UNIVERSIDAD DE GRANADA

Recuperación de Información

Práctica 2

Preprocesado de documentos

Parte II. Análisis del texto

David Carrasco Chicharro
Daniel Terol Guerrero

1. Estudio estadístico

Sobre un documento de la práctica anterior ("The Tragedy of Romeo and Juliet") se han ejecutado distintos analizadores ya predefinidos en Lucene 8, además del propio analizador realizado para la práctica 1. Los analizadores utizados han sido: SimpleAnalyzer, StopAnalyzer, StandardAnalyzer y WhiteSpaceAnalyzer.

En primer lugar hemos hecho un recuento de los 10 términos más frecuentes con nuestro analizador (AnalizadorP1) y los hemos comparado con la frecuencia con la que aparecen dichos términos al procesarlos con el resto de analizadores.

	An.P1	**SimpleAn.**	**StopAn.**	**StandardAn.**	**WhiteSpaceAn.**
and	708	720	0	0	468
the	679	681	0	0	601
i	575	658	658	586	0
to	538	577	0	0	451
a	460	470	0	0	396
of	401	401	0	0	369
my	359	360	360	360	310
that	347	355	0	0	251
is	344	349	0	0	295
in	319	320	0	0	257

Tabla 1: Términos más frecuentes y ocurrencias por analizador

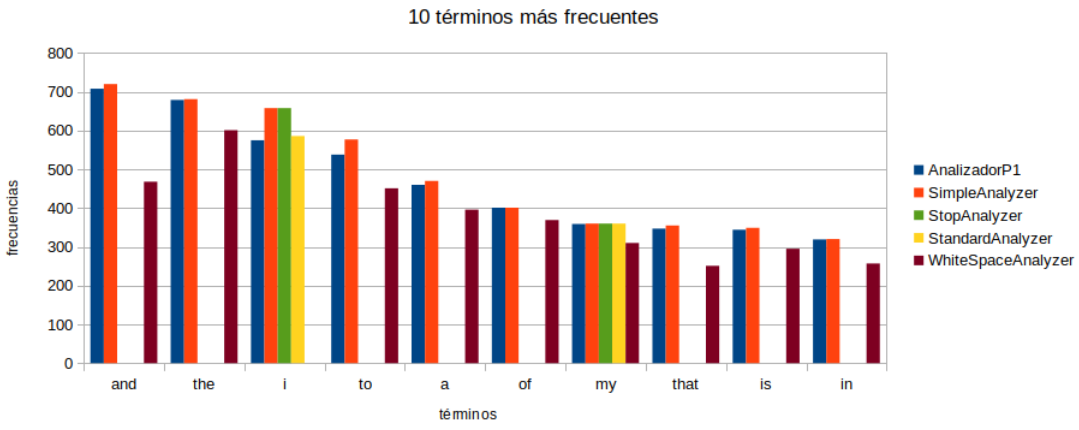

Figura 1: Gráfica de términos más frecuentes

En la tabla y gráfica anteriores podemos observar que tanto StandardAnalyzer como StopAnalyzer eliminan palabras vacías, por lo que aparecen con frecuencia igual a 0 los términos: and, the, to, a, of, that, is, in.

WhiteSpaceAnalyzer tiene para cada término menos frecuencia que el resto. Esto se debe a que los separadores que son distintos a los espacios en blanco (puntos, comas, etc.) se toman como parte de cada término, por lo que para uno, en este analizador, se toman como varios distintos; así, el término 'i', en WhiteSpaceAnalyzer, se toma como *I*, *I,*,*I;*, *'I*, etc.

SimpleAnalyzer y nuestro analizador de la primera práctica arrojan resultados muy parecidos, ya que el cometido era similar: dividir el texto separando por todo aquello que no sean letras y convertir a minúsculas. Aún así el analizador predefinido por Lucene afina con más precisión y da algunas ocurrencias más por cada término.

El segundo estudio estadístico realizado se ha basado en ver cuántos términos produce cada analizador.

Analizador	**Términos**
KeyWordAnalyzer	1
StopAnalyzer	3570
SimpleAnalyzer	3603
StandardAnalyzer	3740
AnalizadorP1	3946
WhiteSpaceAnalyzer	6392

Tabla 2: Términos totales por analizador

Figura 2: Gráfica de términos totales por analizador

StandarAnalyzer contiene un mayor número de términos distintos que StopAnalyzer, puesto que no considera determinantes, abreviaciones, genitivos, etc. como términos tal cual, sino que los considera como términos enteros (*beauty's*, *she'll*, ...) de modo que con StopAnalyzer queda un léxico más probre.

SimpleAnalyzer, por el motivo ya explicado dada su mayor precisión, obtiene menos términos que nuestro analizador.

WhiteSpaceAnalyzer, como era de esperar, contiene un número de términos muy elevado puesto que el texto utilizado contiene una gran cantidad de separadores que este analizador no tiene en cuenta, lo que da lugar a que una misma palabra se cuente varias veces de diversos modos distintas.

Por último cabe destacar KeyWordAnalyzer, que no se mencionó en el análisis anterior, puesto que considera todo el texto como un único token.

2. Aplicación de filtros

En este ejercicios hemos seleccionado un breve texto para probar el efecto de distintos filtros de Lucene:

- ***StopFilter.*** Elimina las palabras vacías.
- ***LowerCaseFilter.*** Transforma todos los caracteres alfabéticos a minúscula.
- ***ShingleFilter.*** Selecciona cada término y produce dos: a él mismo y a este en pareja con el siguiente término. Ejemplo: "Have a good day" = Have — Have a — a — a good — good — good day — day
- ***SynonymFilter.*** Crea un mapa de sinónimos.
- ***SnowballFilter.*** Realiza un proceso de Stemming con un conjunto de palabras en inglés.
- ***CommonGramsFilter.*** Utiliza un conjunto de palabras (palabras vacías) de modo que si una palabra del conjunto aparece en el texto, la muestra unida a la palabra anterior, sola, y unida a la palabra siguiente (A_x — x — x_B)

- ***NGramTokenFilter.*** Dada una palabra de L caracteres y definido el corte en X caracteres: elimina las palabras con $L < X$, muestra tal cual aquellas con $L = X$, y muestra N cortes de las palabras con $L > X$, donde $N = L - X + 1$.

Las salidas producidas por estos filtros se muestran por pantalla al ejecutar el programa.

3. Analizador propio

En este ejercicio hemos elegido *CustomAnalyzer* para realizar nuestro analizador propio. Para ello hemos encadenado distintos filtros con los cuales hemos convertido el texto a minúscula; eliminado un conjunto de palabras vacías, no solo el predeterminado por *StopFilter*, sino que hemos añadido un archivo con más términos: stopwords.txt; y aplicado un stemmer para extraer la raíz de las palabras.

```
Analyzer ana = CustomAnalyzer.builder(Paths.get("."))
.withTokenizer(StandardTokenizerFactory.class)
.addTokenFilter(LowerCaseFilterFactory.class)
.addTokenFilter(StopFilterFactory.class,
    "ignoreCase", "false", "words",
    "stopwords.txt", "format", "wordset")
.addTokenFilter(SnowballPorterFilterFactory.class,
    "language", "English")
.build();
```

La salida produce un texto customizado en base a los filtros aplicados.

4. Ejecución

Para ejecutar el programa hay que introducir el siguiente comando en el terminal:

```
  $ ./ejecucionP2.sh docs
```

Esto volcará los datos de los análisis del primer ejercicio en ficheros CSV en `docs/contador/`, y mostrará por pantalla tanto las salidas de los distintos filtros del segundo ejercicio como el texto customizado del tercero.