Ivan Cala Mesa
Grupo 13

Informe CAIM Sesión 2

Programación con ElasticSearch

Para la realización de este informe se utilizó como fuente de datos el repositorio de novelas, que como se vio en el análisis previo, tenía un comportamiento cercano al esperado por los textos, de acuerdo con la ley de Zipf. Con lo cual, la frecuencia de todas las palabras de dicho repositorio será mostrada, empleando y probando la diferencia entre las diferentes configuraciones que Elasticsearch nos provee para el filtrado y tokenización del texto.

Así pues, asumiremos que la configuración por defecto del índice de Elasticsearch incluye un filtrado del tipo lowercasing, es decir, convierte a minúscula toda letra encontrada en cada string. No solo eso, analizaremos los diferentes parámetros de filtrado asumiendo un diferenciador de palabras de tipo ‘letter’, tal y como está definido en la documentación de Elasticsearch.

Lowercasing	Asciifolding	Stopwords	KStem	Porter	Snowball	Palabras
X						54455
X	X					54365
X	X	X				54332
X	X	X	X			37893
X	X	X		X		34246
X	X	X			X	33660

En la tabla anterior podemos ver cuáles son los efectos de la aplicación de cada uno de los filtros. Además, hemos compuesto el uso de los filtros, siendo el orden de aplicación de izquierda a derecha por columna.
En primer lugar, el filtrado de tipo lowercasing en sí mismo no descarta palabras, sin embargo podemos asumir que hay palabras (como acrónimos) que, puestos en minúscula, se ‘convierten’ en palabras factuales como otra cualquiera. Este es el caso de la palabra ‘it’, la cual podemos encontrar en una posición alta, y podemos asumir que acrónimos como ‘IT’ serían pues transformados e incorporados al recuento de ‘it’.
Podemos ver cómo de los filtrados posteriores que no son una forma de stemming no tienen un impacto (en número de entradas) demasiado alto. Al aplicar Asciifolding, el número de entradas sólo se reduce en 90. Podemos suponer que dada la naturaleza del texto (escrito formal en lengua inglesa) el número de caracteres diferentes no ASCII que podemos encontrar es bastante más reducido. El propio ASCII se diseñó con la intención de poder cubrir las necesidades de la lengua inglesa en textos digitales.
La aplicación del filtro de Stopwords sólo extrae 33 entradas, pero son entradas de las más numerosas. En este caso, pasamos de que la palabra con el mayor número de repeticiones pase de ser 'the' con 206706 apariciones, a 'it' con 32740. No extrae muchas, pero extrae

palabras muy frecuentemente consideradas no relevantes para el análisis del contenido del texto.
Entrando en el mundo del stemming, al aplicar cualquiera de los 3 algoritmos la cantidad de entradas cae muy significativamente. Y además, vemos que el más agresivo de todos fue el algoritmo de Snowball.
En el análisis de tokens, concluimos que el token 'Letter' es el más agresivo de todos, mientras que el Whitespace el que menos. La diferencia en número entre opciones es enorme entre cualquiera de los tokens y el whitespace.

Token	Amount of different words
Letter	54455
Classic	59569
Standard	61825
Whitespace	167063

Experimentación:

Para la segunda parte de este informe, completamos el código basado en las fórmulas presentadas en la asignatura para calcular la similitud de 2 documentos cualesquiera. En cuanto a la realización, no ha sido del todo difícil. Quizás uno de los puntos más importantes ha sido el control de la eficiencia de estas operaciones. Siempre existe el método por fuerza bruta para todo, pero es más inteligente aprovechar las propiedades de la información y estructuras de datos que tenemos. En este caso, partíamos de que la lista de términos y sus pesos se encontraba ordenada alfabéticamente para ambos casos, con lo cual era una cuestión de utilizar una implementación particular de la fusión del mergesort, lo que limitaría el coste.

Comparación de un archivo contra sí mismo

```
/session2ES  master ± python3 TFIDFViewer.py --index novels --files ../novels/KiplingJungleBook.txt ../novels/KiplingJungleBook.t
xt
/home/ivan/.local/lib/python3.10/site-packages/elasticsearch/connection/base.py:200: ElasticsearchWarning: Elasticsearch built-in secu
rity features are not enabled. Without authentication, your cluster could be accessible to anyone. See https://www.elastic.co/guide/en
/elasticsearch/reference/7.17/security-minimal-setup.html to enable security.
  warnings.warn(message, category=ElasticsearchWarning)
Similarity = 1.00000
```

Bastante directo. Definimos el primer documento como A y el segundo como B. Partiendo de que A=B, como podemos suponer, la suma del producto de pesos tiene de resultado 1. El documento A es idéntico al B.

Comparación de archivos que no contienen ninguna palabra en común.

```
 ivan@torrelinuxivan  ~/uni/CAIM/session2ES/docs  master ± python3 ../TFIDFViewer.py --index test  --files ./6 ./7
/home/ivan/.local/lib/python3.10/site-packages/elasticsearch/connection/base.py:200: ElasticsearchWarning: Elasticsearch built-in secu
rity features are not enabled. Without authentication, your cluster could be accessible to anyone. See https://www.elastic.co/guide/en
/elasticsearch/reference/7.17/security-minimal-setup.html to enable security.
  warnings.warn(message, category=ElasticsearchWarning)
Similarity = 0.00000
 ivan@torrelinuxivan  ~/uni/CAIM/session2ES/docs  master ±
```

El otro extremo. Tomando los archivos aportados por el ejercicio de la carpeta docs, los archivos 6 y 7 no contienen ninguna palabra en común con el otro. Y como podemos ver, se ve reflejado en el output de nuestra implementación del algoritmo.

Comparación de archivos en que A es un subconjunto de B.

```
 ivan@torrelinuxivan ~/uni/CAIM/session2ES/docs  master ± python3 ../TFIDFViewer.py --index test  --files ./3 ./7
/home/ivan/.local/lib/python3.10/site-packages/elasticsearch/connection/base.py:200: ElasticsearchWarning: Elasticsearch built-in secu
rity features are not enabled. Without authentication, your cluster could be accessible to anyone. See https://www.elastic.co/guide/en
/elasticsearch/reference/7.17/security-minimal-setup.html to enable security.
  warnings.warn(message, category=ElasticsearchWarning)
Similarity = 0.90883
 ivan@torrelinuxivan ~/uni/CAIM/session2ES/docs  master ±
```

Todas las palabras del archivo 7 existen en el archivo 3, lo que nos hace indicar que en grado en que la similitud entre ambos se aleje del 100% dependerá directamente de las palabras existentes en 3 que no estén en 7. Por ejemplo, si añadimos una palabra distinta a 3:

```
 ivan@torrelinuxivan ~/uni/CAIM/session2ES/docs  master ± python3 ../TFIDFViewer.py --index test  --files ./3 ./7
/home/ivan/.local/lib/python3.10/site-packages/elasticsearch/connection/base.py:200: ElasticsearchWarning: Elasticsearch built-in secu
rity features are not enabled. Without authentication, your cluster could be accessible to anyone. See https://www.elastic.co/guide/en
/elasticsearch/reference/7.17/security-minimal-setup.html to enable security.
  warnings.warn(message, category=ElasticsearchWarning)
Similarity = 0.89984
 ivan@torrelinuxivan ~/uni/CAIM/session2ES/docs  master ±
```

El grado de similitud decae, aunque no en un alto porcentaje (no llega a 1% de diferencia).

Observaciones

Adicionalmente de las técnicas y detalles particulares usados para la implementación del algoritmo, no hay demasiado a destacar. He realizado otras pruebas que no he incluido en el informe, como por ejemplo coger 2 archivos que son mitad y mitad iguales entre sí, y cómo fluctúa la similitud según las palabras de la otra mitad aparezcan o no en el resto del índice. Buscaba comprobar que efectivamente, en ninguno de los casos la similitud podría alcanzar el 50%. Y así fue.