Memoria de la práctica 1

Búsqueda y minería de la información

Jorge Cifuentes, Alejandro Martín, pareja 01

Ejercicio 1

Sobre la búsqueda:

Hemos implementado la búsqueda de LuceneEngine como una búsqueda booleana donde cada término debería aparecer (ver código). Esto hace que cuantos más y más veces aparezcan los términos de la consulta, más alto será el score. Hay otras opciones como un PhraseQuery para buscar una frase entera, pudiendo modificar la distancia máxima deseada entre términos.

Ejercicio 2

Sobre el modelo vectorial:

Para calcular la similitud por coseno de dos vectores, en este caso del vector documento d y del vector query q, la fórmula es la siguiente:

$$\cos(d,q) = \frac{\vec{d}\cdot\vec{q}}{mod(\vec{d}) * mod(\vec{q})}$$

Para calcular estos productos de vectores, hemos decidido usar el modelo tf-idf, que convierte documentos y términos de búsqueda en vectores, y cuya versión general es:

$$tf(term,doc) = \begin{cases} 1+\log_2(frec(term,doc)) & si\ frec(term,doc) > 0 \\ 0\ en\ otro\ caso \end{cases}$$

$$idf(term) = \log_2 \frac{\#Docs\ corpus}{\#Docs(term)}$$

Interpretando corpus como toda nuestra colección de documentos y *Docs(term)* los documentos que contiene un término term.

Con estos datos podemos sustituir la fórmula del coseno de similitud, para una query de longitud q y una colección de términos de longitud c:

$$\cos(d,q) = \frac{\vec{d}\cdot\vec{q}}{mod(\vec{d}) * mod(\vec{q})} = \frac{\sum_1^q tf(d,t_q) * idf(t_q)}{\sqrt{\sum_1^c tf(d,t_c)^2 * idf(t_c)^2} * \sqrt{mod(\vec{q})}}$$

En la parte de abajo, aunque el tf-idf se calcula sobre toda la colección, por practicidad y eficiencia lo hemos implementado solo sobre los términos contenidos en el documento d, ya que los demás términos tendrán un valor de tf de 0 (al tener frecuencia 0), y no aportarán al sumatorio.

Hemos usado esta versión como primea aproximación, creando sendas funciones estáticas para tf e idf en la clase CosineSimilarity. Por ejemplo, esta variante ejecutada sobre una colección de archivos igual que la del ejemplo del hockey de las transparencias de teoría:

```
 Top 8 results for query "liga street hockey"
              0.5773502691896257      /home/jorgecf/Documents/bminf/bmi-p1-01/collections/hockey/documento2.txt
              0.5163977794943223      /home/jorgecf/Documents/bminf/bmi-p1-01/collections/hockey/documento1.txt
              0.447213595499958       /home/jorgecf/Documents/bminf/bmi-p1-01/collections/hockey/documento3.txt
              0.40824829046386296     /home/jorgecf/Documents/bminf/bmi-p1-01/collections/hockey/documento4.txt
```

Como vemos son los mismos resultados. Al trabajar con colecciones más grandes, como la de docs.zip, nos hemos dado cuenta de que surge un problema si uno de los términos de la query no está en ningún documento, ya que en ese caso el idf sería un infinito. Para solucionarlo modificamos idf de tal manera:

$$idf(term) = 1 + \log_2 \frac{\#Docs\ corpus}{1 + \#Docs(term)}$$

Con el 1 de abajo conseguimos que no pueda haber divisiones entre cero, ya que un documento siempre va a estar en, al menos, 0 documentos. Con el primer 1, conseguimos que el idf no sea negativo. Esta es la versión que hemos usado finalmente, aunque podría ser refinada aún más, por ejemplo, para que no penalice de manera tan excesiva a los documentos largos.

En cuanto a nuestro código, construimos el coseno en dos partes: primero, al crear el índice (*LuceneIndexBuilder.storeVectorMod*) calculamos el módulo del documento d, es decir, la primera raíz de abajo del coseno. Esto es lo que más tiempo toma, ya que hay que calcular el tf-idf para todos los términos de cada documento, y por eso se ejecuta al crear el índice, lo que se ejecutaría offline. El archivo se guarda en la carpeta del índice que crea Lucene. Por otro lado, el sumatorio de encima, es decir, el relacionado con la query, se calcula cuando se realice la query, igual que el módulo de q (lo que se haría online).

Nota: hemos detectado que para colecciones tan largas como docs.zip, StoreVectorMod tarda **mucho,** lo cual es asumible al ser en offline, pero lo hemos desactivado por defecto.

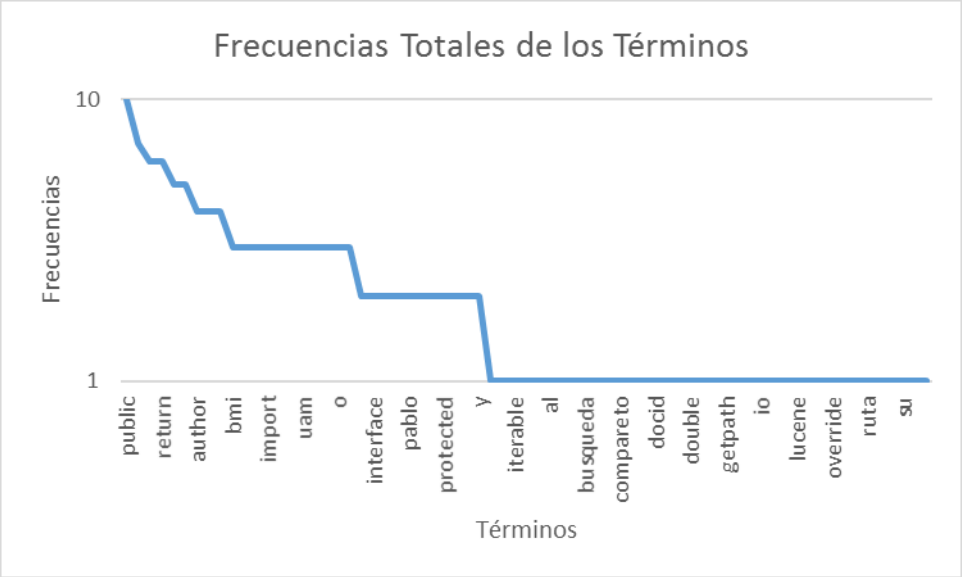
Una salida ejemplo:

```
<terminated> TestEngine [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (16 feb. 2017 17:43:46)
Indexing (999): C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-enwp03-27-20810.html
Indexing (1000): C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-enwp03-29-5471.html
Indexing (1001): C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-enwp03-29-5472.html
Indexing (1002): C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-enwp03-36-9316.html
------------------------------
Collection: collections/docs.zip

  Most frequent terms:
       family  175862
       tree    46734
       history 45776
       genealogy       45607
       surname 44968

  A few term frequencies for docID = 0 - es.uam.eps.bmi.search.index.freq.lucene.LuceneFreqVector@14a2f921: iterable (1) pablo (1) package (1) public (2) ranking (1)

  Frequency of word "seat" in document 0 - es.uam.eps.bmi.search.index.freq.lucene.LuceneFreqVector@3c87521: 0

  Total frequency of word "seat" in the collection: 1445 occurrences over 119 documents

Warning: created index pointing to prueba, no index files there.
        at: es.uam.eps.bmi.search.index.lucene.LuceneIndex.load(LuceneIndex.java:47)
        at: es.uam.eps.bmi.search.index.AbstractIndex.load(AbstractIndex.java:25)
        at: es.uam.eps.bmi.search.index.AbstractIndex.<init>(AbstractIndex.java:15)
        at: es.uam.eps.bmi.search.index.lucene.LuceneIndex.<init>(LuceneIndex.java:31)
        at: es.uam.eps.bmi.search.lucene.LuceneEngine.loadIndex(LuceneEngine.java:56)
        at: es.uam.eps.bmi.search.AbstractEngine.loadIndex(AbstractEngine.java:26)
        at: es.uam.eps.bmi.search.AbstractEngine.<init>(AbstractEngine.java:18)
        at: es.uam.eps.bmi.search.lucene.LuceneEngine.<init>(LuceneEngine.java:24)
        at: es.uam.eps.bmi.search.test.TestEngine.testCollection(TestEngine.java:86)
        at: es.uam.eps.bmi.search.test.TestEngine.main(TestEngine.java:32)
This is just an informative warning and requires no immediate action ;-)

  No index found in prueba!

  Top 5 results for query "obama family tree"
               10.07801342010498       C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-en0010-79-2218.html
               9.857902526855469       C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-en0001-02-21241.html
               9.673702239990234       C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-en0010-57-32937.html
               9.550849914550781       C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-enwp01-59-16163.html
               9.550849914550781       C:\Users\eps\Desktop\bminf-master\bmi-p1-01\tmp\clueweb09-enwp02-06-15081.html

Indexing (986): https://en.wikipedia.org/wiki/Entropy
Indexing (987): https://en.wikipedia.org/wiki/Information_theory
Indexing (988): http://www.uam.es/ss/Satellite/EscuelaPolitecnica/es/home.htm
Indexing (989): http://sigir.org/sigir2017/submit/call-for-full-papers/
Indexing (990): http://nlp.stanford.edu/IR-book/
Indexing (991): http://www.fmp.es
Indexing (992): http://www.laliga.es
------------------------------
Collection: collections/urls.txt

  Most frequent terms:
       family  175862
```

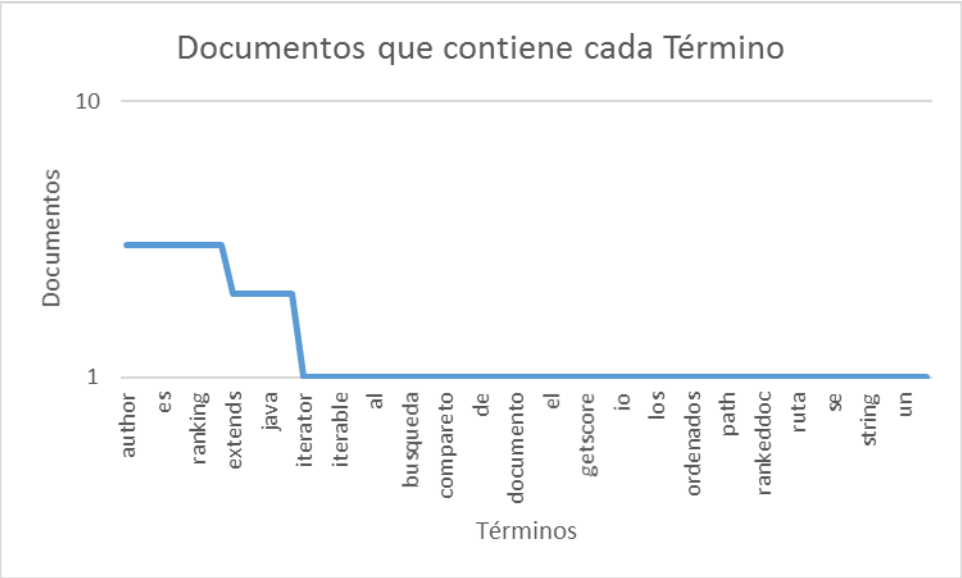
Ejercicio 3.1

Para este apartado se ha implementado la clase *es.uam.eps.bmi.search.test.TeamStats* con un método main para generar los dos ficheros que se enuncian a continuación (cada uno con su gráfico correspondiente en escala logarítmica):

- Frecuencias totales en la colección de los términos (termfrec.txt): este fichero representa el número de veces que aparece cada término en el total de la colección de documentos. En el fichero se generan dos columnas distintas, cada una de ellas corresponde a un eje del gráfico: el eje x corresponde a los términos, mientras que el eje y corresponde a la frecuencia de cada uno de ellos en el total de la colección. Como podemos ver, la gráfica sigue una distribución exponencial, ya que en el fichero generado, los términos se ordenan por frecuencia de mayor a menor.



- Número de documentos que contiene cada término (termdocfrec.txt): este fichero representa el número de documentos en el que aparece cada término. En él, se incluyen dos columnas, las cuales corresponden a un eje del gráfico: el eje x representa los términos de la colección, mientras que el eje y corresponde al número de documentos que aparece cada término. Ésta gráfica, como la anterior, sigue también una distribución exponencial, debido a que los términos del fichero se ordenan en orden decreciente por el número de documentos en los que aparece cada uno de ellos.



Debido al hecho de que utilizar todos los documentos de docs.zip para generar los ficheros tardaba bastante en ejecutar, se ha decidido realizar las gráficas con los documentos del ranking. Independientemente de que se ejecuten con todos los ficheros o con los del ranking, la distribución de las gráficas no va a cambiar y seguirá siendo exponencial.

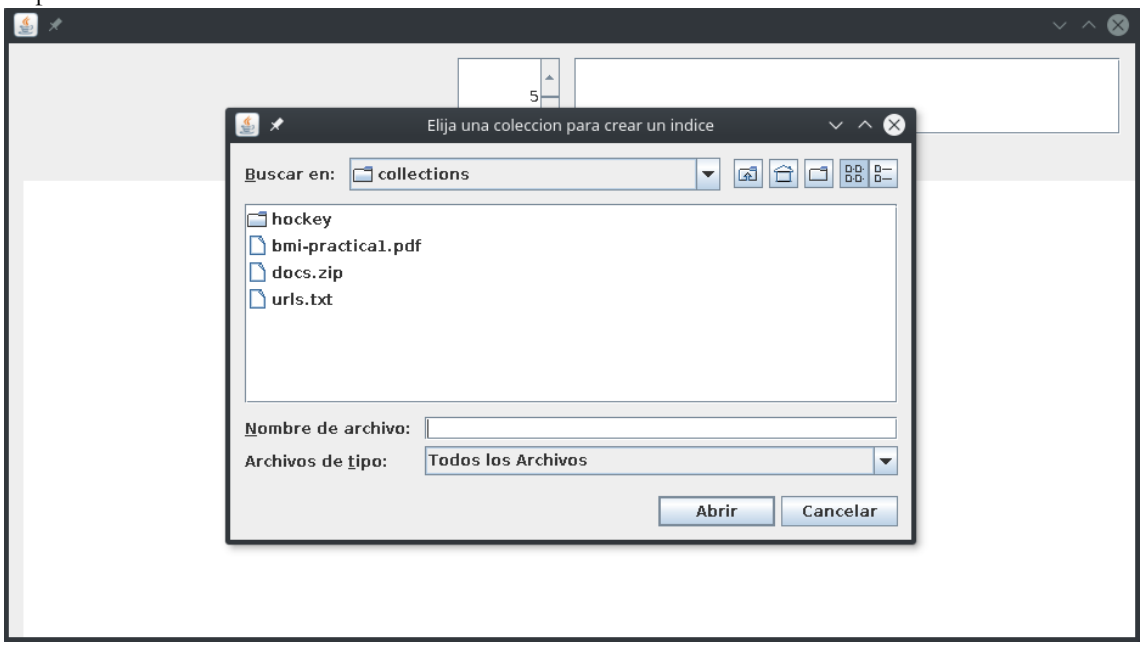
Ejercicio 3.2

Se ha implementado un nuevo caso de carga e indexado de documentos pdf. Se le pasa al creador de índices de igual manera que los anteriores tres casos, y con pdfbox se lee el contenido.

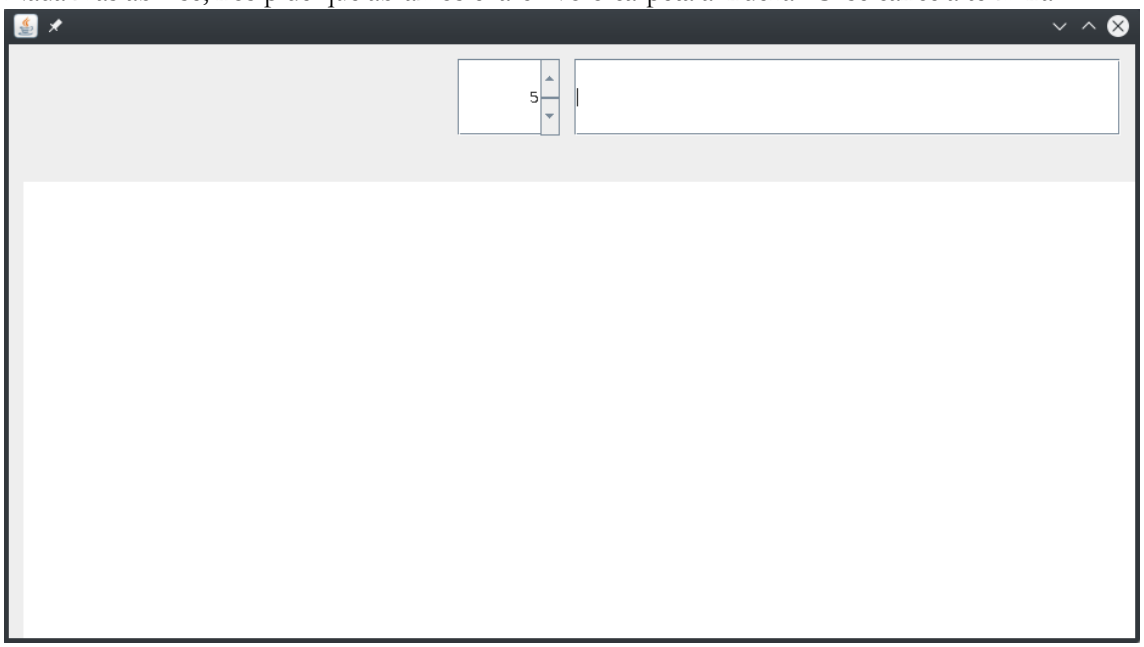
Ejercicio 3.3

Hemos implementado una simple interfaz creada en Java Swing que funciona como un buscador. Se ha implementado intentando seguir un esquema modelo-vista-controlador (MVC): el modelo nuestro código previo, la vista el código de Swing en GUI.java y el controlador la clase Controller.java que conecta ambas partes.

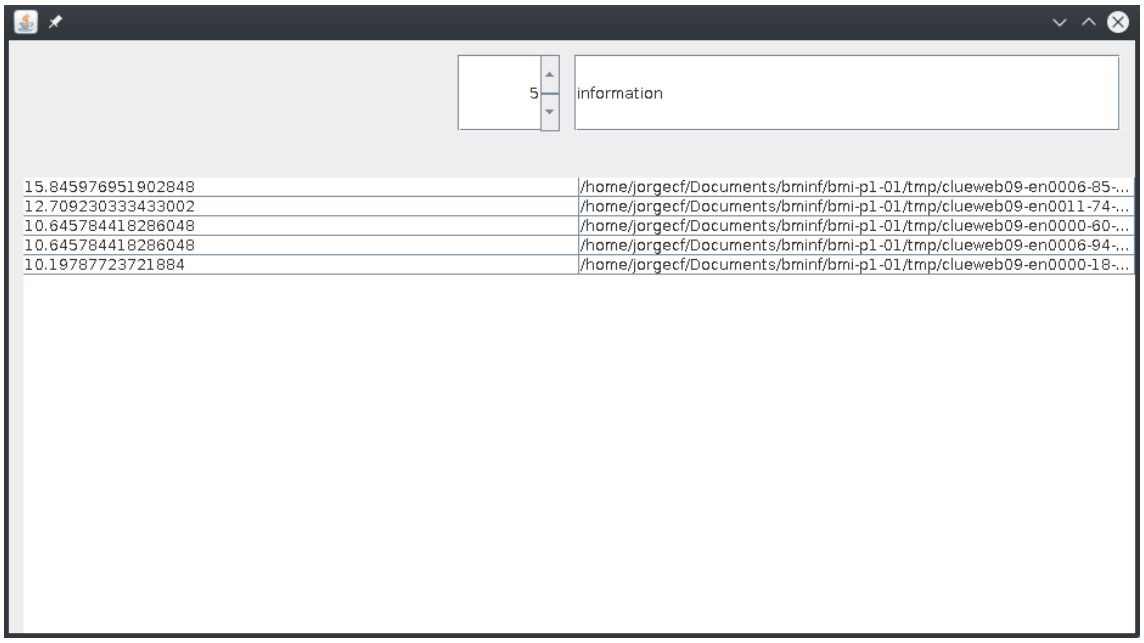
Capturas del funcionamiento:



Nada más abrirse, nos pide que abramos el archivo o carpeta a indexar. Si se cancela termina.



Se abre con un campo de texto de búsqueda, un spinner que modula los resultados y la tabla que muestra los resultados.



Al buscar, aparecen los resultados ordenados por su score.

Si clickamos en una fila, se abre el archivo local como en la imagen o una url en el navegador por defecto del equipo.