UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

CAIM Lab - Sesión 2:
Programming with ElasticSearch

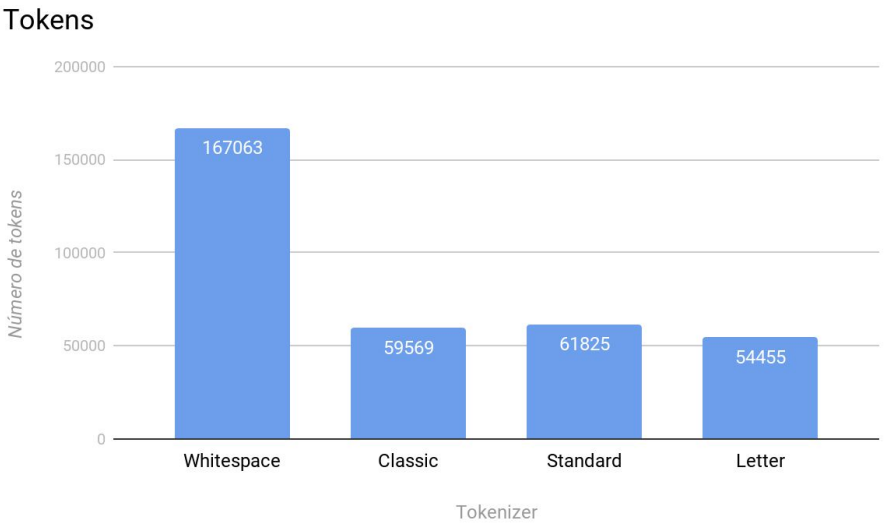
Alberto Giménez
Yimin Pan
Q1 2019-2020

Introducción

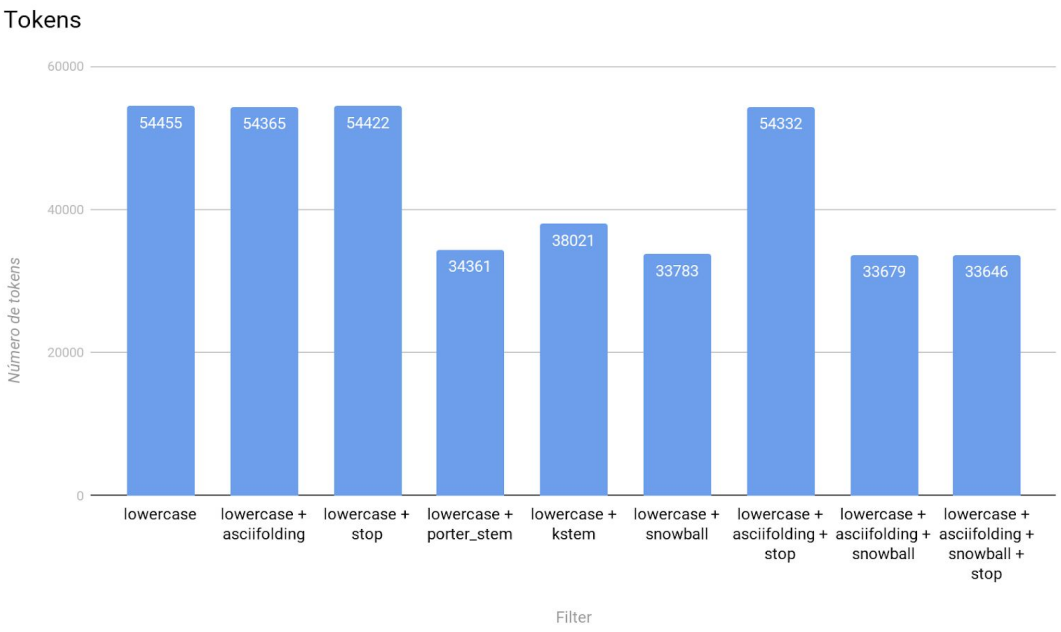
El objetivo de esta práctica es aprender a configurar ElasticSearch para aplicar diferentes filtros y "tokenizadores" y ver cómo afectan en el índice. En una segunda parte de la práctica, completaremos el código proporcionado para calcular el tf-idf y la similitud entre documentos.

Análisis del índice

En esta parte, vamos a analizar cómo afectan los diferentes flags (*--token* y *--filter*) a los tokens. Primero vamos a probar los diferentes "tokenizers" para *novels* y ver el número de palabras diferentes que quedan.



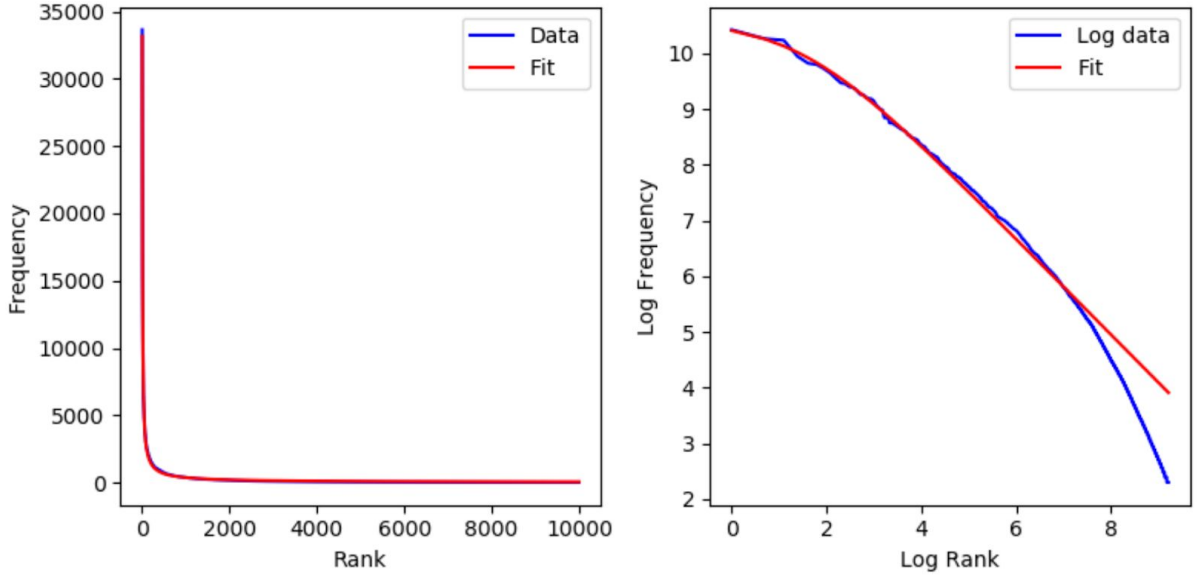
Como observamos, el tokenizer más agresivo es *"Letter"*, que elimina todas las palabras que tenga un carácter no alfabético. Al contrario, el whitespace separa los términos por cada espacio blanco que encuentra, sin eliminar nada, entonces lógicamente es el que deja más palabras.
Letter va bastante bien para lenguajes europeos, por lo que es el que utilizaremos para analizar los diferentes filtros, tanto individualmente como combinados con otros filtros. Veamos la comparativa en el siguiente gráfico.



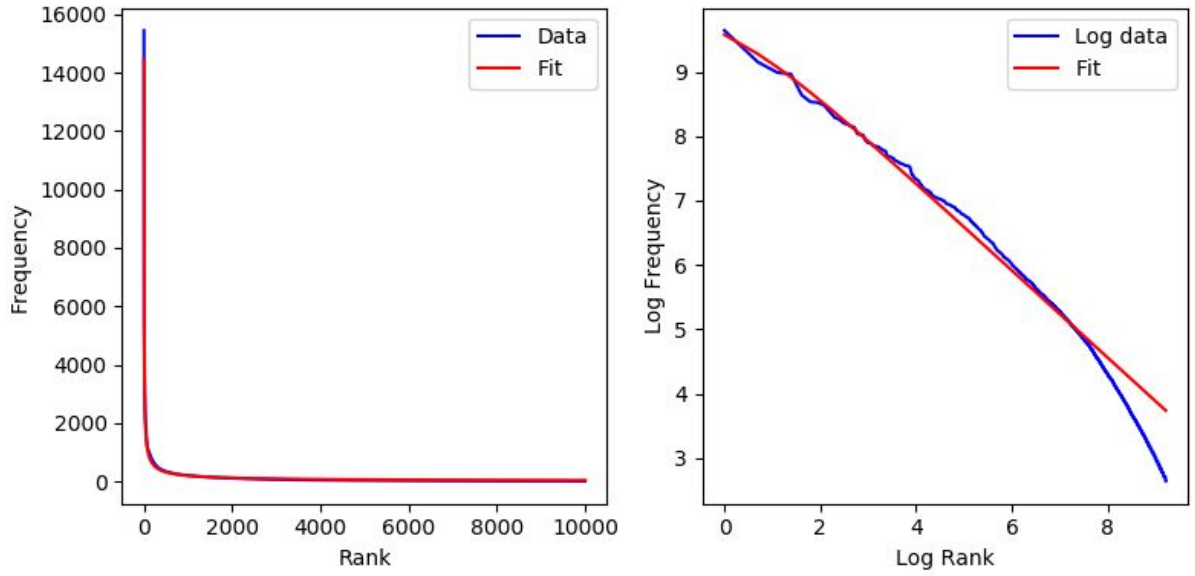
Como vemos el filtro más agresivo es *snowball.* Al combinarlo con todos los demás, obtenemos el mejor resultado en cuanto a eliminación de palabras. Sin embargo, es importante tener en cuenta el orden de aplicación de los filtros y de su utilización, ya que podemos llegar a eliminar palabras que nos interesan, como acrónimos o números, o bien empobrecer el contenido léxico debido a la eliminación o transformación de ciertas palabras, como por ejemplo los acentos o palabras con significado muy distinto que puedan compartir la raíz.

Como vemos en el output, la palabra más frecuente en *novels* es “I”, mientras que en *arxiv corpus* es “we”, debido a su orientación más científica.

Si estudiamos si se cumple la ley de Zipf (caso con todos los filtros y rango 10000), vemos que se sigue cumpliendo. Es difícil determinar si empeora o mejora respecto a la versión de la práctica anterior, ya que el resultado parece muy parecido, aunque se podría decir que ahora se ajusta ligeramente mejor.



Plot de novels con tokenizer letter y todos los filtros



Plot de novels de la práctica anterior

Cálculo del tf-idf y similitud coseno

En esta segunda parte de la práctica, completaremos el código proporcionado en *TFIDFViewer.py* que calcula el Tf-Ifd de los términos de los documentos para calcular su vector de pesos y obtener la similitud coseno entre dos documentos.
En general, no hemos encontrado grandes dificultades para completarlo, ya que en la mayoría de casos era aplicar directamente las fórmulas que aparecen en las transparencias.
En la función para calcular la similitud coseno, estábamos obligados a recorrer como mucho una vez cada vector de pesos. Para conseguirlo, tenemos la ventaja de que los vectores están ordenados alfabéticamente. Para poder hacerlo lineal, hemos utilizado un puntero para cada vector inicializados en la primera posición de sus vectores respectivos. Mientras los punteros apunten a posiciones válidas de los vectores, si las palabras a las que apuntan son diferentes, desplazamos una posición el puntero del vector que tenga un término menor (alfabéticamente hablando). En el caso que sean iguales, actualizamos el resultado y desplazamos ambos punteros. Será más sencillo visualizarlo consultando el código adjunto.

Experimentación

La forma más fácil de ver si nuestro programa funciona, es comparar un documento consigo mismo. Lo hicimos, y la similitud que calculó el programa es lógicamente 1. Por lo que nuestra primera impresión es que el programa funciona.
Además, para cada índice, hemos seleccionado al azar 20 documentos para compararlos entre entre ellos. Hemos calculado la similitud dos a dos (10 comparaciones) y hemos calculado la media (con un script). El resultado que hemos obtenido se muestra en la siguiente tabla:

Ejecución	Arxiv	News	Novels
1	0.0206	0.0096	0.0338
2	0.0453	0.0149	0.0246
3	0.0163	0.0074	0.0316
4	0.0334	0.0152	0.0208
5	0.0276	0.0172	0.0239
6	0.0249	0.0085	0.0394
7	0.0162	0.0167	0.0275

Aquí se observa claramente que los documentos de tipo noticias tienen una similitud más baja que los de otro tipo. Esto es debido a cómo se organiza los documentos de News, están divididos en carpetas y luego subcarpetas de modo que los documentos son muy específicos y cortos.

Observación y conclusión

Después de hacer la experimentación, observamos cosas interesantes. Los documentos de tipo novelas tienen una similitud bastante estable, la varianza de similitud es muy pequeña entre los documentos de este tipo, todos alrededor de 0.03. Esto en realidad lo podemos razonar con la ley de Heap. Las novelas son largas, por lo tanto tienen más número de palabras que documentos de otro tipo, por Heaps sabemos que cuando aumentamos el número total de palabras, el crecimiento de palabras distintas decrece. Por lo que habrás muchas palabras repetidas, esto explica la baja varianza de similitud entre novelas. Un caso contrario sería los documentos de arxiv o news, que son relativamente cortos, podemos encontrar pares de documentos con similitud bastante más variada, ya que dependiendo de los documentos escogidos, el resultado puede ser muy distinto, sobre todo para *arxiv*. Para *news* encontramos también no demasiada varianza aunque en este caso la similitud es mucho menor. Por lo tanto, podemos concluir que la longitud de los documentos afecta a la similitud.
Quizás en la tabla no se ve tan claro ya que cogemos la media de las ejecuciones, pero si ejecutamos el programa con dos documentos se notará más.

Por el otro lado, también cabe decir que “indexar” con tokens y filtros tiene un efecto importante sobre el cálculo de similitud. Cuando más agresivos sean, mayor similitud calculará el programa. Ya que por ejemplo con los verbos, tenemos una multitud de conjugaciones, lo que hará que dos documentos tenga menor similitud. Pero si lo transformamos todos a infinitivo, la similitud obviamente crece. A este proceso se le llama lemmatizing.