6ª práctica de laboratorio de SIW
“Resolviendo consultas con un índice”

Motivación

En la sesión anterior se desarrolló código para indexar una colección de documentos así como código para poder almacenar dicho índice en disco y volver a cargarlo en memoria. Hoy se verá cómo utilizar dicho índice para resolver de la manera más eficiente posible consultas textuales.

Descripción del ejercicio y del entregable

Desarrollar un *script* que demuestre la resolución de consultas mediante el índice.

En su versión más sencilla el *script* cargará el índice de disco (el proceso de indexado tiene que realizarse previamente con otro script) y procesará una o más consultas textuales obteniendo para cada una de ellas una lista de resultados ordenada por relevancia decreciente.

En una versión más elaborada el índice será cargado en memoria una única vez por una aplicación que ofrecerá un servicio web. Dicho servicio web será atacado por un *script* que será el que procese las consultas y reciba desde el servicio web los resultados para las mismas. Si se desea implementar esta opción lo más recomendable sería utilizar el framework *Flask* para desplegar el motor de búsqueda (el *script* que carga el índice y resuelve las consultas) y *Requests* para el *script* que envía las consultas al buscador (ya se tuvo que usar *Requests* en el ejercicio del crawler).

Con independencia de cómo se implemente, el *script* que reciba la lista de resultados debe proporcionar en la salida no solo el identificador de los documentos sino también un fragmento—p.ej., los primeros 280 caracteres—de cada documento resultante.

De manera opcional (supondría puntos extra en la evaluación) pueden utilizarse los juicios de relevancia que acompañan a las colecciones sugeridas en la quinta práctica para evaluar el rendimiento del buscador (p.ej., con P@1—precisión en 1, P@10—precisión en 10, o con una curva precisión vs exhaustividad). **Si alguien quiere realizar esta parte debe contactar con `dani@uniovi.es`.**

Para resolver las consultas se procederá como sigue:

1. La consulta se vectorizará del mismo modo en que el indexador vectoriza los documentos. Es decir, pasará a minúsculas, tokenizará en términos, aplicará *stemming* o lematización según se haga al indexar y eliminará palabras vacías o no en función de lo que haga el programa de indexado.
2. Para cada término de la consulta se obtendrá la lista de documentos que lo contienen[1], es decir, sus *post-lists*.
3. Se combinarán todas esas *post-lists* en una única lista de documentos candidato.
4. Se calculará la similitud del coseno entre el vector de la consulta y cada uno de los documentos candidato.

Para ello se utilizará la siguiente ecuación:

$$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Veamos un ejemplo para entenderlo mejor. Supongamos la colección de documentos que se usó como ilustración en la práctica anterior:

- `The quick brown fox jumps over the lazy dog.`
- `The brown dog is jumped over by the fast brown fox.`
- `A fox is jumping over a dog.`
- `The brown fox jumps over the dog.`
- `The fast brown fox jumps over the sleeping dog.`

[1] Al hacerse de esta manera no es obligatorio que todos los términos de la consulta aparezcan en los documentos resultantes aunque obviamente deberían ser más relevantes aquellos que contengan más términos en común con la consulta. Dicho de otro modo, en estas consultas se aplica por defecto el operador `OR`. Una posible ampliación de la práctica consistiría en explorar—no necesariamente implementar—cómo se podrían aceptar consultas más elaboradas que incluyesen el uso de `AND` o expresiones anidadas con paréntesis. **¡Atención!** Puesto que al indexar no se recogen los *offsets* de los términos dentro del texto de los documentos no es posible aceptar consultas con frases exactas (i.e., entrecomillados) ni con comodines (`*`).

Y supongamos la consulta siguiente:

```
A brown fox jumped over a lazy dog.
```

Al vectorizar esa consulta siguiendo los mismos criterios que se supusieron en los ejemplos del enunciado anterior tendríamos lo siguiente:

```
a:0.25 brown:0.125 dog:0.125 fox:0.125
jump:0.125 lazy:0.125 over:0.125
```

Dado el índice que pusimos como ejemplo en la práctica anterior está claro que habría que comparar con todos los documentos. De hecho, incluso si se hubiera optado por "purgar" los términos con *idf* nulo—`dog`, `fox`, `jump`, `lazy`, `over`—esta consulta en concreto seguiría teniendo a todos los documentos como candidatos.

Así pues podemos tomar cualquier documento para calcular su similitud coseno; por ejemplo, el primero cuyo vector sería:

```
brown:0.11 dog:0.11 fox:0.11 jump:0.11
lazy:0.11 over:0.11 quick:0.11 the: 0.22
```

La cuestión es que estos vectores solo tienen los valores *tf* y necesitamos (estamos implementado el modelo vectorial de Salton) vectores con valores *tf·idf*. Los valores *idf* se obtendrían igualmente del índice y, en consecuencia, la consulta con los valores *tf·idf* sería así:

```
a:0.25*0.70 brown:0.125*0.1 dog:0.125*0 fox:0.125*0
jump:0.125*0 lazy:0.125*0.70 over:0.125*0
```

Es decir,

```
a:0.175 brown:0.013 lazy:0.088
```

Y el documento en cuestión tendría el siguiente vector:

```
brown:0.11*0.1 dog:0.11*0 fox:0.11*0 jump:0.11*0
lazy:0.11*0.70 over:0.11*0 quick:0.11*0.70 the:0.22*0.10
```

Es decir,

```
brown:0.01 lazy:0.08 quick:0.08 the:0.02
```

Si en la ecuación anterior suponemos que *A* es la consulta y *B* es el documento tendríamos entonces:

$$\sqrt{\sum_{i=1}^{n} A_i^2}$$

√(0.175·0.175 + 0.013·0.013 + 0.088·0.088) = √(0.031 + 0.000 + 0.008) = √0.039 = 0.20

$$\sqrt{\sum_{i=1}^{n} B_i^2}$$

√(0.01·0.01 + 0.08·0.08 + 0.08·0.08 + 0.02·0.02) = √(0.0001 + 0.0064 + 0.0064 + 0.0004) = √(0.0133) = 0.115

$$\sum_{i=1}^{n} A_i B_i$$

Puesto que los únicos componente en común y que no tengan un peso nulo entre ambos vectores son `brown` y `lazy` tendríamos:

0.013·0.01 + 0.088·0.08 = 0.00717

$$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

0.00717/(0.20·0.115) = 0.00717/0.023 = 0.312

Para generar la lista de resultados sería preciso calcular la similitud coseno entre la consulta y todos los documentos candidatos y ordenar esa lista por valores decrecientes—es decir, de mayor a menor similitud.

En este momento es preciso señalar que la implementación que se ha sugerido para el índice es "transparente" pero muy poco eficiente. Téngase en cuenta que para obtener la ponderación *tf·idf* de un documento habría que acceder a esos valores para todos los términos que aparecen en el mismo—de hecho, si solo se ha implementado la estructura de datos en dos niveles sería harto complicado.

Es por ello que cuando se habla de vectorización de documentos se refiere a que estos se transforman en vectores numéricos donde cada componente (un valor entero) está asociada con un término en el vocabulario y el índice es, en consecuencia, una matriz de números reales.

Así, una vez se tiene una matriz con valores *tf* y un vector con valores *idf* es posible conseguir una matriz con valores *tf·idf* mediante producto de matrices—el vector *idf* se usa para producir una matriz diagonal cuadrada.

En consecuencia, si no se ha implementado ninguna estructura auxiliar que permita acceder a los valores *tf·idf* de todos los términos de un documento se aceptará que los vectores de los documentos candidatos se reduzcan a aquellas componentes que aparezcan en la consulta. De ese modo, los valores *idf* a localizar serán muchos menos y serán siempre los mismos para todos los documentos candidatos.

Resulta interesante que la función de similitud **BM25** que es bastante mejor que la similitud del coseno no utiliza *idf* para los términos de los documentos sino su frecuencia absoluta con lo que resulta relativamente sencilla de implementar con la estructura propuesta para el índice en estos ejercicios y es eficiente sin tomar decisiones tan radicales como ignorar términos del documento que no aparecen en la consulta. Quien lo desee puede implementar opcionalmente BM25—por supuesto, con puntuación extra.

Se entregará en el campus virtual un archivo comprimido que contendrá:

- Todo el código desarrollado. Al menos debería haber un *script* para generar el índice de la colección y almacenarlo en disco y otro para resolver consultas contra dicho índice.
- Una selección aleatoria de 5 consultas de la colección elegida con la lista de documentos resultantes (solo sus identificadores).
- Un archivo LEEME que detalle todas las decisiones tomadas en la implementación así como cualquier información útil para la ejecución y evaluación del ejercicio.