Práctica 3
Buscador

Iván San Martín Fernández

48726989V

Introducción

Esta práctica consistirá en la implementación de un buscador. Partiremos del indexador realizado en la práctica anterior y nuestro objetivo será, aplicando los modelos BM25 y DFR, devolver una lista ordenada de los documentos indexados según una pregunta o query introducida por el usuario en el menor tiempo posible.

Análisis de la solución

Para lograr esto simplemente debemos aplicar la fórmula correspondiente para cada palabra y documento indexado y, posteriormente, ordenar los documentos según el resultado obtenido. Con simplemente esto, obtendremos los resultados esperados.

Pero antes de implementar el código hay que tener en cuenta varias cuestiones para obtener un algoritmo óptimo.

Por un lado, de primeras el algoritmo se puede enfocar de 2 formas:

1. Recorrer todos los documentos aplicando la fórmula correspondiente para cada término de la query.
2. Recorrer todos los términos de la query aplicando la fórmula correspondiente para cada documento.

Si bien parece que da lo mismo una que otra forma, lo cierto es que la segunda es mucho más óptima que la primera. Vamos a realizar un breve estudio de la complejidad temporal de cada uno de ellos:

1. Número de documentos * Número de términos de la query
2. Número de términos de la query * Documentos en los que existe el término de la query

Como vemos, al aplicar la fórmula para cada término de la query, ahorramos el tiempo invertido en aplicar la fórmula para documentos en los que el resultado será 0.

Por otro lado, las ecuaciones necesarias incluyen logaritmos, lo que implica un alto coste temporal. Es por esto que intentaremos reducir el número de logaritmos al mínimo. Para ello, realizaremos los logaritmos lo antes posible. Es decir, para el cálculo de estos logaritmos nos sobra con los datos del término a analizar, por lo que los calcularemos para el término y guardaremos su valor para utilizarlo posteriormente en las ecuaciones para cada documento.

Por tanto nuestro algoritmo podría resumirse en el siguiente pseudocódigo:

1. Indexación: Indexamos los documentos y la query como se explica en la práctica anterior.
2. Búsqueda
    a. Para cada query:
        i. Buscar el mayor id de los documentos para crear los vectores indexados por id comentados en el punto “Mejoras realizadas”.

ii. Guardar nombres y rutas absolutas de cada documento en los vectores generados.
iii. Para cada término de la query:
    1. Calcular valores que únicamente dependen del término para la ecuación correspondiente
    2. Para cada documento:
        a. Calcular valores restantes y resultado de la ecuación
iv. Crear una cola de prioridad con los documentos calculados
v. Guardar en docs_ordenados los n documentos más relevantes.

3. Impresión: Recorremos docs_ordenados imprimiendo cada uno de ellos con el formato deseado.

Justificación de la solución

Tal como se muestra en el apartado anterior, esta solución optimiza el problema, ya que ahorramos el análisis de documentos irrelevantes y optimizamos las ecuaciones minimizando el número de logaritmos y valores a calcular.

Mejoras realizadas

Para la optimización del código se han llevado a cabo las siguientes mejoras:

- **Minimización de logaritmos.**
- **Análisis únicamente de documentos relacionados** con cada término.
- **Cambio de priority_queue a list**: Inicialmente, el atributo “docs_ordenados” se propone como una lista de prioridad que se ordena en función de la pregunta y el valor de similitud de cada resultado. Pero tal como está implementado el código, las preguntas siempre se analizarán de forma ascendente, por lo que podemos generar la cola de prioridad en la función “buscar” y añadir de forma ordenada los resultados a docs_ordenados al finalizar. De esta forma al insertar un nuevo resultado la complejidad O(log2(numero de documentos*numero de preguntas)) se reducirá a O(log2(numero de documentos)).
- **Implementación de vector namesDocs**: Debido a que la indexación tiene una estructura en la que para acceder a la información de los documentos necesitamos conocer su dirección absoluta, guardaremos esta dirección en un vector indexado por su correspondiente identificador, logrando el acceso a la ruta absoluta con una complejidad constante.
- **Uso del .size()**: Hay que llevar mucho cuidado con esta función, ya que tiene complejidad lineal, es por ello que optamos por llamar a esta función el mínimo número de veces posible guardando su valor en una variable para poder utilizarlo posteriormente sin necesidad de volver a llamar a dicha función.
- **Implementación de vector namesNoRuteDocs**: Con el objetivo de no realizar el split de las rutas absolutas directamente al imprimir, ya que realizaríamos splits innecesarios, se propone realizar el split una única vez y guardar los nombres en un vector al que llamaremos “namesNoRuteDocs”.

Eficiencia computacional

En este punto vamos a analizar el coste computacional de nuestro algoritmo:

1. Indexación: Indexamos los documentos y la query como se explica en la práctica anterior.
   O(Indexacion) Ω**(Indexacion)**

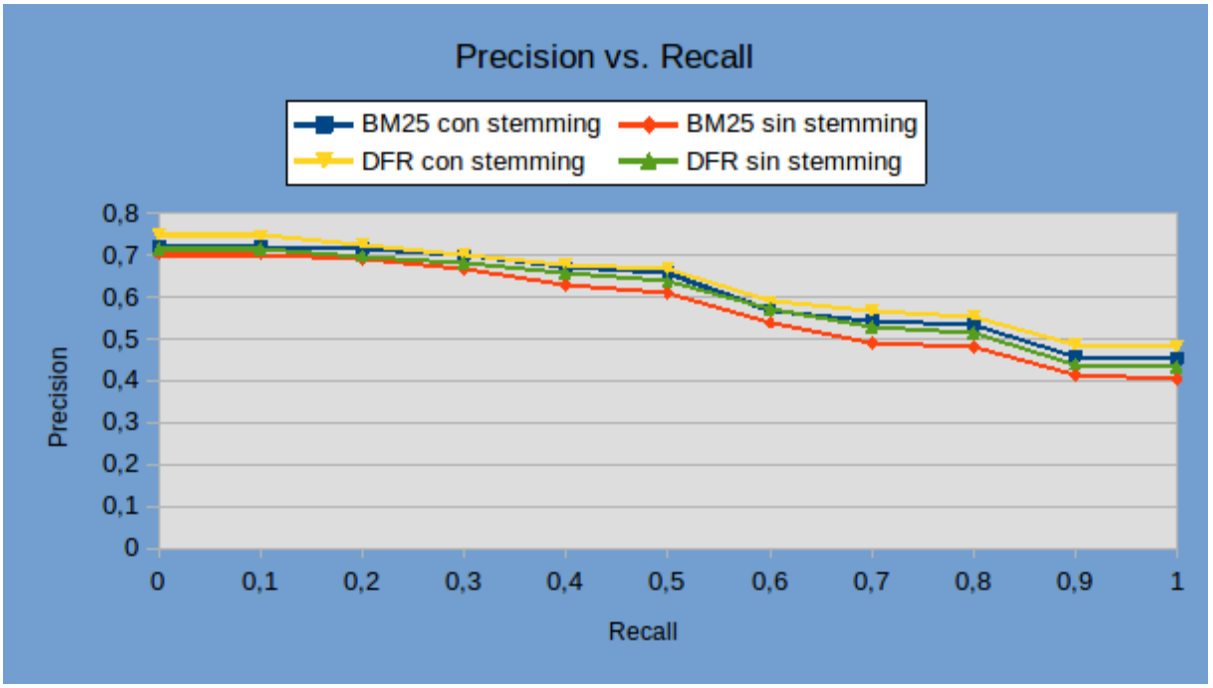
2. Búsqueda
   O(nTerminosQuery * nDocsIndexados * nQueries)
   Ω**(nTerminosQuery * nDocsIndexados * nQueries)**

   a. Para cada query:
      O(nTerminosQueries * nDocsIndexados)
      Ω**(nTerminosQueries * nDocsIndexados)**
      i. Buscar el mayor id de los documentos para crear los vectores indexados por id comentados en puntos anteriores.
         O(nDocsIndexados)
         Ω**(nDocsIndexados)**
      ii. Guardar nombres y rutas absolutas de cada documento en los vectores generados.
         O(nDocsIndexados)
         Ω**(nDocsIndexados)**
      iii. Para cada término de la query:
         O(nTerminosQueries * nDocsIndexados)
         Ω**(nTerminosQueries * nDocsIndexados)**
         1. Calcular valores que únicamente dependen del término para la ecuación correspondiente
            O(1) Ω **(1)**
         2. Para cada documento:
            O(nDocsIndexados) Ω**(nDocsIndexados)**
            a. Calcular valores restantes y resultado de la ecuación
               O(1) Ω**(1)**
      iv. Crear una cola de prioridad con los documentos calculados
         O(nDocsIndexados)
         Ω**(nDocsIndexados)**
      v. Guardar en docs_ordenados los n documentos más relevantes.
         O(n) Ω**(n)**
3. Impresión: Recorremos docs_ordenados imprimiendo cada uno de ellos con el formato deseado.
   O(n * nTerminosQuery * nQueries)
   Ω**(n * nTerminosQuery * nQueries)**

Donde:
nTerminosQuery: Número de términos por query.
nDocsIndexados: Número de docuentos indexados.
nQueries: Número de queries.
n: Número máximo de documentos por query a almacenar en la búsqueda.

Precisión y cobertura

Tras analizar los resultados de buscar 83 preguntas con 423 documentos obtenemos la siguiente tabla de precisión y cobertura. Como se puede observar al aumentar la cobertura la precisión disminuye, ya que implica aumentar el número de documentos devueltos y eso conlleva un mayor número de documentos irrelevantes.



(El resto de archivos relacionados con esta tabla se encuentran en la carpeta src)