Memoria Práctica 1

Miguel Laseca, Pablo Marcos

20 de febrero de 2019

1. Implementación basada en Lucene

1.1. Indexación

Hemos implementado las clases e interfaces pedidas partiendo del código del ejemplo dado y del fichero **TestEngine.java** en el que se llamaba a los métodos solicitados para estas.

En el caso de LuceneIndex incuye a los métodos que se hacen referencia en `TestEngine` y se ha añadido un método *close* y otro para acceder al IndexReader de Lucene dentro del Engine para eludir una doble instanciación.

En `LuceneBuilder` el método *build* construye el índice en función de la estructura del path (si es un directorio, un .zip, o un texto plano). Además hemos incorporado un argumento para poder elegir dónde guardar los módulos de los documentos usados en el coseno.

En el caso de la creación a partir de las URL como había una que fallaba que causaba que todo el programa crasheara, hemos wrapeado la funcionalidad en un try/catch que parsea este tipo de fallo.

En el momento en el que parseamos con JSoup la colección dada de ejemplo nos percatamos de que se analizaba también las cabeceras de los retornos HTTP, que al no tratarse de contenido HTML no se eliminaba y ocurría anomalías como que HTTP era la palabra con más frecuencia en los documentos. Para solucionarlo hicimos una función que localizaba y eliminaba la cabecera.

1.2. Búsqueda

Se nos ha pedido en este apartado implementar cuatro clases: `LuceneEngine`, `LuceneRanking`, `TextResultDocRenderer` y `SearchRankingDoc` para implementar el motor de búsqueda.

Para la implementación del engine hemos utilizado las consutas booleanas de Lucene (BooleanQuery). Hemos ampliado la funcionalidad base permitiendo al usuario introducir diferentes cláusulas booleanas (BooleanClause). Si delante del término de búsqueda escribe '-' entonces se usará `MUST_NOT` y si escribe '+' se usará `MUST`. Por defecto se usará `SHOULD`. Por otra parte las clases relacionadas con el ranking han sido completadas para almacenar los resultados de las consultas, así como los rankings.

2. Modelo vectorial

Para este apartado se ha necesitado completar una implementación del ranking que no utilizara a Lucene, que se encuentran en el paquete *es.uam.eps.bmi.search.ranking.impl*, y son las clases `Ranking`, `RankingDoc` y `RankingIterator`.

2.1. Producto escalar

En primer lugar se calculan todos los IDFs de las palabras ya que no dependen de la ocurrencia de la palabra en un documento concreto.

La fórmula del cálculo de los IDFs utilizada es la siguiente:

$$\log_2 \frac{|D+1|}{|D_t+0{,}5|}$$

La cual se corresponde con la corrección de Laplace aplicada al IDF puro para suavizar las consultas.

2.2. Coseno

Para el cálculo del coseno la única variación con respecto al producto escalar es la división por los módulos del documento. Se ha omitido la division por el módulo de la query al tratarse de un factor común a todos los documentos.

El cálculo de los módulos se realiza en tiempo de creación del índice, tarea realizada por la clase `LuceneBuilder`. En este caso si no se especifica la ruta del fichero de módulos estos se almacenan en **modulos.txt**.

Al tratarse de una operación muy costosa computacionalmente para colecciones grandes, hemos realizado una serie de optimizaciones partiendo de la primera implementación realizada, usando un HashMap para no recalcular IDFs y usando los vectores de frecuencias de los documentos para solo iterar sobre palabras encontradas en los documentos.

El TFIDF utilizado es el mismo expuesto en el apartado 2.1.

3. Extensiones

3.1. Estadísticas de frecuencia

En la clase TermStats se calculan las frecuencias y se vuelcan a los ficheros de frecuencias pertinentes, y con el script **frecuencias.py** dibujamos las gráficas correspondientes.

Se ha empleado la colección docs1k porque el resto de colecciones de ejemplo carecen de un tamaño suficientemente grande como para que se manifiesten las leyes asintóticas.

En la figura 1 se han ordenado los términos por su frecuencia total, y se han ploteado en escala logarítmica, y se manifiesta un comportamiento lineal, que se corresponde con la ley de Zipf. En la figura 2 se muestra el top de las palabras con mayor frecuencia global, y en la que vemos que la palabra *tree* es la que más aparece, cosa que parece lógica dado que la colección es un conjunto de páginas de árboles genealógicos. La figura 3 ordena los datos por el número de documentos en los que aparecen.

3.2. Interfaz de usuario

Para la GUI de nuestra aplicación hemos reciclado una vieja interfaz de un curso anterior, adaptándolo al modelo de nuestra aplicación.

Las imágenes de la figura 4 muestran la interfaz de creación del índice, en las cuales se puede seleccionar la ruta de la colección, el ruta del guardado del índice, el fichero donde almacenar los módulos y el motor, a elegir entre LuceneEngine y el modelo vectorial VSMEngine.

La figura 5 corresponde a la interfaz de búsqueda que aparece tras la creación del índice, solo se ha implementado la visualización de los resultados cuando la colección es un directorio. Al clickar dos veces en un resultado se abre en el navegador, como se muestra en el ejemplo de la figura 6.

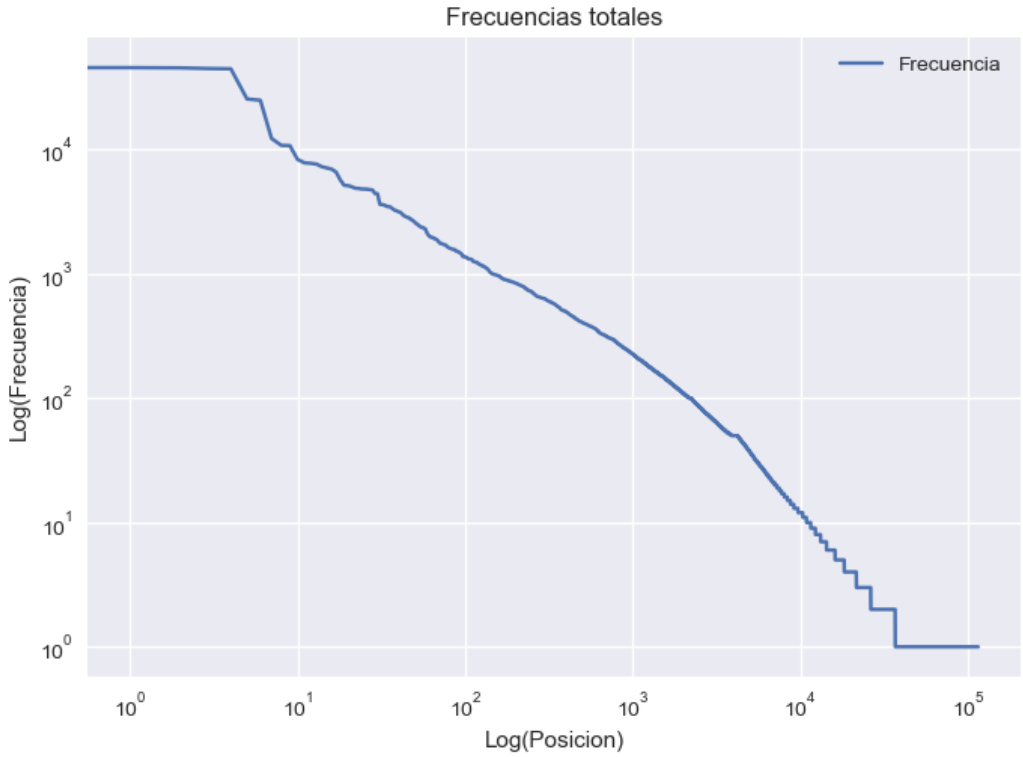


Figura 1: Frecuencia de aparencias de los téminos. Se puede apreciar la linealidad del decrecimiento.

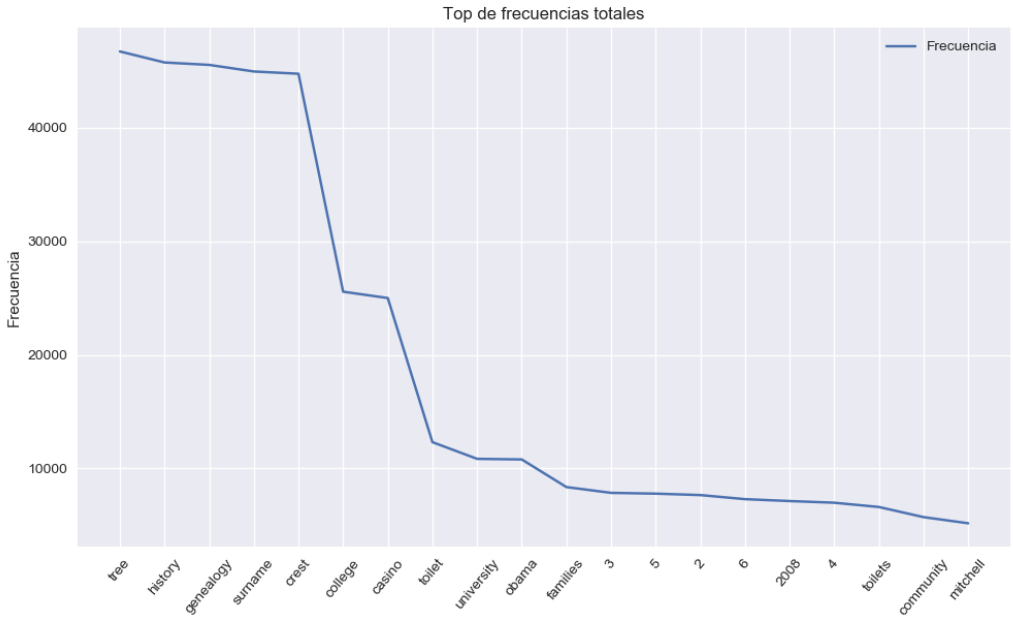


Figura 2: Top 20 de las palabras más encontradas.

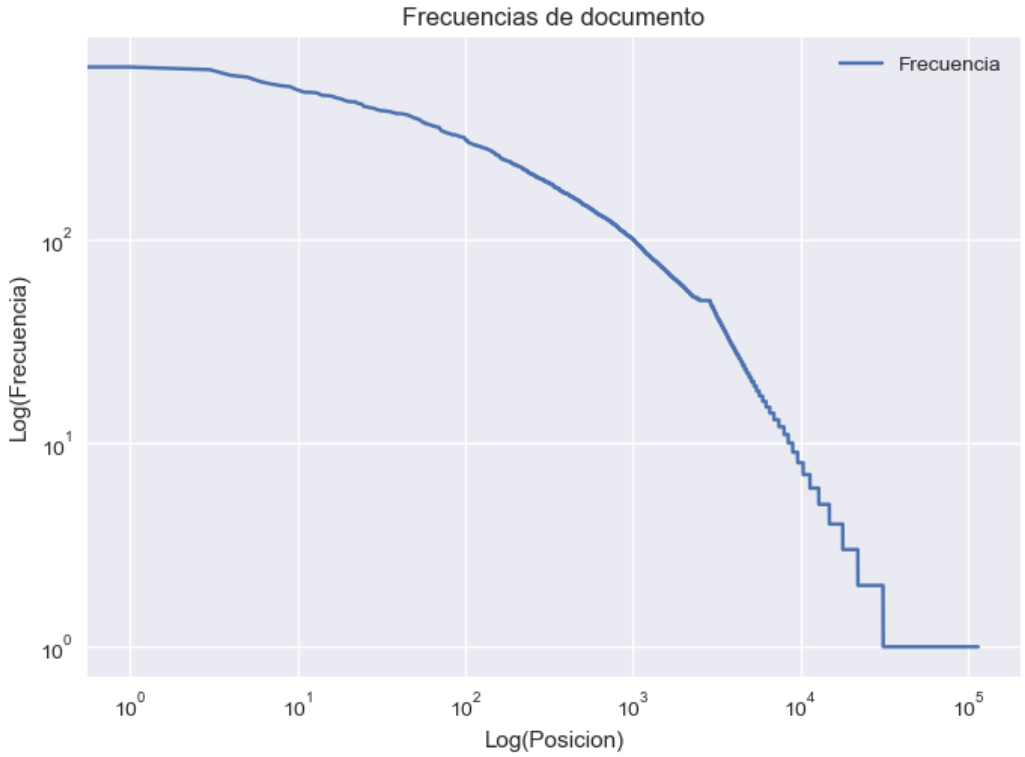
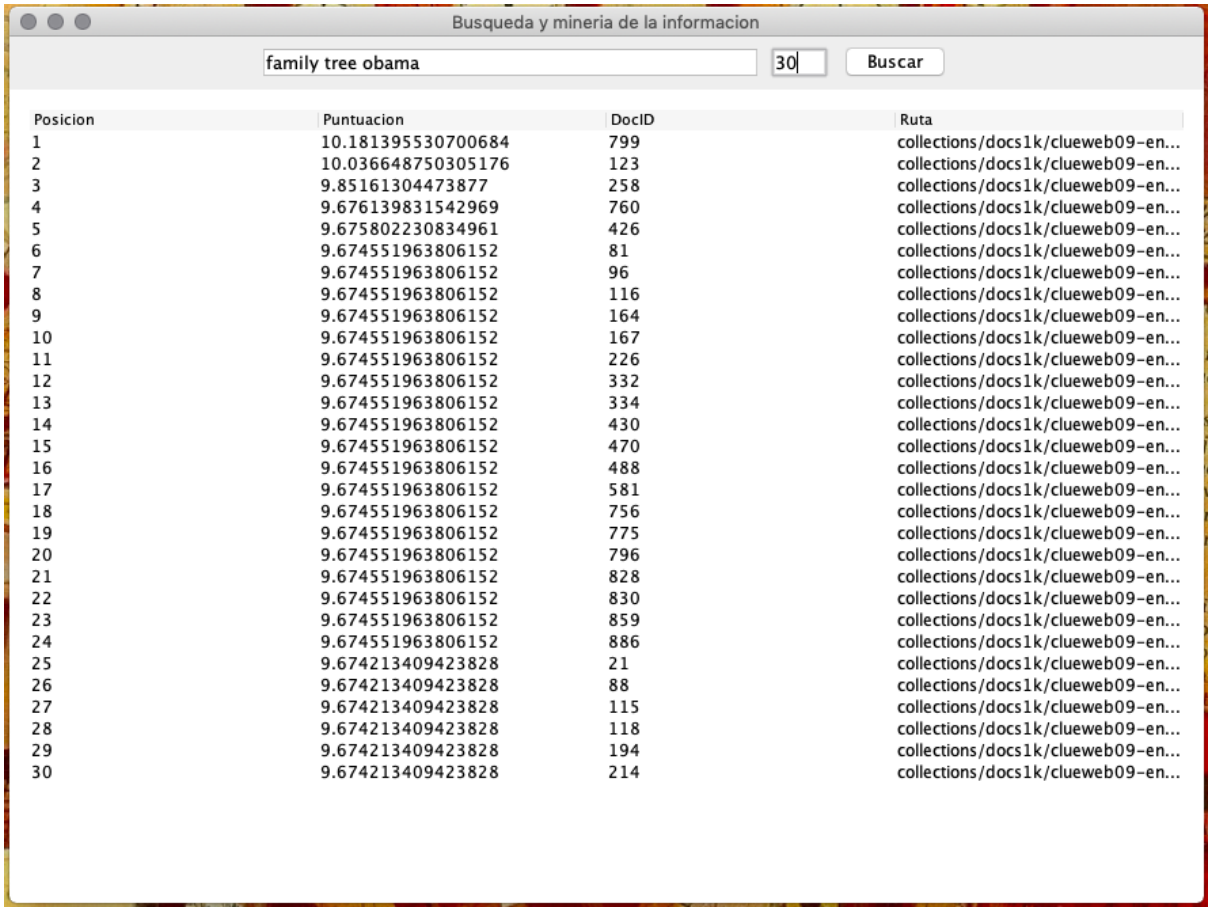


Figura 3: Número de apariciones de las palabras por documento.

Figura 4: UI de la creación del índice



Posicion	Puntuacion	DocID	Ruta
1	10.181395530700684	799	collections/docs1k/clueweb09-en...
2	10.036648750305176	123	collections/docs1k/clueweb09-en...
3	9.85161304473877	258	collections/docs1k/clueweb09-en...
4	9.676139831542969	760	collections/docs1k/clueweb09-en...
5	9.675802230834961	426	collections/docs1k/clueweb09-en...
6	9.674551963806152	81	collections/docs1k/clueweb09-en...
7	9.674551963806152	96	collections/docs1k/clueweb09-en...
8	9.674551963806152	116	collections/docs1k/clueweb09-en...
9	9.674551963806152	164	collections/docs1k/clueweb09-en...
10	9.674551963806152	167	collections/docs1k/clueweb09-en...
11	9.674551963806152	226	collections/docs1k/clueweb09-en...
12	9.674551963806152	332	collections/docs1k/clueweb09-en...
13	9.674551963806152	334	collections/docs1k/clueweb09-en...
14	9.674551963806152	430	collections/docs1k/clueweb09-en...
15	9.674551963806152	470	collections/docs1k/clueweb09-en...
16	9.674551963806152	488	collections/docs1k/clueweb09-en...
17	9.674551963806152	581	collections/docs1k/clueweb09-en...
18	9.674551963806152	756	collections/docs1k/clueweb09-en...
19	9.674551963806152	775	collections/docs1k/clueweb09-en...
20	9.674551963806152	796	collections/docs1k/clueweb09-en...
21	9.674551963806152	828	collections/docs1k/clueweb09-en...
22	9.674551963806152	830	collections/docs1k/clueweb09-en...
23	9.674551963806152	859	collections/docs1k/clueweb09-en...
24	9.674551963806152	886	collections/docs1k/clueweb09-en...
25	9.674213409423828	21	collections/docs1k/clueweb09-en...
26	9.674213409423828	88	collections/docs1k/clueweb09-en...
27	9.674213409423828	115	collections/docs1k/clueweb09-en...
28	9.674213409423828	118	collections/docs1k/clueweb09-en...
29	9.674213409423828	194	collections/docs1k/clueweb09-en...
30	9.674213409423828	214	collections/docs1k/clueweb09-en...

Figura 5: Ejemplo de resultado de búsqueda

Figura 6: Página web accedida a través de los resultados de búsqueda