

Sesión de Aprendizaje N° 15-A

Desarrollo Web

Sistemas Inteligentes

Python / Flask / Django

1. Definición, Alcances y Ámbitos de Aplicación

Los **sistemas inteligentes** son entidades computacionales capaces de percibir su entorno, procesar información, razonar y ejecutar acciones de forma autónoma, con el objetivo de alcanzar metas definidas.

Alcances funcionales:

- **Percepción sensorial:** Captura de datos mediante sensores o fuentes digitales.
- **Razonamiento simbólico:** Inferencia basada en reglas lógicas y conocimiento previo.
- **Aprendizaje adaptativo:** Capacidad de mejorar su desempeño mediante experiencia.

Ámbitos de aplicación:

- Automatización industrial
- Robótica autónoma
- Diagnóstico médico asistido por IA
- Vehículos inteligentes (conducción autónoma)
- Finanzas y mercados predictivos
- Educación personalizada
- Sistemas expertos en atención al cliente

Ejemplos de Sistemas Inteligentes:

Salud

- **Watson Health de IBM:** Analiza historiales médicos y literatura científica para apoyar diagnósticos y tratamientos personalizados.
- **Sistemas de diagnóstico por imagen:** Algoritmos de deep learning que detectan tumores, fracturas o anomalías en radiografías y resonancias.

Automoción

- **Vehículos autónomos (Tesla, Waymo):** Utilizan sensores, redes neuronales y aprendizaje profundo para navegar sin intervención humana.
- **Sistemas ADAS (Advanced Driver Assistance Systems):** Frenado automático, mantenimiento de carril y detección de peatones.

Industria 4.0

- **Robots colaborativos (cobots):** Trabajan junto a humanos en líneas de producción, adaptándose a tareas variables.
- **Sistemas predictivos de mantenimiento:** Analizan datos de sensores para anticipar fallos en maquinaria.

Retail y logística

- **Centros logísticos de Amazon:** Robots inteligentes gestionan inventario, optimizan rutas y preparan pedidos.
- **Sistemas de recomendación (Netflix, Spotify, Yachay):** Aprenden de tus preferencias para sugerir contenido relevante.

Hogar inteligente

- **Asistentes virtuales (Alexa, Google Assistant):** Procesan lenguaje natural para controlar dispositivos, responder preguntas y automatizar rutinas.
- **Termostatos inteligentes (Nest):** Aprenden hábitos del usuario para optimizar el consumo energético.

Ciudades inteligentes

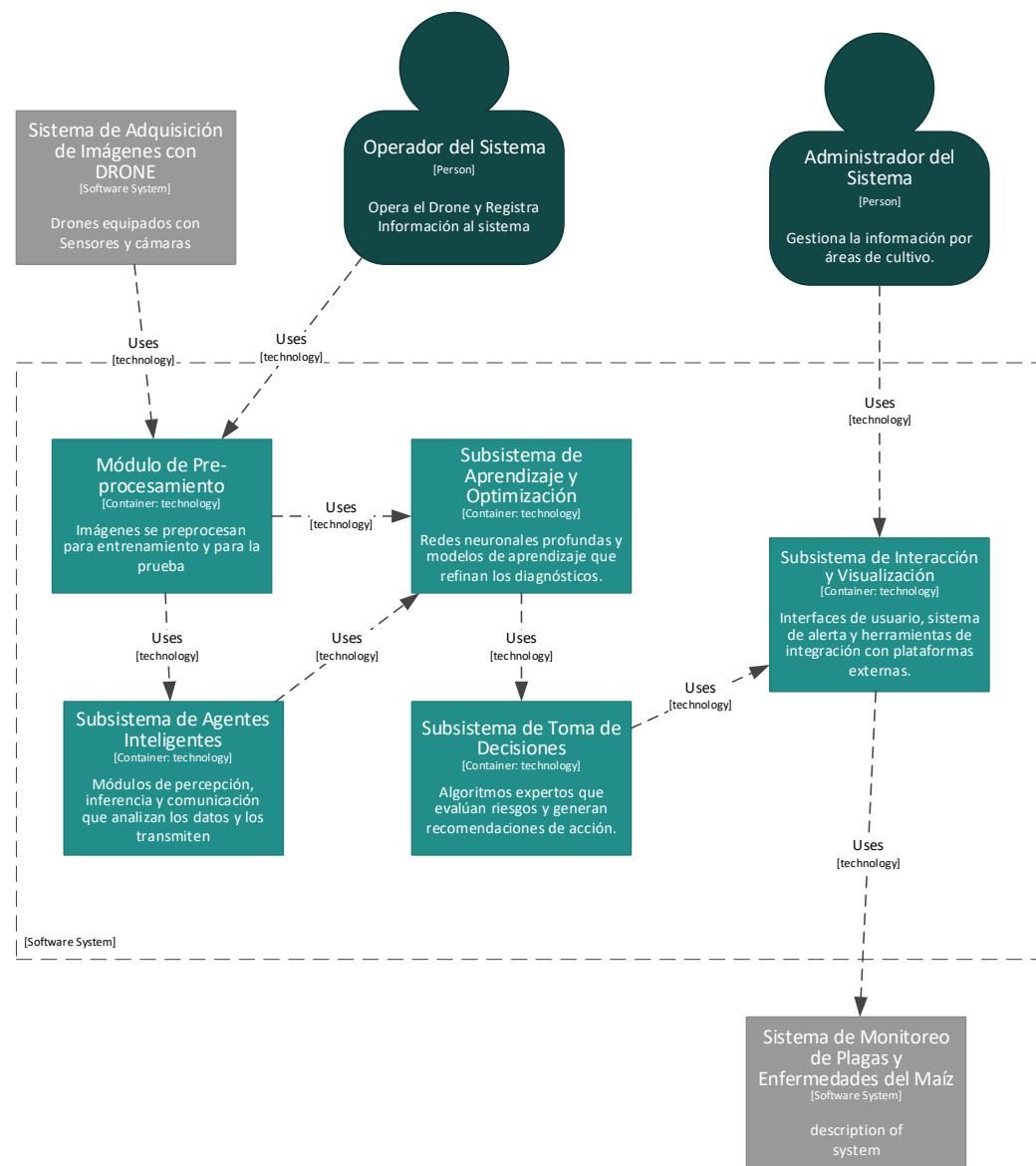
- **Semáforos adaptativos:** Ajustan tiempos según el flujo vehicular en tiempo real.
- **Sistemas de videovigilancia con reconocimiento facial:** Detectan comportamientos sospechosos y alertan a autoridades.

2. Arquitectura de los Sistemas Inteligentes

La arquitectura define la estructura y la interacción de los componentes del sistema, permitiendo una integración coherente de sus capacidades cognitivas.

Componentes principales:

- **Módulo de percepción:** Interfaces con sensores y fuentes de datos, traduciendo estímulos externos en representaciones internas.
- **Módulo de conocimiento:** Base de datos estructurada, ontologías y redes semánticas.
- **Módulo de razonamiento:** Algoritmos de inferencia lógica, toma de decisiones y planificación.
- **Módulo de aprendizaje:** Integración de técnicas de machine learning y adaptación al entorno.
- **Módulo de acción:** Control de actuadores físicos o ejecución de comandos virtuales.
- **Interfaz hombre-máquina:** Canales de interacción cognitiva y comunicacional con usuarios.



3. Redes Neuronales Artificiales

Las redes neuronales constituyen modelos computacionales inspirados en la estructura del cerebro humano, diseñados para el procesamiento paralelo y adaptativo de información.

Características clave:

- Compuestas por nodos (neuronas) y conexiones (sinapsis) organizadas en capas: de entrada, ocultas y de salida.
- Aprenden a través de ajustes en los pesos sinápticos mediante algoritmos como retropropagación.
- Aplicables en clasificación, regresión, segmentación y predicción.

Tipos comunes:

- Perceptrón multicapa (MLP)
- Redes de Kohonen (mapas autoorganizados)
- Redes de Hopfield
- Redes convolucionales (CNNs)
- Redes recurrentes (RNNs y LSTMs)

4. Aprendizaje Profundo (Deep Learning)

El **deep learning** es una *subdisciplina del aprendizaje automático que emplea arquitecturas de redes neuronales profundas con múltiples capas ocultas* para representar funciones de alta complejidad.

Aspectos fundamentales:

- Extracción automática de características: A partir de datos sin necesidad de ingeniería manual.
- Capacidad de escalabilidad: Procesamiento eficiente de grandes volúmenes de datos (big data).
- Backpropagation y optimización: Uso de algoritmos como Adam, RMSprop, SGD.

Principales arquitecturas:

- CNNs para visión artificial
- RNNs y LSTMs para secuencias temporales y procesamiento de lenguaje
- Transformers para modelos generativos y entendimiento semántico (ej. GPT, BERT)

5. Algoritmos de Toma de Decisiones

Son algoritmos que permiten a los sistemas inteligentes seleccionar acciones óptimas en función de objetivos, restricciones y conocimiento del entorno.

Tipos principales:

- **Sistemas basados en reglas:** IF-THEN estructurados, ideales en entornos controlados.
- **Redes Bayesianas:** Probabilísticas, útiles en contextos con incertidumbre.
- **Lógica difusa:** Manejo de imprecisión con variables lingüísticas.
- **Árboles de decisión:** Clasificación y análisis de alternativas.
- **Algoritmos heurísticos:** Búsqueda de soluciones eficientes ante problemas complejos.
- **Algoritmos de planificación:** STRIPS, A*, planificación temporal y secuencial.

6. Clasificación de Modelos en Deep Learning

6.1. Modelos Discriminativos

Son algoritmos diseñados para **clasificar** datos de entrada en categorías. En términos estadísticos, modelan la **probabilidad condicional** $P(y|x)$, es decir, la probabilidad de una etiqueta y dada una observación x. Su objetivo principal es **diferenciar clases** mediante fronteras de decisión.

Características:

- No generan datos nuevos
- Se centran en la separación entre clases
- Requieren datos etiquetados
- Usan funciones de pérdida discriminativa

Ejemplos:

- Regresión logística
- SVM (Support Vector Machines)
- Árboles de decisión
- Redes neuronales clásicas (MLP)
- Modelos BERT y variantes en NLP

6.2. Modelos Generativos

Aprenden la **distribución conjunta** $P(x, y)$ o solo $P(x)$, lo que les permite **generar nuevas muestras de datos**. Son utilizados tanto para clasificación como para **síntesis de contenido**, simulaciones, y detección de anomalías.

Características:

- Pueden reconstruir o simular datos similares a los reales
- Capturan relaciones profundas entre variables
- Se aplican en dominios como visión, texto, audio, video

Ejemplos:

- Naive Bayes
- Hidden Markov Models
- GANs (Generative Adversarial Networks)
- Autoencoders
- VAEs (Variational Autoencoders)

6.3. Large Language Models (LLM)

Los LLM son modelos generativos avanzados especializados en **lenguaje natural**, entrenados con arquitecturas de tipo **Transformer** sobre grandes volúmenes de texto. **Modelan la probabilidad $P(\text{texto}|\text{contexto})$** y permiten **generar texto coherente**, traducir, responder preguntas, redactar código y más.

Características:

- Generan texto nuevo basado en contexto
- Usan atención y aprendizaje profundo (Transformers)
- Se entrenan en corpus masivos (billones de palabras)
- Capaces de adaptarse a tareas discriminativas mediante **fine-tuning**

Ejemplos:

- GPT (OpenAI)
- Gemini (Google)
- Claude (Anthropic)
- LLaMA (Meta)
- Mistral

Segunda Unidad: Desarrollo Backend
GUÍA DE LABORATORIO N° 15-1:
DESARROLLO SISTEMAS INTELIGENTES EN PYTHON – FLASK - DJANGO

Docente: Jaime Suasnabar Terrel

Fecha:

Duración: 90 minutos

Instrucciones:

OBJETIVOS

- Desarrollar sistemas inteligentes en Python aplicando algoritmos de toma de decisiones tipo IF-THEN estructurados.
- Familiarizarse con la creación de clases, herencia, y lógica condicional para emular sistemas expertos simples.

EJERCICIO 01. CREACIÓN DE ALGORITMOS DE TOMA DE DECISIONES

1. **Crear una carpeta y su archivo principal**

```
mkdir diagnostico-salud  
cd diagnostico-salud  
touch main.py
```

2. **Crear la clase base: Persona**

```
class Persona:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad  
  
    def presentarse(self):  
        return f"Hola, soy {self.nombre} y tengo {self.edad} años."
```

3. **Clase derivada: Paciente (hereda de Persona)**

```
class Paciente(Persona):  
    def __init__(self, nombre, edad, sintomas):  
        super().__init__(nombre, edad)  
        self.sintomas = sintomas  
  
    def mostrar_sintomas(self):  
        return f"Síntomas reportados: {', '.join(self.sintomas)}"
```

4. **Crear el algoritmo de decisión basado en reglas IF-THEN**

```
def diagnosticar(sintomas):  
    if "fiebre" in sintomas and "tos" in sintomas and "dolor de garganta" in sintomas:  
        return "Diagnóstico: Posible gripe común"  
    elif "fiebre" in sintomas and "dolor muscular" in sintomas and "cansancio" in sintomas:  
        return "Diagnóstico: Posible influenza"
```

```

        elif "dolor de cabeza" in sintomas and "mareos" in sintomas and
        "visión borrosa" in sintomas:
            return "Diagnóstico: Posible migraña"
        elif "tos" in sintomas and "dificultad para respirar" in sintomas:
            return "Diagnóstico: Posible bronquitis"
        else:
            return "Diagnóstico: No se identificó una condición clara. Se
recomienda consulta médica."
    
```

5. Ejecutar el sistema con un ejemplo práctico

```

if __name__ == "__main__":
    sintomas_ingresados = ["fiebre", "tos", "dolor de garganta"]
    paciente = Paciente("Lucía", 22, sintomas_ingresados)

    print(paciente.presentarse())
    print(paciente.mostrar_sintomas())
    resultado = diagnosticar(paciente.sintomas)
    print(resultado)
    
```

EJERCICIO 02. SISTEMA: CONTROL DIFUSO DE CLIMATIZACIÓN

La lógica difusa permite utilizar **variables lingüísticas** como "temperatura alta" o "clima templado" para evaluar condiciones imprecisas. Se definen conjuntos difusos para cada rango de temperatura, y luego reglas del tipo **IF-THEN** para decidir la acción.

Ejemplo Fuzzy AC Controller

Este ejemplo usa la biblioteca scikit-fuzzy (skfuzzy) que facilita la implementación de lógica difusa.

1. Instalación requerida

```
pip install scikit-fuzzy
```

2. Código del sistema

```

import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# 1. Variables de entrada y salida
temperatura = ctrl.Antecedent(np.arange(15, 41, 1), 'temperatura')
potencia = ctrl.Consequent(np.arange(0, 101, 1), 'potencia')

# 2. Definición de conjuntos difusos
temperatura['baja'] = fuzz.trimf(temperatura.universe, [15, 15, 25])
temperatura['media'] = fuzz.trimf(temperatura.universe, [20, 27, 34])
temperatura['alta'] = fuzz.trimf(temperatura.universe, [30, 40, 40])

potencia['baja'] = fuzz.trimf(potencia.universe, [0, 0, 50])
potencia['media'] = fuzz.trimf(potencia.universe, [30, 50, 70])
potencia['alta'] = fuzz.trimf(potencia.universe, [60, 100, 100])

# 3. Reglas difusas IF-THEN
regla1 = ctrl.Rule(temperatura['baja'], potencia['baja'])
regla2 = ctrl.Rule(temperatura['media'], potencia['media'])
    
```

```
regla3 = ctrl.Rule(temperatura['alta'], potencia['alta'])

# 4. Sistema de control difuso
controlador = ctrl.ControlSystem([regla1, regla2, regla3])
simulador = ctrl.ControlSystemSimulation(controlador)

# 5. Prueba con temperatura de entrada
simulador.input['temperatura'] = 33
simulador.compute()

print(f"Potencia sugerida del aire acondicionado:
{simulador.output['potencia']:.2f}%)")
```

3. ¿Qué hace este sistema inteligente?

- Define rangos difusos para temperaturas: "baja", "media", "alta".
- Usa reglas lingüísticas para decidir la potencia del aire según el clima.
- Evalúa múltiples niveles intermedios, permitiendo una **respuesta más natural y gradual** que la lógica tradicional.

4. 6. Ejecutar la app

```
python climatizador_fuzzy.py
```

Segunda Unidad: Desarrollo Backend
GUÍA DE LABORATORIO N° 15-2:
DESARROLLO SISTEMAS INTELIGENTES EN PYTHON – FLASK - DJANGO

Docente: Jaime Suasnabar Terrel

Fecha:

Duración: 90 minutos

Instrucciones:

OBJETIVOS

- Implementar una interfaz web funcional con Flask y tecnologías frontend (HTML, CSS, JavaScript) que permita la interacción dinámica con el modelo Llama 3 de Meta, ejecutado localmente mediante la librería Transformers de Hugging Face.
- Familiarizar al desarrollador con el proceso completo de integración de un modelo de lenguaje a gran escala (LLM) en un entorno web personalizado, incluyendo la configuración del entorno, carga del modelo, diseño de rutas de comunicación entre cliente y servidor, y gestión eficiente del flujo de datos conversacionales.

EJERCICIO 01. CREACIÓN DE UNA INTERFAZ WEB PARA UN LLM

1. Descargar el modelo Llama 3

Requisitos previos

- Python 3.9+
- Git
- Cuenta en [Hugging Face](#) con acceso aprobado al modelo
- GPU recomendada (para modelos grandes)

Instalación de dependencias

```
pip install torch transformers flask python-dotenv
```

2. Crear la aplicación web con Flask + HTML/CSS/JS

 **Estructura del proyecto**

```
llama_web_app/
└── app.py
└── templates/
    └── index.html
└── static/
    ├── style.css
    └── script.js
└── .env
```

Descargar el modelo

Backend: app.py (Primera parte)

```
from transformers import AutoTokenizer, AutoModelForCausalLM
model_id = "meta-llama/Meta-Llama-3-8B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id,
device_map="auto")
```

Puedes usar otros tamaños como Llama-3-70B si tu hardware lo permite.

Backend: app.py (Segunda parte)

```
from flask import Flask, render_template, request, jsonify
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
import os

app = Flask(__name__)

model_id = "meta-llama/Meta-Llama-3-8B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id,
device_map="auto")

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

@app.route("/generate", methods=["POST"])
def generate():
    prompt = request.json.get("prompt", "")
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    outputs = model.generate(**inputs, max_new_tokens=200)
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return jsonify({"response": response})

if __name__ == "__main__":
    app.run(debug=True)
```

Frontend: templates/index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Chat con Llama 3</title>
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <div id="chat-box">
        <textarea id="prompt" placeholder="Escribe tu mensaje..."></textarea>
        <button onclick="sendPrompt()">Enviar</button>
        <div id="response"></div>
    </div>
    <script src="/static/script.js"></script>
</body>
```

```
</html>
```

Estilos: static/style.css

```
body {  
    font-family: sans-serif;  
    background-color: #f4f4f4;  
    display: flex;  
    justify-content: center;  
    padding: 50px;  
}  
#chat-box {  
    width: 500px;  
    background: white;  
    padding: 20px;  
    border-radius: 8px;  
}  
textarea {  
    width: 100%;  
    height: 100px;  
}  
button {  
    margin-top: 10px;  
    padding: 10px;  
}  
#response {  
    margin-top: 20px;  
    background: #eaeaea;  
    padding: 10px;  
    border-radius: 4px;  
}
```

JavaScript: static/script.js

```
async function sendPrompt() {  
    const prompt = document.getElementById("prompt").value;  
    const responseBox = document.getElementById("response");  
  
    responseBox.innerHTML = "Generando respuesta...";  
    const res = await fetch("/generate", {  
        method: "POST",  
        headers: { "Content-Type": "application/json" },  
        body: JSON.stringify({ prompt })  
    });  
  
    const data = await res.json();  
    responseBox.innerHTML = data.response;  
}
```

3. Ejecutar la aplicación

```
python app.py
```

Abre tu navegador en <http://localhost:5000> y ¡comienza a interactuar con Llama 3

PREGUNTAS DE REPASO

1. ¿Cómo se define formalmente la Inteligencia Artificial General (AGI) y en qué se diferencia de la IA débil?
2. ¿Qué características permiten clasificar un sistema como agente inteligente meta-cognitivo?
3. ¿Cuál es la diferencia entre un modelo generativo y uno discriminativo en aprendizaje automático? Da un ejemplo de cada tipo.
4. ¿Qué tipo de tareas puede realizar un modelo de lenguaje a gran escala (LLM) como Llama 3? ¿Qué arquitectura utiliza?
5. ¿Cuáles son las principales categorías de redes neuronales profundas (Deep Learning) según su estructura? Menciona al menos tres.
6. ¿Qué es la lógica difusa y cómo se aplica en la toma de decisiones en sistemas inteligentes?
7. ¿Qué componentes forman parte de la arquitectura de un sistema inteligente autónomo típico?
8. ¿Qué rol juega el aprendizaje profundo en la generación de contenido original mediante IA generativa?
9. ¿Cómo se estructura un sistema experto basado en reglas (IF–THEN) y cuál sería un caso práctico de su uso?
10. ¿Qué diferencias existen entre los modelos de IA generativa, los LLM y los modelos predictivos tradicionales?