Explicación General del Programa

Este programa utiliza un algoritmo evolutivo (más específicamente, un algoritmo genético) implementado con la librería DEAP para descubrir automáticamente un conjunto de reglas de diagnóstico a partir de un pequeño conjunto de datos de entrenamiento simulado.

Imagina que quieres que una computadora aprenda a identificar enfermedades basándose en una descripción de síntomas. En lugar de programar explícitamente cada regla ("Si tiene fiebre Y tos Y fatiga, entonces es Influenza"), este programa "hace evolucionar" esas reglas, probando diferentes combinaciones y mejorando las mejores a lo largo de muchas generaciones, hasta encontrar un conjunto de reglas que sea lo más preciso posible para los datos que ha visto.

El objetivo final es obtener un "sistema de diagnóstico basado en reglas" que pueda tomar una descripción de síntomas de un paciente y predecir cuál es el diagnóstico más probable.

Partes Principales del Código

El código se estructura siguiendo los pasos típicos de un algoritmo genético:

1. Definición del Contexto (Palabras Clave y Diagnósticos)

- KEYWORDS: Una lista de cadena de texto que representan los posibles síntomas o "palabras clave" que podrían aparecer en la descripción de un paciente (Ej: 'fiebre', 'tos', 'dolor cabeza'). Estas son las fichas con las que el algoritmo construirá las condiciones de sus reglas.
- CLASSES: Una lista que contiene los posibles diagnósticos o "clases" a las que se puede asignar un paciente (Ej: 'Influenza', 'Resfriado Comun'). Estas son las predicciones que las reglas intentarán hacer.

2. Conjunto de Datos de Entrenamiento (DATASET)

- Este es el "conocimiento" que el algoritmo utilizará para aprender. Cada elemento del DATASET es una tupla: (texto_documento, diagnostico_real).
- texto_documento: Una cadena de texto que simula la descripción de los síntomas de un paciente.
- diagnostico_real: El diagnóstico correcto para ese set de síntomas.
- Función: Permite al algoritmo evaluar la exactitud de sus reglas. Una regla es "buena" si predice correctamente el diagnostico_real para muchos texto_documento.

3. Configuración de DEAP (creator, toolbox)

- creator.create("FitnessMax", base.Fitness, weights=(1.0,)): Define la aptitud (fitness) del individuo. weights=(1.0,) indica que queremos maximizar un único valor (porque la precisión es un valor más alto es mejor).
- creator.create("Individual", list, fitness=creator.FitnessMax): Define lo que es un "individuo" en nuestra población. En este caso, un individuo es una list de reglas de diagnóstico. Cada individuo tiene asociado un valor de fitness que heredará de FitnessMax.
- toolbox = base.Toolbox(): Es el contenedor central de DEAP donde se registran todas las funciones específicas del problema (cómo crear individuos, cómo evaluarlos, cómo mutarlos, cómo cruzarlos, etc.).

4. Generadores para la Inicialización de Individuos

Estas funciones definen cómo se crean las reglas y los individuos desde cero:

- generate_antecedent(): Crea la parte "SI" de una regla (las palabras clave de los síntomas). Selecciona aleatoriamente entre 1 y 3 palabras clave del KEYWORDS.
- generate_consequent(): Crea la parte "ENTONCES" de una regla (el diagnóstico). Selecciona aleatoriamente uno de los CLASSES.
- generate_rule(): Combina un antecedente y un consecuente para formar una (antecedente, consecuente) tupla que es una regla completa.
- generate_individual(): Crea un individuo completo, que es una lista de entre 1 y 5 generate_rule()s.
- toolbox.register(...): Vincula estas funciones de creación a los nombres que DEAP utilizará internamente (attr_rule, individual, population).

5. Función de Aptitud (evaluate_ruleset)

- Propósito: Este es el corazón del algoritmo. Mide qué tan "bueno" es un conjunto de reglas dado.
- Cómo funciona:
    1. Recibe un individual (un conjunto de reglas) y el DATASET.
    2. Para cada documento en el DATASET (Ej: "paciente con fiebre, tos, fatiga"), intenta predecir el diagnóstico usando las reglas del individual.

3. Lógica de predicción: Recorre las reglas del individual en orden. Si una regla cumple con todas sus palabras clave en el texto del documento, esa regla "dispara" y su consecuente se convierte en la predicción. La primera regla que cumple sus condiciones gana (esto es un tipo de "sistema de reglas ordenadas").
4. Compara la predicción con el diagnostico_real.
5. Calcula la precisión (número de predicciones correctas / total de documentos).
6. Devuelve esta precisión como una tupla (precisión,), que es el formato que DEAP espera para la aptitud.

- toolbox.register("evaluate", ...): Registra esta función para que DEAP la use para evaluar cada individuo.

6. Operadores Genéticos Personalizados

Estas funciones son las que introducen la variación y permiten la mejora de los individuos a lo largo de las generaciones, imitando los procesos biológicos de evolución.

- mutate_individual():
    - Propósito: Modifica un gen (una característica) del individuo aleatoriamente.
    - Cómo actúa: Puede añadir o quitar una regla completa del individuo. O, para una regla existente, puede añadir/quitar una palabra clave en su antecedente, o cambiar su consecuente (diagnóstico).
    - Probabilidad: indpb_rule, indpb_keyword, indpb_class, indpb_add_remove_rule controlan la probabilidad de cada tipo de mutación.
- crossover_individual() (mate):
    - Propósito: Combina material genético de dos "padres" (individuos) para crear "descendencia".
    - Cómo actúa: Implementa un "cruce de un punto" (one-point crossover) a nivel de reglas. Toma dos individuos, elige un punto de corte aleatorio en cada uno, e intercambia las "colas" de sus listas de reglas.
- toolbox.register("select", tools.selTournament, tournsize=3):

- Propósito: Elige qué individuos de la población actual pasarán a la siguiente generación o serán usados para reproducción.
- Cómo actúa: Utiliza la "selección por torneo" (tournament selection). Se selecciona un pequeño grupo aleatorio de individuos (tournsize=3), y el mejor de ese grupo es elegido. Este proceso se repite hasta obtener el número deseado de individuos para la próxima generación.

7. Bucle Principal de la Evolución (main function)

Esta es la orquestación de todo el algoritmo genético:

- Parámetros: Define population_size (cuántos individuos en cada generación), num_generations (cuántas veces el algoritmo evoluciona), cx_prob (probabilidad de que ocurra el cruce), y mut_prob (probabilidad de que ocurra la mutación).
- toolbox.population(n=population_size): Crea la primera población de individuos completamente aleatoria.
- tools.HallOfFame(1): Un objeto especial que guarda una copia del individuo con la mejor aptitud encontrado *en toda la historia* de la evolución.
- tools.Statistics(...): Configura cómo se recopilan las estadísticas (promedio, desviación estándar, mínimo, máximo de la aptitud) de la población en cada generación. La corrección clave aquí fue usar lambda ind: ind.fitness.values[0] para asegurar que se pasara un número flotante a statistics.mean y statistics.stdev.
- algorithms.eaSimple(...): Este es la función de alto nivel de DEAP que ejecuta el algoritmo genético. Se encarga de todo el ciclo:
    1. Evaluar la aptitud de cada individuo.
    2. Seleccionar los individuos para la reproducción.
    3. Aplicar cruce y mutación para crear la nueva generación.
    4. Repetir durante el num_generations especificado.
- print y Demostración Final: Después de la evolución, el programa imprime la mejor regla de diagnóstico encontrada (la que está en hof[0]), su precisión en los datos de entrenamiento, y luego demuestra cómo usar esa regla para predecir diagnósticos en algunos textos de prueba nuevos.

E programa es un ejemplo de cómo los algoritmos genéticos pueden ser utilizados para aprender a resolver un problema (clasificación de diagnósticos médicos) mediante la evolución de soluciones (conjuntos de reglas) a lo largo de múltiples generaciones, guiados por una función de aptitud que mide su rendimiento.