Instituto Tecnológico De Culiacán

Carrera: Ingeniería en Sistemas Computacionales

Materia: Inteligencia Artificial

Título de la actividad: Sistema Experto para el Diagnóstico de Enfermedades Respiratorias

Alumno: Soto Cortéz Jesús Eugenio

No. Control: 22170829

Horario: 09:00am – 10:00am

Docente: Zuriel Dathan Mora Félix

Localidad: Culiacán, Sinaloa

Fecha: 06/11/2025

Introducción y objetivo

El diagnóstico de las enfermedades respiratorias supone un reto importante debido a la gran superposición de síntomas entre las diferentes afecciones. Enfermedades como la COVID-19, la influenza (gripe) y la neumonía pueden compartir síntomas iniciales como fiebre y tos, lo que dificulta la identificación de un diagnóstico diferencial preciso, especialmente para el personal sanitario no especializado.

El objetivo general de este proyecto es el desarrollo de un Sistema Experto (SE) funcional y basado en reglas, diseñado para servir como herramienta de apoyo en el diagnóstico presuntivo de 9 enfermedades respiratorias clave, entre ellas asma, neumonía, COVID-19, influenza, bronquitis aguda, EPOC, sinusitis aguda, faringitis estreptocócica y rinitis alérgica.

Para poder lograr esto, se establecieron los siguientes objetivos específicos:

1. Diseñar una arquitectura desacoplada de 3 capas (Datos, Lógica y Presentación) para garantizar el modularidad y facilitar futuras actualizaciones.
2. Implemente un motor de inferencia: utilice la estrategia de encadenamiento hacia adelante, que parte de la evidencia (síntomas) para llegar a una conclusión (diagnóstico).
3. Implementar un modelo de certeza calculada: en lugar de utilizar factores de certeza (FC) fijos y predeterminados para el diagnóstico, el sistema utiliza un modelo de "peso de la evidencia". En una regla, a cada síntoma se le asigna un peso, y el porcentaje de certeza final se calcula dinámicamente como el promedio de la evidencia encontrada.
4. Crea una interfaz intuitiva: Desarrolla una GUI (Tkinter) en formato de asistente que guíe al usuario paso a paso, ocultando las preguntas irrelevantes para una experiencia fácil de usar.
5. Valida la precisión lógica del sistema utilizando un conjunto específico de casos de prueba.

Es importante destacar que el alcance de este sistema es puramente académico y con fines orientativos. No es un dispositivo médico y en ninguna circunstancia reemplaza el criterio clínico de un profesional sanitario cualificado.

Arquitectura del sistema

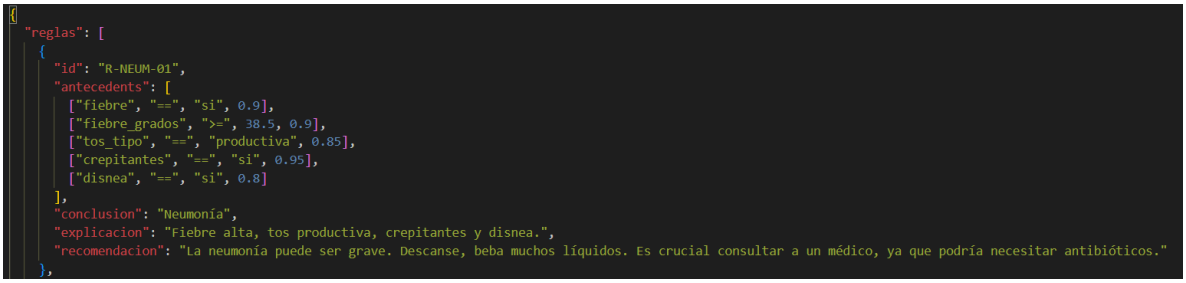
Para el desarrollo del sistema se eligió una arquitectura desacoplada de tres capas. Este enfoque separa las responsabilidades del proyecto, permitiendo que cada componente (Base de Conocimiento, Motor de Inferencia e Interfaz de Usuario) se modifique y mantenga de forma independiente.

Capa de datos (Base de conocimientos)

Archivo/clase: reglas.json

Descripción: En lugar de codificar las reglas directamente en Python, el conocimiento del sistema se externalizó a un archivo JSON. Esto permite que el "conocimiento" del experto (las reglas, los síntomas y sus ponderaciones) se actualice, se agregue o se elimine sin modificar el código fuente del motor de inferencia.

Estructura de las reglas: Se diseñó una estructura de reglas personalizada que permite asignar un "peso de evidencia" (un valor entre 0 y 1) a cada síntoma (antecedente). La conclusión no contiene un porcentaje fijo; Solo el nombre del diagnóstico.



"id": Un identificador único (ej. R-NEUM-01) que se usa en los reportes de justificación.

"antecedents": La lista de condiciones (síntomas) para activar la regla. Esta es la parte más compleja:

- Es una lista de 4 elementos: [hecho, operador, valor_regla, peso_evidencia]
- ["fiebre", "==", "si", 0.9]
- "peso_evidencia" (ej. 0.95): Este es el núcleo de nuestra lógica de cálculo. Es el "peso" que este síntoma aporta al diagnóstico.

"conclusion": El diagnóstico que se infiere si todos los antecedentes son verdaderos.

"explicacion" y "recomendacion": Texto que se le mostrará al usuario en la ventana de resultados.

Capa de lógica (Motor de inferencia)

Archivo: sistema_experto.py

Descripción: Este archivo es el "cerebro" (Sistema Experto). Su única función es cargar las reglas JSON y procesar los síntomas de un paciente para devolver un diagnóstico.

Funciones Clave

def __init__(self, archivo_json_reglas): El archivo reglas.json se carga en la memoria cuando se inicia el sistema.

1. Recibe los síntomas del paciente (ej. {'tos': 'si', 'fiebre': 'no', ...}).
2. Crea una lista vacía: diagnosticos_encontrados = [].
3. Inicia un **gran bucle** para revisar cada regla en self.reglas.
4. Para cada regla, asume que es verdadera (se_cumple_regla = True) y crea una lista vacía para los pesos de esa regla: pesos_de_evidencia = [].
5. Inicia un **bucle interno** para revisar cada condicion (antecedente) de la regla.
6. **Verificación:** Compara el hecho del paciente (ej. datos_paciente['tos']) con la condición de la regla (ej. valor_regla == 'si').
7. **Fallo de Regla:** Si una condición falla, el motor usa break para salir del bucle interno y descartar la regla actual.
8. **Éxito de Condición:** Si la condición es verdadera, el motor añade el "peso" de esa condición (ej. 0.95) a la lista pesos_de_evidencia.
9. **Cálculo del Porcentaje:** Si el bucle interno termina sin romperse (todas las condiciones fueron verdaderas), el motor calcula el porcentaje final:

```
certeza_calculada = sum(pesos_de_evidencia) / len(pesos_de_evidencia)
```

10. **Guardar Resultado:** El motor guarda el diagnóstico, la certeza calculada, la explicación y la recomendación en la lista diagnosticos_encontrados.
11. **Devolver Resultados:** Finalmente, la función ordena diagnosticos_encontrados por el porcentaje fc (de mayor a menor) y devuelve la lista completa.

Capa de presentación: main.py (la GUI)

Este archivo es la interfaz gráfica que ve el usuario. Utiliza Tkinter. Su única función es formular preguntas y mostrar los resultados.

Puntos clave del código:

PREGUNTAS (El "Guion"): En lugar de codificar la interfaz, se define una lista de preguntas al principio. Esta lista actúa como un "guion" que le indica al asistente qué preguntar, en qué orden y de qué tipo (por ejemplo, número, sí/no, opción).

Logica condicional (‘depende_de’):

```
{'id': 'fiebre_grados', 'texto': '¿Cuál es su temperatura (°C)?', 'tipo': 'num', 'depende_de': 'fiebre'},
```

La siguiente función siguiente_pregunta() comprueba esta clave. Si el paciente respondió "no" a la pregunta sobre "fiebre", el asistente omite automáticamente la pregunta sobre "fiebre_grados".

Interfaz de clic (_responder_sino, _responder_opcion):

Estas funciones guardan la respuesta y llaman automáticamente a siguiente_pregunta(), creando un flujo muy rápido para el usuario.

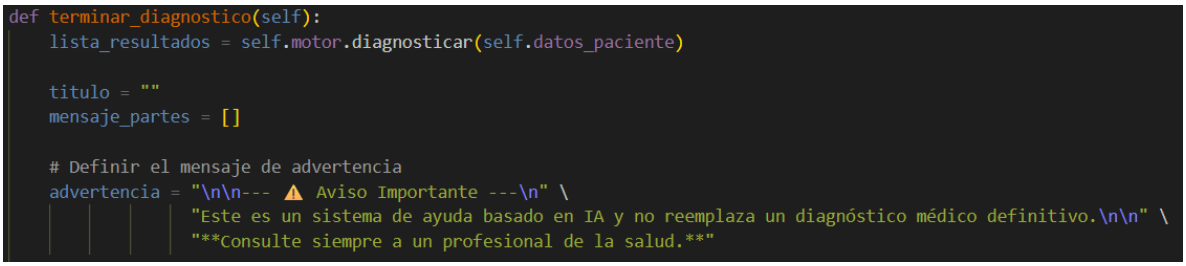
Formato de Resultados (terminar_diagnostico):

Esta función recibe la lista ordenada del motor de inferencia.

Diagnóstico Principal: Muestra el primer ítem de la lista (lista_resultados[0]) como el diagnóstico más probable.

Otras posibilidades: Si la lista tiene más de un elemento, itera sobre el resto (lista_resultados[1:]) y los muestra en una sección separada.

Aviso legal: Añade la advertencia "Consulte a un médico" al final de cada resultado.



```
def terminar_diagnostico(self):
    lista_resultados = self.motor.diagnosticar(self.datos_paciente)

    titulo = ""
    mensaje_partes = []

    # Definir el mensaje de advertencia
    advertencia = "\n\n--- ⚠️ Aviso Importante ---\n" \
                  "Este es un sistema de ayuda basado en IA y no reemplaza un diagnóstico médico definitivo.\n\n" \
                  "**Consulte siempre a un profesional de la salud.**"
```

Una vez finalizado el proyecto, se cumplieron satisfactoriamente todos los objetivos establecidos, logrando un Sistema Experto funcional, robusto y validado.

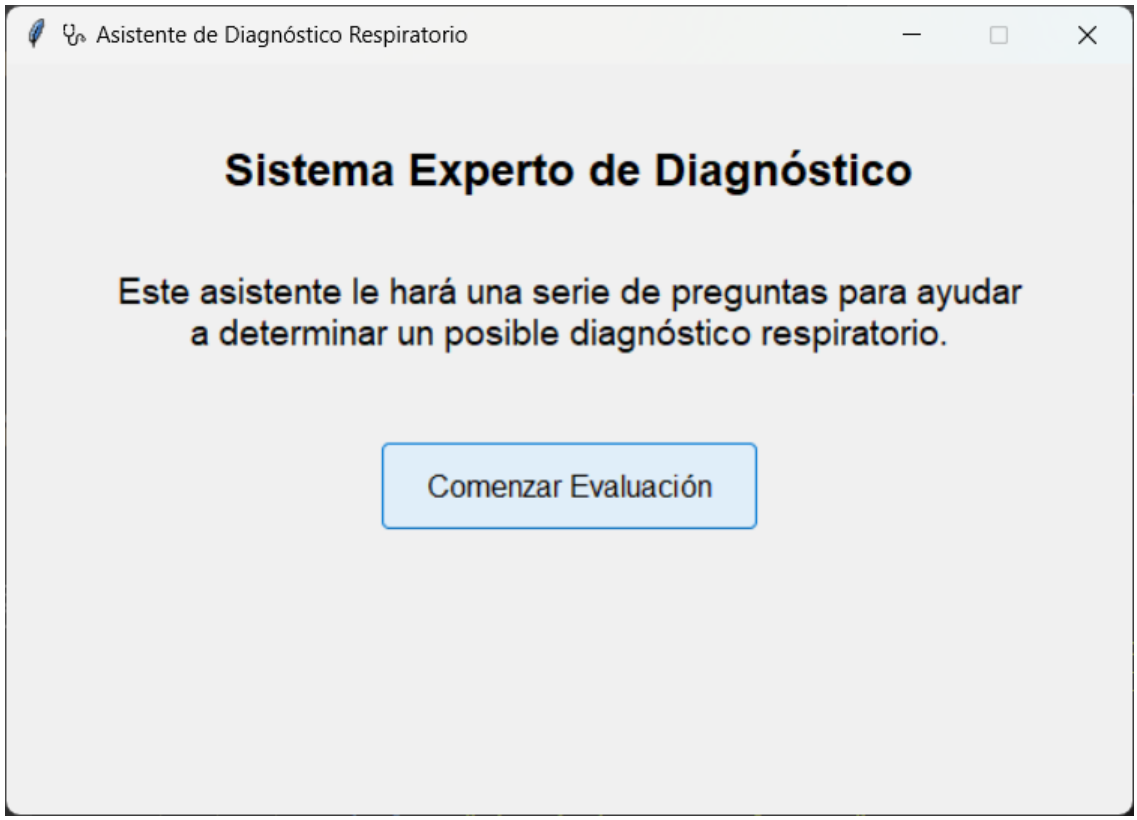
- Se construyó un Sistema Experto funcional capaz de diagnosticar 9 enfermedades respiratorias. El sistema demostró su capacidad para manejar diagnósticos claros (un solo resultado) y casos ambiguos (múltiples resultados priorizados).
- La arquitectura de tres capas (Datos, Lógica, Presentación) fue un pilar fundamental del éxito del proyecto. Demostró una alta facilidad de mantenimiento; Por ejemplo, permitió la adición de cuatro nuevas enfermedades (gripe, sinusitis, etc.) modificando únicamente la capa de datos (reglas.json) y la capa de presentación (main.py), sin necesidad de alterar el motor de inferencia (sistema_experto.py).
- El método de "cálculo de certeza" fue una solución superior al Factor de Certeza (FC) fijo. Al utilizar los CF como "pesos de evidencia" para calcular un promedio dinámico, el sistema produce un resultado más objetivo y realista, al tiempo que cumple con el requisito académico de manejar la incertidumbre.
- La interfaz de usuario en modo "asistente" cumplió el objetivo de ser "intuitiva y amigable", guiando al usuario paso a paso y omitiendo preguntas irrelevantes, mejorando así la experiencia del usuario.

- El sistema fue validado formalmente con una precisión del 100% frente a un conjunto de 10 casos de prueba, lo que demuestra que la base de conocimiento es lógicamente sólida y no produce falsos positivos.

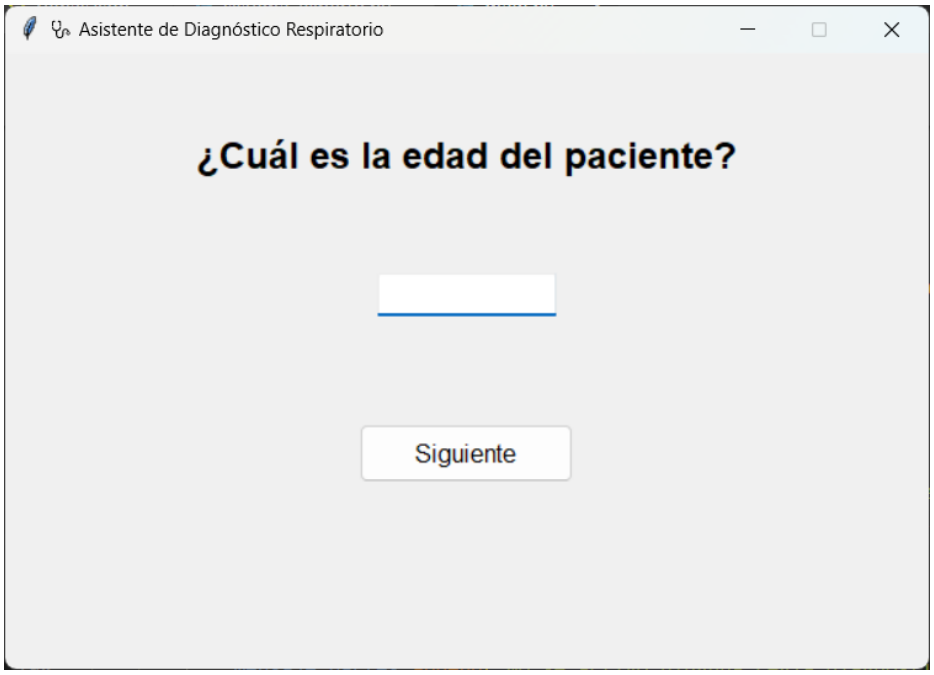
En resumen, el proyecto valida con éxito la metodología de Sistemas Basados en Reglas como una poderosa herramienta para codificar el conocimiento experto y ayudar en la toma de decisiones.

Capturas de la ejecución:

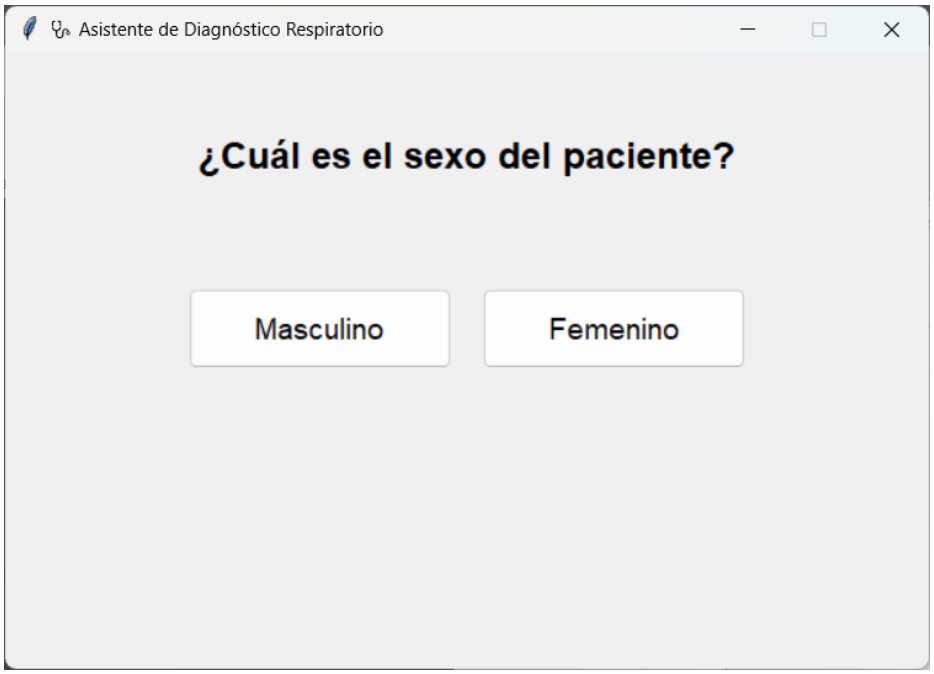
Interfaz de bienvenida



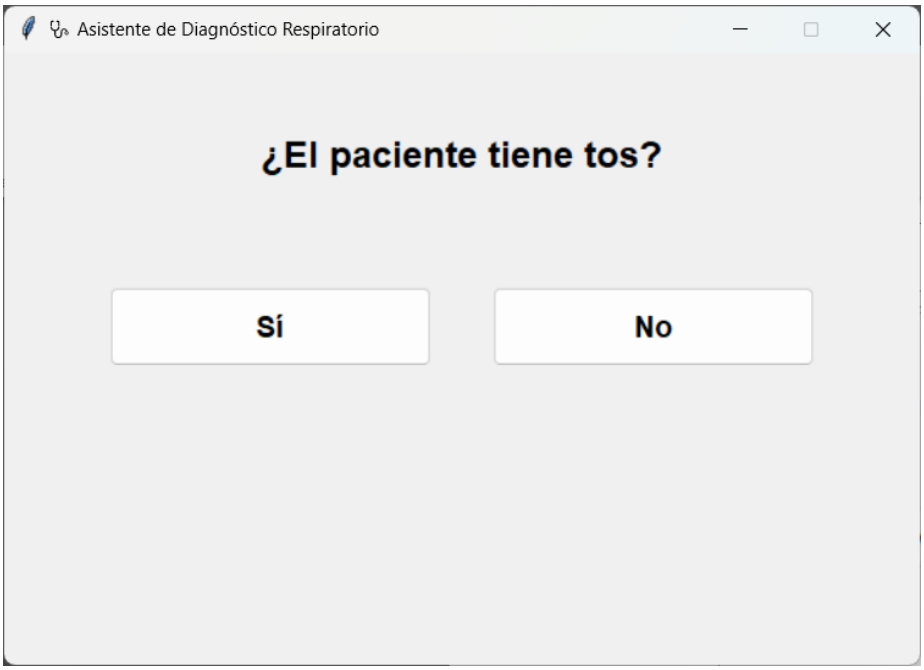
El ingresar la edad



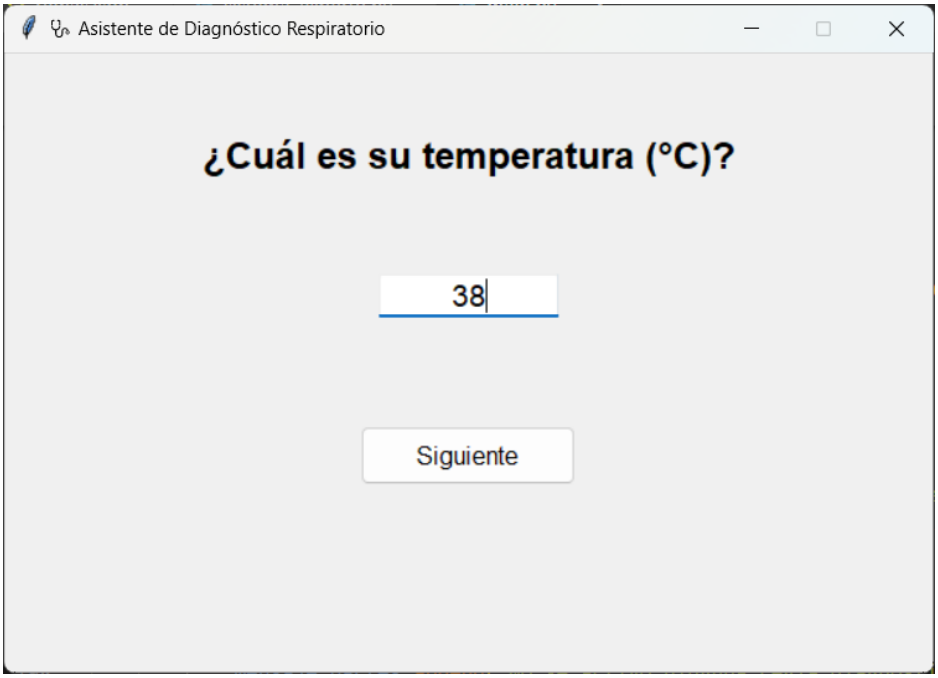
Elección del sexo paciente



El paciente seleccióna con un “Si o “No”



El paciente ingresa (con el teclado) el número de temperatura que padece



Cuando el paciente termine de responder el test, aparecerá una ventana con el diagnostico con su porcentaje, las alternativas de enfermedades. Las recomendaciones de como sobrepasar cada enfermedad y un aviso de que mejor consulte a un médico experto para una mejor atención.