4ª práctica de laboratorio de SIW "Detección de documentos cuasi-duplicados"

Motivación

En la práctica anterior nos enfrentamos a dos problemas fundamentales en un buscador:

> (1) cómo representar un texto en base a sus términos constituyentes, y
>
> (2) cómo comparar un documento dado con otro texto—p.ej., una consulta—para determinar su similitud y, por extensión, su relevancia.

También vimos que comparar cada consulta con todos los documentos de una colección no es viable pero antes de ver cómo resolver consultas de forma eficiente debemos solucionar otro problema muy habitual en la web: la existencia de documentos duplicados o cuasi-duplicados.

Imaginemos que al explorar la web acabasen en la colección los documentos *A*, *B* y *C* junto con copias (o cuasi-copias) de los mismos A_2, A_3, A_4, B_2, B_3 y C_2, C_3, C_4. Imaginemos, además, que dada una consulta *q* los documentos más relevantes son *A*, *B* y *C* en ese orden. ¿Qué aspecto tendría la lista de resultados obtenidos? El siguiente:

> $A, A_2, A_3, A_4, B, B_2, B_3, C, C_2, C_3, C_4, \ldots$

La lista obtenida contendría información duplicada que es similar a la consulta y, en consecuencia, relevante pero que ha dejado de ser relevante por ya no ser novedosa—p.ej., para la persona usuaria el documento A_2 no aporta nada diferente al documento *A*. En consecuencia, un motor de búsqueda debería detectar documentos similares y eliminarlos o no mostrarlos en la lista de resultados.

Para ello podrían usarse cualquiera de las medidas de similitud ya vistas pero de nuevo nos encontraríamos con un problema de eficiencia: Simplemente no podemos comparar todos los documentos entre sí.

Para resolver ese problema existen las técnicas denominadas *SimHashing*[1].

---

[1] Como no podía ser menos existe un paquete Python llamado **simhash** pero, por razones obvias, no podéis usarlo en este ejercicio. En caso de necesitar detección de cuasi-duplicados en un proyecto más realista sería recomendable su uso.

Descripción del ejercicio y del entregable

La primera parte de este ejercicio consiste en la lectura atenta y análisis del artículo *"Simple Simhashing: Clustering in linear time"* de Ryan Moulton. En dicho artículo, publicado originalmente en Knol, se explica cómo se pueden usar funciones hash para obtener "firmas" de documentos que permitan determinar si son o no similares.

En el pseudocódigo que ofrece aparece un parámetro muy importante, `restrictiveness`, que indica cuántos elementos van a usarse para construir dicha firma. Parte del objetivo de este ejercicio es realizar pruebas con distintos valores de `restrictiveness` y documentar el proceso de decisión para fijar un valor por defecto.

A tal fin se debe implementar en Python la función `SimHash` que recibirá un documento y un valor para `restrictiveness` y retornará su firma.

¡Atención! El documento que reciba la función no será una cadena de texto sino un vector. Para obtenerlo debería reutilizarse todo el código necesario de la tercera práctica (es decir, pueden usarse tokenizadores, estematizadores, listas de palabras vacías, etc.)

En principio, los elementos de dichos vectores serán términos aislados pero **se valorará muy positivamente** el uso de n-gramas de términos—véase el apartado *"Turning anything into a set"* del artículo en el que se menciona específicamente el uso de 3-gramas de términos. En caso de implementar una versión que use n-gramas, también será necesario realizar las pruebas pertinentes para fijar un valor por defecto y documentar ese proceso.

Para las pruebas del ejercicio hay varias opciones:

- Puede utilizarse la adaptación de la colección Cranfield usada en la tercera práctica.
- Se puede utilizar el archivo **`2008-Feb-02-04.json.gz`**. Dicho dataset incluye todos los tuits (en todos los idiomas) publicados entre el 2 y el 4 de febrero de 2008. Puesto que los retuits son elementos cuasi-duplicados dicho dataset contiene multitud de tales ítems.
- Pueden usarse los archivos ofrecidos por Chris McCormick para su tutorial sobre *MinHash.* Los archivos con extensión `.train` contienen documentos asociados a identificadores y los archivos con extensión `.truth` señalan parejas de documentos cuasi-duplicados que un algoritmo de *SimHashing* debería ser capaz de detectar.

Con independencia de la colección que se emplee se deberá usar el código desarrollado para encontrar al menos 5 elementos cuasi-duplicados. En consecuencia, se deberá incluir un archivo de texto en el que se incluirán los ejemplos de documentos cuasi-duplicados encontrados de

manera automática así como el código desarrollado para procesar la colección y detectar los documentos cuasi-duplicados—no sólo el código de la función de *SimHashing*.

Bibliografía y material de consulta

Sería recomendable que, además del artículo de Moulton, consulteis estos materiales. Así apreciaréis que la implementación de *SimHashing* que habéis desarrollado es en concreto una variedad de *MinHash.*

- Broder, Andrei. *"On the resemblance and containment of documents"*
- Broder, Andrei *et al.* *"Syntactic Clustering of the Web"*
- Manku, Gurmeet Singh *et al.* *"Detecting Near-Duplicates for Web Crawling"*
- Manning, Christopher *et al.* *"Near-duplicates and shingling"*
- Moulton, Ryan. *"Simple simhashing: Clustering in linear time"*.
- Vassilvitskii, Sergei. *"Duplicate Detection On the Web: Finding Similar Looking Needles In Very Large Haystacks"*
- MinHash Tutorial with Python Code