SISTEMAS DE GESTIÓN DE DATOS

TEMA 4

Recuperación de información

1º Introducción

Def. La recuperación de información es la disciplina encargada de la estructura, el análisis, la organización, el almacenamiento, la búsqueda y la obtención de dicha información cuando es solicitada.

-- Tradicionalmente, la recuperación de información siempre ha estado basada en documentos de texto en diferentes formatos, ya sea mediante páginas webs, e-mail, artículos etc. Un documento de texto puede llegar a contener algunos atributos relevantes, pero lo más importante es el contenido del mismo, loq ue hace que su manero sea muy diferente al realizado en las bases de datos.

-- La recuperación de información también puede emplearse en otros tipos de documentos como pueden ser imágenes, vídeos, audios etc. Para hacer esto se emplean descripciones textuales asociadas a dichos documentos, las cuales pueden crearse de diversas maneras.

-- Un aspecto principal de la recuperación de información es que los resultados deben ser relevantes, es decir, deben tener importancia con respecto a la consulta realizada por el usuario. Hay dos tipos:

- **Relevancia temática:** Los resultados obtenidos deben tratar sobre el mismo tema que se consulta.
- **Relevancia de usuario:** Los resultados obtenidos deben aportan algo al usuario concreto que ha realizado la petición.

-- Para hacer que los resultados proporcionados cumplan con el aspecto de relevancia temática, se utilizan los llamados modelos de recuperación, los cuales constituyen la base de las funciones de ranking que sirven para ordenar los resultados.

2º Modelos de recuperación de información

Def. Los modelos de recuperación de información (MRI) son los encargados de formalizar el proceso para asociar una determinada consulta con un documento. Estos modelos están compuestos por:

- **D:** Vista lógica de los documentos en la colección donde se realizará la búsqueda. Cada uno de dichos documentos se denota mediante “d”
- **Q:** Vista lógica de las consultas de los usuarios.
- **R(d, q):** Función que relaciona un número real a un documento “d” y a una consulta “q”.

-- Los documentos están formados por secuencias de caracteres y las consultas por secuencias de términos clave unidos mediante conectores, pero para el MRI, ambos elementos pueden estar representados de diversas formas, como pueden ser tuplas binarios o vectores.

-- Un término clave es una palabra o secuencia de palabras contenidas en un documentos, las cuales denominaremos mediante “k”, siendo “t” el número de palabras de una colección. Se denomina *vocabulario* al conjunto de los distintos términos clave contenidos en una colección de documentos.

2.1 Modelo booleano

-- Las consulta empleados en este modelo se representan mediante fórmulas booleanas de la forma: **q ::= [“conector”] “k1” [“conector”] [“k2”] | … | [[“conector”] [“kn-1”] [“conector”] [“kn”]]**, donde “kn” son los términos clave y los conectores serán las usuales: AND, OR y NOT.

ej. q ::= (deporte OR ocio) AND agua , c::= pesca AND mar AND (NOT rio)

-- El modelo booleano devolverá aquellos documentos de la colección que encajen completamente con la consulta indicada, es decir, que cumplan con las condiciones indicadas en la misma respecto a los términos solicitados. Esto quiere decir que los documentos que cumplan la condición de forma parcial no serán devueltos.

-- Si en un documento *di* aparecen únicamente los términos *kl*, *km* y *kn*, diremos que se observa el patrón de ocurrencia [kl, km, kn].

-- Si el vocabulario tiene una longitud "t", existirán un total de 2^t patrones de co-ocurrencia posibles y al conjunto de estos los denotaremos como *K*. Los distintos patrones de co-ocurrencia serán denominados mediante "p".

-- Los patrones de co-ocurrencia son representados mediante tuplas binarias, donde por cada uno de los términos clave buscados se indicará un 1 si se encuentra en el documento o un 0 si no se encuentra. Cada una de dichas tuplas se llama *componente conjuntivo de términos clave*.

-- Para cada documento *di* asociaremos un determinado patrón de co-ocurrencia *c(di)*, el cual describe los términos clave que aparecen en dicho documento. A cada consulta *qi* se le asocian uno o varios patrones de co-ocurrencia, los cuales indican los términos clave que contienen los documentos sobre los que se realizan las consultas. Normalmente *c(q)* devuelve un subconjunto de *K*.

Modelo boolano

q = Consulta.
d = Documento.
k = Término clave.
t = Número de términos clave distintos en una colección.

V = {k1, k1, … kt}: Vocabulario, conjunto de términos clave de una colección.
P = (1|0, [1|0], …): Patrón de co-ocurrencia, el cual tiene t valores.
K = Conjunto de todos los patrones de co-ocurrencia.

c(di) = Ejecución de una consulta sobre un documento di, el cual da como resultado un patrón de co-ocurrencia.
c(q) = Ejecución de una consulta sobre la colección de documentos, la cual suele dar como resultado un subconjutno de K.

Función de similitud, $\mathbf{R(d,q)} = \begin{cases} 1 \Leftrightarrow c(d) \in c(q) \\ 0 \Leftrightarrow e.o.c \end{cases}$

– La función de similitud toma la consulta a realizar sobre un documento en concreto y devuelve:

- **0:** Si el documento no contiene todos los términos clave de la consulta, es decir, si es relevante.
- **1:** Si el documento contiene todos los términos clave de la consulta, es decir, es relevante.

– Debido a esto, aplicar la función de similitud sobre un conjunto de documentos dará como resultado dos conjuntos, uno relevante para el usuario y otro no. Sin embargo, la aplicación de este modelo no nos aporta información sobre que documento es más relevante, por lo que no podremos realizar una ordenación de los mismos por orden de relevancia.

– Debido a esto, el modelo booleano es sencillo de implementar y eficiente, pero esta limitado en lo que a la ordenación de los resultados se refiere.

2.2 Modelo vectorial

– El modelo vectorial mitiga las limitaciones del modelo booleano en cuanto a la capacidad de diferenciar el grado de relevancia de los resultados obtenidos. En este modelo, tanto los documentos como las consultas se representa mediante vectores de números reales de dimensión *t*.

– La función de simulitud calculará la relevancia entre documentos y consultas en base a comparar el ángulo que forman el vector de dicha consulta con el vector del documento a examinar. Al realizar la comparación se obtiene como resultado valores continuos en lugar de los valores discretos {0, 1}, los cuales se pueden utilizar como medida de relevancia entre los distintos documentos.

– Los vectores están compuestos por un conjunto de pesos, los cuales representan la importancia de cada uno de los términos clave sobre el documento o consulta. Tienen la siguiente forma:

- **dj = (w_{1j}, w_{2j}, …, w_{tj})**, donde w_{ij} es el peso del término clave i-ésimo en el documento j-ésimo.
- **q = (w_{1q}, w_{2q}, …, w_{tq})**, donde w_{iq} es el peso del término clave i-ésimo en la consulta q-ésima.

– El peso de cada uno de los términos clave es calculado en base a cuantificar el número de apariciones de dicho término dentro del documento o consulta. Los términos claves no pueden ser comunes a todos los documentos, ya que entonces no se puede hacer una correcta diferenciación de aquellos que son relevantes.

– La frecuencia f_{ij} es el número de apariciones del término clave *ki* en el documento *dj*, mientras que la frecuencia f_{iq} es la frecuencia del término clave *ki* en la consulta *q*. Por otra parte, *ni* es el número de documentos en los que aparece el término clave *ki*.

Modelo Vectorial

q = Consulta.
d = Documento.
N = Número total de documentos en la colección.
k = Término clave.
t = Número de términos clave distintos en una colección.

dj = (w_{1j}, w_{2j}, …, w_{tj}): Vector de pesos del documento j.
q = (1|0, [1|0], …): Vector de pesos de la consulta q.

f_{ij} = Número de apariciones del término clave ki en el documento dj.
f_{iq} = Número de apariciones del término clave ki en la consulta q.
ni = Número de documentos en los que aparece el término clave ki.

2.2.1 Esquema de pesos TF-IDF

– Es uno de los esquemas más utilizaos para asignar los pesos a los términos clave. Este se basa en la combinación de dos valores relativos a la frecuencia de aparición del término en cuestión:

- **Term Frecuency (tf_{ij}):** Frecuencia del término clave *ki* en el documento *dj*.
- **Inverted Document Frecuency (idf_i):** Frecuencia inversa del término *ki* en el conjunto de documentos de la colección.

– El término de frecuencia (tf) para una determinada clave *ki* en un documento *dj* es proporcional a la frecuencia de aparición de dicha clave en el documento en cuestión.

$$tf_{ij} = \begin{cases} 1+\log_2(f_{ij}) & \Leftrightarrow \ f_{ij}>0 \\ 0 & \Leftrightarrow \ f_{ij}=0 \end{cases}$$

tf_{ij} = Termino de frecuencia de la clave ki en el documento dj.
f_{ij} = Número de apariciones del término clave ki en el documento dj.

– La utilización de logaritmos tiene dos motivaciones: Para amortiguar el crecimiento y para que la magnitud sea comparable de manera directa con la Frecuencia Inversa de Documento.

– La Frecuencia Inversa de Documento (idf_i) trata de reflejar la importancia de la clave *ki* dentro de la colección de documentos donde se realizará la consulta.

$$Idf_i = log_2(N/n_i)$$

Idf_i = Frecuencia Inversa de la clave ki en la colección de documentos N.
N = Número total de documentos en la colección.
ni = Número de documentos en los que aparece el término clave ki.

Suponemos que no tenemos colecciones vacias (N>0) y que el término clave aparece en algún momento dentro de la colección (n_i>0).

– Si el término clave aparece aparece en gran parte de los documentos de la colección, este será poco discriminante, por lo que deberá tener un valor *idf* bajo. Por el contrario, si el término aparece en pocos documentos, este será bastante discriminante, por lo que deberá tener un valor de *idf* alto.

– La fórmula utilizada para calcular la Frecuencia Inversa de Documento (idf_i) se ha obtenido de forma empírica, observando que produce buenos resultados al medir la discriminación. Para obtener el peso w_{ij} del término clave k_i sobre un determinado documento d_j:

$$w_{ij} = tf_{ij} * idf_i$$

w_{ij} = Peso de la clave k_i para el documento d_j perteneciente a la colección N.
tf_{ij} = Termino de frecuencia de la clave ki en el documento dj.
Idf_i = Frecuencia Inversa de la clave ki en la colección de documentos N.

2.2.2 Cálculo de la similitud

– En el momento de calcular la similitud entre una consulta y los documentos donde se va a aplicar, necesitaremos utilizar el vector de pesos de la consulta *q = (1|0, [1|0], …)*, el cual asociaremos un 0 los términos que no aparecen y 1 en los que sí aparecen.

– La similitud se calcula en base al coseno del ángulo que forman (θ) los vectores del documento *dj* y de la consulta *q*. El coseno toma valores entre [1 y -1], pero todos nuestros pesos tienen un valor mayor que 0, todos los ángulos estarán comprendidos entre 0º (valor 1) y 90º (valor 0).

– La función de similitud *R(a,b)* utilizará el producto escalar ($a \circ b = |a| * |b| * \cos(\theta)$):

$$R(a,b) = a \circ b / (|a| * |b|)$$

R(a,b) = Similitud entre los vectores a y b.

$$R(dj,q) = \frac{dj \cdot q}{|dj||q|} = \frac{\sum_{i=1}^{t} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \sqrt{\sum_{i=1}^{t} w_{iq}^2}}$$

3º Procesos de recuperación de información

– Un sistema de recuperación de información se compone por dos fases principales:

- **Indexado:** Tratamiento de la colección de documentos para generar el índice de búsqueda.
- **Consulta:** Tratamiento de la consulta del usuario y enlace con los documentos adecuados de la colección, utilizando para ello el índice generado en el proceso de indexado.

3.1 Proceso de Indexado

Def. Fase encargada de procesar los documentos de la colección con el objetivo de extraer los términos clave. Se trata de la fase que más recursos consume pero se realiza una única vez y su coste se amortiza con el gran volumen de consultas contestadas.

– El indexado se divide en tres subfases principales:

1. Adquisición de los datos.
2. Transformación de los datos.
3. Creación del índice.

3.1.1 Subfase 1: Adquisición de los datos

– La adquisición de los datos comprende varios pasos:

1. **Obtención de los documentos:** Se obtienen los documentos sobre los que se van a realizar las posteriores búsquedas, lo cual puede ser mas o menos complicado dependiendo del tipo de documentos a tratar.
   - La adquisición se realiza mediante un crawler, el cual recorre el contenido de todos los documentos y localizaciones web para encontrar todos los posibles documentos a incluir en la colección. El crawler puede aplicar filtros para únicamente incluir documentos concretos.
2. **Conversión de los datos:** Los documentos obtenidos en el primer paso pueden tener formatos muy diversos, por lo que este paso consiste en convertir todos los datos a un formato de texto común. Algunos elementos, como imágenes pueden seguir necesitando ser interpretados.
3. **Almacenamiento:** Una vez tenemos los ficheros localizados y hemos convertido los datos a un texto uniforme, debemos almacenar los mismos en un almacén de datos.

3.1.2 Subfase 2: Transformación de los datos

– Una vez tenemos todos los documentos que vamos a utilizar, necesitaremos examinarlos y transformar su contenido con el fin de poder operar con el de forma adecuada. Dependiendo del sistema a implementar, se pueden llevar a cabo más o menos pasos. Estos son algunos básicos:

1. **Análisis léxico (tokenizing):** Se recorre el documento separando las unidades mínimas (tokens) que se van a utilizar. Normalmente estas unidades serán palabras, por lo que deberemos tener cuidados con los caracteres especiales.
2. **Eliminación de palabras vacías (stop words):** Paso opcional basado en la eliminación de las palabras que no contribuyen significativamente al contenido del texto. Estas palabras tienen poco impacto en la relevancia de los documentos devueltos debido a que suelen estar presente en todos los documentos.
   - Estas palabras no pueden eliminarse si se realiza una búsqueda por frases.

3. **Obtención de raíces (stemming):** Transformación a nivel de palabra que busca agrupar todas las palabras de significado similar basándose en la raíz de las mismas.
4. **Obtención de lemas (lemanización):** Transformación a nivel de palabra que busca sustituir cada palabra por otra general que las representa a todas.

3.1.3 Subfase 3: Creación del índice

– Una vez hemos obtenido los documentos y finalizado el tratamiento de los mismos, debemos proceder a la creación del índice invertido. Esta fase dependerá de la estructura de datos utilizada y del modelo de recuperación de datos que vayamos a emplear, pero en general realizaremos operaciones con el fin de obtener posiciones, pesos etc.

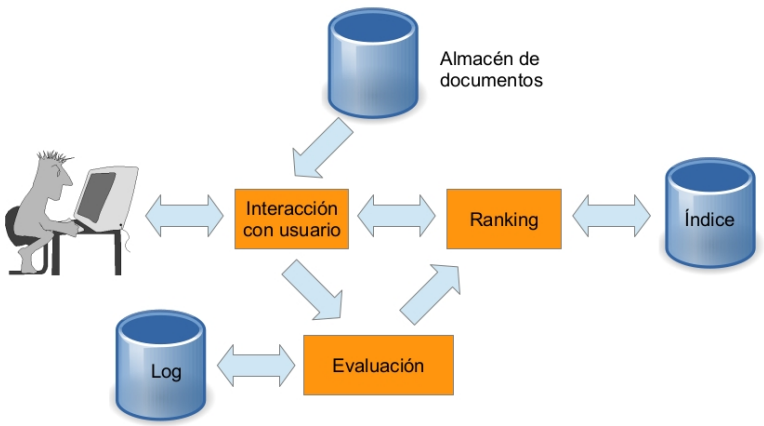
3.2 Realización de consultas

– En el proceso de realización de consultas intervienen los siguientes actores:

- **Usuario:** Realiza la consulta, obtiene los resultados y navega entre ellos.
- **Almacén de documentos:** Empleado para obtener fragmentos con los cuales acompañar los resultados.
- **Log de consultas previas:** Almacén de las consultas realizadas anteriormente y las interacciones llevadas a cabo por el usuario, con el fin de proporcionar resultados más relevantes.
- **Indice:** Generado en la fase de indexación y utilizado para proyectar los resultados.

– Además de los actores, la realización de la consulta esta compuesto por distintos componentes:

- **Componente de interacción con el usuario:** Su trabajo consiste en recibir las consultas en el lenguaje establecido, transformarlas ofreciendo sugerencias y expandiéndola a términos similares. Por último, muestra los resultados al usuario y monitoriza su actividad.
- **Componente de ranking:** Se encarga de obtener los documentos más relevantes ahora la consulta y de organizarlos en orden de relevancia. Para llevar esto a cabo se hace uso de un modelo de recuperación de información y de la información almacenada en el log.
- **Componente de evaluación:** Almacena las consultas e interacciones realizadas por el usuario en el log y analiza la correspondencia entre la relevancia del modelo de recuperación empleado y la relevancia real de los resultados para el usuario. Este componentes es importante para mantener el sistema con buenos standares de calidad y mejorar el modelo.



4º Índices

Def. Un índice es una estructura de datos utilizada para la recuperación de información, cuyo objetivo es almacenar la información importante de los documentos para acceder a la misma de manera rápida a la hora de realizar consultas.

– Los índices almacenan los términos clave de los documentos e información útil a cerca de los mismos para realizar las consultas. Esta información puede ser más o menos detallada, pero incluye aspectos como: Número de apariciones del término clave o las posiciones en las que se encuentra.

– El principal tipo de índice utilizado en la recuperación de datos es el índice invertido, el cual contiene los términos clave existentes en la colección y las relaciones de cada uno de estos con los diferentes documentos que la conforman.

– Debido a la gran cantidad de documentos a tratar y que la mayoría de términos no aparecen en los mismos, lo más normal es almacenar las ocurrencias en forma de listas ordenadas.

4.1 Tipos de índices

4.1.1 Índices invertidos completos

– Cuando queremos responder a consultas sobre frases o que tengan en cuenta la proximidad de los términos clave empleados, deberemos utilizar más información que únicamente los documentos en los que aparecen dichos términos. Para esto se utilizan los índices invertidos completos, cuyo objetivos es almacenar toda la información posible para las posteriores consultas.

– Los índices invertidos completos están formados por una lista, donde cada posición se corresponde con un término clave y para los que se guardan los siguientes datos:

- El número de documentos que contienen dicho término clave.
- Una lista de tuplas que guardan información completa sobre las apariciones del término. Estas tuplas son de la forma ***(doc, n, [pos1,…,posn])***, donde “doc” es el documento, “n” el número de apariciones y “pos” la posición de dichas apariciones dentro del documento.

Vocabulario	*ni*	Lista de apariciones
el	3	[(1,5,[1,3,5,7,9]), (2,1,[3]), (3,2,[17,29])]
amigo	1	[(2,2,[4,86])]
hola	3	[(1,4,[22,24,56,71]), (2,3,[1,10,20]),(3,1,[1])
verde	3	[(1,5,[19,25,32,40,59]),(2,1,[5]),(3,4,[76,80,90,111])]
pez	1	[(2,2,[45,123])]
coral	1	[(1,6,[101,134,145,204,250,278])]
agua	3	[(1,3,[67,78,106]), (2,3,[56,78,99,301]), (3,2,[1,34])]
mar	2	[(1,3,[500,509,700]), (3,1,[72])]
abuela	1	[(2,4,[303,581,600,708])]

– La información almacenada respecto a la posición de las apariciones puede ser más o menos detallada y expresarse de varias formas, como: Indicando el carácter en el que empieza el término clave, el número de palabra que ocupa en el documento o el párrafo al cual pertenece.

4.1.2 Índices invertidos vectoriales

– Los índices invertidos vectoriales toman la misma información que se almacena en los índices invertidos completos para calcular los pesos de cada uno de los términos clave, los cuales son necesarios para obtener la similitud entre consultas vectoriales.

– Estos índices almacenan los en pesos en lugar de las apariciones, lo que hace que las consultas vectoriales sean más eficientes.

Vocabulario	*ni*	Pesos
abrigo	3	[(1, 0.0), (2, 0.0), (3, 0.0)]
gol	1	[(1, 4.097)]
abrazo	3	[(1, 0.0), (2, 0.0), (3, 0.0)]
pie	2	[(1, 0.585), (2, 0.585)]
paella	1	[(1, 1.585)]

4.2 Búsqueda con índices

– Una forma sencilla de realizar las búsquedas sería recorrer todos los documentos de la colección comprobando para cada uno de ellos si encaja o no con la consulta realizada. Esta solución no es para nada eficiente, ya que necesitamos recorrer todos los elementos en cada una de las consultas.

– Independientemente del modelo de recuperación que utilicemos, necesitaremos emplear el uso de índices para que las búsquedas se realicen de forma eficiente.

4.2.1 Búsquedas empleando el modelo booleano

– En base al modelo booleano, en el caso de que queramos obtener los documentos donde se cumpla *t1 AND t2*, deberemos realizar los siguientes pasos:

1. Recuperar las listas de apariciones de los términos clave *t1* y *t2* registradas en el índice. Esto es sencillo, únicamente necesitaremos consultar el índice.
2. Cruzar ambas listas y quedarnos con los documentos que aparecen en ambas. Esto lo realizaremos mediante la función *INTERSECT(p1, p2)*, la cual nos devolverá una lista de identificadores de documentos ordenados.

```
INTERSECT(p1,p2):
  answer = []
  while p1 != [] and p2 != []:
    if docID(p1) == docID(p2):
      add(answer, docID(p1))
      p1 = next(p1)
      p2 = next(p2)
    elif docID(p1) < docID(p2):
      p1 = next(p1)
    else:
      p2 = next(p2)
  return answer
```

– Debido a que las listas están ordenados por *id* del documento, como máximo, necesitaremos recorrer cada una de ellas una única vez. Esto hace que el orden de coste para ejecutar dicha función sea como máximo la suma de las dimensiones de los dos vectores pasados como parámetro.

– Para minimizar el crecimiento de la lista de resultados, podemos hacer que se intersecciones las listas provenientes del índice con menor tamaño, es decir, las más cortas.

– Para obtener el resultados de las peticiones *t1 OR t2* se necesitará implementar la función *UNION(p1, p2)*, la cual también sacará partidos de que las listas estén ordenadas. Por otra parte, el operador NOT es poco eficiente de forma aislada, pero pueden crearse funciones concretas para la negación de *INTERSECT* y de *UNION*.

4.2.2 Búsquedas con índices completos

– Los índices completos se utilizan, entre otras cosas, para realizar consultas de frases, las cuales requieres que se encuentren todos los términos situados de forma contigua en el documento.

– Teniendo en cuenta que estos índices almacenan todas las posiciones en las que aparece cada término, para resolver una consulta de frase se utilizan las listas de aparición de todas las palabras involucradas en la misma. Esto se realiza mediante una variación de la función *INTERSECT*, la cual se denomina *INTERSECT_PHRASE*.

```
INTERSECT_PHRASE(<p1,...,pn>):
  answer = []
  while allNotEmpty(<p1,...,pn>):
    if sameDocID(<p1,...,pn>):
      if consecutive(<p1,...,pn>):
        add(answer, docID(p1))

    <p1,...,pn> = advanceMin(<p1,...,pn>)
  return answer
```

- `sameDocID()` → bool: las cabezas de p1,...,pn almacenan el mismo docID.
- `consecutive()` → bool: las cabezas de p1,...,pn son consecutivas.
- `advanceMin()` : avanza las listas con menor docID en su cabeza (cada lista está ordenadas por docID). Puede avanzar todas las listas
- `allNotEmpty()` → bool: ninguna lista p1,...,pn está vacía

4.2.3 Búsquedas empleando el modelo vectorial

– Para entender como se realizará la búsqueda mediante el modelo vectorial, partiremos de la función de relevancia. En principio necesitaremos conocer todos los pesos de tanto del documento w_{ij}, como de la consulta w_{iq}, lo cual provoca que las consultas tengan que recorrer todos los documentos.

$$R(dj,q) = \frac{dj \cdot q}{|dj||q|} = \frac{\sum_{i=1}^{t} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \sqrt{\sum_{i=1}^{t} w_{iq}^2}}$$

– Para que el uso del modelo vectorial sea eficiente, realizaremos las siguientes actualizaciones:

- No es necesario conocer todos los términos clave k_i, sino únicamente aquellos en los que w_{iq} sea mayor que cero. Esto es debido a que si $w_{iq} = 0$, entonces $w_{ij} * w_{iq} = 0$.
- Debido a que q es un vector con cuyos valores pueden ser 1 o 0 dependiendo de si el término es buscado o no, podremos simplificar el numerador de la función eliminando el término w_{iq} y quedándonos solo con w_{iq}.
- La normal $|q|$ es un factor positivo constante en todas las similitudes, por lo que podemos omitirlo sin que afecte al orden resultante de los documentos.

– Aunque hemos simplificado en gran medida la función de relevancia, el cálculo de $|dj|$ continúa necesitando todos los pesos de la misma, incluso para los términos clave que no aparecen la consulta. Sin embargo, esto no depende de la consulta, por lo que podremos precalcularlo y guardar los resultados en la estructura del índice.

```
FASTCOSINESCORE(q, index, k):
1 Scores[N] = 0 # Diccionario de resultados
2 for t in q:
3   p = index[t] # Lista de pesos 'w' del término t en cada
                 # documento 'd'
4   for (d,w) in p:
5     Scores[d] += w # Evitamos el  producto
6 for d in Scores:     # Dividimos por |d|
7   Scores[d] = Scores[d]/Length[d]
8 return best(k,Scores)
```

1) Inicializamos el producto escalar parcial asociado a cada documento d (como máximo habrá N)

3) Obtenemos la lista [(doc,peso),...(doc,peso)] asociada al término t

4-5) Sumamos el peso en cada producto escalar parcial

6-7) Dividimos entre la norma |d|

8) Obtenemos los k documentos con mayor relevancia