Universidad de Costa Rica – Recinto de Paraiso
Sistemas Expertos
Laboratorio 2
Maria Celeste Montoya Bonilla
C14999

Motor de inferencia para el reconocimiento de enfermedades en plantas

1. Importar bibliotecas

```
import tkinter as tk
from tkinter import (Label, Button, Frame, messagebox,
                     filedialog, ttk, Scrollbar, VERTICAL, HORIZONTAL, font)
import json
```

- **tkinter as tk:** Tkinter es una biblioteca gráfica de Python que permite crear interfaces gráficas de usuario (GUI). Al importarla como tk, estás asignándole un alias para hacer referencia a ella de forma más sencilla en tu código.
- **Label:** La clase Label se utiliza para crear etiquetas de texto en la interfaz gráfica.
- **Button:** La clase Button se utiliza para crear botones en la interfaz gráfica.
- **Frame**: La clase Frame se utiliza para crear contenedores en la interfaz gráfica.
- **messagebox:** El módulo messagebox proporciona funciones para mostrar cuadros de diálogo de mensaje en la interfaz gráfica.
- filedialog: El módulo filedialog proporciona funciones para mostrar cuadros de diálogo de selección de archivos y directorios en la interfaz gráfica.
- **ttk:** TTK (Temas de Tk) es un conjunto de widgets adicionales que proporcionan una apariencia más moderna y consistente en comparación con los widgets estándar de Tkinter.
- **Scrollbar:** La clase Scrollbar se utiliza para crear barras de desplazamiento en la interfaz gráfica.
- **VERTICAL y HORIZONTAL:** Estas son constantes que representan la orientación de las barras de desplazamiento. VERTICAL se utiliza para barras de desplazamiento verticales y HORIZONTAL para barras de desplazamiento horizontales.
- **font:** La clase font se utiliza para crear y configurar fuentes de texto personalizadas en la interfaz gráfica.
- **json:** El módulo json proporciona funciones para codificar y decodificar datos en formato JSON (JavaScript Object Notation), que es un formato de intercambio de datos ligero y fácil de leer y escribir para humanos.

2. Importar reglas de diagnóstico

```python
# Importar las reglas de diagnóstico divididas por síntomas puntuales
with open("reglas.json", "r", encoding="utf-8") as archivo:
    rules = json.load(archivo)
```

Abre el archivo "reglas.json" en modo de lectura ("r") con la codificación UTF-8, esta codificación es para leer correctamente los caracteres especiales de ASCII.

Carga el contenido del archivo JSON en la variable rules utilizando la función json.load(). Esta función decodifica el contenido JSON del archivo y lo convierte en un diccionario de Python

3. Método de evaluación de síntomas

```python
#Metodo para evaluar los sintomas de json con las reglas
1 usage
def evaluar_reglas(sintomas, reglas):
    diagnosticos = []
    explicaciones = []
    for regla in reglas:
        if all(sintomas.get(s, False) for s in regla["sintomas_presentes"]) and \
           not any(sintomas.get(s, False) for s in regla["sintomas_ausentes"]):
            diagnosticos.append(regla["diagnostico"])
            explicaciones.append(regla["explicacion"])
    return diagnosticos, explicaciones
```

Tiene parámetros de entrada que son:

- **sintomas:** Esta es la entrada de donde se toman los datos de las casos que se envían por un json.
- **reglas**: Estas son las que se extraen del punto anterior y traen uns estructura de { “nombre”: string , “sintomas_presentes” : [], “sintomas_ausentes” : [], “diagnostico”: string, “explicacion” : string}

Los parámetros de entrada:

- **diagnosticos:** El diagnóstico asociado si se cumple la regla.
- **explicaciones:** Una explicación asociada al diagnóstico.

Funcionamiento:

- Itera sobre cada regla en la lista de reglas.

- Verifica si todos los síntomas presentes en la regla están presentes en el diccionario de síntomas (sintomas), y si todos los síntomas ausentes están ausentes.
- Si una regla se cumple, agrega el diagnóstico y la explicación asociados a esa regla a listas separadas.
- Finalmente, devuelve dos listas: una lista de diagnósticos y una lista de explicaciones asociadas a los diagnósticos que se cumplieron.

4. Método para abrir el archivo JSON

```
def open_archive():
    archive = filedialog.askopenfilename(initialdir='/',
                                         title='Select archive',
                                         filetype=(('JSON files', '*.json'), ('All files', '*.*')))
    if archive:
        with open(archive, 'r') as f:
            # Cargar el contenido del archive JSON
            datos_json = json.load(f)
    # Actualizar el texto en el widget 'indica' con la ruta del archive seleccionado
    path['text'] = archive if archive else "No se seleccionó ningún archive"
```

- Muestra un cuadro de diálogo para que el usuario seleccione un archivo utilizando filedialog.askopenfilename(), que funciona para abrir archivos del explorador de archivos.
- Si el usuario selecciona un archivo y hace clic en "Abrir" (o equivalente):
- Abre el archivo seleccionado en modo de lectura ('r').
- Carga el contenido del archivo JSON utilizando json.load().
- Actualiza el texto en un widget llamado path para mostrar la ruta del archivo seleccionado. Si no se selecciona ningún archivo, se mostrará un mensaje indicando que no se seleccionó ningún archivo.

5. Método para cargar el archivo mostrarlo y evaluar las reglas

```python
def data_json():
    archive_json = path['text']  # Obtenemos la ubicación del archivo desde el Label
    try:
        # Abrir y cargar el archivo JSON
        with open(archive_json, 'r') as f:
            data = json.load(f)

        # Limpiar los resultados anteriores
        CleanResults()

        # Iterar sobre cada entrada en los datos
        for entry in data:
            symptoms = entry  # Los síntomas se toman directamente de la entrada
            # Evaluar las reglas para obtener diagnósticos y explicaciones
            diagnosis, explications = evaluar_reglas(symptoms, rules)
            # Agregar los resultados al Treeview de resultados
            if diagnosis:
                for diagnose, explication in zip(diagnosis, explications):
                    results.insert( parent: '', index: 'end', values=[diagnose, explication])
            else:
                results.insert( parent: '', index: 'end', values=["No se encontraron enfermedades", ""])
```

Obtención del archivo JSON y carga de datos:

- Obtiene la ubicación del archivo desde el texto del widget path.
- Intenta abrir y cargar el archivo JSON.
- Si hay un error de formato JSON (ValueError), muestra un mensaje de error.
- Si el archivo no se encuentra (FileNotFoundError), muestra un mensaje de error.

Proceso de limpieza de resultados:

- Llama a la función CleanResults() para limpiar los resultados anteriores antes de mostrar los nuevos resultados.

Iteración sobre los datos y evaluación de reglas:

- Itera sobre cada entrada en los datos.
- Para cada entrada, obtiene los síntomas y luego utiliza la función evaluar_reglas() para obtener diagnósticos y explicaciones basados en esas reglas y síntomas.
- Agrega los diagnósticos y explicaciones al Treeview de resultados.

```
    # Mostrar los datos en la table
    table['column'] = list(data[0].keys())
    table['show'] = "headings"  # encabezado

    for column in table['column']:
        table.heading(column, text=column)

    for fila in data:
        table.insert( parent: '', index: 'end', values=list(fila.values()))
except ValueError:
    messagebox.showerror( title: 'Informacion', message: 'Formato incorrecto')
except FileNotFoundError:
    messagebox.showerror( title: 'Informacion', message: 'El archivo no se encontró')
```

Mostrar datos en la tabla:

- Configura la estructura de la tabla (table) basada en las claves de la primera entrada de los datos.
- Inserta los datos en la tabla.

6. Método para eliminar los datos de la tabla

```
#Metodo para eliminar los datos de la tabla
2 usages
def CleanResults():
    results.delete(*results.get_children())
    table.delete(*table.get_children())
```

- Utiliza el método delete de los widgets results y table para eliminar todas las filas existentes de ambos widgets.
- **results.get_children() y table.get_children()** devuelven una lista de los identificadores de todas las filas presentes en los widgets results y table, respectivamente.
- **results.delete(*results.get_children()) y table.delete(*table.get_children())** eliminan todas las filas en los widgets results y table.

7. Interfaz grafica (GUI)

```
# GUI
root = tk.Tk()
root.configure(background='black')
root.geometry('1000x600')
root.minsize(width=600, height=400)
root.title('Motor de Inferencia')

root.columnconfigure( index: 0, weight=25)
root.rowconfigure( index: 0, weight=25)
root.columnconfigure( index: 0, weight=1)
root.rowconfigure( index: 1, weight=1)

frame1 = Frame(root, bg='gray')
frame1.grid(column=0, row=0, sticky='nsew')
frame1.columnconfigure( index: 0, weight=1)
frame1.rowconfigure( index: 0, weight=1)

frame2 = Frame(root, bg='dark gray')
frame2.grid(column=0, row=1, sticky='nsew')
frame2.columnconfigure( index: 0, weight=1)
frame2.rowconfigure( index: 0, weight=1)
```

- Se crea una instancia de la clase Tk() de Tkinter, que representa la ventana principal de la aplicación.
- Se configuran algunas propiedades de la ventana, como el fondo (background), la geometría (geometry), el tamaño mínimo (minsize) y el título (title).
- Se configuran las columnas y filas de la ventana principal para que puedan expandirse y contraerse de manera apropiada.
- Se crean dos frames (frame1 y frame2) dentro de la ventana principal para organizar y contener otros widgets.
- Cada frame se configura con un color de fondo (bg) y se ajusta para que se expanda y contraiga con la ventana principal.
- Los frames se colocan en posiciones específicas dentro de la ventana principal utilizando el método grid().

8. Tablas de la interfaz grafica

```
# Tablas del GUI
results = ttk.Treeview(frame2, height=10)
results.grid(column=0, row=1, columnspan=4, sticky='nsew')

results['column'] = ['Diagnóstico', 'Explicación']
results['show'] = 'headings'

results.heading('Diagnóstico', text='Diagnóstico')
results.heading('Explicación', text='Explicación')

frame2.rowconfigure( index: 1, weight=1)

table = ttk.Treeview(frame1, height=10)
table.grid(column=0, row=0, sticky='nsew')

ladox = Scrollbar(frame1, orient=HORIZONTAL, command=table.xview)
ladox.grid(column=0, row=1, sticky='ew')

ladoy = Scrollbar(frame1, orient=VERTICAL, command=table.yview)
ladoy.grid(column=1, row=0, sticky='ns')

table.configure(xscrollcommand=ladox.set, yscrollcommand=ladoy.set)
```

Tabla de resultados (results):

- Se crea un widget Treeview (ttk.Treeview) dentro del frame2 para mostrar los resultados de diagnóstico.
- Se configura la estructura de columnas de la tabla con los encabezados "Diagnóstico" y "Explicación".
- Se asigna la configuración headings para mostrar solo los encabezados de las columnas.

- Se ajusta la configuración de la fila 1 del frame2 para que se expanda y contraiga con la ventana principal.

Tabla de datos (table):

- Se crea otro widget Treeview dentro del frame1 para mostrar los datos de entrada.
- Se configura un scrollbar horizontal (ladox) y un scrollbar vertical (ladoy) para permitir el desplazamiento horizontal y vertical en la tabla de datos.
- Se configuran los comandos de desplazamiento (xscrollcommand y yscrollcommand) de la tabla para que respondan a los movimientos de los scrollbars.

9. Estilos de la interfaz grafica

```
# Estilos del GUI
style = ttk.Style(frame1)
style.configure( style: "Treeview", font=('Helvetica', 11), background="#272727",
                foreground="white", fieldbackground="black")
style.configure( style: "Treeview.Heading", font=('Tahoma', 11, 'bold'), foreground="#4F4F4F")
```

Creación de un objeto Style:

- Se crea un objeto Style utilizando ttk.Style(frame1), lo que indica que los estilos definidos se aplicarán al frame1.

Configuración de estilos:

- Se configura el estilo para los widgets Treeview utilizando style.configure("Treeview", ...).
- Se establece la fuente (font) como "Helvetica" con un tamaño de 11 puntos y se especifica el color de fondo (background), el color del texto (foreground) y el color de fondo del campo (fieldbackground).
- Se configura el estilo para los encabezados de columna de Treeview utilizando style.configure("Treeview.Heading", ...).
- Se establece la fuente como "Tahoma" con un tamaño de 11 puntos, negrita y se especifica el color del texto (foreground) para los encabezados de columna.

10. Botones de la interfaz grafica

```
# Botones del GUI
button1 = Button(frame2, text="Open", bg="#44C6DF", fg="black", command=open_archive, font=("Arial", 11, "bold"),
                 width=-10, height=1)
button1.grid(column=0, row=0, sticky='nsew', padx=10, pady=10)

button2 = Button(frame2, text='Show', bg='#EC8123', fg="black", command=data_json, font=("Arial", 11, "bold"))
button2.grid(column=1, row=0, sticky='nsew', padx=10, pady=10)

button3 = Button(frame2, text='Clean', bg='#FF3062', fg="black", command=CleanResults, font=("Arial", 11, "bold"))
button3.grid(column=2, row=0, sticky='nsew', padx=10, pady=10)

path = Label(frame2, fg='black', bg='dark gray', text='Ubicación del archivo', font=('Arial', 10, 'bold'))
path.grid(column=3, row=0,  sticky='nsew', padx=10, pady=10)
```

Botón "Open" (button1):

- Crea un botón con el texto "Open".
- Establece el color de fondo (bg) en "#44C6DF" y el color del texto (fg) en negro.
- Asigna la función open_archive como comando que se ejecutará cuando se haga clic en el botón.
- Especifica la fuente como "Arial" con un tamaño de 11 puntos y negrita.
- Ajusta el ancho del botón a un valor negativo para que se ajuste automáticamente al texto.
- Coloca el botón en la columna 0 y fila 0 del frame2, con un relleno (padx, pady) de 10 píxeles.

Botón "Show" (button2):

- Crea un botón con el texto "Show".
- Establece el color de fondo en "#EC8123" y el color del texto en negro.
- Asigna la función data_json como comando que se ejecutará cuando se haga clic en el botón.
- Especifica la fuente como "Arial" con un tamaño de 11 puntos y negrita.
- Coloca el botón en la columna 1 y fila 0 del frame2, con un relleno de 10 píxeles.

Botón "Clean" (button3):

- Crea un botón con el texto "Clean".
- Establece el color de fondo en "#FF3062" y el color del texto en negro.
- Asigna la función CleanResults como comando que se ejecutará cuando se haga clic en el botón.
- Especifica la fuente como "Arial" con un tamaño de 11 puntos y negrita.

- Coloca el botón en la columna 2 y fila 0 del frame2, con un relleno de 10 píxeles.

Etiqueta "Ubicación del archivo" (path):

- Crea una etiqueta con el texto "Ubicación del archivo".
- Establece el color de fondo en "dark gray" y el color del texto en negro.
- Especifica la fuente como "Arial" con un tamaño de 10 puntos y negrita.
- Coloca la etiqueta en la columna 3 y fila 0 del frame2, con un relleno de 10 píxeles.