# Curso: Inteligencia Artificial

**Unidad 2:** Aprendizaje automático

**Sesión 11:** Aprendizaje de reglas

**Docente:** Carlos R. P. Tovar

Universidad Tecnológica del Perú

# INICIO

## ¿Tienen dudas o consultas sobre la clase previa?

# Objetivo de la sesión

Al finalizar la sesión, el alumno será capaz de:

- Comprender los fundamentos del aprendizaje de reglas.
- Implementar algoritmos básicos de aprendizaje de reglas.
- Evaluar la calidad de conjuntos de reglas aprendidas.
- Aplicar estas técnicas en problemas de clasificación.

# UTILIDAD
# ¿Por qué es importante el Aprendizaje de reglas?

**Ventajas**

- Interpretabilidad.
- Flexibilidad.
- Sin necesidad de escalado.
- Base para modelos ensemble.

**Casos de uso**

- Diagnóstico médico.
- Scoring crediticio.
- Segmentación de clientes.
- Auditoría de procesos.

# TRANSFORMACIÓN
# Introducción al Aprendizaje de Reglas

El aprendizaje de reglas es una técnica de machine learning que extrae reglas lógicas de los datos. Estas reglas siguen la estructura: **SI** (condición) **ENTONCES** (conclusión).
Ejemplo:
*SI "Llueve" Y "Hace frío" ENTONCES "Llevar paraguas".*

**Ventajas**:

- Interpretación fácil para humanos.
- Ideal para sistemas expertos y diagnósticos.

# Aplicaciones en el Mundo Real

- **Medicina:** Diagnóstico de enfermedades
- **Finanzas:** Detección de fraudes
- **Marketing:** Segmentación de clientes
- **Manufactura:** Control de calidad

# Algoritmos Principales

- **CN2**: Genera reglas iterativamente cubriendo ejemplos.
- **RIPPER**: Optimizado para datasets grandes y ruidosos.
- **OneR**: Basado en un solo atributo (sencillo pero limitado).
- **PART**: Combina árboles de decisión y reglas.

# Prolog para Aprendizaje de Reglas – Ventajas:

- **Lenguaje nativo para reglas**: Sintaxis diseñada específicamente para lógica de reglas
- **Inferencia automática**: Motor de inferencia incorporado
- **Transparencia**: Las reglas son explícitas y fácilmente interpretables
- **Unificación**: Potente para manejar relaciones complejas

# Prolog para Aprendizaje de Reglas – Implementación típica:

```
% Sistema de reglas médico en Prolog
sintoma(fiebre_alta).
sintoma(tos_persistente).
sintoma(dificultad_respirar).

enfermedad(gripe) :-
    sintoma(fiebre_alta),
    sintoma(tos_persistente).

enfermedad(neumonia) :-
    sintoma(fiebre_alta),
    sintoma(dificultad_respirar).

% Consulta
?- enfermedad(X).
```

# Python para Aprendizaje de Reglas - Ventajas

- **Ecosistema completo**: Bibliotecas como scikit-learn, PyKEEN, Orange
- **Integración con ML**: Fácil transición a otros algoritmos de aprendizaje automático
- **Comunidad amplia**: Más recursos, tutoriales y soporte
- **Preprocesamiento**: Herramientas robustas para limpieza y preparación de datos

# Métricas de Evaluación

- **Precisión:** Proporción de aciertos
- **Cobertura:** Proporción de instancias cubiertas
- **Confianza:** Fiabilidad de la regla

**Fórmula:**

$$\text{Confianza} = \frac{\text{Instancias correctas}}{\text{Instancias cubiertas}}$$

# Implementación en Python

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.rules import RuleFitClassifier

# Cargar datos
data = load_wine()
X, y = data.data, data.target

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

# Entrenar modelo
model = RuleFitClassifier()
model.fit(X_train, y_train)

# Evaluar
print("Precisión:", model.score(X_test, y_test))
```

# Algoritmo OneR (One Rule)

- **Concepto:** Crea reglas basadas en un solo atributo.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

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

# Entrenar modelo
model = DecisionTreeClassifier(max_depth=1)  # Solo una regla
model.fit(X, y)
```

# Algoritmo RIPPER

**Ventajas:**

- Maneja datasets grandes
- Resistente a ruido
- Reglas interpretables

**Proceso:**

1. Crear reglas iniciales
2. Podar reglas (eliminar condiciones)
3. Optimizar conjunto de reglas

# Algoritmo CN2:

**Ventajas**

- Manejo de Ruido y Datos Incompletos
- Flexibilidad en la Búsqueda
- Reglas Comprehensibles
- Eficiencia Computacional
- Evaluación Estadística

**Proceso**

- **Fase 1:** Inicialización
- **Fase 2:** Búsqueda de Reglas (Beam Search)
- **Fase 3:** Generación de Especializaciones
- **Fase 4:** Evaluación de Reglas

# CN2 Implementación en Python

**Bibliotecas:**

- Scikit-learn: DecisionTree (max_depth=1)
- MLxtend: Apriori algorithm
- Orange3: Rule induction

```
import Orange
data = Orange.data.Table('deportes')
learn = Orange.classification.rules.CN2Learner()
clasificador = learn(data)
```

# PART - Algoritmo Híbrido de Reglas

**¿Qué es PART?**

PART (*Partial Decision Trees*) es un algoritmo híbrido que combina:

- La estructura de los **árboles de decisión** (como C4.5).
- La estrategia de **aprendizaje de reglas** (como RIPPER).

**Objetivo principal**:

Generar reglas interpretables manteniendo la precisión de un árbol.

**Cómo funciona:**

- Construye un árbol de decisión parcial (no completo).
- Convierte la rama más "pura" del árbol en una regla.
- Elimina los ejemplos cubiertos por esa regla.
- Repite el proceso con los datos restantes.

# PART - Algoritmo Híbrido de Reglas

```
import weka.core.jvm as jvm
from weka.core.converters import Loader
from weka.classifiers import Classifier, Evaluation
from weka.core.classes import Random

# Iniciar la JVM de Java
jvm.start()
```

```
# Cargar dataset
loader = Loader(classname="weka.core.converters.ArffLoader")
data = loader.load_file("iris.arff")  # Asegúrate de tener el archivo iris.arff
data.class_is_last()  # Indicar que la clase está en la última columna

# Crear el clasificador PART
part = Classifier(classname="weka.classifiers.rules.PART")
```

# PART - Algoritmo Híbrido de Reglas

```
# Evaluar el clasificador con validación cruzada
evaluation = Evaluation(data)
evaluation.crossvalidate_model(part, data, 10, Random(1))

print("Resultados de evaluación:")
print(f"Correctos: {evaluation.percent_correct}")
print(f"Incorrectos: {evaluation.percent_incorrect}")
print(f"Kappa: {evaluation.kappa}")
print(f"Matriz de confusión:\n{evaluation.confusion_matrix}")
```

```
# Entrenar el modelo en todos los datos
part.build_classifier(data)

# Imprimir el modelo (las reglas)
print("\nReglas generadas por PART:")
print(part)

# Detener la JVM
jvm.stop()
```

# Comparación con Otros Algoritmos

| Característica | CN2 | RIPPER | OneR | PART |
|---|---|---|---|---|
| **Tipo de algoritmo** | Inductivo, basado en cobertura | Optimización de reglas | Basado en un solo atributo | Híbrido (árboles + reglas) |
| **Manejo de datos** | Numéricos y categóricos | Numéricos y categóricos | Requiere discretización | Numéricos y categóricos |
| **Complejidad** | Alta (iterativo) | Media-alta (optimizado) | Muy baja | Media |
| **Escalabilidad** | Moderada | Alta (eficiente en grandes datos) | Muy alta | Moderada-alta |
| **Calidad de reglas** | Reglas precisas pero complejas | Reglas compactas y generalizables | Reglas simples pero limitadas | Reglas balanceadas |

# Comparación con Otros Algoritmos

| Característica | CN2 | RIPPER | OneR | PART |
|---|---|---|---|---|
| **Ruido en datos** | Sensible | Robusto | Muy sensible | Moderadamente robusto |
| **Interpretabilidad** | Media (reglas pueden ser largas) | Alta (reglas concisas) | Muy alta (reglas simples) | Alta (reglas estructuradas) |
| **Aplicaciones típicas** | Diagnóstico, sistemas expertos | Clasificación general | Baseline rápido | Sistemas de recomendación |
| **Ventajas** | Reglas detalladas | Eficiencia en grandes volúmenes | Simplicidad y velocidad | Equilibrio precisión-interpretación |
| **Desventajas** | Lento en datos grandes | Complejidad de implementación | Bajo rendimiento en problemas complejos | Puede sobreajustar si no se ajusta bien |

# Ejercicio Práctico

**Tarea**:

- Cargar el dataset wine de Scikit-Learn.
- Entrenar un modelo RuleFitClassifier.
- Extraer las 3 reglas más importantes.
- Calcular la matriz de confusión.

**Entrega**:

- Código completo en Python.
- Comentarios sobre el significado de las reglas generadas.

# CIERRE
# Conclusiones

**1. Importancia del Aprendizaje de Reglas**

- El aprendizaje de reglas proporciona **modelos interpretables** que se asemejan al razonamiento humano
- Es fundamental en aplicaciones donde la **explicabilidad** es crucial (medicina, finanzas, derecho)
- Los sistemas basados en reglas son la base de los **sistemas expertos** en IA.

# Conclusiones

**2. Algoritmos Clave Aprendidos**

- **OneR**: Ideal para introducción al concepto, pero limitado en expresividad
- **RIPPER**: Eficiente para datasets grandes, con buena resistencia al ruido
- **CN2**: Potente para descubrimiento de reglas complejas con evaluación estadística

# Conclusiones

**3. Aplicaciones Prácticas**

- Diagnóstico médico y sistemas de recomendación
- Detección de fraudes y análisis de riesgo crediticio
- Sistemas de clasificación interpretables
- Automatización de procesos de decisión

# Conclusiones

**4. Ventajas Comparativas del uso de Reglas**

- Interpretabilidad
- Similitud al razonamiento humano
- Manejo datos incompletos
- Evaluación Estadística.

# Limitaciones y Consideraciones

- **Sensibilidad al ruido** (excepto RIPPER y CN2)
- **Dificultad con atributos continuos** sin discretización
- **Posible sobreajuste** con reglas demasiado específicas
- **Computacionalmente costoso** para datasets muy grandes

# Recursos adicionales

**1. Libros Recomendados**

- **"Machine Learning"** de Tom Mitchell - Capítulo 3: Aprendizaje de Reglas
- **"Data Mining: Practical Machine Learning Tools and Techniques"** de Ian H. Witten - Capítulo 4: Algoritmos de Reglas
- **"An Introduction to Machine Learning"** de Miroslav Kubat - Capítulo 6: Rule Learning

**2. Tutoriales y Guías Prácticas**

- Scikit-learn: Decision Trees for Rule Extraction
- Orange Data Mining: Rule Induction
- KNIME: Rule-Based Learning Nodes

# Recursos adicionales

**3. Conjuntos de Datos para Practicar**

- UCI Machine Learning Repository - Buscar "rule learning"
- Kaggle: Medical Diagnosis Datasets
- OpenML: Rule Learning Benchmarks

# Recursos adicionales

**4. Herramientas y Bibliotecas**

# Bibliotecas Python para aprendizaje de reglas
- Scikit-learn: Decision Trees (para extracción de reglas)
- Skope-rules: Implementación de algoritmos de reglas
- Orange3: Plataforma visual con herramientas de rule learning
- PyKEEN: Para reglas en knowledge graphs

# Herramientas especializadas
- WEKA: Implementación de RIPPER y otros algoritmos
- RapidMiner: Extensiones para rule learning
- LISP/Prolog: Para sistemas expertos basados en reglas


Universidad
Tecnológica
del Perú