Contador y buscador

Arturo Daza - 506231039

I. Introducción

En el mundo actual, el procesamiento de texto y el análisis de datos desempeñan un papel esencial en una amplia gama de campos, desde la programación hasta la toma de decisiones empresariales. Uno de los aspectos fundamentales en este proceso es la identificación y clasificación de palabras clave en el texto. Para abordar este desafío, se ha desarrollado un algoritmo que aprovecha la técnica de memoización para contar y clasificar las palabras más repetidas en un conjunto de documentos. Este algoritmo es versátil, ya que permite realizar esta tarea tanto con como sin la inclusión de "stop words", palabras comunes que a menudo se excluyen del análisis para centrarse en las palabras más significativas. Además hay una función de búsqueda de palabras, que proporciona los índices de posición de una palabra (que se pide al usuario) en la lista de documentos, esto permite una identificación rápida y efectiva de dónde y cómo se utiliza una palabra en el conjunto de datos.

II. Desarollo

II-A. Algoritmo con técnica memoización

El algoritmo presentado aquí se compone de varias funciones interconectadas. En primer lugar, se realiza un proceso de preprocesamiento del texto para convertir todas las letras a minúsculas y dividir el texto en palabras individuales. A continuación, se utiliza la técnica de memoización para contar las ocurrencias de cada palabra en el texto. La memoización es una estrategia que almacena resultados previos para evitar cálculos repetitivos y, de esta manera, mejora la eficiencia del algoritmo.

El algoritmo permite dos modos de funcionamiento: uno en el que se incluyen "stop wordsz otro en el que se excluyen. Las "stop words"son palabras comunes como "de", .[el]", "la", etc., que a menudo se eliminan del análisis de texto debido a su frecuente aparición y falta de relevancia en la identificación de palabras clave.

En el proceso de evaluación de rendimiento utilizando la biblioteca time de Python, se observa una diferencia notable al optar por incluir o excluir las "stop words". Esto se debe a una característica clave del algoritmo: al excluir las "stop words", se genera una lista adicional que contiene todas estas palabras. Luego, se realiza una comprobación extra para verificar si cada palabra en la lista de documentos no se encuentra en este conjunto de palabras excluidas antes de contarla. A continuación, se detallan los resultados de este análisis temporal de rendimiento:

- ***Resultados con "stop word": tiempo de ejeción de 0.024244 segundos***
- ***Resultados sin "stop word": tiempo de ejeción de 0.074673 segundos***

Añadiendo al análisis anterior, ***la complejidad algorítmica utilizando la notación Big O, tomando el peor de los casos, es O(n) para ambos algoritmos***. Esto se debe a que solo se ha agregado una declaración ïf.ªdicional, lo que significa que aún hay un solo bucle que itera hasta el tamaño "n", que corresponde al tamaño de la lista.

II-B. Algoritmo de búsqueda

Este es un algoritmo que aprovecha la función de Python llamada .[en]umerate". Esta función crea un índice para cada elemento en la lista, lo que simplifica el seguimiento de los documentos que contienen la palabra buscada. A continuación, se verifica dentro de un bucle si la palabra coincide con la palabra buscada. Si hay una coincidencia, se agrega el índice del documento a una lista que almacena los índices correspondientes. Este algoritmo tiene una ***complejidad de O(n)***, donde "n"será igual al tamaño de la lista. Con la lista proporcionada, la ***complejidad temporal será de aproximadamente 0.000989*** segundos después de introducir la palabra que se desea buscar.

III. Conclusión

En conclusión, podemos decir que tanto el algoritmo de conteo de palabras como el algoritmo de búsqueda de palabras en los documentos presentan una complejidad algorítmica de O(n) en el peor de los casos.

Esto se debe a que, en ambos casos del algoritmo de conteo, solo se ha agregado una declaración ïf.ªdicional en el código, lo que significa que solo hay un bucle que itera hasta el tamaño "n", que es el tamaño de la lista de documentos.

Esta eficiencia en la complejidad temporal es una ventaja significativa, ya que significa que el rendimiento de los algoritmos no degrada significativamente a medida que aumenta el tamaño de la lista de documentos. En otras palabras, estos algoritmos son escalables y pueden manejar grandes conjuntos de datos de manera eficiente.