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.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**:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt

```

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

```
# Cargar datos
df = pd.read_csv('titanic.csv')

# Preprocesamiento b sico
df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)

# Seleccionar caracter sticas relevantes
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', '
    Embarked_Q', 'Embarked_S']
X = df[features]
y = df['Survived']

```

4.2. Implementación de Algoritmo OneR

1. **Implementación de OneR**:

```
def one_rule_algorithm(X, y, feature_name):
    # Encontrar el mejor umbral para la caracter stica
    best_accuracy = 0
    best_threshold = 0

    for threshold in np.linspace(X[feature_name].min(), X[
    feature_name].max(), 100):
        predictions = (X[feature_name] > threshold).astype(int)
        accuracy = accuracy_score(y, predictions)

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_threshold = threshold

    return best_threshold, best_accuracy

# Probar con diferentes caracter sticas
for feature in ['Age', 'Fare', 'Pclass']:
    threshold, accuracy = one_rule_algorithm(X, y, feature)
```

```
    print(f"Caracter stica: {feature}, Umbral: {threshold:.2f},
    Precisi n: {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**:

```
# Entrenar  rbol  de decisi n con una sola regla
model = DecisionTreeClassifier(max_depth=1)
model.fit(X, y)

# Evaluar el modelo
y_pred = model.predict(X)
accuracy = accuracy_score(y, y_pred)
print(f"Precisi n del  rbol  de una regla: {accuracy:.2f}")

```

2. **Visualizar la regla**:

```
from sklearn.tree import plot_tree

plt.figure(figsize=(10, 8))
plot_tree(model, feature_names=features, class_names=['No
    Sobrevivi ', 'Sobrevivi '], filled=True)
plt.title(" rbol  de Decisi n de Una Regla")
plt.show()

```

5.2. Extracción de Reglas de Árboles de Decisión

1. **Uso del dataset Iris**:

```
from sklearn.datasets import load_iris
from sklearn.tree import export_text

# Cargar dataset Iris
X_iris, y_iris = load_iris(return_X_y=True)
feature_names = load_iris().feature_names
target_names = load_iris().target_names

# Entrenar  rbol  con profundidad 3
clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_iris, y_iris)

# Extraer reglas
rules = export_text(clf, feature_names=feature_names)
print("Reglas del  rbol  de decisi n:")
print(rules)

```

6. Parte 3: Implementación Manual de Sistemas de Reglas

6.1. Reglas Simples

1. **Clasificación por edades**:

```python
def clasificar_edad(edad):
    if edad < 12:
        return "Ni o"
    elif edad < 18:
        return "Joven"
    elif edad < 60:
        return "Adulto"
    else:
        return "Anciano"

print(clasificar_edad(25))  # Adulto

```

6.2. Reglas con Múltiples Condiciones

1. **Clasificador de frutas**:

```python
def clasificar_fruta(color, tamano, sabor):
    if color == "rojo" and sabor == "dulce":
        return "Manzana"
    elif color == "naranja" and tamano == "mediano":
        return "Naranja"
    elif color == "amarillo" and tamano == "largo":
        return "Pl tano"
    else:
        return "Desconocido"

print(clasificar_fruta("rojo", "peque o", "dulce"))

```

6.3. Sistema de Reglas con Prioridad

1. **Sistema de aprobación de créditos**:

```python
def evaluar_credito(ingreso, deuda, historial):
    # Regla 1: Prioridad alta
    if ingreso > 5000 and deuda < 2000 and historial == "bueno":
        return "Aprobado"
    # Regla 2: Prioridad media
    elif ingreso > 3000 and deuda < 3000:
        return "Revisi n manual"
    # Regla 3: Prioridad baja (default)
    else:
        return "Rechazado"

print(evaluar_credito(6000, 1500, "bueno"))

```

7. Parte 4: Programación Lógica con Prolog

7.1. Base de Conocimiento y Reglas en Prolog

1. **Sistema de diagnóstico médico**:

```
% Base de hechos
fiebre(juan).
tos(juan).
dolor_cabeza(maria).
congestion(luis).

% Reglas
gripe(X) :- fiebre(X), tos(X).
resfriado(X) :- tos(X), congestion(X), \+ fiebre(X).
migrana(X) :- dolor_cabeza(X), \+ fiebre(X).

% Consultas de ejemplo:
% ?- gripe(juan).      % true
% ?- migrana(maria).   % true
% ?- resfriado(luis).  % true

```

8. Parte 5: Evaluación de Reglas

8.1. Cálculo de Métricas

1. **Evaluación de reglas**:

```
def evaluate_rule(X, y, feature, threshold, operator='>'):
    if operator == '>':
        predictions = (X[feature] > threshold).astype(int)
    else:
        predictions = (X[feature] <= threshold).astype(int)

    accuracy = accuracy_score(y, predictions)
    coverage = len(predictions) / len(y)

    # Calcular precisi n, recall y F1-score
    tn, fp, fn, tp = confusion_matrix(y, predictions).ravel()
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (
   precision + recall) > 0 else 0

    return accuracy, coverage, precision, recall, f1

# Evaluar regla para edad
accuracy, coverage, precision, recall, f1 = evaluate_rule(X, y, '
   Age', 30, '>')
print(f"Precisi n: {accuracy:.2f}, Cobertura: {coverage:.2f}")
print(f"Precisi n: {precision:.2f}, Recall: {recall:.2f}, F1: {f1
   :.2f}")

```

8.2. Comparación de Enfoques

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

```python
# Crear datos de prueba
datos_edad = [(10, "Ni o"), (15, "Joven"), (25, "Adulto"), (70, "
    Anciano")]

# Evaluar reglas manuales
correctos = 0
for edad, esperado in datos_edad:
    if clasificar_edad(edad) == esperado:
        correctos += 1

precision_manual = correctos / len(datos_edad)
print(f"Precisi n del sistema manual: {precision_manual:.2f}")

# Comparar con  rbol  de decisi n (usando Iris como ejemplo)
y_pred_iris = clf.predict(X_iris)
precision_arbol = accuracy_score(y_iris, y_pred_iris)
print(f"Precisi n del  rbol  de decisi n: {precision_arbol:.2f}")

```

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