PRÁCTICA 1

LÓGICA

Inteligencia Artificial para la Ciencia de Datos

2º Curso de Ciencia e Ingeniería de Datos
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Aris Vazdekis Soria y Alejandra Ruiz de Adana Fleitas

MEMORIA SOBRE EL PROYECTO DE LÓGICA EN PYTHON

El proyecto se enfoca en el desarrollo de un sistema de lógica y generación de tablas de verdad en Python. El programa permite validar y generar tablas de verdad para fórmulas lógicas ingresadas por el usuario. Para garantizar el correcto funcionamiento del sistema, se implementaron diversas funciones y algoritmos.

Inicialmente, se lleva a cabo una etapa de regularización de la fórmula ingresada para verificar su validez y corregir posibles errores en su escritura. Esto se realiza mediante un proceso que analiza la estructura de la fórmula, verificando la correcta asociación de operadores lógicos y la presencia de caracteres no permitidos. Si la fórmula supera esta validación, se procede con su procesamiento.

El procesamiento de la fórmula se realiza mediante la generación de un diccionario que contiene las letras de la fórmula como claves y sus respectivas tablas de verdad como valores asociados. Este diccionario se utiliza para evaluar la fórmula y generar su tabla de verdad correspondiente.

Durante el proceso de evaluación, se aplican diversas operaciones lógicas, tales como negaciones, disyunciones, conjunciones, implicaciones y dobles implicaciones, según la estructura de la fórmula ingresada. Estas operaciones se realizan de manera recursiva, resolviendo cada parte de la fórmula hasta obtener un resultado final.

Una vez evaluada la fórmula, se imprime la tabla de verdad correspondiente, mostrando los valores de verdad de todas las variables involucradas en la fórmula, así como el resultado final.

El sistema se desarrolló utilizando exclusivamente la biblioteca estándar de Python, sin requerir la instalación de bibliotecas adicionales. Esto se hizo asi para cumplir los requisitos del proyecto dados por el profesor.

INDICE

FUNCIONES

En esta sección vamos a hablar sobre las diferentes funciones utilizadas en el programa.

- **FUNCIONES 1.1 —> *regularizacion***

  Esta función está diseñada para verificar si una fórmula lógica está escrita correctamente o si tiene errores de sintaxis. Lo que va hacer la función es verificar que los operadores y letras que tiene la fórmula son correctos. También va a comprobar que los paréntesis escritos estén balanceados, es decir que no haya ninguno sin pareja y que primero esté el paréntesis de apertura y luego el de cierre. Por último se va a comprobar que los operadores estén correctamente asociados a sus operandos, por ejemplo se verifica que un operador de negación "!" este seguido por una letra minúscula, que no tenga ninguna letra minúscula delante y que puede tener delante un paréntesis de apertura o otro operador como"&", "=", etc.

- **FUNCIONES 1.2 —> *contador de letras***

  Esta función recorre la fórmula y cuenta todas las letras que aparecen, almacenándolas en una lista llamada letras_vistas asegurando de no incluir letras duplicadas en la lista.

- **FUNCIONES 1.3 —> *generador diccionario***

  Esta función va a utilizar la función contador_letras para obtener la lista de letras únicas en la fórmula. Utilizando esta lista de letras se va a calcular el número de variables presentes en la fórmula y el número total de filas necesarias para la tabla de verdad basado en el número de variables.

  Luego, para cada fila de la tabla de verdad, genera una secuencia binaria que representa los valores de verdad posibles para las variables.

  Por último crea un diccionario llamado diccionario_comprobador donde las claves son las letras de la fórmula y los valores son los vectores de la tabla de verdad correspondientes para esas letras.

- **FUNCIONES 1.4 —> *comprobador***

  Esta función va a comprobar que exista la letra que se le pasa en diccionario_comprobador, devolviendo el vector de 1 y 0 correspondiente a esa letra.

- **FUNCIONES 1.5 —>generar_letra_aleatoria**

  Esta fórmula se utiliza para generar una letra aleatoria que no sea ninguna de las letras utilizadas en la fórmula.

- **FUNCIONES 1.6 —> borrador**

  Esta función está diseñada para modificar una fórmula lógica, reemplazando una parte resuelta por una letra identificadora nueva, facilitando seguir realizando operaciones sobre la fórmula

- **FUNCIONES 1.7 —> separador**

  Esta función opera sobre una fórmula lógica, dividiéndola en partes más pequeñas y resolviéndolas recursivamente hasta que la fórmula se reduce a una única letra. A continuación se detallan los pasos que sigue:

  1. **Iteración hasta que quede una única letra:**

     La función utiliza un bucle while para iterar sobre la fórmula hasta que solo quede una única letra.

  2. **Manejo de paréntesis:**

     Si hay paréntesis en la fórmula, la función los identifica, extrae la subfórmula dentro de los paréntesis y resuelve recursivamente esta subfórmula utilizando la misma función separador. Luego, reemplaza la subfórmula por su resultado en la fórmula original.

  3. **Manejo de negaciones:**

     La función maneja las negaciones ("!") dentro de la fórmula. Para cada negación encontrada, se resuelve su variable consultando el diccionario de valores de verdad. Luego, se aplica la operación correspondiente (isNegacion) sobre el vector de verdad del operando. Por último, se genera una letra aleatoria nueva y se actualiza la fórmula original sustituyendo la negación y su variable por esta nueva letra.

  4. **Manejo de disyunciones y conjunciones:**

     La función maneja las disyunciones ("|") y las conjunciones ("&") dentro de la fórmula. Para cada operador encontrado, se resuelven sus operandos consultando el diccionario de valores de verdad. Luego, se aplica la operación correspondiente (isDisyuncion o isConjuncion) sobre los vectores de verdad de los operandos. Se genera una letra aleatoria nueva y se

actualiza la fórmula original sustituyendo el operador y sus operandos por esta nueva letra.

5. **Manejo de implicaciones y dobles implicaciones:**

   La función maneja las implicaciones (">") y las dobles implicaciones (“=”) dentro de la fórmula. Para cada operador encontrado, se resuelven sus operaciones consultando el diccionario de valores de verdad. Luego, se aplica la operación correspondiente (isImplicacion o isDobleimplicacion) sobre los vectores de verdad de los operandos. Se genera una letra aleatoria nueva y se actualiza la fórmula original sustituyendo el operador y sus operandos por esta nueva letra.

6. **Verificación de operadores restantes:**

   Después de manejar cada tipo de operador, la función verifica si quedan operadores por manejar. Si no hay ninguno, se detiene el bucle.

- **FUNCIONES 1.8 —> Funciones lógicas**

  Las funciones lógicas consisten en según la interpretación de la función separador, realizar la operativa lógica de negación, disyunción, conjunción, implicación y doble implicación requeridas por el profesor en las pautas del proyecto.

  Cada operación lógica fue implementada como una función diferente, a continuacion explicamos como funciona cada una de ellas:

  - **FUNCIONES 1.8.1 —> isNegacion**

    La función de negación toma el valor de la letra a negar de la fórmula e invierte todos los 0 en 1 y los 1 en 0. Es la función más básica puesto que solo trabaja con una lista de valores y no hay ningún tipo de operativa más allá de la conversión de números.

  - **FUNCIONES 1.8.2 —> isDisyuncion**

    La función de disyunción observa dos listas de dos letras de la fórmula, puesto que siempre una disyunción va a realizarse tomando dos valores. Una disyunción consiste en comprobar si de las dos listas si alguna contiene un valor positivo. Con los condicionales de esta función se interpreta como un recorrido de ambos valores de la lista y si ambos son 1 entonces se guarda un 1, si los valores son distintos se guarda 1, pues significa que una de las listas contiene un 1 y por último si ambos valores son 0 entonces se guarda un 0.

- **FUNCIONES 1.8.3 —> isConjuncion**

  La conjunción sigue una estructura muy similar a la 1.8.2 (Disyunción) pero solamente se guardará el valor 1 cuando ambas listas tengan un valor 1, pues el operador sería equivalente a un AND.

- **FUNCIONES 1.8.4 —> isImplicacion**

  Implicación también toma dos listas y va recorriendo valor por valor. Este guardará un 0 solamente cuando vea que la implicación es falsa, es decir que la segunda lista (o la letra del lado derecho de la implicación) tiene un 0 y la primera tiene un 1.

- **FUNCIONES 1.8.5 —> isDobleImplicacion**

  Doble implicación almacena un 1 cuando ambos valores son iguales, si ambos valores son negativos es un 1 y si los valores son distintos entonces el resultado es 0.

- **FUNCIONES 1.9 —> imprimir_tabla_verdad**

  La función imprimir_tabla_verdad recorre el diccionario de letras y sus valores asociados para construir una tabla de verdad correspondiente a la fórmula lógica proporcionada. Se asegura de evitar duplicados al almacenar las letras vistas y sus valores en listas separadas.

  Luego, genera el encabezado de la tabla combinando las letras vistas y la fórmula original, y crea un separador para mejorar la legibilidad. Finalmente, itera sobre los valores de las letras y su resultado final, formateando cada fila de la tabla y mostrándola La función proporciona una representación clara y concisa de la tabla de verdad para la fórmula lógica especificada.

EXPERIMENTACIÓN

Como experimentación realizamos varias pruebas a lo largo del desarrollo del programa para verificar tanto posibles errores de escritura como el funcionamiento del diccionario.

Entre las pruebas se encuentran varios printeos sobre las claves y valores del diccionario, inicialización de falsos vectores de valores para comprobar la lógica de funcionamiento de disyunción, negación, conjunción, implicación y doble implicación. También fueron realizados varios experimentos sobre fallos de escritura de la fórmula con el fin de comprobar que la función de regularización realiza su labor de indicar los errores de escritura de la fórmula.

CONCLUSIONES

La realización de este proyecto representó un desafío considerable debido a la complejidad del manejo de fórmulas lógicas y la generación de tablas de verdad. A lo largo del proceso de desarrollo, nos enfrentamos a diversos obstáculos que pusieron a prueba nuestra comprensión y habilidades en programación pero que supimos afrontar..

Uno de los principales desafíos fue el manejo adecuado del diccionario utilizado para almacenar las letras de la fórmula y sus respectivas tablas de verdad. Nos encontramos con dificultades para organizar y manipular eficientemente este diccionario, lo que afectó el proceso de evaluación de la fórmula y la generación de la tabla de verdad.

Además, la implementación de la lógica para moverse hacia adelante y hacia atrás en la fórmula, sustituyendo partes resueltas por letras aleatorias, resultó ser más compleja de lo esperado. La gestión de los diferentes casos y escenarios posibles requirió de varias depuraciones y múltiples intentos para llegar a una solución funcional.

En general, el mayor desafío radica en comprender cómo hacer funcionar el programa de manera eficiente y correcta, dado la dificultad puesta por el profesor de no poder utilizar ninguna librería adicional. En uno de nuestros resultados finales nos dimos cuenta de que todo funcionaba bien hasta que nos dimos cuenta de que la tabla de verdad siempre tenía una longitud de 4 valores, no tenia un dimensionamiento variable según el número de letras de la fórmula, puesto que siempre inicializamos los vectores con una longitud de 4. Este proceso de comprensión y resolución de problemas demandó tiempo y esfuerzo significativos.

A pesar de los desafíos encontrados, el proyecto nos proporcionó una valiosa experiencia en el desarrollo de algoritmos lógicos y la manipulación de estructuras de datos complejas en Python. A través de algunas preguntas realizadas al profesor, trabajo en equipo y una pizarra para reestructurar el trabajo, logramos superar los obstáculos y alcanzar una solución funcional que cumple con los requisitos del proyecto.

En última instancia, este proyecto nos permitió fortalecer nuestra comprensión de la lógica proposicional y desarrollar nuestras habilidades en programación, especialmente en el manejo de estructuras de datos y la implementación de algoritmos complejos como el de regularización que se encarga de llamar a todas las funciones necesarias para la ejecución del programa. Aunque enfrentamos dificultades, el resultado final representa un avance en nuestro aprendizaje y comprensión de Python puro.

RECURSOS UTILIZADOS

La estructura y contenido de la memoria se han desarrollado siguiendo las pautas del proyecto de lógica. Entre los recursos utilizados encontramos como entorno de desarrollo Google Collab, puesto que hemos recibido varias recomendaciones de dicha

aplicación web de Google. No se utilizó ninguna herramienta de control de versiones y como herramienta de documentación se utilizó Documentos de Google.

BIBLIOGRAFÍA

- http://elclubdelautodidacta.es/wp/2012/11/python-los-operadores-logicos-y-sus-tablas-de-verdad/
- Apuntes campus virtual