: Prioridad media
6     elif ingreso > 3000 and deuda < 3000:
7         return "Revisión manual"
8     # Regla 3: Prioridad baja (default)
9     else:
10        return "Rechazado"
11
12 print(evaluar_credito(6000, 1500, "bueno"))
13
```

## 7. Parte 4: Programación Lógica con Prolog

### 7.1. Base de Conocimiento y Reglas en Prolog

#### 1. Sistema de diagnóstico médico:

```
1 % Base de hechos
2 fiebre(juan).
3 tos(juan).
4 dolor_cabeza(maria).
5 congestion(luis).
6
7 % Reglas
8 gripe(X) :- fiebre(X), tos(X).
9 resfriado(X) :- tos(X), congestion(X), \+ fiebre(X).
10 migrana(X) :- dolor_cabeza(X), \+ fiebre(X).
11
12 % Consultas de ejemplo:
13 % ?- gripe(juan).           % true
14 % ?- migrana(maria).       % true
15 % ?- resfriado(luis).      % true
16
```

## 8. Parte 5: Evaluación de Reglas

### 8.1. Cálculo de Métricas

#### 1. Evaluación de reglas:

```
1 def evaluate_rule(X, y, feature, threshold, operator='>'):
2     if operator == '>':
3         predictions = (X[feature] > threshold).astype(int)
4     else:
5         predictions = (X[feature] <= threshold).astype(int)
6
7     accuracy = accuracy_score(y, predictions)
8     coverage = len(predictions) / len(y)
9
10    # Calcular precisi n, recall y F1-score
11    tn, fp, fn, tp = confusion_matrix(y, predictions).ravel()
12    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
13    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
14    f1 = 2 * (precision * recall) / (precision + recall) if (
15    precision + recall) > 0 else 0
16
17    return accuracy, coverage, precision, recall, f1
18
19 # Evaluar regla para edad
20 accuracy, coverage, precision, recall, f1 = evaluate_rule(X, y, '
21 Age', 30, '>')
22 print(f"Precisi n: {accuracy:.2f}, Cobertura: {coverage:.2f}")
23 print(f"Precisi n: {precision:.2f}, Recall: {recall:.2f}, F1: {f1
24 :.2f}")
```

## 8.2. Comparación de Enfoques

### 1. Comparación entre árbol de decisión y reglas manuales:

```
1 # Crear datos de prueba
2 datos_edad = [(10, "Niño"), (15, "Joven"), (25, "Adulto"), (70, "
    Anciano")]
3
4 # Evaluar reglas manuales
5 correctos = 0
6 for edad, esperado in datos_edad:
7     if clasificar_edad(edad) == esperado:
8         correctos += 1
9
10 precision_manual = correctos / len(datos_edad)
11 print(f"Precisión del sistema manual: {precision_manual:.2f}")
12
13 # Comparar con árbol de decisión (usando Iris como ejemplo)
14 y_pred_iris = clf.predict(X_iris)
15 precision_arbol = accuracy_score(y_iris, y_pred_iris)
16 print(f"Precisión del árbol de decisión: {precision_arbol:.2f}")
17
```

## 9. Conclusión

En esta guía de laboratorio unificada, hemos explorado tres enfoques diferentes para el aprendizaje de reglas en inteligencia artificial. Hemos implementado el algoritmo OneR para reglas simples, utilizado árboles de decisión para generar conjuntos de reglas automáticamente, creado sistemas de reglas manualmente en Python, y explorado la programación lógica con Prolog para representar conocimiento mediante reglas.

Cada enfoque tiene sus ventajas: los algoritmos automáticos como OneR y los árboles de decisión son excelentes para extraer patrones de datos, mientras que los sistemas de reglas manuales y la programación lógica permiten incorporar conocimiento experto y son más interpretables.

El aprendizaje de reglas es una técnica poderosa para crear modelos interpretables que se asemejan al razonamiento humano, siendo valiosa en aplicaciones donde la transparencia del modelo es importante, como en sistemas expertos, diagnóstico médico y toma de decisiones.

## 10. Recursos Adicionales

- Documentación de scikit-learn: <https://scikit-learn.org/>
- SWI-Prolog: <https://www.swi-prolog.org/>
- Libro: "Machine Learning" by Tom Mitchell (Capítulo 3)
- Paper: "Fast Effective Rule Induction" (Algoritmo RIPPER)
- Libro: "Programming in Prolog" by Clocksin and Mellish