Sesión 2: Programming with ElasticSearch

ELENA RUIZ CANO

1. El índice recargado

1.1 Comparación de resultados según el 'token'

Para esta comparación se ejecutan los siguientes scripts:

- *IndexFilesPreprocess.py* con '--token [name_token]
- *CountWords.py* original sin ningún tipo de filtro com se hizo en la sesión anterior.

WHITESPACE
Usando la opción de whitespace da como resultado todas esas palabras que están separadas por un espacio, es decir, en este token puede incluir desde palabras compuestas, como *I'm* o *teacher's,* hasta números y sobretodo con las palabras contenidas en puntos aunque sea para indicar el final de una frase.

CLASSIC
En la opción de classic no solo separa los términos por espacio sino que ya distingue los símbolos de puntuación usados en la lengua inglesa.

STANDARD
En esta opción amplía un poco más los símbolos que utilizará como 'token' respecto el classic. De manera que a parte de reconocer los espacios y símbolos de puntuación, también reconoce aquellos símbolos que forman palabras compuestas.

LETTER
Esta opción es la que más filtraje realiza ya que separa las palabras por cada símbolo que encuentra.

Después de comparar las diferentes opciones con los mismos índices se observan los siguientes resultados:

	WHITESPACE	CLASSIC	STANDARD	LETTER
Total palabras	3099105	3144033	3146845	3160329
Total palabras dif.	167063	59569	61823	54455

	WHITESPACE	CLASSIC	STANDARD	LETTER
the	203231	206706	206546	206706
of	115552	116802	116786	116803
and	97314	101593	101586	101593
to	82329	84215	84200	84279
a	63102	65288	65238	65487

Exceptuando classic y standard, que tienen resultados muy similares, ya que como se ha mencionado antes separan las palabras por un criterio muy similar, se observa en la tabla de derecha a izquierda, que a medida que se tienen en cuenta más caràcteres para separar las palabras, más ocurrencias de las palabras aparecen.

TOKEN MÁS AGRESIVO

El token más agresivo encontrado después de comparar entre las diferentes opciones es el de **LETTER,** ya que el texto se separa en más términos, todo y que a la vez se observa que existe menos palabras diferentes de las demás. Por tanto realiza una partición más afinada con el objetivo de contabilizar lo mejor posible las frecuencias.

Sobretodo se observa en la tabla de las palabras con más ocurrencias, ya que con la opción letter se acaban contabilizando más ocurrencias en las palabras más usadas, que las otras opciones por los símbolos, no permite contarlas.

1.2 Palabra más usada en inglés

Después de elegir el token más 'agresivo' y ejecutando el mismo script con la opción --filter, tenemos que la palabra más utilizada es: **the** con 206706 recurrencias seguida de las palabras que resultan ser las más usadas en los diferentes métodos de separación.

2. La computación del tf-idf y su coseno de similitud

1.1 Explicación del código

Las modificaciones del script TFIFViewer.py para completar sus funciones han sido las siguientes:

- **toTFIDF:** A partir de las fórmulas dadas en las transparencias de teoría dado la lista de frecuencias de los términos sobre cada documentos y del índice se calcula el valor de TFIDF para cada término. Devolviendo la lista de pares (termino, TFIDF)

- **normalize:** Cálculo de la normalización del vector que devuelve la anterior función.
- **print_term_weigth_vector:** Escribe por consola el vector de pares (term, peso).
- **cosine_similarity:** Se llama a la función normalizar y se calcula el producto de las dos listas.

1.2 Experimento 1: Único documento

PREPARACIÓN DEL TEST

Dado el código completo del fichero TDIDFViewer.py, se ha generado una colección de un solo documento para comprobar los pesos sobre el mismo.

RESULTADOS

El resultado es que su **similitud es de 1**, como era de esperar porque se han comparado dos documentos idénticos.

ANÁLISIS

Si comparamos la lista de términos que devuelve CountWords.py sobre el mismo fichero, se observa que conservan el mismo orden. Como es natural, al no calcular los pesos sobre diferentes documentos, solo en uno, los pesos son equivalentes a las frecuencias del mismo documento.

1.2 Experimento 2: Dos documentos

PREPARACIÓN DEL TEST

Dado el código completo del fichero TDIDFViewer.py, se ha generado una colección de dos documentos. A la hora de indexar el documento se filtra únicamente por cuatro palabras con el objetivo de comparar sus pesos en los distintos documentos. Los documentos comparados han sido **DarwinOriginofSpecies.txt** y **DickensAChristmasCarol.txt**

RESULTADOS

El resultado que se obtiene después de ejecutar el script es una similitud de: **0.039**

3. Análisis sobre la práctica

Después de trabajar sobre el script TDFIFViewer.py, puedo afirmar que he comprendido la mayoría del funcionamiento del fichero. Desde que se indexa, compara, ordena y calcula todos los pesos y finalmente su similitud. Ahora bien, existen partes del código, donde por poco conocimiento de python confía que esa línea hace lo que debería hacer.

A la hora de implementar el código, todas las funciones excepto de la print_term_weigth_vector, y cousine_similarity. Todas desde un principio parecen haber sido implementadas correctamente, ya que simplemente se ha seguido las fórmulas de las

transparencias, pero sobretodo por la de calcular el coseno, cuando se encuentra dos vectores de tamaño distinto quizás no se ha resuelto de la mejor manera. Es por eso que en el experimento 2, sale una similitud extremadamente baja.

La mayor dificultad que he encontrado ha sido el hecho de programar en python, ya que como he comentado anteriormente, es un lenguaje no conocido por mi. Y el hecho de buscar constantemente como se realizan cálculos y se gestionan las listas es lo que ha hecho tardar más en realizar la práctica.