20-2-2018

Búsqueda y minería de Información

Practica 1

Óscar García de Lara Parreño
Jorge Gómez Conde
GRUPO 11

Tabla de contenido

Ejercicio 2 – Modelo vectorial

2.1 Producto vectorial

En el desarrollo del modelo vectorial, hemos decido implementarlo en orientación a términos. La lógica se encuentra implementada en VSMEngine.java y las clases contenidas en el directorio /es/uam/eps/bmi/ranking/impl, más concretamente la clase IMPLDocVector.java:

- VSMENgine.java: Se encarga de recorrer los documentos del índice en la orientada a términos. Se ocupa de llamar a IMPLDocVector para generar cada vector de documento.
- IMPLDocVector.java: esta clase guarda toda la información relativa a generar una puntuación del documento en el índice. Cada vez que se añade una palabra al vector, la clase solicita la frecuencia de la palabra del documento y la frecuencia total de la palabra, guardando en el vector la puntuación de la función tf*idf del término en el documento.

Una vez revisado todos los términos de la consulta, podemos llamar al método sumPuntuaciones, en un primer momento planteado para este apartado, devuelve el la suma de tf-idf de los términos de la query que contiene el documento, ver numerador del coseno.

A continuación, especificamos matemáticamente las funciones utilizadas:

$$tf(t,d) = 1 + log2(frec(t,d)) \; si \; frec(t,d) > 0, \; en \; otro \; caso \; 0$$

$$idf(t) = log2(1 + \frac{\#\, Documentos}{1 + \#Documentos\ con\ t})$$

2.2 Coseno

En este apartado hemos realizado unas pequeñas modificaciones sobre el apartado anterior y sobre LuceneBuildier para generar el módulo del documento y guardarlo en un documento junto al índice de Lucene.

En LuceneIndex generamos un HashMap para almacenar a cada docId su modulo que hemos leído del fichero .

En IMPLDocVector hemos modificado la función sumPuntuaciones para generar el valor del coseno. Este método de la clase utilizara el vector de consulta ya generado por el índice si al construir el vector introducimos este dato por parámetro. El método sumPuntuaciones devolverá el coseno si contiene un vector de términos, si no, devolverá el numerador como en el apartado anterior.

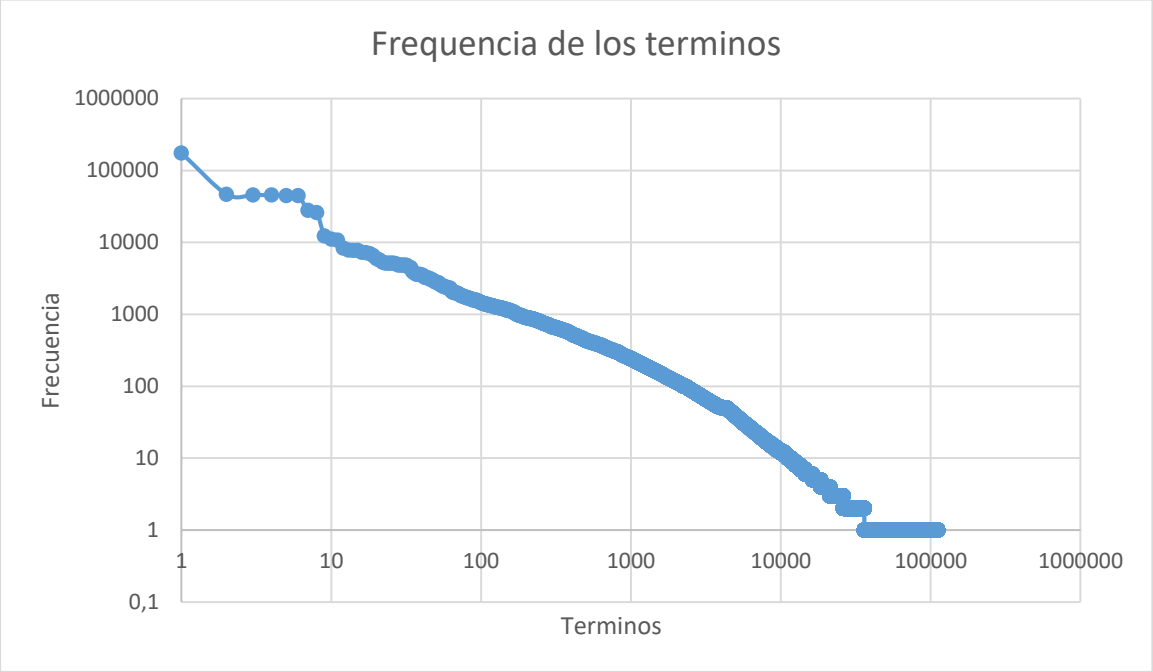
A continuación, especificamos matemáticamente las funciones utilizadas:

$$\cos(d,q)\ \alpha \frac{\sum_{t \in q} tf(t,d) * idf(t)}{\sqrt{\sum_{t \in d} (f(t,d) * idf(t))^2}}$$

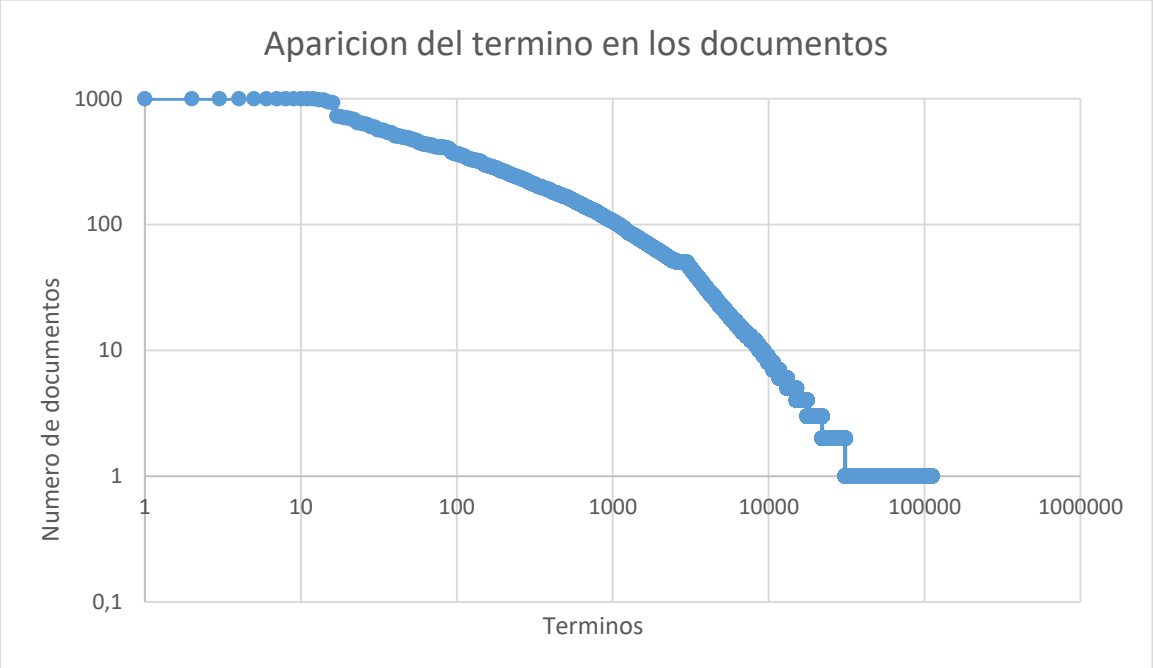
Ejercicio 3 - Extensiones

3.1 Frecuencias

El ejercicio no especifica que índice tenemos que usar para mostrar las gráficas, así que hemos optado por el del zip que contiene muchos más documentos.



La grafica sigue la tendencia de la ley de Ley de Zipf pero no la cumple exactamente ya que como se puede observar del segundo al sexto, ambos inclusive, son prácticamente planos.



En la segunda grafica se puede ver que hay ciertas palabras que salen en todos los documentos, lo mas seguro que con un Stopwords mas especifico desaparezcan.

3.2 Stopwords y Stemming

Hemos probado usando EnglishAnalyzer y SpanishAnalyzer para ver la diferencias con el estándar en los documentos del zip. Para ello hemos añadido un jar nuevo que no ha sido entregado al considerar que la versión final tiene que contener solo la implementación de LuceneBuilder y LuceneEngine con el StandardAnalyzer.

StandardAnalyzer

Most frequent terms:

1. family 175800
2. tree 46731
3. history 45770
4. genealogy 45549
5. surname 44968

Total frequency of word "seat" in the collection: 1428 occurrences over 119 documents

LuceneEngine: top 5 for query 'obama family tree'

10.169378280639648 resources/docs1k.zip/clueweb09-en0010-79-2218.html

10.021995544433594 resources/docs1k.zip/clueweb09-en0001-02-21241.html

9.842021942138672 resources/docs1k.zip/clueweb09-en0010-57-32937.html

9.691999435424805 resources/docs1k.zip/clueweb09-enwp01-59-16163.html

9.691673278808594 resources/docs1k.zip/clueweb09-enwp02-06-15081.html

EnglishAnalyzer

Most frequent terms:

1. famili 184216
2. tree 46948
3. genealog 46127
4. histori 46040
5. surnam 45237

Total frequency of word "seat" in the collection: 2126 occurrences over 149 documents

LuceneEngine: top 5 for query 'obama family tree'

9.882296562194824 resources/docs1k.zip/clueweb09-en0010-79-2218.html
9.821781158447266 resources/docs1k.zip/clueweb09-en0001-02-21241.html
9.600945472717285 resources/docs1k.zip/clueweb09-en0010-57-32937.html
9.53982162475586 resources/docs1k.zip/clueweb09-enwp02-06-15081.html
9.53982162475586 resources/docs1k.zip/clueweb09-enwp01-49-16274.html

SpanishAnalyzer

Most frequent terms:

1. family 175800
2. tree 46731
3. history 45770
4. genealogy 45549
5. surnam 45237

Total frequency of word "seat" in the collection: 1428 occurrences over 119 documents

LuceneEngine: top 5 for query 'obama family tree'

10.163031578063965 resources/docs1k.zip/clueweb09-en0010-79-2218.html

10.014495849609375 resources/docs1k.zip/clueweb09-en0001-02-21241.html

9.853029251098633 resources/docs1k.zip/clueweb09-en0010-57-32937.html

9.658024787902832 resources/docs1k.zip/clueweb09-enwp01-59-16163.html

9.657711029052734 resources/docs1k.zip/clueweb09-enwp02-06-15081.html

Si comparamos el Standard con Spanish vemos que la única diferencia es la puntuaciones que no afecta al orden del top 5, esto indica que la gran mayoría de documentos proporcionados no están en español por tanto el Stopwords y Stemming que aplica no afecta porque no reconoce los términos.

En cambio con el EnglishAnalyzer sí que hay mas diferencia se nota en el top 5 de los términos mas frecuentes ya que las palabras tienen aplicados Stemming, lo que hace que aumente la frecuencia y cambie el orden del top 5, ya que reconoce más palabras con raíz genealogy que con history. Las puntaciones de la consulta cambian drásticamente pero el top 3 se mantiene el mismo documento, haciendo que sean otros el cuarto y el quinto, lo mas seguro que sea debido a la palabra family.

3.2 Modelos IR

IBM 25

Hemos cambiado el IndexSearcher y el IndexWriterConfig con setSimilarity(new BM25Similarity()); y no hay diferencia en las puntaciones

BooleanSimilarity

Con este modelo si que hay mas diferencia ya que la puntuación va de 0 - Nºterminos de la query, por es un modelo malo para query con pocos términos ya que hay mas probabilidad que varios documentos contengan todas las palabras y se produzcan empates.

3.2 Documentos admitidos

Hemos admitido que cuando lea una carpeta, los documentos leidos puedan ser también un pdf, Para ello usamos una librería externa llamada PDFBox, con ella limpiamos el pdf para dejar solos los términos y poder añadirlo al índice.

3.3 UI

Para ejecutar nuestra UI hay un requisito y es comprobar el Enum *IndexSorce*, para ello solo hay que comprobar que las rutas lleven a una carpeta con un índice Lucene ya creado ya que entendemos que una UI no se encarga de crear un índice, básicamente seguimos el esquema proporcionado.

También se podrían borrar, añadir o modificar el nombre de las variables del Enum ya que la UI los carga dinámicamente.

Una vez completado el requisito podemos ejecutar *InterfazUI* y nos saldrá la siguiente pantalla.

```
Bienvenido:

Opciones de indice:

Id: 0 Nombre: SRC
Id: 1 Nombre: DOCS
Id: 2 Nombre: URLS
Introduzca id:
```

Solo valdrá introducir un numero que este entre el rango de los id, otro tipo de carácter no avanzara y otro número hace que te lo pida de nuevo.

```
Accion a realizar:

Opcion 1: Insertar query
Opcion 2: Cambiar indice
Opcion 3: Salir

Introduzca opcion(numero):
```

La siguiente acción sea insertar un el numero de la Opción:

1. La primera nos da la elección de insertar una consulta.
2. Nos lleva al paso anterior para cambiar el índice para la consulta.
3. Termina la ejecución del programa.

Vamos a elegir la primera opción ya que es la única que nos permite hacer mas cosas no explicadas.

```
Query, separar por espacios las distintas palabras: Information theory
```

Nos saldría la siguiente línea y como se puede observar nos permite escribir las palabras de la consultas separadas por espacios.

```
Query procesada:'information theory'
    Score: 0.2701743110126112 Ruta: https://en.wikipedia.org/wiki/Information_theory
    Score: 0.26889750620929403 Ruta: https://nlp.stanford.edu/IR-book/
    Score: 0.17556624015767278 Ruta: http://sigir.org/sigir2017/submit/call-for-full-papers/
    Score: 0.15073585435490558 Ruta: https://en.wikipedia.org/wiki/Entropy


Accion a realizar:

Opcion 1: Insertar query
Opcion 2: Cambiar indice
Opcion 3: Salir

Introduzca opcion(numero):
```

La query es procesada para evitar caracteres no alfanuméricos y ponerla en minúsculas. En el caso de enlaces se podrían pinchar y llevaría a la pagina web correspondiente, esto no podría estar disponible en cualquier IDE en nuestro caso usamos IntelliJ IDEA.

Vamos a hacer otra consulta pero con una palabra que no existe y símbolos no admitidos.

```
Query, separar por espacios las distintas palabras: fesdfe*e!!efdas !!¿¿ hola

Query procesada:'fesdfeeefdas  hola'
Sin resultados
```

Se puede observar que borra los símbolos dejando solo los admitidos, y como la consulta no obtiene resultados, te muestra un mensaje.