Aprendizaje por Refuerzo

CAD612023642C-A

Estudiantes:
Daniela Alejandra Álvarez
Cristian Camilo Naranjo
Andrés Felipe Venegas
Juan Sebastián Bustos

Universidad de Cundinamarca
18 de noviembre del 2025
Chía

Estructura

Modelo AprendizajeRefuerzo.py

Importaciones principales:

- **NumPy** - para manejar vectores y cálculos numéricos.
- **Matplotlib** - generar gráficas (se usa en modo "sin pantalla").
- **TensorFlow/Keras** - construir y entrenar la red neuronal.
- **Gymnasium** - el entorno CartPole.
- **Pickle y os** - guardar/cargar datos.
- **deque** - memoria para experiencia (replay buffer).

```python
import numpy as np
import matplotlib

matplotlib.use("Agg")  # Para usar matplotlib sin GUI
import matplotlib.pyplot as plt
import pickle
import os
from collections import deque
import base64
from io import BytesIO

# TensorFlow y Keras
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"  # Suprimir warnings de TensorFlow
import tensorflow as tf
import keras
from keras import layers
from keras import metrics

# Gym para el entorno
import gymnasium as gym
```

Clase DQNAgent

Este es el agente que aprende a resolver CartPole usando una red neuronal.

```python
class DQNAgent:
    """
    Agente de Deep Q-Learning para el entorno CartPole.

    Parámetros:
    -----------
    state_size : int
        Tamaño del espacio de estados (4 para CartPole)
    action_size : int
        Número de acciones posibles (2 para CartPole: izquierda/derecha)
    learning_rate : float
        Tasa de aprendizaje (α)
    gamma : float
        Factor de descuento (γ)
    epsilon : float
        Tasa de exploración inicial (ε)
    epsilon_decay : float
        Decaimiento de epsilon por episodio
    epsilon_min : float
        Valor mínimo de epsilon
    """
```

Constructor __init__()

Define:

Parámetros del DQN

- **state_size = 4** - CartPole tiene 4 variables de estado.
- **action_size = 2** - mover izquierda o derecha.
- **learning_rate = 0.001**
- **gamma = 0.99** - factor de descuento.
- **epsilon = 1.0** - exploración inicial.
- **epsilon_decay = 0.995**
- **epsilon_min = 0.01**
- Memoria: **deque(maxlen=2000).**

Crea la red neuronal

Llama a _build_model().

Crea listas para almacenar historial

- Recompensa
- Epsilon por episodio
- Promedio móvil de recompensas

```
53      def __init__(
54          self,
55          state_size=4,
56          action_size=2,
57          learning_rate=0.001,
58          gamma=0.99,
59          epsilon=1.0,
60          epsilon_decay=0.995,
61          epsilon_min=0.01,
62      ):
63
64          self.state_size = state_size
65          self.action_size = action_size
66          self.memory = deque(maxlen=2000)
67
68          # Hiperparámetros
69          self.gamma = gamma  # Factor de descuento
70          self.epsilon = epsilon  # Tasa de exploración
71          self.epsilon_decay = epsilon_decay
72          self.epsilon_min = epsilon_min
73          self.learning_rate = learning_rate
74          self.batch_size = 32
75
76          # Red neuronal
77          self.model = self._build_model()
78
79          # Historial de entrenamiento
80          self.rewards_history = []
81          self.epsilon_history = []
82          self.avg_rewards_history = []
```

Método _build_model()

Define una red neuronal simple:

Entrada: 4 valores (estado)

Capa oculta: 24 neuronas ReLU

Capa oculta: 24 neuronas ReLU

Salida: 2 neuronas (Q-value de cada acción)

Compila con:

- Pérdida: MSE
- Optimizador: Adam (learning_rate configurado)

Es una arquitectura estándar de DQN.

```python
    def _build_model(self):
        """
        Construye la red neuronal para aproximar la función Q.

        Arquitectura:
        - Capa de entrada: state_size neuronas
        - Capa oculta 1: 24 neuronas, activación ReLU
        - Capa oculta 2: 24 neuronas, activación ReLU
        - Capa de salida: action_size neuronas, activación lineal
        """
        model = keras.Sequential(
            [
                layers.Dense(24, input_dim=self.state_size, activation="relu"),
                layers.Dense(24, activation="relu"),
                layers.Dense(self.action_size, activation="linear"),
            ]
        )

        model.compile(
            loss="mse",
            optimizer=keras.optimizers.Adam(learning_rate=self.learning_rate),
        )

        return model
```

remember()

Guarda experiencias en la memoria (replay buffer):

Cada experiencia = (state, action, reward, next_state, done)

```python
    def remember(self, state, action, reward, next_state, done):
        """
        Almacena una experiencia en la memoria de replay.

        Parameters:
        -----------
        state : array
            Estado actual
        action : int
            Acción tomada
        reward : float
            Recompensa recibida
        next_state : array
            Estado siguiente
        done : bool
            Si el episodio terminó
        """
        self.memory.append((state, action, reward, next_state, done))
```

act()

Selecciona una acción según epsilon-greedy:

- Con probabilidad **epsilon**: acción aleatoria (explorar)
- Con probabilidad **1-epsilon**: acción con mayor Q-value (explotar)

Usa la red neuronal para predecir Q-values del estado actual.

```python
    def act(self, state):
        """
        Selecciona una acción usando política epsilon-greedy.

        Con probabilidad epsilon: acción aleatoria (exploración)
        Con probabilidad 1-epsilon: acción según Q-value (explotación)

        Parameters:
        -----------
        state : array
            Estado actual

        Returns:
        --------
        int : Acción seleccionada
        """
        if np.random.rand() <= self.epsilon:
            return np.random.randint(self.action_size)

        q_values = self.model.predict(state, verbose=0)
        return np.argmax(q_values[0])
```

replay()

Este método es el núcleo del aprendizaje.

Etapas:

1. Verifica si hay suficientes experiencias en memoria (≥ 32).
2. Selecciona aleatoriamente 32 experiencias.
3. Para cada experiencia:
    - Si done, target = reward.
    - Si no: target = reward + gamma * max(Q(next_state))
4. Actualiza el Q-value de la acción tomada en el vector de salida.
5. Entrena la red neuronal un solo paso (epochs=1).
6. Actualiza epsilon: epsilon *= epsilon_decay

```python
    def replay(self):
        """
        Entrena la red neuronal usando experiencias aleatorias de la memoria.
        Implementa Experience Replay para romper correlaciones temporales.
        """
        if len(self.memory) < self.batch_size:
            return

        # Muestreo aleatorio de experiencias
        minibatch = np.random.choice(len(self.memory), self.batch_size, replace=False)

        states = []
        targets = []

        for idx in minibatch:
            state, action, reward, next_state, done = self.memory[idx]

            target = reward
            if not done:
                # Q-Learning: Q(s,a) = r + γ * max(Q(s',a'))
                target = reward + self.gamma * np.amax(
                    self.model.predict(next_state, verbose=0)[0]
                )

            # Obtener Q-values actuales
            target_f = self.model.predict(state, verbose=0)
            target_f[0][action] = target

            states.append(state[0])
            targets.append(target_f[0])

        # Entrenar la red
        self.model.fit(np.array(states), np.array(targets), epochs=1, verbose=0)

        # Decaimiento de epsilon
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

train()

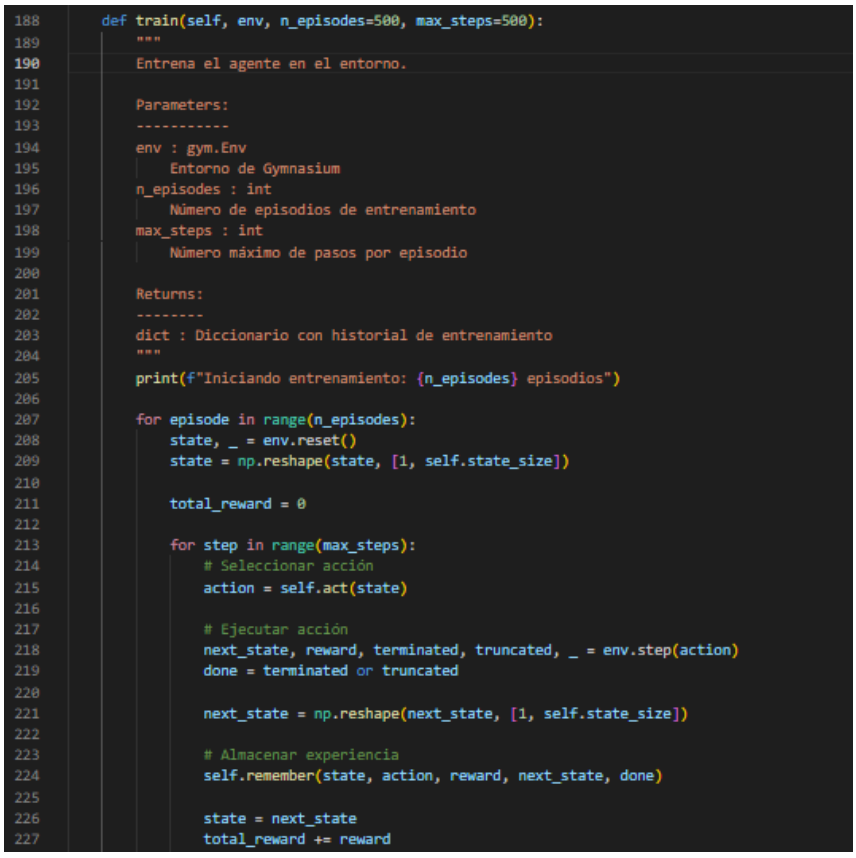
Entrena el agente durante N episodios.

Flujo:

1. Reinicia el entorno.
2. Por episodio:
    - Selecciona acciones con act().
    - Ejecuta en el entorno env.step().
    - Guarda experiencia remember().
    - Entrena usando replay().
    - Acumula recompensas.
3. Guarda:
    - recompensa total
    - epsilon
    - promedio de últimos 100 episodios

Cada 10 episodios imprime progreso.

Al final retorna un diccionario con todo el historial.



```python
    def train(self, env, n_episodes=500, max_steps=500):
        """
        Entrena el agente en el entorno.

        Parameters:
        -----------
        env : gym.Env
            Entorno de Gymnasium
        n_episodes : int
            Número de episodios de entrenamiento
        max_steps : int
            Número máximo de pasos por episodio

        Returns:
        --------
        dict : Diccionario con historial de entrenamiento
        """
        print(f"Iniciando entrenamiento: {n_episodes} episodios")

        for episode in range(n_episodes):
            state, _ = env.reset()
            state = np.reshape(state, [1, self.state_size])

            total_reward = 0

            for step in range(max_steps):
                # Seleccionar acción
                action = self.act(state)

                # Ejecutar acción
                next_state, reward, terminated, truncated, _ = env.step(action)
                done = terminated or truncated

                next_state = np.reshape(next_state, [1, self.state_size])

                # Almacenar experiencia
                self.remember(state, action, reward, next_state, done)

                state = next_state
                total_reward += reward
```

```
                if done:
                    break

            # Entrenar con experiencias
            self.replay()

            # Guardar historial
            self.rewards_history.append(total_reward)
            self.epsilon_history.append(self.epsilon)

            # Calcular recompensa promedio de los últimos 100 episodios
            if len(self.rewards_history) >= 100:
                avg_reward = np.mean(self.rewards_history[-100:])
            else:
                avg_reward = np.mean(self.rewards_history)

            self.avg_rewards_history.append(avg_reward)

            # Mostrar progreso cada 50 episodios
            if (episode + 1) % 10 == 0:
                print(
                    f"Episodio {episode + 1}/{n_episodes} | "
                    f"Recompensa: {total_reward:.0f} | "
                    f"Promedio: {avg_reward:.2f} | "
                    f"Epsilon: {self.epsilon:.3f}"
                )

        print("✓ Entrenamiento completado")

        return {
            "rewards": self.rewards_history,
            "avg_rewards": self.avg_rewards_history,
            "epsilon": self.epsilon_history,
        }
```

Guardar y cargar modelos

```
    def save_model(self, filepath="models/dqn_cartpole.h5"):
        """Guarda el modelo entrenado."""
        os.makedirs(os.path.dirname(filepath), exist_ok=True)
        self.model.save(filepath)
        print(f"Modelo guardado en: {filepath}")

    def load_model(self, filepath="models/dqn_cartpole.h5"):
        """Carga un modelo previamente entrenado."""
        if os.path.exists(filepath):
            self.model = keras.models.load_model(
                filepath,
                custom_objects={
                    "mse": metrics.mean_squared_error  # O solo metrics.mse si está disponible
                },
            )
            self.epsilon = self.epsilon_min  # Modo explotación
            print(f"Modelo cargado desde: {filepath}")
            return True
        return False

    def save_training_data(self, filepath="models/training_data.pkl"):
        """Guarda el historial de entrenamiento."""
        os.makedirs(os.path.dirname(filepath), exist_ok=True)
        data = {
            "rewards": self.rewards_history,
            "avg_rewards": self.avg_rewards_history,
            "epsilon": self.epsilon_history,
        }
        with open(filepath, "wb") as f:
            pickle.dump(data, f)
        print(f"Datos de entrenamiento guardados en: {filepath}")

    def load_training_data(self, filepath="models/training_data.pkl"):
        """Carga el historial de entrenamiento."""
        if os.path.exists(filepath):
            with open(filepath, "rb") as f:
                data = pickle.load(f)
            self.rewards_history = data["rewards"]
            self.avg_rewards_history = data["avg_rewards"]
            self.epsilon_history = data["epsilon"]
            print(f"Datos de entrenamiento cargados desde: {filepath}")
            return True
        return False
```

Funciones para la interfaz web

```python
def initialize_agent():
    """
    Inicializa el agente DQN.
    Carga el modelo si existe, si no, crea uno nuevo.
    """
    agent = DQNAgent()

    # Intentar cargar modelo existente
    if agent.load_model() and agent.load_training_data():
        return agent, True  # Modelo cargado

    return agent, False  # Modelo nuevo


def train_agent(n_episodes=500):
    """
    Entrena el agente y guarda el modelo.

    Parameters:
    -----------
    n_episodes : int
        Número de episodios de entrenamiento

    Returns:
    --------
    dict : Resultados del entrenamiento
    """
    # Crear entorno
    env = gym.make("CartPole-v1")

    # Crear agente
    agent = DQNAgent()

    # Entrenar
    results = agent.train(env, n_episodes=n_episodes)

    # Guardar modelo y datos
    agent.save_model()
    agent.save_training_data()

    env.close()

    return results
```

```python
def test_agent(n_episodes=5):
    """
    Prueba el agente entrenado.

    Parameters:
    -----------
    n_episodes : int
        Número de episodios de prueba

    Returns:
    --------
    dict : Resultados de las pruebas
    """
    # Crear entorno
    env = gym.make("CartPole-v1")

    # Cargar agente
    agent = DQNAgent()
    if not agent.load_model():
        return {"error": "No hay modelo entrenado disponible"}

    test_rewards = []

    for episode in range(n_episodes):
        state, _ = env.reset()
        state = np.reshape(state, [1, agent.state_size])

        total_reward = 0
        done = False
        steps = 0

        while not done and steps < 500:
            action = agent.act(state)
            next_state, reward, terminated, truncated, _ = env.step(action)
            done = terminated or truncated

            state = np.reshape(next_state, [1, agent.state_size])
            total_reward += reward
            steps += 1

        test_rewards.append(total_reward)

    env.close()

    return {
        "rewards": test_rewards,
        "avg_reward": np.mean(test_rewards),
        "max_reward": np.max(test_rewards),
        "min_reward": np.min(test_rewards),
    }
```

```python
def plot_training_progress():
    """
    Genera un gráfico del progreso de entrenamiento.

    Returns:
    --------
    str : Imagen en formato base64
    """
    agent = DQNAgent()
    if not agent.load_training_data():
        return None

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

    # Gráfico 1: Recompensas
    episodes = range(1, len(agent.rewards_history) + 1)
    ax1.plot(
        episodes, agent.rewards_history, alpha=0.3, label="Recompensa por episodio"
    )
    ax1.plot(
        episodes,
        agent.avg_rewards_history,
        linewidth=2,
        label="Promedio móvil (100 episodios)",
    )
    ax1.set_xlabel("Episodio")
    ax1.set_ylabel("Recompensa Total")
    ax1.set_title("Evolución de la Recompensa durante el Entrenamiento")
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Gráfico 2: Epsilon
    ax2.plot(episodes, agent.epsilon_history, color="orange")
    ax2.set_xlabel("Episodio")
    ax2.set_ylabel("Epsilon (Tasa de Exploración)")
    ax2.set_title("Decaimiento de Epsilon")
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()

    # Convertir a base64
    buffer = BytesIO()
    plt.savefig(buffer, format="png", dpi=100, bbox_inches="tight")
    buffer.seek(0)
    image_base64 = base64.b64encode(buffer.getvalue()).decode()
    plt.close()

    return image_base64
```

```python
def get_training_stats():
    """
    Obtiene estadísticas del entrenamiento actual.

    Returns:
    --------
    dict : Estadísticas de entrenamiento
    """
    agent = DQNAgent()

    if not agent.load_training_data():
        return {
            "trained": False,
            "n_episodes": 0,
            "final_epsilon": 1.0,
            "avg_reward": 0,
            "max_reward": 0,
        }

    return {
        "trained": True,
        "n_episodes": len(agent.rewards_history),
        "final_epsilon": agent.epsilon_history[-1] if agent.epsilon_history else 1.0,
        "avg_reward": (
            np.mean(agent.avg_rewards_history[-100:])
            if len(agent.avg_rewards_history) >= 100
            else 0
        ),
        "max_reward": np.max(agent.rewards_history) if agent.rewards_history else 0,
        "final_avg": agent.avg_rewards_history[-1] if agent.avg_rewards_history else 0,
    }


def reset_model():
    """
    Elimina el modelo entrenado y los datos guardados.

    Returns:
    --------
    bool : True si se eliminó correctamente
    """
    import os

    model_path = "models/dqn_cartpole.h5"
    data_path = "models/training_data.pkl"

    removed = False
    if os.path.exists(model_path):
        os.remove(model_path)
        removed = True
    if os.path.exists(data_path):
        os.remove(data_path)
        removed = True

    print(f"✓ Modelo reiniciado correctamente")
    return removed
```

```python
def test_agent_with_trajectory(n_episodes=5):
    """
    Prueba el agente entrenado y retorna la trayectoria del último episodio.

    Parameters:
    -----------
    n_episodes : int
        Número de episodios de prueba

    Returns:
    --------
    tuple : (test_results, trajectory_data)
    """
    import gymnasium as gym
    import numpy as np

    # Crear entorno
    env = gym.make("CartPole-v1")

    # Cargar agente
    agent = DQNAgent()
    if not agent.load_model():
        return {"error": "No hay modelo entrenado disponible"}, None

    test_rewards = []
    last_states = []
    last_actions = []

    for episode in range(n_episodes):
        state, _ = env.reset()
        state = np.reshape(state, [1, agent.state_size])

        total_reward = 0
        done = False
        steps = 0

        episode_states = []
        episode_actions = []

        while not done and steps < 500:
            action = agent.act(state)
            next_state, reward, terminated, truncated, _ = env.step(action)
            done = terminated or truncated

            # Guardar estados y acciones del último episodio
            if episode == n_episodes - 1:
                episode_states.append(state[0])
                episode_actions.append(action)

            state = np.reshape(next_state, [1, agent.state_size])
            total_reward += reward
            steps += 1
```

```python
        test_rewards.append(total_reward)

        # Guardar la trayectoria del último episodio
        if episode == n_episodes - 1:
            last_states = episode_states
            last_actions = episode_actions

    env.close()

    test_results = {
        "rewards": test_rewards,
        "avg_reward": np.mean(test_rewards),
        "max_reward": np.max(test_rewards),
        "min_reward": np.min(test_rewards),
    }

    trajectory_data = (
        {"states": last_states, "actions": last_actions} if last_states else None
    )

    return test_results, trajectory_data


def plot_trajectory(states, actions):
    """
    Genera un gráfico de la trayectoria del agente.

    Parameters:
    -----------
    states : list
        Lista de estados (posición, velocidad, ángulo, velocidad angular)
    actions : list
        Lista de acciones tomadas

    Returns:
    --------
    str : Imagen en formato base64
    """
    import matplotlib

    matplotlib.use("Agg")
    import matplotlib.pyplot as plt
    import base64
    from io import BytesIO

    if not states or len(states) == 0:
        return None

    positions = [s[0] for s in states]
    angles = [s[2] for s in states]
    steps = list(range(len(states)))

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
```

```python
    # Posición del carrito
    ax1.plot(steps, positions, "b-", linewidth=2)
    ax1.axhline(y=0, color="gray", linestyle="--", alpha=0.5)
    ax1.axhline(y=2.4, color="red", linestyle="--", alpha=0.3, label="Límite")
    ax1.axhline(y=-2.4, color="red", linestyle="--", alpha=0.3)
    ax1.set_xlabel("Paso")
    ax1.set_ylabel("Posición del Carrito")
    ax1.set_title("Posición del Carrito a lo Largo del Tiempo")
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Ángulo del poste
    ax2.plot(steps, angles, "r-", linewidth=2)
    ax2.axhline(y=0, color="gray", linestyle="--", alpha=0.5)
    ax2.fill_between(
        steps, -0.2095, 0.2095, alpha=0.2, color="green", label="Zona segura (±12°)"
    )
    ax2.axhline(y=0.2095, color="red", linestyle="--", alpha=0.3)
    ax2.axhline(y=-0.2095, color="red", linestyle="--", alpha=0.3)
    ax2.set_xlabel("Paso")
    ax2.set_ylabel("Ángulo del Poste (radianes)")
    ax2.set_title("Ángulo del Poste a lo Largo del Tiempo")
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()

    buffer = BytesIO()
    plt.savefig(buffer, format="png", dpi=100, bbox_inches="tight")
    buffer.seek(0)
    image_base64 = base64.b64encode(buffer.getvalue()).decode()
    plt.close()

    return image_base64
```

Código final

```python
if __name__ == "__main__":
    print("=" * 70)
    print("DEEP Q-LEARNING - CARTPOLE")
    print("=" * 70)

    # Entrenar agente
    print("\n1. Entrenando agente...")
    results = train_agent(n_episodes=100)

    # Probar agente
    print("\n2. Probando agente...")
    test_results = test_agent(n_episodes=10)
    print(f"Recompensa promedio en pruebas: {test_results['avg_reward']:.2f}")

    # Generar gráfico
    print("\n3. Generando gráfico...")
    plot_training_progress()
    print("Gráfico generado exitosamente")

    print("\n✓ Proceso completado")
```

Templates

refuerzo_conceptos.html

```html
{% extends "base.html" %} <!-- Hereda de la plantilla base para estructura común -->

{% block title %}Conceptos Básicos - Aprendizaje por Refuerzo{% endblock %} <!-- Define el título de la página -->

{% block content %} <!-- Inicia el bloque de contenido principal -->
<div class="container mt-5 mb-5"> <!-- Contenedor principal con márgenes -->

    <header class="text-center mb-5"> <!-- Encabezado centrado -->
        <h1 class="display-4">Aprendizaje por Refuerzo</h1> <!-- Título principal -->
        <p class="lead text-muted">Teoría completa, Algoritmos y Buenas Prácticas</p> <!-- Subtítulo -->
    </header>

    <!-- Tarjeta 1: Definición General -->
    <div class="card shadow-sm mb-5"> <!-- Tarjeta con sombra -->
        <div class="card-header bg-primary text-white"> <!-- Encabezado azul -->
            <h2 class="h4 mb-0">1. Definición General</h2> <!-- Título de sección -->
        </div>
        <div class="card-body"> <!-- Cuerpo de tarjeta -->
            <p> <!-- Párrafo descriptivo -->
                El aprendizaje por refuerzo es una disciplina distinta donde no hay un supervisor que indique la acción correcta;
                el agente recibe una señal (recompensa) después de actuar. Las consecuencias de las acciones
                pueden ser retrasadas en el tiempo.
            </p>
            <div class="row mt-3"> <!-- Fila comparativa -->
                <div class="col-md-6 border-end"> <!-- Columna izquierda con borde -->
                    <h6>Vs. Supervisado</h6> <!-- Subtítulo comparación -->
                    <small>No hay un "profesor". El feedback es una recompensa, no una etiqueta correcta.</small> <!-- Descripción -->
                </div>
                <div class="col-md-6"> <!-- Columna derecha -->
                    <h6>Vs. No Supervisado</h6>
                    <small>Se busca maximizar una señal de recompensa, no solo encontrar patrones ocultos en datos.</small>
                </div>
            </div>
        </div>
    </div>
```

```html
    <!-- Tarjeta 2: Componentes del Sistema -->
    <div class="card shadow-sm mb-5">
        <div class="card-header bg-success text-white"> <!-- Encabezado verde -->
            <h2 class="h4 mb-0">2. Componentes y Ciclo de Aprendizaje</h2>
        </div>
        <div class="card-body">
            <div class="row mb-4"> <!-- Fila de dos columnas -->
                <div class="col-lg-6"> <!-- Columna elementos -->
                    <h5 class="text-success">Elementos del Modelo</h5> <!-- Subtítulo -->
                    <ul class="list-unstyled"> <!-- Lista sin viñetas -->
                        <li><strong>Agente:</strong> Implementa la política y toma decisiones.</li> <!-- Elemento de lista -->
                        <li><strong>Entorno:</strong> Mundo físico o simulado donde opera el agente.</li>
                        <li><strong>Estados (S):</strong> Información necesaria para decidir.</li>
                        <li><strong>Acciones (A):</strong> Decisiones discretas o continuas.</li>
                        <li><strong>Recompensas (R):</strong> Señal clave. Si se diseña mal, el agente puede aprender conductas indeseadas.</li>
                    </ul>
                </div>
                <div class="col-lg-6"> <!-- Columna principios -->
                    <h5 class="text-success">Principios del Ciclo</h5>
                    <p class="small"> <!-- Texto pequeño -->
                        <strong>Exploración vs Explotación:</strong> El agente usa estrategias como ε-greedy
                        para equilibrar descubrir nuevas acciones (explorar) y usar las mejores conocidas (explotar).
                    </p>
                    <p class="small">
                        <strong>Descuento Temporal (γ):</strong> Regula la visión a futuro. Un γ cercano a 1 prioriza
                        el largo plazo, mientras que uno bajo prioriza lo inmediato.
                    </p>
                </div>
            </div>
        </div>
    </div>
```

```
<!-- Tarjeta 3: Algoritmos con Pestañas -->
<div class="card shadow-sm mb-5">
    <div class="card-header bg-info text-white"> <!-- Encabezado azul claro -->
        <h2 class="h4 mb-0">3. Algoritmos Principales</h2>
    </div>
    <div class="card-body">
        <ul class="nav nav-tabs" id="algoTab" role="tablist"> <!-- Navegación por pestañas -->
            <li class="nav-item" role="presentation"> <!-- Item Q-Learning -->
                <button class="nav-link active" id="qlearning-tab" data-bs-toggle="tab" data-bs-target="#qlearning" type="button" role="tab">Q-Learning</button>
                <!-- Botón pestaña activa -->
            </li>
            <li class="nav-item" role="presentation"> <!-- Item SARSA -->
                <button class="nav-link" id="sarsa-tab" data-bs-toggle="tab" data-bs-target="#sarsa" type="button" role="tab">SARSA</button>
            </li>
            <li class="nav-item" role="presentation"> <!-- Item DQN -->
                <button class="nav-link" id="dqn-tab" data-bs-toggle="tab" data-bs-target="#dqn" type="button" role="tab">Deep Q-Network</button>
            </li>
        </ul>
        <div class="tab-content p-4 border border-top-0 rounded-bottom" id="algoTabContent"> <!-- Contenido pestañas -->

            <div class="tab-pane fade show active" id="qlearning" role="tabpanel"> <!-- Pestaña Q-Learning activa -->
                <h5>Q-Learning (Off-Policy)</h5> <!-- Título algoritmo -->
                <p>Es un algoritmo <strong>off-policy</strong>: aprende la mejor política posible independientemente de la política que está usando para explorar.</p>
                <!-- Descripción -->

                <div class="alert alert-light text-dark text-center border shadow-sm"> <!-- Contenedor fórmula -->
                    <p class="mb-0 font-monospace fw-bold"> <!-- Fórmula matemática -->
                        $$ Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) $$
                    </p>
                </div>

                <small class="text-muted">Muy usado en entornos discretos.</small> <!-- Nota adicional -->
            </div>
```

```
            <div class="tab-pane fade" id="sarsa" role="tabpanel"> <!-- Pestaña SARSA -->
                <h5>SARSA (On-Policy)</h5>
                <p>Es <strong>on-policy</strong>: actualiza los valores usando la acción que realmente selecciona su política actual.</p>
                <p>Es más "conservador" y seguro, ya que considera la exploración dentro del aprendizaje.</p>

                <div class="alert alert-light text-dark text-center border shadow-sm"> <!-- Fórmula SARSA -->
                    <p class="mb-0 font-monospace fw-bold">
                        $$ Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)) $$
                    </p>
                </div>
            </div>

            <div class="tab-pane fade" id="dqn" role="tabpanel"> <!-- Pestaña DQN -->
                <h5>Deep Q-Network (DQN)</h5>
                <p> <!-- Descripción DQN -->
                    Cuando el espacio de estados es muy grande (ej. imágenes), no se pueden usar tablas. DQN usa <strong>redes neuronales profundas</strong>
                    para aproximar Q(s,a).
                </p>
                <div class="mt-3"> <!-- Innovaciones -->
                    <h6>Innovaciones Clave:</h6>
                    <ul> <!-- Lista innovaciones -->
                        <li><strong>Experience Replay:</strong> Almacena experiencias en memoria para entrenar en "lotes" aleatorios, rompiendo correlaciones.</li>
                        <li><strong>Target Network:</strong> Una red separada para calcular el valor objetivo, mejorando la estabilidad.</li>
                    </ul>
                </div>
            </div>
        </div>
    </div>
</div>
```

```
<!-- Tarjeta 4: Buenas Prácticas -->
<div class="card shadow-sm mb-5 border-warning"> <!-- Tarjeta con borde amarillo -->
    <div class="card-header bg-warning text-dark"> <!-- Encabezado amarillo -->
        <h3 class="h5 mb-0"><i class="fas fa-check-circle me-2"></i>Buenas Prácticas en RL</h3> <!-- Título con icono -->
    </div>
    <div class="card-body">
        <div class="row"> <!-- Fila dos columnas -->
            <div class="col-md-6"> <!-- Columna estabilidad -->
                <h6 class="text-primary">Estabilidad y Convergencia</h6> <!-- Subtítulo -->
                <ul class="list-group list-group-flush mb-3"> <!-- Lista grupo -->
                    <li class="list-group-item"> <!-- Item lista -->
                        <strong>Experience Replay:</strong> Vital en DQN para romper la correlación entre datos consecutivos.
                    </li>
                    <li class="list-group-item">
                        <strong>Tasa de aprendizaje (α):</strong> Ajustarla cuidadosamente. Si es muy alta no converge; si es muy baja es demasiado lento.
                    </li>
                    <li class="list-group-item">
                        <strong>Factor de descuento (γ):</strong> Elegirlo según si el problema requiere visión a largo o corto plazo.
                    </li>
                </ul>
            </div>
            <div class="col-md-6"> <!-- Columna exploración -->
                <h6 class="text-primary">Exploración y Recompensas</h6>
                <ul class="list-group list-group-flush mb-3">
                    <li class="list-group-item">
                        <strong>Decay de ε:</strong> Comenzar con alta exploración y reducirla gradualmente para evitar quedar en óptimos locales.
                    </li>
                    <li class="list-group-item">
                        <strong>Diseño de Recompensas:</strong> Evitar recompensas "engañosas" (ej. premiar velocidad si causa choques).
                    </li>
                    <li class="list-group-item">
                        <strong>Generalización:</strong> Usar regularización o aleatorizar el entorno para evitar la memorización.
                    </li>
                </ul>
            </div>
        </div>
    </div>
</div>
```

```
<!-- Tarjeta 5: Conclusiones -->
<div class="card shadow-sm mb-5 border-danger"> <!-- Tarjeta con borde rojo -->
    <div class="card-header bg-danger text-white"> <!-- Encabezado rojo -->
        <h2 class="h4 mb-0">4. Conclusiones Clave</h2>
    </div>
    <div class="card-body">
        <p> <!-- Párrafo introductorio -->
            El Aprendizaje por Refuerzo (RL) es un paradigma poderoso para la toma de decisiones secuenciales, caracterizado por la interacción dinámica entre
            un **Agente** y un **Entorno**. A diferencia del aprendizaje supervisado, el agente aprende a maximizar una recompensa acumulada a largo plazo a través de la
            experimentación (Exploración) y el uso del conocimiento actual (Explotación).
        </p>
        <ul class="list-group list-group-flush"> <!-- Lista conclusiones -->
            <li class="list-group-item"> <!-- Item conclusión 1 -->
                <strong>RL Clásico vs. Profundo:</strong> Algoritmos como **Q-Learning** (Off-Policy) y **SARSA** (On-Policy) son fundamentales y efectivos en espacios de
                estados discretos. Sin embargo, para problemas complejos con grandes espacios de estados (como la visión por computadora o entornos 3D), el **DQN** integra
                redes neuronales profundas para aproximar las funciones de valor, haciendo escalable el RL.
            </li>
            <li class="list-group-item"> <!-- Item conclusión 2 -->
                <strong>Desafíos y Sintonización:</strong> La estabilidad del aprendizaje, la convergencia y la obtención de políticas óptimas dependen críticamente del
                balance entre exploración y explotación (controlado por **ε**), la visión a futuro (**γ**), y la calidad del **diseño de la recompensa**. Una implementación
                exitosa requiere una cuidadosa sintonización de hiperparámetros y el uso de técnicas estabilizadoras como el *Experience Replay*.
            </li>
        </ul>
    </div>
</div>
```

```
<!-- Acordeón de Referencias -->
<div class="accordion" id="accordionReferences"> <!-- Acordeón contenedor -->
    <div class="accordion-item"> <!-- Item acordeón -->
        <h2 class="accordion-header" id="headingOne"> <!-- Encabezado acordeón -->
            <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse" data-bs-target="#collapseOne"> <!-- Botón colapsable -->
                <strong>Referencias Bibliográficas (APA 7)</strong> <!-- Título acordeón -->
            </button>
        </h2>
        <div id="collapseOne" class="accordion-collapse collapse"> <!-- Contenido colapsado -->
            <div class="accordion-body"> <!-- Cuerpo acordeón -->
                <p class="small mb-1">Sutton, R. S., & Barto, A. G. (2018). <i>Reinforcement learning: An introduction</i> (2nd ed.). MIT Press.</p> <!-- Referencia 1 -->
                <p class="small mb-1">Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. <i>Nature, 518</i>(7540), 529-533.</p>
                <!-- Referencia 2 -->
                <p class="small mb-1">Arulkumaran, K., et al. (2017). Deep reinforcement learning: A brief survey. <i>IEEE Signal Processing Magazine</i>.</p>
                <!-- Referencia 3 -->
                <hr> <!-- Separador -->
                <p class="small mb-1">FlowHunt. (s.f.). <i>Glosario RL: Aprendizaje por Refuerzo</i>. Recuperado de flowhunt.io.</p> <!-- Referencia web 1 -->
                <p class="small mb-1">Fundación Bankinter. (s.f.). <i>¿Qué es Q-Learning?</i> Recuperado de fundacionbankinter.org.</p> <!-- Referencia web 2 -->
                <p class="small mb-1">FUOC. (s.f.). <i>Introducción al aprendizaje por refuerzo</i> (Documento PDF). Repositorio de Acceso Abierto UOC.</p>
                <!-- Referencia web 3 -->
                <p class="small mb-1">Gavilán, I. G. R. (s.f.). <i>Notas sobre aprendizaje por refuerzo: SARSA y Q-Learning</i> [Blog]. Recuperado de Ignacio G.R. Gavilán.</
                <!-- Referencia web 4 -->
                <p class="small mb-1">Notus.cl. (s.f.). <i>Qué es el Reinforcement Learning y cuáles son sus aplicaciones</i> [Artículo conceptual]. Recuperado de Notus.</p>
                <!-- Referencia web 5 -->
                <p class="small mb-1">Tópicos Avanzados de IA - Aprendizaje por Refuerzo. (s.f.). Apuntes de curso universitario. Recuperado de rexemin.github.io.</p>
                <!-- Referencia web 6 -->
                <p class="small mb-0">Universidad de Zaragoza / UNIZAR. (s.f.). <i>Tesis sobre modelos de Aprendizaje por Refuerzo</i>. Recuperado de Zagu.</p>
                <!-- Referencia web 7 -->
            </div>
        </div>
    </div>
</div>
```

```
    <!-- Navegación -->
    <div class="d-flex justify-content-between my-4"> <!-- Contenedor flex navegación -->
        <a href="{{ url_for('index') }}" class="btn btn-outline-primary"> <!-- Botón volver inicio -->
            <i class="fas fa-home me-2"></i> Volver al inicio <!-- Icono y texto -->
        </a>
    </div>
</div>
{% endblock %} <!-- Fin bloque contenido -->
```

Propósito: Página teórica que presenta los fundamentos del Aprendizaje por Refuerzo de manera estructurada y educativa.

Estructura Principal:

- Definición General - Explica qué es el aprendizaje por refuerzo y sus diferencias con otros paradigmas
- Componentes del Sistema - Describe agentes, entornos, estados, acciones y recompensas
- Algoritmos Principales - Presenta Q-Learning, SARSA y DQN mediante pestañas interactivas
- Buenas Prácticas - Ofrece recomendaciones para entrenamiento estable
- Conclusiones Clave - Resume los puntos más importantes
- Referencias Bibliográficas - Incluye fuentes académicas en formato APA 7

Características Técnicas:

- Diseño responsive con tarjetas Bootstrap
- Navegación por pestañas para algoritmos
- Acordeón colapsable para referencias
- Fórmulas matemáticas para ecuaciones clave
- Iconografía consistente con Font Awesome

refuerzo_practico.html

```
{% extends "base.html" %} <!-- Hereda plantilla base -->
{% block title %}Caso Práctico - Aprendizaje por Refuerzo{% endblock %} <!-- Título página -->
{% block content %} <!-- Inicio contenido -->
<div class="container py-4"> <!-- Contenedor principal -->
  <!-- Header Section -->
  <div class="text-center mb-5"> <!-- Encabezado centrado -->
    <h1 class="display-5 fw-bold"> <!-- Título principal -->
      <i class="fas fa-robot text-primary me-2"></i> <!-- Icono robot -->
      Caso Práctico: Deep Q-Learning
    </h1>
    <p class="lead text-muted"> <!-- Subtítulo -->
      Entrenamiento de un agente para equilibrar el poste en CartPole
    </p>
  </div>

  <!-- Alert Messages -->
  {% with messages = get_flashed_messages(with_categories=true) %} <!-- Obtener mensajes flash -->
  {% if messages %} <!-- Si hay mensajes -->
  {% for category, message in messages %} <!-- Iterar mensajes -->
  <div
    class="alert alert-{{ category }} alert-dismissible fade show" <!-- Alert con categoría -->
    role="alert"
  >
    <i class="fas fa-info-circle me-2"></i>{{ message }} <!-- Icono y mensaje -->
    <button type="button" class="btn-close" data-bs-dismiss="alert"></button> <!-- Botón cerrar -->
  </div>
  {% endfor %} <!-- Fin iteración -->
  {% endif %} <!-- Fin condición -->
  {% endwith %} <!-- Fin bloque mensajes -->
```

```
  <div class="row g-4 mb-4"> <!-- Fila principal -->
    <!-- Environment Information Card -->
    <div class="col-lg-6"> <!-- Columna información entorno -->
      <div class="card border-0 shadow-sm h-100"> <!-- Tarjeta sin borde -->
        <div class="card-header bg-primary text-white border-0"> <!-- Encabezado azul -->
          <h4 class="card-title mb-0"> <!-- Título tarjeta -->
            <i class="fas fa-info-circle me-2"></i> <!-- Icono información -->
            Entorno: CartPole-v1
          </h4>
        </div>
        <div class="card-body"> <!-- Cuerpo tarjeta -->
          <div class="text-center mb-3"> <!-- Icono centrado -->
            <i class="fas fa-shopping-cart fa-4x text-primary"></i> <!-- Icono carrito grande -->
          </div>

          <h5 class="mb-3">Objetivo</h5> <!-- Subtítulo -->
          <p> <!-- Descripción objetivo -->
            Mantener el poste en posición vertical el mayor tiempo posible
            aplicando fuerzas al carrito.
          </p>

          <h5 class="mb-3">Especificaciones</h5> <!-- Subtítulo especificaciones -->
          <ul class="list-group list-group-flush"> <!-- Lista especificaciones -->
            <li class="list-group-item d-flex justify-content-between"> <!-- Item con flex -->
              <strong>Estados:</strong> <!-- Etiqueta -->
              <span class="badge bg-info">4 dimensiones</span> <!-- Valor badge -->
            </li>
            <li class="list-group-item d-flex justify-content-between">
              <strong>Acciones:</strong>
              <span class="badge bg-success">2 posibles</span>
            </li>
            <li class="list-group-item d-flex justify-content-between">
              <strong>Máx. pasos:</strong>
              <span class="badge bg-warning text-dark">500</span>
            </li>
          </ul>
```

```
        <div class="mt-3"> <!-- Sección estados observables -->
          <h6 class="fw-bold">Estados observables:</h6> <!-- Subtítulo -->
          <ul class="small"> <!-- Lista pequeña -->
            <li>Posición del carrito</li> <!-- Estado 1 -->
            <li>Velocidad del carrito</li> <!-- Estado 2 -->
            <li>Ángulo del poste</li> <!-- Estado 3 -->
            <li>Velocidad angular del poste</li> <!-- Estado 4 -->
          </ul>
        </div>

        <div class="mt-3"> <!-- Sección acciones -->
          <h6 class="fw-bold">Acciones disponibles:</h6>
          <ul class="small">
            <li><strong>0:</strong> Empujar hacia la izquierda</li> <!-- Acción 0 -->
            <li><strong>1:</strong> Empujar hacia la derecha</li> <!-- Acción 1 -->
          </ul>
        </div>
      </div>
    </div>
  </div>
```

```
  <!-- Training Control Panel -->
  <div class="col-lg-6"> <!-- Columna panel control -->
    <div class="card border-0 shadow-sm h-100"> <!-- Tarjeta panel -->
      <div class="card-header bg-success text-white border-0"> <!-- Encabezado verde -->
        <h4 class="card-title mb-0"> <!-- Título -->
          <i class="fas fa-sliders-h me-2"></i> <!-- Icono controles -->
          Panel de Control
        </h4>
      </div>
      <div class="card-body"> <!-- Cuerpo panel -->
        <form
          id="trainingForm" <!-- ID formulario -->
          method="POST" <!-- Método POST -->
          action="{{ url_for('refuerzo_practico') }}" <!-- Acción endpoint -->
        >
          <input type="hidden" name="action" value="train" /> <!-- Campo oculto acción -->

          <div class="mb-4"> <!-- Grupo episodios -->
            <label for="n_episodes" class="form-label fw-bold"> <!-- Etiqueta -->
              <i class="fas fa-layer-group me-1"></i> <!-- Icono capas -->
              Número de episodios
            </label>
            <input
              type="number" <!-- Input numérico -->
              class="form-control form-control-lg" <!-- Clases control grande -->
              id="n_episodes" <!-- ID input -->
              name="n_episodes" <!-- Nombre campo -->
              min="10" <!-- Mínimo valor -->
              max="1000" <!-- Máximo valor -->
              value="100" <!-- Valor por defecto -->
              required <!-- Campo requerido -->
            />
            <div class="form-text"> <!-- Texto ayuda -->
              <i class="fas fa-info-circle me-1"></i> <!-- Icono información -->
              Recomendado: 500 episodios <!-- Recomendación -->
            </div>
          </div>
```

```
          <div class="mb-4"> <!-- Sección hiperparámetros -->
            <h6 class="fw-bold">Hiperparámetros del modelo:</h6> <!-- Subtítulo -->
            <ul class="list-unstyled small text-muted"> <!-- Lista sin estilo -->
              <li><strong>α (learning rate):</strong> 0.001</li> <!-- Hiperparámetro 1 -->
              <li><strong>γ (discount factor):</strong> 0.99</li> <!-- Hiperparámetro 2 -->
              <li><strong>ε inicial:</strong> 1.0</li> <!-- Hiperparámetro 3 -->
              <li><strong>ε mínimo:</strong> 0.01</li> <!-- Hiperparámetro 4 -->
              <li><strong>Decay de ε:</strong> 0.995</li> <!-- Hiperparámetro 5 -->
              <li><strong>Batch size:</strong> 32</li> <!-- Hiperparámetro 6 -->
            </ul>
          </div>

          <button
            type="submit" <!-- Botón enviar -->
            class="btn btn-success btn-lg w-100" <!-- Clases botón éxito grande -->
            id="trainBtn" <!-- ID botón -->
          >
            <i class="fas fa-play me-2"></i> <!-- Icono play -->
            Iniciar Entrenamiento <!-- Texto botón -->
          </button>
```

```
        {% if is_training %} <!-- Si está entrenando -->
        <div class="alert alert-info mt-3" role="alert"> <!-- Alert información -->
          <div class="d-flex align-items-center"> <!-- Flex alineado -->
            <div
              class="spinner-border spinner-border-sm me-2" <!-- Spinner loading -->
              role="status"
            ></div>
            <div>
              <strong>Entrenamiento en progreso...</strong> <!-- Texto progreso -->
              <br />
              <small <!-- Texto pequeño -->
                >Recarga la página en unos minutos para ver los
                resultados.</small
              >
            </div>
          </div>
        </div>
        {% endif %} <!-- Fin condición entrenamiento -->
        {% if training_stats and training_stats.trained %} <!-- Si hay stats entrenamiento -->
        <div class="mt-3"> <!-- Espacio superior -->
          <button
            type="submit" <!-- Botón probar -->
            name="action" <!-- Nombre acción -->
            value="test" <!-- Valor test -->
            class="btn btn-info btn-lg w-100" <!-- Clases botón info -->
          >
            <i class="fas fa-vial me-2"></i> <!-- Icono probeta -->
            Probar Agente Entrenado <!-- Texto botón -->
          </button>
        </div>
```

```
          <div class="mt-3"> <!-- Espacio superior -->
            <button
              type="submit" <!-- Botón reiniciar -->
              name="action" <!-- Nombre acción -->
              value="reset" <!-- Valor reset -->
              class="btn btn-warning btn-lg w-100" <!-- Clases botón warning -->
              onclick="return confirm('¿Estás seguro de reiniciar? Se perderá el modelo entrenado.')" <!-- Confirmación -->
            >
              <i class="fas fa-redo me-2"></i> <!-- Icono reiniciar -->
              Reiniciar Modelo <!-- Texto botón -->
            </button>
          </div>
          {% endif %} <!-- Fin condición stats -->
        </form>

        <!-- Loading indicator -->
        <div
          id="loadingIndicator" <!-- ID indicador -->
          class="alert alert-info mt-3 d-none" <!-- Alert oculto inicialmente -->
          role="alert"
        >
          <div class="d-flex align-items-center"> <!-- Flex alineado -->
            <div class="spinner-border spinner-border-sm me-2" role="status"> <!-- Spinner -->
              <span class="visually-hidden">Entrenando...</span> <!-- Texto accesibilidad -->
            </div>
            <div>
              <strong>Entrenando agente...</strong> <!-- Texto entrenamiento -->
              <br />
              <small>Este proceso puede tardar varios minutos.</small> <!-- Texto pequeño -->
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

```
<!-- Training Statistics -->
{% if training_stats and training_stats.trained %} <!-- Si hay estadísticas -->
<div class="row g-3 mb-4"> <!-- Fila estadísticas -->
  <div class="col-md-3"> <!-- Columna episodios -->
    <div class="card border-0 shadow-sm h-100 card-hover"> <!-- Tarjeta hover -->
      <div class="card-body text-center"> <!-- Cuerpo centrado -->
        <div
          class="bg-primary text-white rounded-circle d-inline-flex align-items-center justify-content-center mb-3" <!-- Círculo azul -->
          style="width: 60px; height: 60px" <!-- Tamaño círculo -->
        >
          <i class="fas fa-layer-group fa-lg"></i> <!-- Icono capas -->
        </div>
        <h6 class="text-muted text-uppercase mb-2">Episodios</h6> <!-- Etiqueta -->
        <h2 class="display-6 fw-bold text-primary mb-0"> <!-- Valor grande -->
          {{ training_stats.n_episodes }} <!-- Número episodios -->
        </h2>
      </div>
    </div>
  </div>

  <div class="col-md-3"> <!-- Columna recompensa final -->
    <div class="card border-0 shadow-sm h-100 card-hover">
      <div class="card-body text-center">
        <div
          class="bg-success text-white rounded-circle d-inline-flex align-items-center justify-content-center mb-3" <!-- Círculo verde -->
          style="width: 60px; height: 60px"
        >
          <i class="fas fa-trophy fa-lg"></i> <!-- Icono trofeo -->
        </div>
        <h6 class="text-muted text-uppercase mb-2">Recompensa Final</h6>
        <h2 class="display-6 fw-bold text-success mb-0"> <!-- Valor éxito -->
          {{ "%.1f"|format(training_stats.final_avg) }} <!-- Recompensa formateada -->
        </h2>
      </div>
    </div>
  </div>
```

```
    <div class="col-md-3"> <!-- Columna mejor recompensa -->
      <div class="card border-0 shadow-sm h-100 card-hover">
        <div class="card-body text-center">
          <div
            class="bg-warning text-white rounded-circle d-inline-flex align-items-center justify-content-center mb-3" <!-- Círculo amarillo -->
            style="width: 60px; height: 60px"
          >
            <i class="fas fa-star fa-lg"></i> <!-- Icono estrella -->
          </div>
          <h6 class="text-muted text-uppercase mb-2">Mejor Recompensa</h6>
          <h2 class="display-6 fw-bold text-warning mb-0"> <!-- Valor warning -->
            {{ "%.0f"|format(training_stats.max_reward) }} <!-- Máxima recompensa -->
          </h2>
        </div>
      </div>
    </div>

    <div class="col-md-3"> <!-- Columna epsilon final -->
      <div class="card border-0 shadow-sm h-100 card-hover">
        <div class="card-body text-center">
          <div
            class="bg-info text-white rounded-circle d-inline-flex align-items-center justify-content-center mb-3" <!-- Círculo info -->
            style="width: 60px; height: 60px"
          >
            <i class="fas fa-search fa-lg"></i> <!-- Icono búsqueda -->
          </div>
          <h6 class="text-muted text-uppercase mb-2">Epsilon Final</h6>
          <h2 class="display-6 fw-bold text-info mb-0"> <!-- Valor info -->
            {{ "%.3f"|format(training_stats.final_epsilon) }} <!-- Epsilon formateado -->
          </h2>
        </div>
      </div>
    </div>
  </div>
```

```
  <!-- Training Progress Chart -->
  <div class="card border-0 shadow-sm mb-4"> <!-- Tarjeta gráfico -->
    <div class="card-header bg-dark text-white border-0"> <!-- Encabezado oscuro -->
      <h5 class="card-title mb-0"> <!-- Título -->
        <i class="fas fa-chart-line me-2"></i> <!-- Icono gráfico -->
        Evolución del Entrenamiento
      </h5>
    </div>
    <div class="card-body text-center"> <!-- Cuerpo centrado -->
      {% if plot_image %} <!-- Si hay imagen gráfico -->
      <img
        src="{{ plot_image }}" <!-- Imagen base64 -->
        alt="Progreso de entrenamiento" <!-- Texto alternativo -->
        class="img-fluid rounded shadow" <!-- Clases imagen -->
        style="max-height: 600px" <!-- Estilo altura máxima -->
      />
      <p class="text-muted mt-3 mb-0"> <!-- Descripción gráfico -->
        <small> <!-- Texto pequeño -->
          Gráfico superior: Recompensa por episodio y promedio móvil (100
          episodios)
          <br /> <!-- Salto línea -->
          Gráfico inferior: Decaimiento de epsilon (exploración → explotación)
        </small>
      </p>
      {% else %} <!-- Si no hay imagen -->
      <p class="text-muted">No hay datos de entrenamiento disponibles</p> <!-- Mensaje vacío -->
      {% endif %} <!-- Fin condición imagen -->
    </div>
  </div>
  {% endif %} <!-- Fin condición estadísticas -->
```

```
  <!-- Test Results -->
  {% if test_results %} <!-- Si hay resultados prueba -->
  <div class="card border-0 shadow-sm mb-4"> <!-- Tarjeta resultados -->
    <div class="card-header bg-info text-white border-0"> <!-- Encabezado info -->
      <h5 class="card-title mb-0"> <!-- Título -->
        <i class="fas fa-vial me-2"></i> <!-- Icono probeta -->
        Resultados de Prueba
      </h5>
    </div>
    <div class="card-body"> <!-- Cuerpo resultados -->
      {% if test_results.error %} <!-- Si hay error -->
      <div class="alert alert-danger" role="alert"> <!-- Alert error -->
        <i class="fas fa-exclamation-triangle me-2"></i> <!-- Icono advertencia -->
        {{ test_results.error }} <!-- Mensaje error -->
      </div>
      {% else %} <!-- Si no hay error -->
      <div class="row"> <!-- Fila métricas -->
        <div class="col-md-4 text-center mb-3"> <!-- Columna promedio -->
          <h6 class="text-muted text-uppercase">Recompensa Promedio</h6> <!-- Etiqueta -->
          <h2 class="display-4 fw-bold text-info"> <!-- Valor grande info -->
            {{ "%.1f"|format(test_results.avg_reward) }} <!-- Promedio formateado -->
          </h2>
        </div>
        <div class="col-md-4 text-center mb-3"> <!-- Columna máximo -->
          <h6 class="text-muted text-uppercase">Máximo</h6>
          <h2 class="display-4 fw-bold text-success"> <!-- Valor grande éxito -->
            {{ "%.0f"|format(test_results.max_reward) }} <!-- Máximo formateado -->
          </h2>
        </div>
        <div class="col-md-4 text-center mb-3"> <!-- Columna mínimo -->
          <h6 class="text-muted text-uppercase">Mínimo</h6>
          <h2 class="display-4 fw-bold text-warning"> <!-- Valor grande warning -->
            {{ "%.0f"|format(test_results.min_reward) }} <!-- Mínimo formateado -->
          </h2>
        </div>
```

```
    <hr class="my-4" /> <!-- Separador -->

    <h6 class="fw-bold mb-3">Recompensas por episodio de prueba:</h6> <!-- Subtítulo tabla -->
    <div class="table-responsive"> <!-- Tabla responsive -->
      <table class="table table-striped table-hover"> <!-- Tabla con estilo -->
        <thead class="table-dark"> <!-- Encabezado tabla oscuro -->
          <tr> <!-- Fila encabezado -->
            <th class="text-center">Episodio</th> <!-- Columna episodio -->
            <th class="text-center">Recompensa</th> <!-- Columna recompensa -->
          </tr>
        </thead>
        <tbody> <!-- Cuerpo tabla -->
          {% for reward in test_results.rewards %} <!-- Iterar recompensas -->
          <tr> <!-- Fila datos -->
            <td class="text-center">{{ loop.index }}</td> <!-- Número episodio -->
            <td class="text-center"> <!-- Celda recompensa -->
              <span
                class="badge {% if reward >= 400 %}bg-success{% elif reward >= 200 %}bg-info{% else %}bg-warning text-dark{% endif %}" <!-- Badge color condicional -->
              >
                {{ "%.0f"|format(reward) }} <!-- Recompensa formateada -->
              </span>
            </td>
          </tr>
          {% endfor %} <!-- Fin iteración -->
        </tbody>
      </table>
    </div>
    {% endif %} <!-- Fin condición error -->
  </div>
</div>
{% endif %} <!-- Fin condición resultados -->
```

```
<!-- Interpretation Guide -->
<div class="card border-0 shadow-sm mb-4"> <!-- Tarjeta guía -->
  <div class="card-header bg-secondary text-white border-0"> <!-- Encabezado secundario -->
    <h5 class="card-title mb-0"> <!-- Título -->
      <i class="fas fa-question-circle me-2"></i> <!-- Icono pregunta -->
      Guía de Interpretación
    </h5>
  </div>
  <div class="card-body"> <!-- Cuerpo guía -->
    <div class="row"> <!-- Fila dos columnas -->
      <div class="col-md-6"> <!-- Columna recompensas -->
        <h6 class="fw-bold">¿Qué significan las recompensas?</h6> <!-- Subtítulo -->
        <ul class="small"> <!-- Lista pequeña -->
          <li> <!-- Item recompensa -->
            <strong>Recompensa = 500:</strong> Rendimiento perfecto (límite
            del entorno)
          </li>
          <li><strong>Recompensa > 400:</strong> Excelente desempeño</li>
          <li><strong>Recompensa > 200:</strong> Buen desempeño</li>
          <li> <!-- Item recompensa baja -->
            <strong>Recompensa < 100:</strong> El agente necesita más
            entrenamiento
          </li>
        </ul>
      </div>
      <div class="col-md-6"> <!-- Columna indicadores -->
        <h6 class="fw-bold">Indicadores de convergencia:</h6> <!-- Subtítulo -->
        <ul class="small"> <!-- Lista pequeña -->
          <li>El promedio móvil se estabiliza en valores altos</li> <!-- Indicador 1 -->
          <li>Epsilon decae hasta el mínimo configurado</li> <!-- Indicador 2 -->
          <li>La varianza entre episodios disminuye</li> <!-- Indicador 3 -->
          <li>El agente mantiene el poste vertical consistentemente</li> <!-- Indicador 4 -->
        </ul>
      </div>
    </div>
  </div>
</div>
```

```
  <!-- Navigation -->
  <div class="d-flex justify-content-between my-4"> <!-- Navegación flex -->
    <a
      href="{{ url_for('refuerzo_conceptos') }}" <!-- Enlace conceptos -->
      class="btn btn-outline-primary d-flex align-items-center" <!-- Botón outline -->
    >
      <i class="fas fa-arrow-left me-2"></i> <!-- Icono flecha izquierda -->
      Conceptos Básicos <!-- Texto botón -->
    </a>
    <a
      href="{{ url_for('index') }}" <!-- Enlace inicio -->
      class="btn btn-primary d-flex align-items-center" <!-- Botón primario -->
    >
      <i class="fas fa-home me-2"></i> <!-- Icono casa -->
      Volver al Inicio <!-- Texto botón -->
    </a>
  </div>
</div>
{% endblock %} <!-- Fin bloque contenido -->
```

```
{% block extra_js %} <!-- Bloque JavaScript adicional -->
<script>
  // Show loading indicator when training starts
  document
    .getElementById("trainingForm") // Obtener formulario -->
    .addEventListener("submit", function (e) { // Escuchar evento submit -->
      const action = e.submitter.value; // Obtener acción del botón -->
      if (action === "train") { // Si acción es entrenar -->
        document.getElementById("loadingIndicator").classList.remove("d-none"); // Mostrar indicador -->
        document.getElementById("trainBtn").disabled = true; // Deshabilitar botón -->
      }
    });

  // Initialize tooltips
  var tooltipTriggerList = [].slice.call( //Inicializar tooltips Bootstrap
    document.querySelectorAll('[data-bs-toggle="tooltip"]') //Seleccionar elementos con tooltip
  );
  var tooltipList = tooltipTriggerList.map(function (tooltipTriggerEl) { //Crear tooltips
    return new bootstrap.Tooltip(tooltipTriggerEl); //Instanciar tooltip
  });
</script>
{% endblock %} <!-- Fin bloque JavaScript -->
```

Propósito: Página interactiva para entrenar y probar un agente de Deep Q-Learning en el entorno CartPole.

Funcionalidades Principales:

- Panel de Información - Explica el entorno CartPole-v1 y sus especificaciones
- Control de Entrenamiento - Permite configurar y ejecutar el entrenamiento
- Estadísticas en Tiempo Real - Muestra métricas de rendimiento del agente
- Visualización de Progreso - Gráficos de recompensas y evolución de epsilon
- Prueba del Modelo - Evalúa el agente entrenado en episodios de prueba
- Guía de Interpretación - Ayuda a entender los resultados

Características Técnicas:

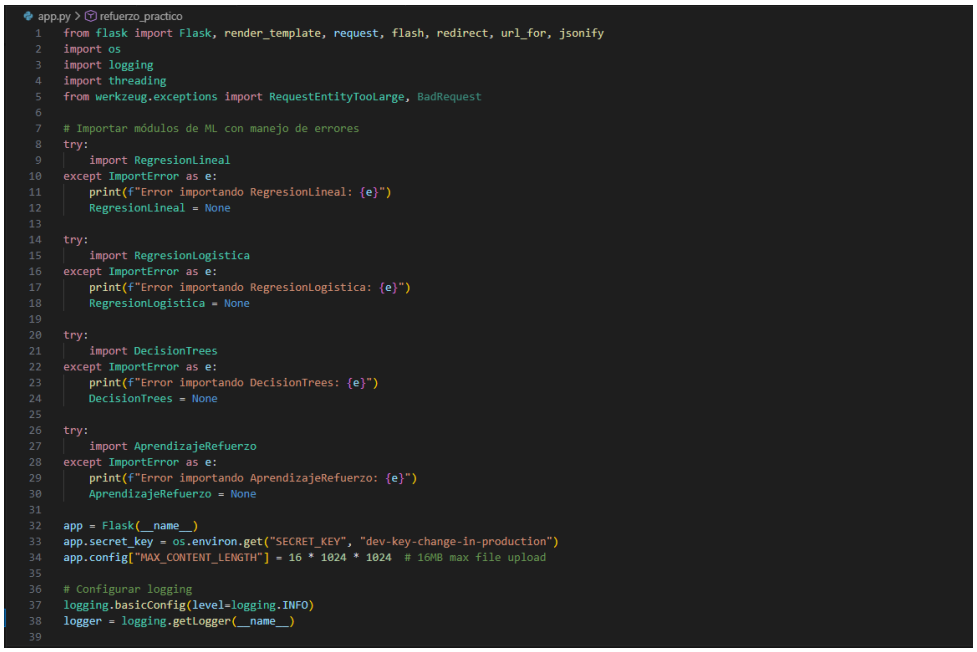
- Formulario interactivo con validación
- Indicadores visuales de progreso
- Tablas responsivas con colores condicionales
- Manejo de estados de entrenamiento
- JavaScript para interactividad
- Mensajes flash para feedback al usuario
- Diseño modular con tarjetas informativas

Integración:

Ambas páginas están completamente integradas con la navegación principal y comparten estilos consistentes con el resto de la aplicación Flask.

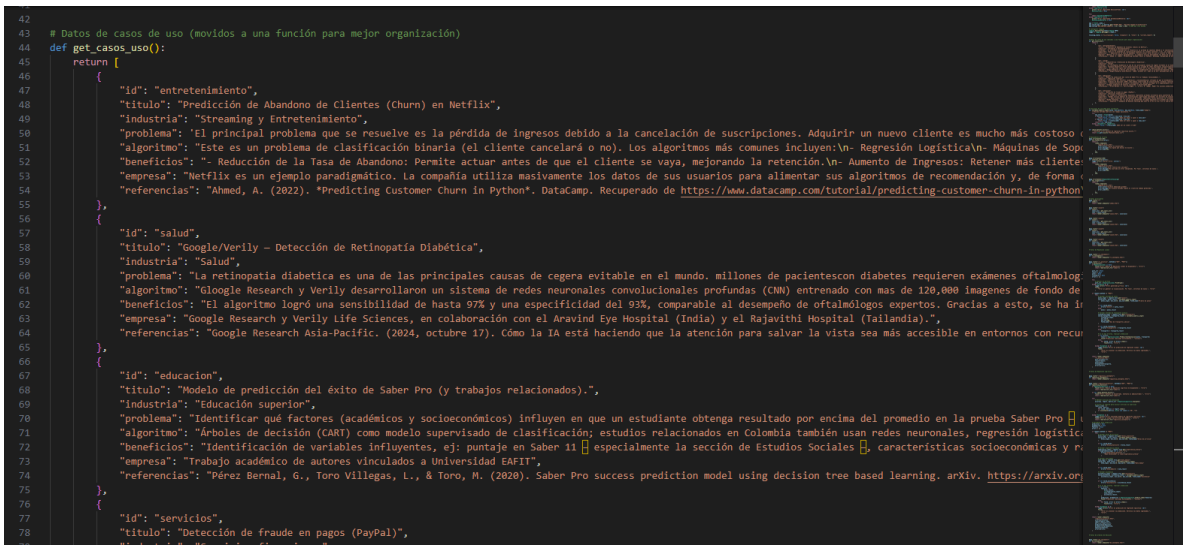
app.py

Esta sección se encarga de importar las librerías necesarias, configurar la aplicación Flask y manejar la importación de los módulos de Machine Learning de manera segura.



```python
from flask import Flask, render_template, request, flash, redirect, url_for, jsonify
import os
import logging
import threading
from werkzeug.exceptions import RequestEntityTooLarge, BadRequest

# Importar módulos de ML con manejo de errores
try:
    import RegresionLineal
except ImportError as e:
    print(f"Error importando RegresionLineal: {e}")
    RegresionLineal = None

try:
    import RegresionLogistica
except ImportError as e:
    print(f"Error importando RegresionLogistica: {e}")
    RegresionLogistica = None

try:
    import DecisionTrees
except ImportError as e:
    print(f"Error importando DecisionTrees: {e}")
    DecisionTrees = None

try:
    import AprendizajeRefuerzo
except ImportError as e:
    print(f"Error importando AprendizajeRefuerzo: {e}")
    AprendizajeRefuerzo = None

app = Flask(__name__)
app.secret_key = os.environ.get("SECRET_KEY", "dev-key-change-in-production")
app.config["MAX_CONTENT_LENGTH"] = 16 * 1024 * 1024  # 16MB max file upload

# Configurar logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

Función get_casos_uso(): Esta función centraliza toda la información descriptiva sobre los casos de uso de Machine Learning que se presentan en la plataforma (Netflix Churn, Retinopatía Diabética, Saber Pro, Fraude PayPal).



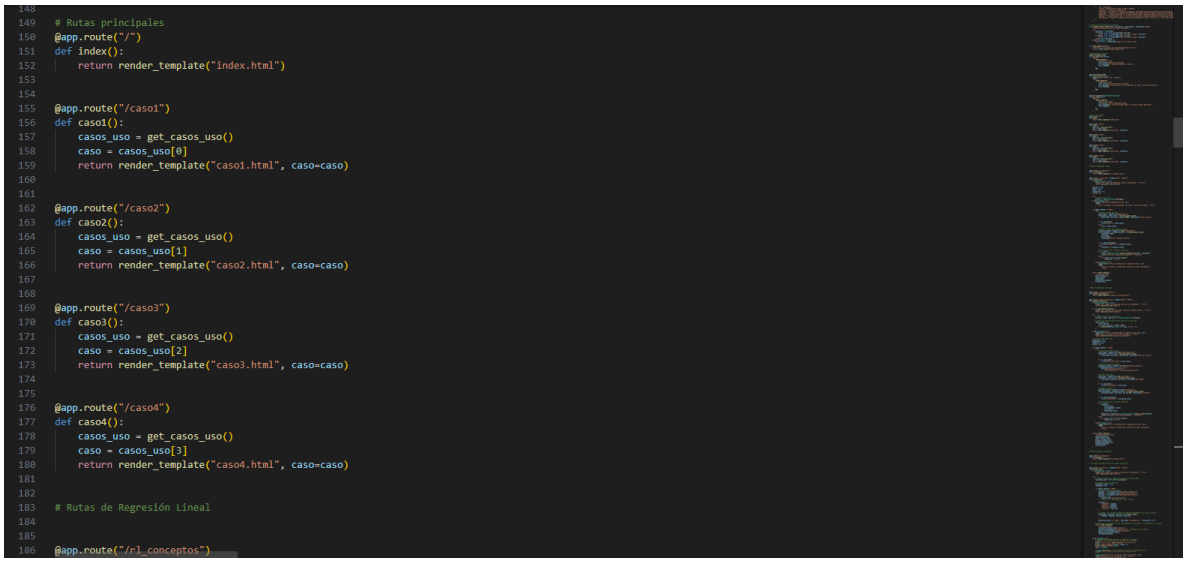
```python
# Datos de casos de uso (movidos a una función para mejor organización)
def get_casos_uso():
    return [
        {
            "id": "entretenimiento",
            "titulo": "Predicción de Abandono de Clientes (Churn) en Netflix",
            "industria": "Streaming y Entretenimiento",
            "problema": 'El principal problema que se resuelve es la pérdida de ingresos debido a la cancelación de suscripciones. Adquirir un nuevo cliente es mucho más costoso (
            "algoritmo": "Este es un problema de clasificación binaria (el cliente cancelará o no). Los algoritmos más comunes incluyen:\n- Regresión Logística\n- Máquinas de Sop
            "beneficios": "- Reducción de la Tasa de Abandono: Permite actuar antes de que el cliente se vaya, mejorando la retención.\n- Aumento de Ingresos: Retener más cliente
            "empresa": "Netflix es un ejemplo paradigmático. La compañía utiliza masivamente los datos de sus usuarios para alimentar sus algoritmos de recomendación y, de forma (
            "referencias": "Ahmed, A. (2022). *Predicting Customer Churn in Python*. DataCamp. Recuperado de https://www.datacamp.com/tutorial/predicting-customer-churn-in-python"
        },
        {
            "id": "salud",
            "titulo": "Google/Verily – Detección de Retinopatía Diabética",
            "industria": "Salud",
            "problema": "La retinopatia diabetica es una de las principales causas de cegera evitable en el mundo. millones de pacientescon diabetes requieren exámenes oftalmolog
            "algoritmo": "Gloogle Research y Verily desarrollaron un sistema de redes neuronales convolucionales profundas (CNN) entrenado con mas de 120,000 imagenes de fondo de
            "beneficios": "El algoritmo logró una sensibilidad de hasta 97% y una especificidad del 93%, comparable al desempeño de oftalmólogos expertos. Gracias a esto, se ha i
            "empresa": "Google Research y Verily Life Sciences, en colaboración con el Aravind Eye Hospital (India) y el Rajavithi Hospital (Tailandia).",
            "referencias": "Google Research Asia-Pacific. (2024, octubre 17). Cómo la IA está haciendo que la atención para salvar la vista sea más accesible en entornos con recu
        },
        {
            "id": "educacion",
            "titulo": "Modelo de predicción del éxito de Saber Pro (y trabajos relacionados).",
            "industria": "Educación superior",
            "problema": "Identificar qué factores (académicos y socioeconómicos) influyen en que un estudiante obtenga resultado por encima del promedio en la prueba Saber Pro u
            "algoritmo": "Árboles de decisión (CART) como modelo supervisado de clasificación; estudios relacionados en Colombia también usan redes neuronales, regresión logística
            "beneficios": "Identificación de variables influyentes, ej: puntaje en Saber 11 especialmente la sección de Estudios Sociales , características socioeconómicas y r
            "empresa": "Trabajo académico de autores vinculados a Universidad EAFIT",
            "referencias": "Pérez Bernal, G., Toro Villegas, L., & Toro, M. (2020). Saber Pro success prediction model using decision tree based learning. arXiv. https://arxiv.org
        },
        {
            "id": "servicios",
            "titulo": "Detección de fraude en pagos (PayPal)",
```

Estas funciones son la columna vertebral para garantizar la calidad de los datos ingresados por el usuario y para proporcionar una experiencia de usuario robusta ante fallos.Validación de Entrada (validate_numeric_input): Es una función fundamental para la seguridad y robustez de la aplicación. Se utiliza en todas las rutas prácticas (Regresión Lineal, Regresión Logística, etc.) para:

1. Asegurar que el valor ingresado sea efectivamente un número.

```
88
89  # Funciones de utilidad para validación
90  def validate_numeric_input(value, min_val=None, max_val=None, field_name="Campo"):
91      """Valida entrada numérica con rangos opcionales."""
92      try:
93          num_value = float(value)
94          if min_val is not None and num_value < min_val:
95              return False, f"{field_name} debe ser mayor o igual a {min_val}"
96          if max_val is not None and num_value > max_val:
97              return False, f"{field_name} debe ser menor o igual a {max_val}"
98          return True, num_value
99      except (ValueError, TypeError):
100         return False, f"{field_name} debe ser un número válido"
101
102
103 def check_dataset_exists():
104     """Verifica si el dataset de regresión logística existe."""
105     return os.path.exists("Datasets/data.csv")
106
107
108 # Manejadores de errores
109 @app.errorhandler(404)
110 def not_found_error(error):
111     return (
112         render_template(
113             "pagina_error.html",
114             error_title="Página no encontrada",
115             error_message="La página que buscas no existe.",
116             error_code=404,
117         ),
118         404,
119     )
120
121
122 @app.errorhandler(500)
123 def internal_error(error):
124     logger.error(f"Server Error: {error}")
125     return (
```

Rutas de *Landing* y Casos de Uso: Las rutas de tipo GET (acceso directo desde el navegador) manejan la carga de la página de inicio (/) y las páginas de los ejemplos prácticos (/caso1 a /caso4), pasando los datos estáticos de get_casos_uso() a las plantillas HTML correspondientes.



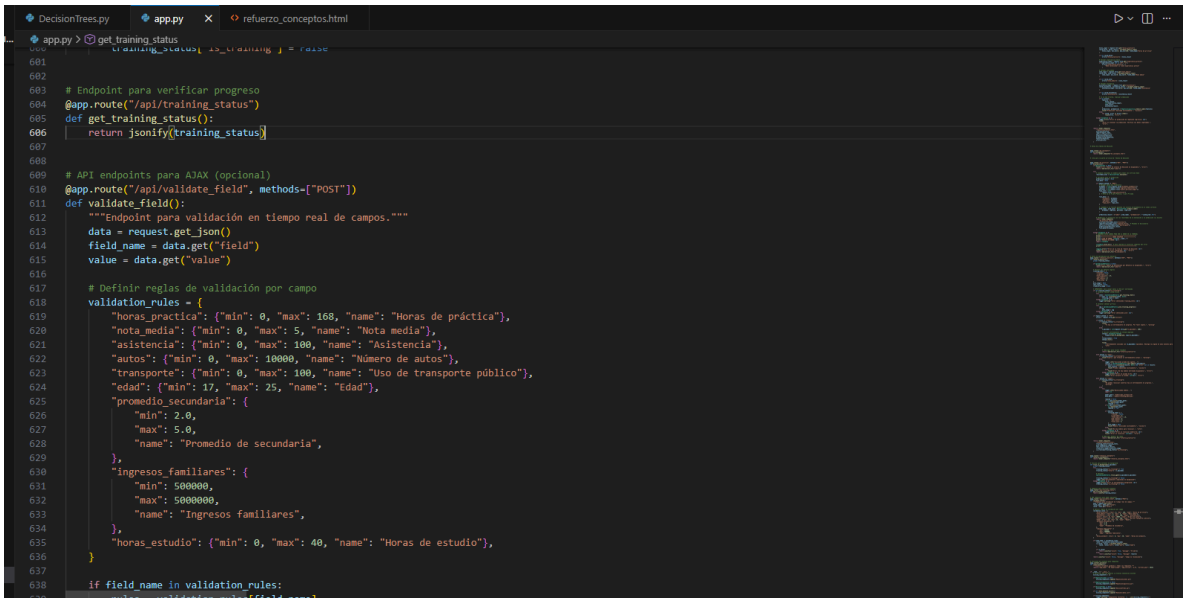
```python
# Rutas principales
@app.route("/")
def index():
    return render_template("index.html")


@app.route("/caso1")
def caso1():
    casos_uso = get_casos_uso()
    caso = casos_uso[0]
    return render_template("caso1.html", caso=caso)


@app.route("/caso2")
def caso2():
    casos_uso = get_casos_uso()
    caso = casos_uso[1]
    return render_template("caso2.html", caso=caso)


@app.route("/caso3")
def caso3():
    casos_uso = get_casos_uso()
    caso = casos_uso[2]
    return render_template("caso3.html", caso=caso)


@app.route("/caso4")
def caso4():
    casos_uso = get_casos_uso()
    caso = casos_uso[3]
    return render_template("caso4.html", caso=caso)


# Rutas de Regresión Lineal


@app.route("/rl_conceptos")
```

Estas son las secciones más complejas, ya que manejan la interacción del usuario con los modelos de Machine Learning (visualización, predicción y entrenamiento).Métodos GET y POST: Todas las rutas prácticas (/rl_practico, /logistica_practico, etc.) usan ambos métodos:

- GET: Carga la página por primera vez, muestra visualizaciones (gráficos, matriz de confusión) y métricas de evaluación iniciales.
- POST: Se ejecuta cuando el usuario envía un formulario para realizar una predicción o iniciar un entrenamiento.

Estas secciones finales facilitan la interacción asíncrona (AJAX) y aseguran que ciertas variables estén disponibles en todas las plantillas.

get_training_status (API): Un endpoint que devuelve el estado del entrenamiento (si está activo, el progreso, etc.) en formato JSON. Esto es ideal para que las páginas web puedan consultar el progreso de forma asíncrona (AJAX) sin necesidad de recargar toda la página.



ruta nueva:

La sección de Aprendizaje por Refuerzo (RL) se distingue de las demás (Regresión Lineal, Logística, Árboles de Decisión) porque no se enfoca en la predicción a partir de datos etiquetados, sino en la toma de decisiones secuenciales por un agente en un entorno dinámico.

```python

@app.route("/refuerzo_conceptos")
def refuerzo_conceptos():
    return render_template("refuerzo_conceptos.html")


# Función para entrenar en background
def train_in_background(n_episodes):
    global training_status
    try:
        training_status["is_training"] = True
        training_status["total"] = n_episodes

        # Entrenar
        AprendizajeRefuerzo.train_agent(n_episodes=n_episodes)

        training_status["is_training"] = False
        logger.info("Entrenamiento completado en background")
    except Exception as e:
        logger.error(f"Error en entrenamiento background: {e}")
        training_status["is_training"] = False


# Endpoint para verificar progreso
@app.route("/api/training_status")
def get_training_status():
    return jsonify(training_status)


# API endpoints para AJAX (opcional)
@app.route("/api/validate_field", methods=["POST"])
def validate_field():
    """Endpoint para validación en tiempo real de campos."""
    data = request.get_json()
    field_name = data.get("field")
```

Propósito Global: Esta es la ruta simple de tipo GET que sirve la página HTML (refuerzo_conceptos.html). Su única función es proporcionar la base teórica necesaria para entender el paradigma RL, que es complejo. Probablemente explica conceptos como:

- Agente y Entorno: El agente que toma decisiones (ej. un coche autónomo) y el entorno donde opera (ej. la carretera).
- Estado, Acción y Recompensa: Los tres elementos centrales de la interacción RL.
- Función de Valor y Política: Cómo el agente aprende a elegir acciones para maximizar la recompensa futura.

script.js

showToast(message, type) (Notificaciones Toast):

- Propósito: Muestra una notificación emergente (Toast), usando el componente de Bootstrap, para informar al usuario sobre el éxito o fracaso de una acción (similar a los flash de Flask en el backend).
- Funcionamiento: Localiza el elemento Toast, actualiza su mensaje (toastMessage.textContent) y modifica su clase CSS (text-bg-${type}) para cambiar el color o estilo (ej., 'success', 'danger', 'primary').

- validateForm(formId):
    - Propósito: Verifica que todos los campos marcados como required (obligatorios) en un formulario tengan un valor antes de permitir el envío.
    - Funcionamiento:
        1. Selecciona todos los elementos (input, select, textarea) que tienen el atributo required.
        2. Itera sobre ellos, eliminando cualquier estado de error previo.
        3. Si un campo está vacío (!input.value.trim()), lo marca como inválido (is-invalid) y crea dinámicamente un mensaje de error ( ) debajo del campo, adhiriéndose al estándar de estilos de Bootstrap.
        4. Retorna true solo si todos los campos requeridos han sido llenados. Esta función debe ser llamada en el evento submit del formulario para detener el envío si retorna false.

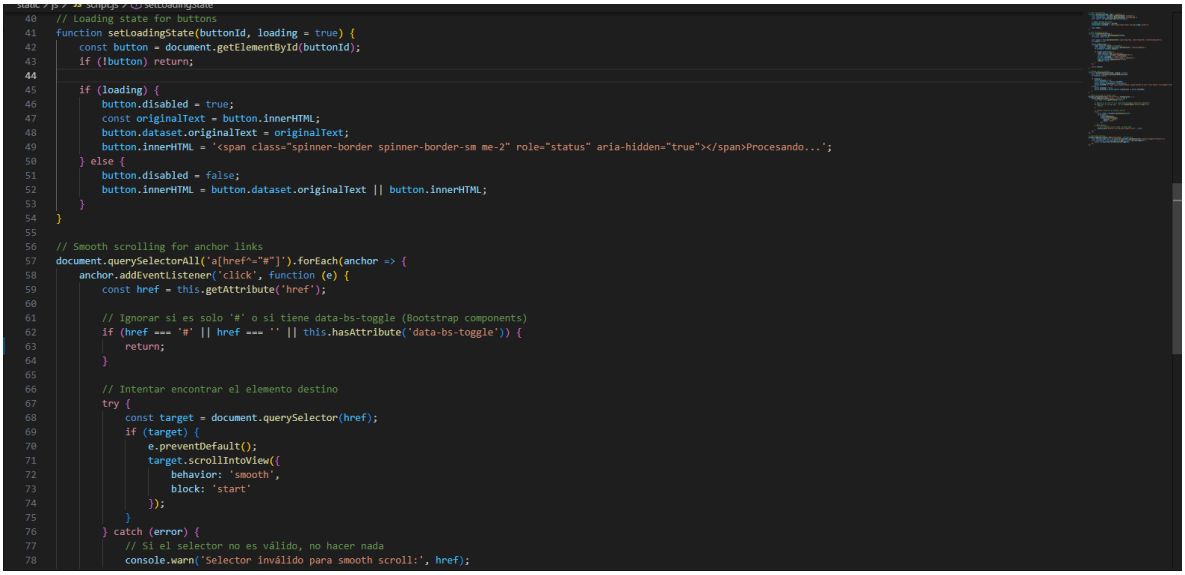
```
// Form validation helper
function validateForm(formId) {
    const form = document.getElementById(formId);
    if (!form) return false;

    const inputs = form.querySelectorAll('input[required], select[required], textarea[required]');
    let isValid = true;

    inputs.forEach(input => {
        input.classList.remove('is-invalid');
        const feedback = input.parentNode.querySelector('.invalid-feedback');
        if (feedback) feedback.remove();

        if (!input.value.trim()) {
            input.classList.add('is-invalid');
            const errorDiv = document.createElement('div');
            errorDiv.className = 'invalid-feedback';
            errorDiv.textContent = 'Este campo es requerido.';
            input.parentNode.appendChild(errorDiv);
            isValid = false;
        }
    });

    return isValid;
}
```

Desplazamiento Suave (Smooth Scrolling):

- Propósito: Reemplaza el salto abrupto a una sección al hacer clic en un ancla (<a href="#section-id">) por un desplazamiento animado y suave.
- Funcionamiento: Escucha el evento click en todos los enlaces cuyo href comienza con #. Si el enlace apunta a un elemento válido, previene la acción por defecto y usa el método nativo de JavaScript scrollIntoView con la opción behavior: 'smooth'.

Inicialización de Tooltips:

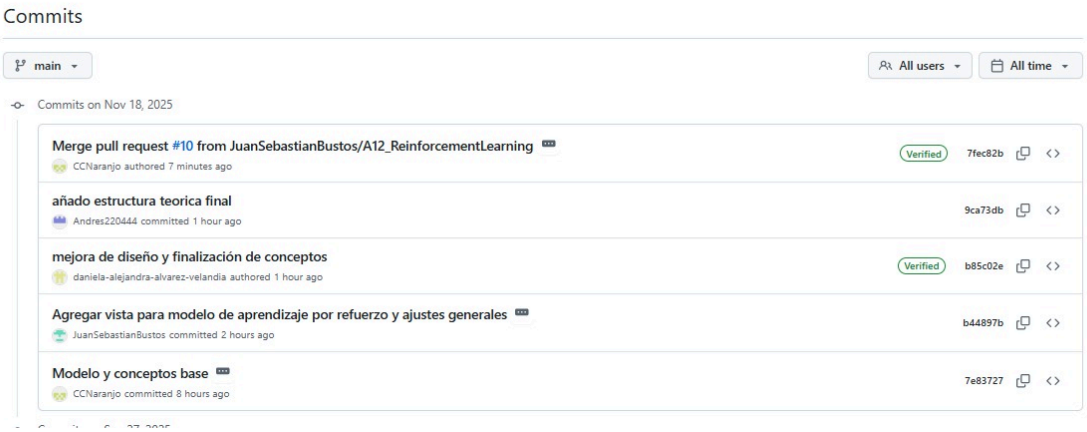
- Propósito: Activa el componente de Bootstrap Tooltip (mensajes de ayuda flotantes) en todos los elementos que tienen el atributo data-bs-toggle="tooltip".
- Funcionamiento: Se asegura de que los tooltips funcionen correctamente inicializándolos solo después de que todo el Document Object Model (DOM) de la página esté completamente cargado (DOMContentLoaded).

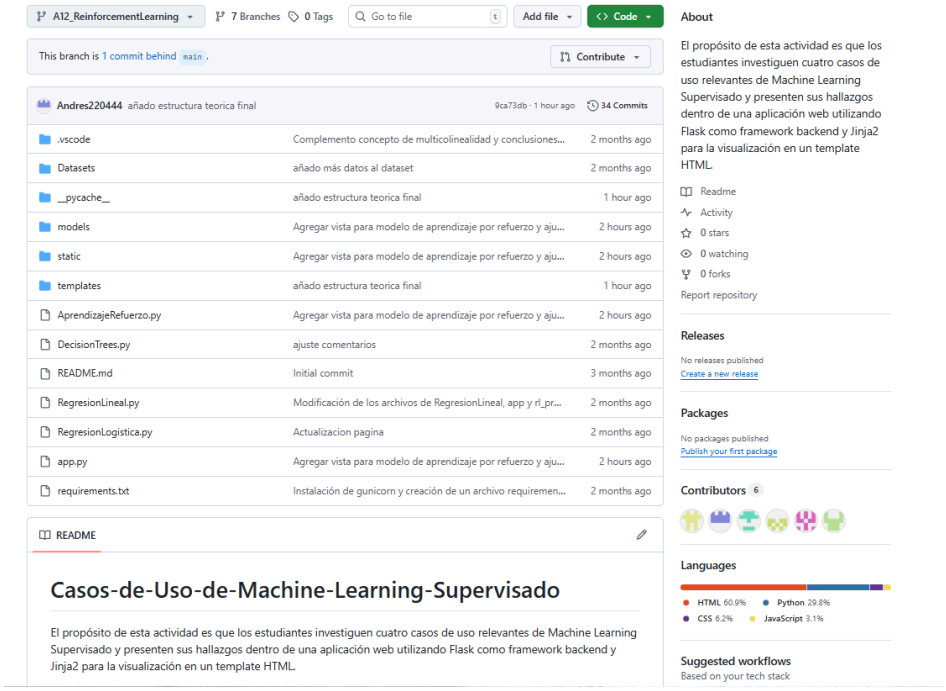


```
// Loading state for buttons
function setLoadingState(buttonId, loading = true) {
    const button = document.getElementById(buttonId);
    if (!button) return;

    if (loading) {
        button.disabled = true;
        const originalText = button.innerHTML;
        button.dataset.originalText = originalText;
        button.innerHTML = '<span class="spinner-border spinner-border-sm me-2" role="status" aria-hidden="true"></span>Procesando...';
    } else {
        button.disabled = false;
        button.innerHTML = button.dataset.originalText || button.innerHTML;
    }
}

// Smooth scrolling for anchor links
document.querySelectorAll('a[href^="#"]').forEach(anchor => {
    anchor.addEventListener('click', function (e) {
        const href = this.getAttribute('href');

        // Ignorar si es solo '#' o si tiene data-bs-toggle (Bootstrap components)
        if (href === '#' || href === '' || this.hasAttribute('data-bs-toggle')) {
            return;
        }

        // Intentar encontrar el elemento destino
        try {
            const target = document.querySelector(href);
            if (target) {
                e.preventDefault();
                target.scrollIntoView({
                    behavior: 'smooth',
                    block: 'start'
                });
            }
        } catch (error) {
            // Si el selector no es válido, no hacer nada
            console.warn('Selector inválido para smooth scroll:', href);
```

GITHUB

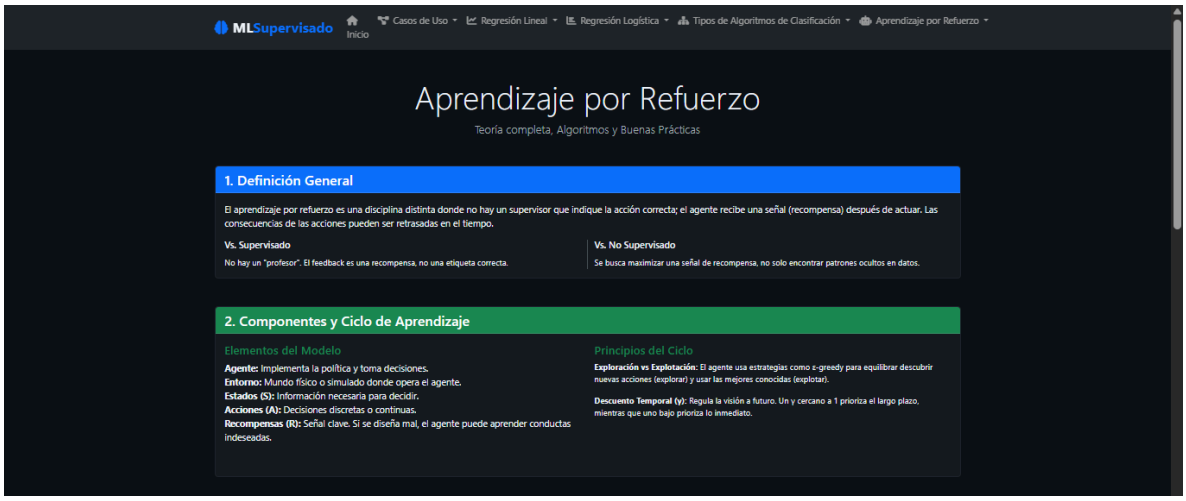
https://github.com/JuanSebastianBustos/Arboles-de-Desicion.git

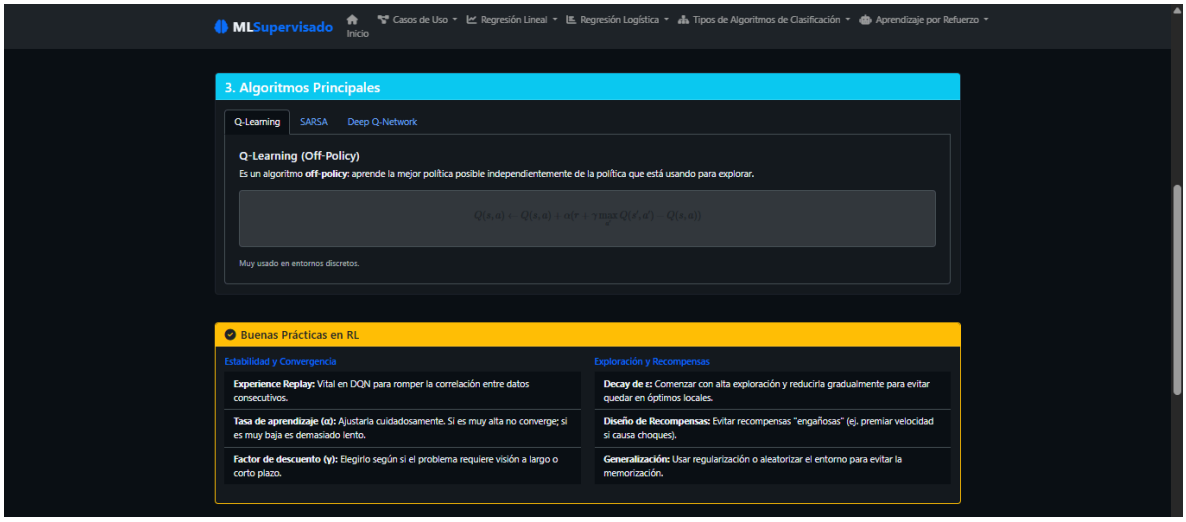


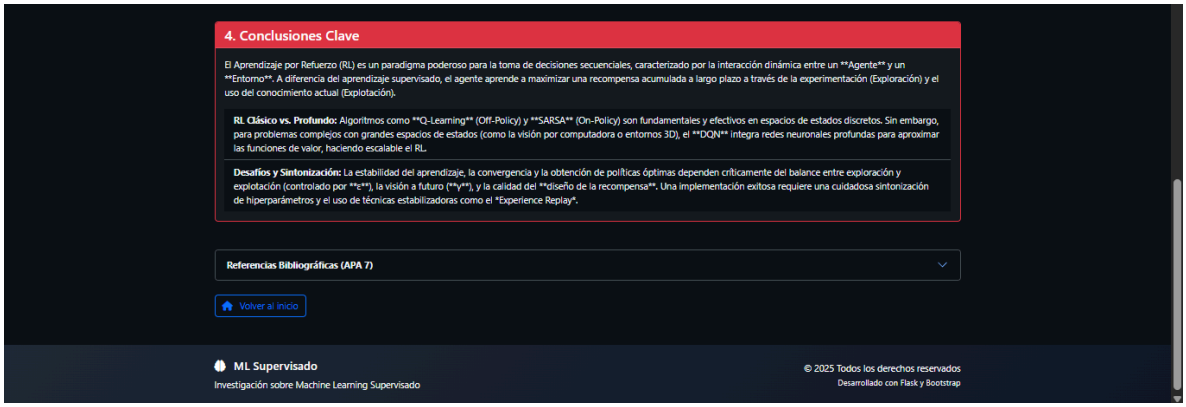


Página

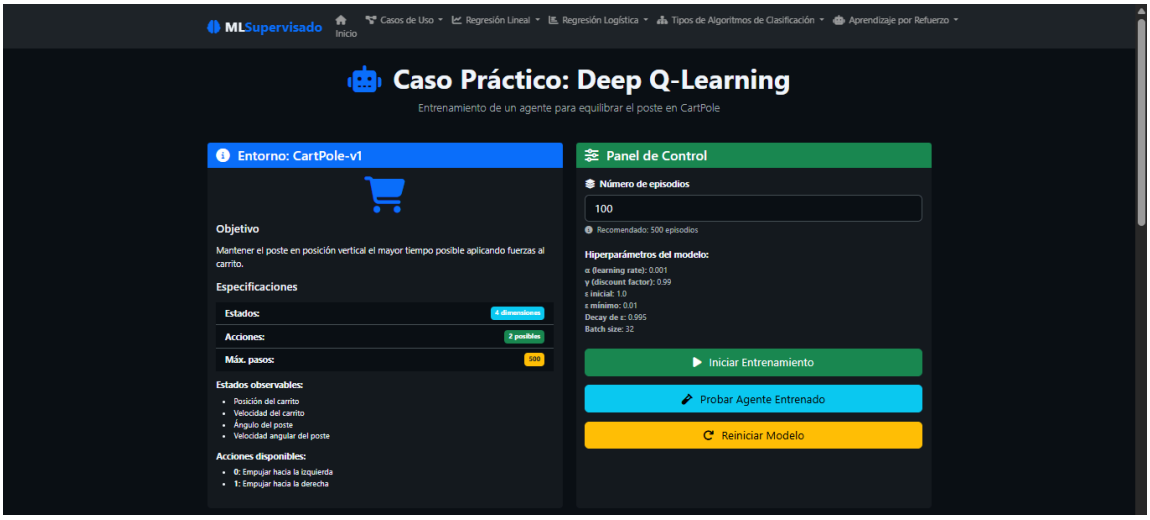
Vista de Conceptos

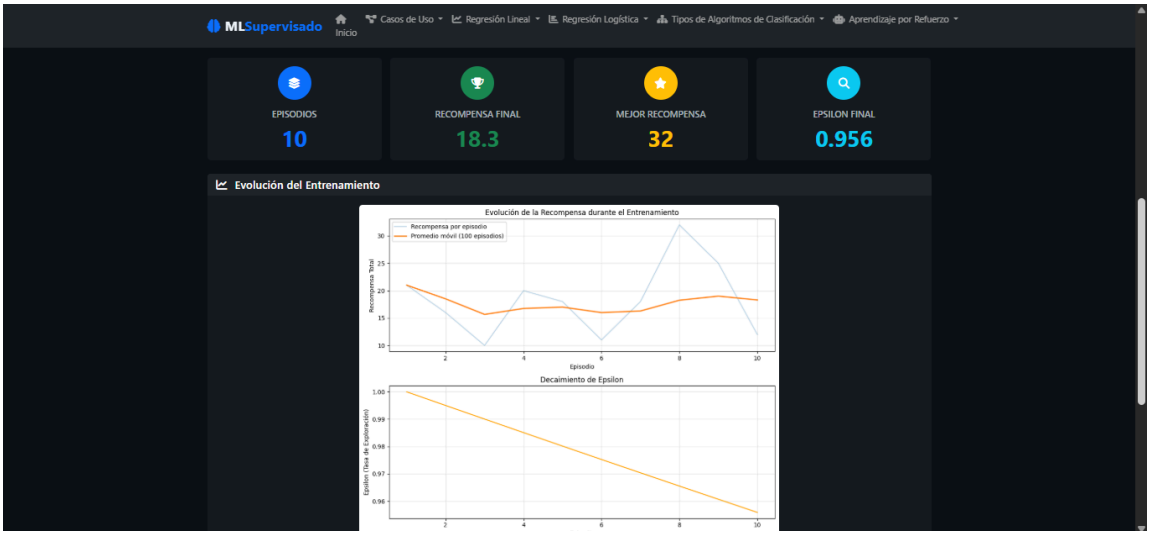


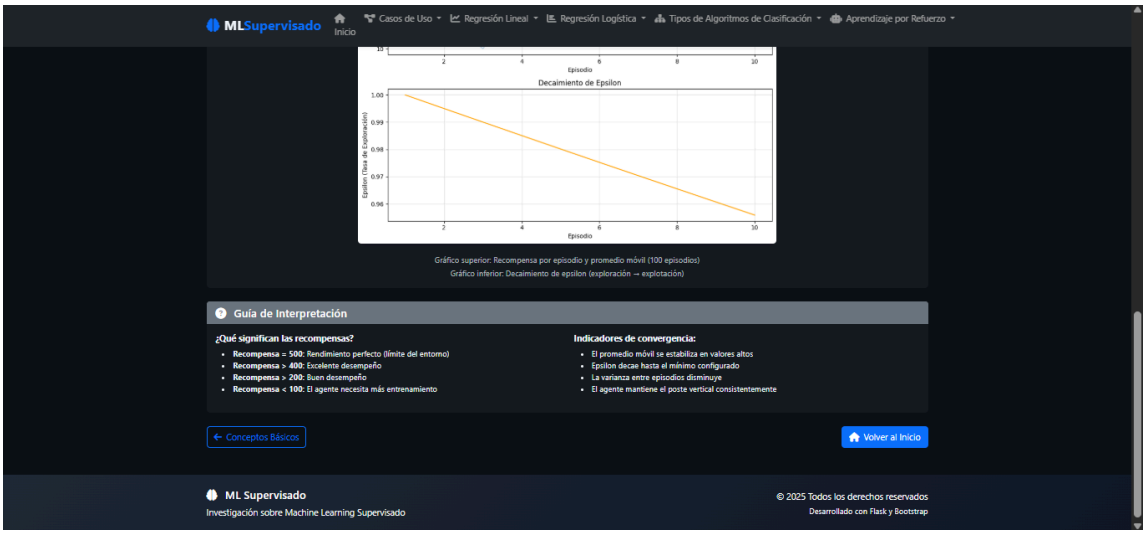




Vista de Caso Práctico







Investigación

Definición general y diferencias con otros aprendizajes

- El aprendizaje por refuerzo es una disciplina distinta dentro del aprendizaje automático. Según la FUOC, no hay un supervisor que indique “esta acción es la correcta”: el agente recibe una señal (recompensa) después de actuar, pero no una etiqueta directa.
- Además, como dice la FUOC, las consecuencias de las acciones pueden ser retrasadas en el tiempo: una acción puede tener un resultado (recompensa) mucho después de ser tomada.
- Comparado con el aprendizaje supervisado, no hay un “profesor” que indique la respuesta correcta. Y en comparación con lo no supervisado, no solo se busca patrones en datos: se trata de tomar decisiones para maximizar una señal (la recompensa).

Componentes del modelo RL (más detalle)

1. Agente: Como persona, robot o programa. El agente implementa una política (π) que decide acciones según el estado actual.
2. Entorno: Es el “mundo” donde opera el agente. Puede ser físico (robot) o simulado (juego, simulador).
3. Estados (S): Representan toda la información necesaria para decidir. Por ejemplo, en un juego de Atari, el estado podría ser la imagen de la pantalla.
4. Acciones (A): Decisiones que el agente puede realizar. En espacios discretos, hay un número limitado; en espacios continuos, pueden ser valores reales (por ejemplo, velocidad de un robot).
5. Recompensas (R): Señales numéricas que el agente recibe para indicar si su acción fue “buena” o “mala”. El diseño de recompensas es clave: recompensas mal diseñadas pueden provocar comportamientos indeseables (por ejemplo, que el agente “engañe” el sistema de recompensa).
6. Política (π): Función (determinista o probabilística) que asigna a cada estado una acción, o una distribución de acciones.
7. Función de valor:
    - $V(s)$: valor esperado del estado s bajo cierta política, mide “qué tan bueno es estar en s”.
    - $Q(s, a)$: valor esperado de tomar la acción a en el estado s, y luego seguir la política.

Principios del ciclo de aprendizaje

- Exploración vs explotación:
  El agente debe explorar para descubrir acciones nuevas que podrían dar recompensas mayores, pero también debe explotar lo que ya ha aprendido para maximizar recompensa. Estrategias como ε-greedy son muy comunes: con probabilidad ε el agente explora (elige acción al azar), con probabilidad $1 - \varepsilon$elige la acción “mejor conocida”.

- Retorno acumulado:
  El agente no solo mira la recompensa inmediata; su objetivo es maximizar el retorno total esperado, típicamente modelado como

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

donde γ(factor de descuento) regula cuánto valor se le da al futuro.

- Descuento temporal (γ):
  - Si γestá cerca de 1 → el agente valora mucho las recompensas futuras (planea a largo plazo).
  - Si γes bajo (por ejemplo, 0.1) → se enfoca más en recompensas inmediatas.

Algoritmos principales (más detalle)

1. Q-Learning
   - Es *off-policy*: el agente aprende la mejor política independientemente de la política con la que actúe.
   - Fórmula de actualización típica del Q:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'} Q(s', a') - Q(s,a))$$

Esta fórmula aparece, por ejemplo, en los apuntes de UNISON.

   - Es muy usado en entornos discretos.

2. SARSA
   - Es *on-policy*, lo que significa que el agente actualiza sus valores Q usando la acción que realmente selecciona su política actual (no la acción óptima hipotética).

- Fórmula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Explicado en las notas de Ignacio Gavilán.

- Es más “conservador”: puede llevar a políticas más “seguras” porque incorpora la exploración dentro del proceso de actualización.

3. Deep Q-Network (DQN)
    - Cuando el espacio de estados o acciones es muy grande (por ejemplo imágenes), no es práctico usar una tabla Q. DQN usa redes neuronales profundas para aproximar $Q(s, a)$.
    - Incluye técnicas como *Experience Replay* (almacenar experiencias para entrenar en “lotes”) y *Target Network* (una red separada para calcular el valor objetivo) que mejoran la estabilidad del aprendizaje.
    - Aplicaciones típicas: videojuegos (Atari), control robótico, sistemas con muchos estados posibles.

Buenas prácticas en Aprendizaje por Refuerzo (más detalle)

- Estabilidad del aprendizaje:
    - En DQN, usar *experience replay* para romper la correlación entre experiencias consecutivas.
    - Usar una *target network* para calcular los valores a largo plazo, actualizándola cada cierto número de pasos.
- Tasa de exploración:
    - Usar un decay de ε: comenzar con un valor alto (muchas exploraciones) y reducirlo conforme el agente aprende.
    - Evitar que ε sea demasiado bajo desde el principio para no quedar atrapado en una política subóptima por falta de exploración.
- Diseño de recompensas:
    - Evitar “recompensas engañosas” que hacen que el agente aprenda comportamientos no deseados (por ejemplo, recompensar “ir rápido” si eso hace que choque repetidamente).
    - Dar recompensas intermedias si la tarea es compleja, para guiar al agente.

- Convergencia:
    - Ajustar la tasa de aprendizaje (α): si es muy alta, el agente puede “saltar” demasiado y no converger bien; si es muy baja, el aprendizaje será muy lento.
    - Elegir un buen γ (factor de descuento) según la naturaleza del problema.
- Generalización:
    - Evaluar al agente en entornos distintos a los de entrenamiento para comprobar que no solo “memoriza” la solución.
    - En Deep RL, usar técnicas como *regularización*, *dropout* o *domain randomization* (aleatorizar aspectos del entorno durante el entrenamiento) para favorecer que el agente generalice bien.

Conclusiones

El Aprendizaje por Refuerzo (RL) es un paradigma de toma de decisiones secuenciales donde un Agente aprende a maximizar la recompensa acumulada a largo plazo a través de la interacción con el Entorno. Esto requiere un balance crítico entre Exploración (descubrir nuevas acciones) y Explotación (usar las mejores conocidas).

Algoritmos y Escalabilidad

- Q-Learning es off-policy: aprende la política óptima independientemente de la exploración.
- SARSA es on-policy: actualiza valores usando la acción que realmente selecciona su política actual, siendo más conservador.
- Para espacios de estados grandes (como imágenes), DQN utiliza redes neuronales profundas para aproximar la función $Q(s,a)$.

Estabilidad y Desafíos Prácticos

La implementación exitosa depende de la cuidadosa sintonización de hiperparámetros y el manejo de la estabilidad:

- Estabilidad: El uso de Experience Replay y la Target Network en DQN es vital para romper la correlación de datos y mejorar la estabilidad.
- Diseño de Recompensas: Las recompensas deben ser diseñadas para evitar conductas indeseadas o "engañosas" por parte del agente.
- Convergencia: La tasa de aprendizaje (α) y el factor de descuento (γ) son cruciales para la estabilidad y la visión a futuro, respectivamente.
- Óptimos Locales: Reducir gradualmente la exploración (Decay de ϵ) ayuda a evitar quedar atrapado en soluciones subóptimas.

Referencias

- **FUOC – Introducción al aprendizaje por refuerzo**
  Documento PDF con explicación teórica muy clara sobre RL, MDP, agentes, estados, recompensas, etc.
  Repositorio de Acceso Abierto UOC

- **Ignacio G.R. Gavilán – Notas sobre aprendizaje por refuerzo: SARSA y Q-Learning**
  Blog técnico con fórmulas, explicación de la diferencia entre Q-Learning y SARSA, ejemplos.
  Ignacio G.R. Gavilán

- **Tópicos Avanzados de IA – Aprendizaje por Refuerzo (UNISON)**
  Apuntes de un curso universitario que explican Q-Learning, el ciclo de aprendizaje, exploración/explotación, etc.rexemin.github.io

- **Notus.cl – Qué es el Reinforcement Learning y cuáles son sus aplicaciones**
  Artículo conceptual, bien explicado y actualizado sobre RL, sus algoritmos y aplicaciones prácticas.
  Notus

- **Fundación Bankinter – ¿Qué es Q-Learning?**
  Explicación de Q-learning, cómo funciona, sus parámetros importantes, limitaciones.
  fundacionbankinter.org

- **FlowHunt – Glosario RL: Aprendizaje por Refuerzo**
  Define los componentes fundamentales del RL (agente, entorno, estado, acción, recompensa), y explica brevemente algoritmos Q-Learning, SARSA y DQN.
  flowhunt.io+1

- **Universidad de Zaragoza / UNIZAR – Tesis sobre modelos de Aprendizaje por Refuerzo**
  En esta tesis se explican Q-Learning, SARSA, DQN, y se discuten aspectos teóricos como convergencia.
  Zaguán