Grado en Ingeniería Informática - Universidad de Alicante

Práctica 3: Buscador

Explotación de la Información. 2022-2023

Andrei Eduard Duta
7-6-2023

CONTENIDO

1 INTRODUCCIÓN

En esta memoria se presenta la solución implementada para la fase de búsqueda de información y presentación de resultados en un sistema de recuperación de información. Este buscador se construiría a partir de la indexación generada en la práctica anterior, implementando los modelos de similitud Deviation From Randomness (DFR) y Okapi BM25.

El objetivo principal de esta práctica es profundizar en el entendimiento y aplicación de estos modelos de similitud. Se trata de dos modelos de similitud probabilísticos cuyo objetivo es localizar aquellos documentos que maximicen la probabilidad de pertenecer al conjunto de documentos relevantes.

A lo largo de este documento, se detallará el proceso seguido para la implementación de la clase Buscador, los desafíos encontrados y cómo se superaron, así como las decisiones tomadas para optimizar la eficiencia del buscador.

2 ANÁLISIS DE LA SOLUCIÓN IMPLEMENTADA - IMPLEMENTACIÓN DE LOS MODELOS DFR Y BM25

La implementación de los modelos de similitud Deviation From Randomness (DFR) y Okapi BM25 se realizó en el contexto de la clase Buscador en el método Buscar. El objetivo era calcular el valor de los documentos de acuerdo con la fórmula de similitud elegida.

Después de realizar la indexación de los documentos y la pregunta, para cada documento en el índice, se recorren los términos de la pregunta. Si el término está indexado, se obtiene la información del término en el documento y se calcula el peso del término en la consulta y el peso del término en el documento.

En el caso del **modelo DFR**, se calcula el peso del término en la consulta como la frecuencia del término en la consulta entre el número de términos de la consulta. El peso del término en el documento se calcula utilizando la fórmula DFR, que incluye un término de logaritmo y una división que incorpora el número de documentos y la longitud del documento.

En el caso del **modelo BM25**, se calcula el peso en la consulta del término utilizando la fórmula BM25, que incluye un término de logaritmo y una multiplicación que incorpora la frecuencia del término en el documento y la longitud del documento.

El **valor de similitud del documento** será el sumatorio de la multiplicación de estos dos pesos. Una vez obtenido este valor para el documento, este se almacena en un contenedor ordenado junto con los demás detalles necesarios (pregunta, documento, modelo de similitud).

El principal desafío en la implementación de estos modelos fue asegurar la **precisión de los cálculos**. En particular, fue necesario prestar atención a los detalles como asegurarse de utilizar la división de números de punto flotante en lugar de la división entera cuando se necesitaban números decimales. Este desafío se superó a través de un proceso cuidadoso de depuración.

3 JUSTIFICACIÓN DE LA SOLUCIÓN ELEGIDA

La solución implementada implica el uso de **vectores** en lugar de una **cola de prioridad** para guardar los documentos con su valor de similitud. Se optó por los vectores de C++ ya que, para nuestro caso de uso, tal como se menciona en la siguiente sección, son más rápidos y nos dan más flexibilidad.

Además, se ha optado por mantener tanto los índices como los demás parámetros del indexador privados, pero se accede a ellos a través de **punteros** para evitar realizar copias de forma innecesaria.

4 ANÁLISIS DE LAS MEJORAS IMPLEMENTADAS

La eficiencia de la implementación fue una consideración clave durante el desarrollo de la práctica. Para medir la eficiencia, se utilizó el archivo main.cpp propuesto en el enunciado, midiendo el tiempo que tardaba en realizar la búsqueda.

En cuanto a las **estructuras de datos**, inicialmente, se implementó el almacenamiento de los resultados utilizando una **cola de prioridad**, ya que de esta forma los resultados siempre estaban ordenados. Sin embargo, esto presentó problemas debido a que uno de los requisitos en el método de Buscar era limitar los documentos que se almacenan a un cierto número. Para ello, primero se tienen que analizar (y por tanto guardar) todos los documentos con sus valores de similitud y, una vez que están todos ordenados, quedarse con los primeros N. Esto era difícil de hacer con una cola de prioridad, por lo que se decidió implementarlo con un **vector**, donde los resultados se insertan desordenados y luego se ordenan. Además, en C++, un vector suele ser una de las estructuras de datos más rápidas y esto supuso una mejora en los tiempos. Si necesitáramos ordenar después de cada inserción la cola de prioridad sería mucho más eficiente, pero como solamente necesitamos ordenar una vez al final, el vector es más rápido. Esto ha supuesto una mejora de varios segundos en la búsqueda.

Se implementaron varias mejoras para aumentar la eficiencia del **algoritmo**. Por ejemplo, se implementaron **getters** en la parte pública del **indexador** para evitar tener que copiar muchos objetos. La mejora más notable fue en el método Devuelve para la pregunta, que originalmente copiaba todo el objeto InformacionTermino, que contiene un unordered_map. Al cambiar a un doble puntero, se pudo obtener la dirección de memoria para acceder directamente al objeto sin necesidad de copiarlo. Esta mejora redujo significativamente los tiempos de búsqueda, de 18 segundos a 0,45 segundos,

ya que este es un método que se llama muchas veces durante la ejecución. Esto ha provocado que la búsqueda sea 40 veces más rápida.

Otra mejora implementada ha sido el uso de la **instrucción *emplace_back*** para insertar en los vectores en lugar de *push_back*. De esta forma se evita crear el objeto y después usar el constructor de copia para insertarlo.

También se han implementado mejoras en el indexador que evitan búsquedas innecesarias que se hacían en algunos métodos Devuelve.

Estas mejoras en la eficiencia de la implementación permitieron que el buscador funcionara de manera más rápida y eficiente, lo que es esencial para un buscador eficaz.

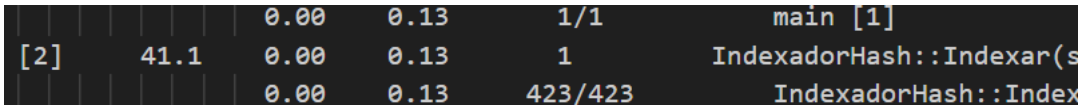
5 ANÁLISIS DE EFICIENCIA COMPUTACIONAL

Se ha medido la eficiencia usando el main.cpp propuesto en el enunciado de la práctica. Este código mide el tiempo que tarda la búsqueda para el Corpus Times para una pregunta y para las 83 preguntas del corpus. Se ha hecho una media de varias ejecuciones.

Modelo similitud	**1 pregunta**	**83 preguntas**
DFR	0.003811s	0.435331s
BM25	0.004459s	0.439121s

No se ha observado una diferencia notable entre los tiempos de los dos modelos ya que presentan una complejidad similar.

Usando el profiler se observa que un 41,1% del tiempo lo ha empleado indexando (lo que implica también la tokenización).



Cabe mencionar que un 12% se ha empleado en recuperar la indexación por lo que si ya se tuviera el índice en memoria, el tiempo de la indexación se reduciría a prácticamente el mismo que la búsqueda.

En la búsqueda se ha empleado un 28% del tiempo.

Podemos ver que, aunque el cálculo de la similitud no es costoso en sí, iterar a través de la colección por cada término de la query sí lo es, por lo que es muy importante optimizar al máximo las estructuras de almacenamiento del índice así como su iteración.

6 ANÁLISIS DE PRECISIÓN Y COBERTURA

Para evaluar el rendimiento del buscador, se realizó una comparación de precisión versus cobertura interpolada a 11 valores entre los modelos BM25 y DFR, con y sin stemming.

DFR sin stemming

```
Queryid (Num):  All
Total number of documents over all queries
    Retrieved:    35109
    Relevant:       324
    Rel_ret:        321
Interpolated Recall - Precision Averages:
    at 0.00       0.6589
    at 0.10       0.6549
    at 0.20       0.6368
    at 0.30       0.6282
    at 0.40       0.6063
    at 0.50       0.5945
    at 0.60       0.5144
    at 0.70       0.4923
    at 0.80       0.4800
    at 0.90       0.4212
    at 1.00       0.4126
Average precision (non-interpolated) for all rel docs(averaged over queries)
                  0.5379
Precision:
  At    5 docs:   0.3349
  At   10 docs:   0.2422
  At   15 docs:   0.1847
  At   20 docs:   0.1452
  At   30 docs:   0.1016
  At  100 docs:   0.0345
  At  200 docs:   0.0181
  At  500 docs:   0.0077
  At 1000 docs:   0.0039
R-Precision (precision after R (= num_rel for a query) docs retrieved):
    Exact:        0.4818
```

BM25 sin stemming

```
Queryid (Num):  All
Total number of documents over all queries
    Retrieved:    35109
    Relevant:       324
    Rel_ret:        321
Interpolated Recall - Precision Averages:
    at 0.00       0.7001
    at 0.10       0.7001
    at 0.20       0.6927
    at 0.30       0.6677
    at 0.40       0.6468
    at 0.50       0.6250
    at 0.60       0.5468
    at 0.70       0.5217
    at 0.80       0.5086
    at 0.90       0.4367
    at 1.00       0.4325
Average precision (non-interpolated) for all rel docs(averaged over queries)
                  0.5740
Precision:
  At    5 docs:   0.3735
  At   10 docs:   0.2627
  At   15 docs:   0.1976
  At   20 docs:   0.1530
  At   30 docs:   0.1040
  At  100 docs:   0.0346
  At  200 docs:   0.0180
  At  500 docs:   0.0077
  At 1000 docs:   0.0039
R-Precision (precision after R (= num_rel for a query) docs retrieved):
    Exact:        0.5182
```

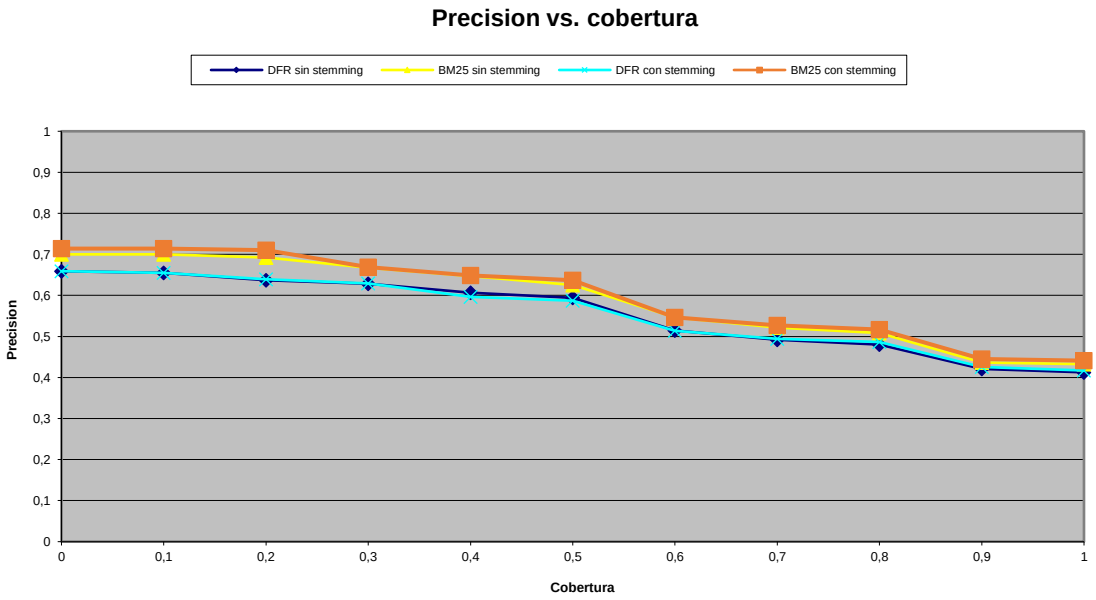
DFR con stemming

```
Queryid (Num):  All
Total number of documents over all queries
    Retrieved:    35109
    Relevant:       324
    Rel_ret:        321
Interpolated Recall - Precision Averages:
    at 0.00       0.6591
    at 0.10       0.6551
    at 0.20       0.6391
    at 0.30       0.6290
    at 0.40       0.5964
    at 0.50       0.5870
    at 0.60       0.5136
    at 0.70       0.4943
    at 0.80       0.4861
    at 0.90       0.4258
    at 1.00       0.4171
Average precision (non-interpolated) for all rel docs(averaged over queries)
                  0.5374
Precision:
  At    5 docs:   0.3301
  At   10 docs:   0.2434
  At   15 docs:   0.1831
  At   20 docs:   0.1470
  At   30 docs:   0.1032
  At  100 docs:   0.0351
  At  200 docs:   0.0183
  At  500 docs:   0.0077
  At 1000 docs:   0.0039
R-Precision (precision after R (= num_rel for a query) docs retrieved):
    Exact:        0.4658
```

BM25 con stemming

```
Queryid (Num):  All
Total number of documents over all queries
    Retrieved:    35109
    Relevant:       324
    Rel_ret:        321
Interpolated Recall - Precision Averages:
    at 0.00       0.7141
    at 0.10       0.7141
    at 0.20       0.7102
    at 0.30       0.6690
    at 0.40       0.6488
    at 0.50       0.6372
    at 0.60       0.5462
    at 0.70       0.5271
    at 0.80       0.5169
    at 0.90       0.4450
    at 1.00       0.4411
Average precision (non-interpolated) for all rel docs(averaged over queries)
                  0.5809
Precision:
  At    5 docs:   0.3639
  At   10 docs:   0.2602
  At   15 docs:   0.1968
  At   20 docs:   0.1524
  At   30 docs:   0.1064
  At  100 docs:   0.0345
  At  200 docs:   0.0181
  At  500 docs:   0.0077
  At 1000 docs:   0.0039
R-Precision (precision after R (= num_rel for a query) docs retrieved):
    Exact:        0.5369
```

Los resultados se presentan en la siguiente gráfica:



El **valor más alto para la precisión** se obtuvo con el modelo BM25 con stemming, alcanzando un valor de 0,7141 con una cobertura de 0. Por otro lado, el valor más bajo se obtuvo con el modelo DFR sin stemming, con una precisión de 0,4126 y una cobertura de 1. En general, tal como se esperaba, se observó que la precisión disminuye a medida que aumenta la cobertura.

Al utilizar **stemming**, se observó una mejora marginal en la precisión en comparación con el modo sin stemming. Esto sugiere que el stemming puede ayudar a mejorar la precisión de la búsqueda al reducir la variabilidad de los términos de búsqueda.

Entre los modelos DFR y BM25, **BM25** mostró mejores resultados en términos de precisión. Sin embargo, se observó que la precisión de BM25 disminuye más rápidamente a medida que aumenta la cobertura en comparación con DFR. Esto podría sugerir que, aunque BM25 puede proporcionar resultados más precisos para búsquedas más específicas (cobertura baja), su rendimiento puede disminuir más rápidamente para búsquedas más generales (cobertura alta).

Es importante mencionar que la diferencia entre los modelos fue más notable sin utilizar el modo de quitar acentos y pasar a minúsculas. Esto sugiere que estas técnicas de **preprocesamiento** pueden ayudar a reducir la diferencia en el rendimiento entre diferentes modelos de similitud.