3ª práctica de laboratorio de SIW
“Similitud entre textos”

Motivación

Un buscador web tiene como misión encontrar en la Web documentos que sean relevantes para una necesidad de información expresada en forma de consulta. Así, si nuestra necesidad de información fuera *“¿Qué es un buscador web?”* podríamos introducir la consulta `buscador+web` en un buscador y quizás se obtendría como primer resultado la URL https://es.wikipedia.org/wiki/Motor_de_b%C3%BAsqueda. Dicha página, contiene en su inicio el texto siguiente:

> Un motor de búsqueda o **buscador** es un sistema informático que busca archivos almacenados en servidores **web** gracias a su spider (también llamado araña web). Un ejemplo son los buscadores de Internet (algunos buscan únicamente en la web, pero otros lo hacen además en noticias, servicios como Gopher, FTP, etc.) cuando se pide información sobre algún tema. Las búsquedas se hacen con palabras clave o con árboles jerárquicos por temas; el resultado de la búsqueda «Página de resultados del **buscador**» es un listado de direcciones **web** en los que se mencionan temas relacionados con las palabras clave buscadas.

En este caso concreto el resultado obtenido parece bastante relevante para la necesidad de información original; sin embargo, el buscador es un sistema totalmente automático, sin ningún tipo de “inteligencia” para determinar qué objetivo se perseguía con la consulta, ni para evaluar la relevancia real del contenido encontrado.

Lo que sucede en realidad es que el buscador no trata de encontrar documentos relevantes sino documentos **similares** a la consulta de entrada. Similitud que, en los métodos más básicos es puramente léxica y no semántica (dicho de otro modo, se buscan coincidencias por palabras clave).

Así, en negrita se muestran términos que coinciden literalmente con alguno de los términos de la consulta; mientras que se han subrayado aquellos términos que, dependiendo de la implementación del buscador, podrían tener o no coincidencia con los términos de búsqueda.

Es preciso señalar que los buscadores modernos utilizan muchísimos más indicios que la simple aparición de palabras clave[1] pero, aun así, **resulta importante comprender cómo funcionan las medidas de similitud más comunes.**

Similitud booleana entre documentos o documentos y consultas

Documentos y consultas se representan mediante *bags-of-words*, es decir, **vectores** donde se tiene en cuenta únicamente los términos que aparecen pero no se considera ni el orden de las palabras en el documento ni su frecuencia de aparición.

Para dicho modelo existen **distintos coeficientes** para determinar **qué documento es más parecido a una consulta** y, en consecuencia, presumiblemente **relevante**.

Coeficiente de Dice	$2\frac{\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$
Coeficiente de Jaccard	$\frac{\lvert X \cap Y \rvert}{\lvert X \cup Y \rvert}$
Coseno	$\frac{\lvert X \cap Y \rvert}{\sqrt{\lvert X \rvert \cdot \lvert Y \rvert}}$
Coeficiente de solapamiento	$\frac{\lvert X \cap Y \rvert}{\min(\lvert X \rvert, \lvert Y \rvert)}$

En las fórmulas anteriores se entiende que **X e Y son vectores que representan documentos** (o consultas) mientras que el **módulo** indica el **número de términos** en un documento, unión de documentos o intersección de documentos.

[1] BERT, MUM y vector search engines. https://app.milanote.com/1OAADK18VM5H6Z?p=CKkb3hZhxyL

¿Qué es un término?

A priori puede parecer sencillo determinar de manera automática qué términos aparecen en un documento, pero en realidad no es una tarea fácil. Así, en el texto anterior aparecen las siguientes secuencias de caracteres que podrían considerarse (o no) términos diferentes:

búsqueda
buscador
busca
buscadores
buscan
búsquedas
búsqueda
buscador»
buscadas.

Obsérvese cómo al usar el espacio como separador de términos hay palabras que incorporan símbolos de puntuación y que serían, por tanto, distintas de las mismas palabras sin esos signos.

Por otro lado, `buscador` y `buscadores` o `búsqueda` y `búsquedas` se consideran términos distintos según aparezcan en plural o singular aunque tengan el mismo "lema".

Por último, si en lugar de usar los términos literalmente se redujesen a su raíz (usando un *stemmer*) habría una serie de ellos que pasarían a ser equivalentes entre sí (p.ej., `busca`, `buscan` y `buscadas` reducen a `busc`, `buscadores` y `buscador` a `buscador` y `búsquedas` y `búsqueda` a `busqued`).

Tokenizar un texto en lenguaje natural no es una tarea trivial y existen diversas bibliotecas que pueden utilizarse para ello (p.ej., *NLTK* o *spaCy*). Por simplicidad en este ejercicio se recomienda utilizar *TextBlob*, además las pruebas se realizarán en inglés[2].

[2] Por razones obvias lematizadores y estematizadores son dependientes del idioma y *TextBlob* no soporta español. Por otro lado, aún cuando la biblioteca soporte el idioma que nos interese sería preciso detectar en qué idioma está escrito un texto en caso de que puedan coexistir textos en múltiples lenguas. A tal fin se podría usar la biblioteca `langdetect`.

Descripción del ejercicio y del entregable

Colección de documentos (y consultas)

Para la realización del ejercicio se trabajará sobre un derivado de la colección Cranfield disponible en el siguiente archivo. Al descomprimirlo se obtienen dos ficheros de texto: `cran-1400.txt` que contiene una serie de documentos (uno por línea) y `cran-queries.txt` que contiene una serie de consultas (también una por línea).

Los documentos tienen el siguiente aspecto (obsérvese que no son excesivamente largos puesto que la colección se elaboró en los años 1960):

> I1 experimental investigation of the aerodynamics of a wing in a slipstream brenckman,m j ae scs 25, 1958, 324 experimental investigation of the aerodynamics of a wing in a slipstream an experimental study of a wing in a propeller slipstream was made in order to determine the spanwise distribution of the lift increase due to slipstream at different angles of attack of the wing and at different free stream to slipstream velocity ratios the results were intended in part as an evaluation basis for different theoretical treatments of this problem the comparative span loading curves, together with supporting evidence, showed that a substantial part of the lift increment produced by the slipstream was due to a /destalling/ or boundary-layer-control effect the integrated remaining lift increment, after subtracting this destalling lift, was found to agree well with a potential flow theory an empirical evaluation of the destalling effects was made for the specific configuration of the experiment

Las consultas tienen el siguiente aspecto. Como se puede ver son bastante elaboradas si las comparamos con una consulta web típica y muy próximas a una verbalización literal de una necesidad de información.

> I 001 what similarity laws must be obeyed when constructing aeroelastic models of heated high-speed aircraft

Objetivo del ejercicio

Se deben implementar en Python todas las medidas de similitud entre vectores de documentos que aparecen en la motivación de este enunciado. Cada medida será una función que recibirá dos argumentos (los documentos a comparar[3]) y retornará un valor real positivo que será su similitud.

¡Atención! Se implementarán medidas de similitud booleana, es decir, los términos aparecen o no en documento/consulta. Se considerará como *bonus* la implementación de medidas de similitud que incorporen frecuencia de aparición de los términos en consultas y documentos.

Deberán implementarse, además, funciones que permitan cargar los documentos de la colección y las consultas en sendos diccionarios.

Se debe demostrar el uso de dichas funciones en un script obteniendo el documento más parecido (para cada coeficiente) a una consulta cualquiera[4].

Queda a criterio del estudiante determinar cómo se obtienen los términos desde documentos y consultas; es decir, si simplemente se divide el texto usando espacios en blanco o se usa *TextBlob* para tokenizar[5]; si se aplica *stemming*[6] o lematización[7] o no; y si se ignoran las palabras vacías[8] o no.

¡Atención! El proceso de transformación de textos en *bags of words* debe ser idéntico para documentos de la colección y consultas.

Entregable

Se subirá al campus virtual **un archivo comprimido** que contendrá el **código fuente correctamente documentado**. Además, se incluirá un breve documento donde se describirán las distintas decisiones tomadas por el estudiante a la hora de implementar el ejercicio.

---

[3] Idealmente deberían ser vectores de términos y no simples cadenas de texto.

[4] Nótese que no resulta práctico comparar una consulta dada con todos los documentos de una colección. En una práctica posterior se verá cómo realizar las búsquedas de forma eficiente.

[5] https://textblob.readthedocs.io/en/dev/quickstart.html#tokenization

[6] https://textblob.readthedocs.io/en/dev/api_reference.html#textblob.blob.Word.stem o https://textblob.readthedocs.io/en/dev/api_reference.html#textblob.blob.WordList.stem

[7] https://textblob.readthedocs.io/en/dev/quickstart.html#words-inflection-and-lemmatization

[8] Una posible lista para inglés: http://snowball.tartarus.org/algorithms/english/stop.txt

Pseudocódigo

Este es el pseudocódigo de la función principal:

```
def main():
  texts = load_lines(“cran-1400.txt”)

  queries = load_lines(“cran-queries.txt”)

  for q in queries:
    best_text = find_best_text(q, texts, dice)
    # show text

    best_text = find_best_text(q, texts, jaccard)
    # show text

    best_text = find_best_text(q, texts, cosine)
    # show text

    best_text = find_best_text(q, texts, overlapping)
    # show text
```

Para transformar una cadena de texto en un *bags-of-words* se puede usar el siguiente algoritmo. Dada la complejidad de la tarea **se recomienda encarecidamente usar TextBlob** (u otra biblioteca de procesamiento de lenguaje natural) para realizar algunos de los pasos.

```
def string_to_bag_of_words(text):
  bow = {}

  # Dividir por espacios en blanco o usando un tokenizador
  tokens = text.tokenize()

  for token in tokens:
    # Paso 1: Pasar a minúsculas. Puede ser innecesario en
      función del tokenizador que se use

    # Paso 2: Eliminar símbolos de puntuación, crucial si se
      divide por espacios en blanco; puede ser innecesario en
```

```
    función del tokenizador

  # Paso 3 (opcional): eliminar palabras vacías (stop-words)

  # Paso 4 (opcional): obtener el stems o lema del token

  # Paso final: Almacenar el término final en un diccionario.
    De manera opcional puede tenerse en cuenta la frecuencia
    de aparición

  bow[token] += 1

return bow
```