Ficha 01

Motores de Búsqueda: El Modelo Vectorial

1.] Introducción.

Cuando se necesita realizar una consulta sobre una base de datos, la tarea se ve facilitada por el hecho que los datos están en un *ambiente estructurado*: se sabe que cada registro de la base contiene ciertos campos o atributos, se sabe de qué tipo son esos atributos, de qué longitud, el nombre de cada uno, etc. Como si fuera poco, también se cuenta desde hace tiempo con un *lenguaje estructurado de consulta* (*SQL*) que permite acceder a los datos y manipularlos en la forma que se requiera. Se puede tener la certeza que la consulta dará los resultados correctos.

Pero el panorama es diferente cuando los datos residen en *ambientes no estructurados*: si en lugar de una base de datos (o un archivo dividido en registros) se tiene un conjunto de documentos de diversos formatos, cada uno conteniendo texto editado sin seguir patrones específicos (como es básicamente la web), entonces encontrar los documentos que respondan a una consulta dada no es tarea sencilla. De hecho, en casos así, lo que se suele obtener es una lista de documentos que *posiblemente* respondan a la consulta, ordenados de alguna forma, y la decisión final sobre qué documento es el mejor la tiene el usuario que formuló la consulta.

Ambos contextos son muy diferentes. En general, si la consulta se realiza sobre un ambiente de datos estructurados, como una *base de datos relacional* usando incluso lenguajes estándar de consulta, hablamos de "*Recuperación de Datos*" (*RD*). Quien hace una consulta sabe exactamente lo que quiere, y puede obtener exactamente lo que necesita:

select dni, nombre **from** Personas **where** sueldo >= 1000

Pero si la consulta se realiza sobre datos no estructurados (en lo que técnicamente sería una base de documentos o *base de datos documental*), hablamos de *Recuperación de Información* (*RI*): la respuesta correcta única no existe y cada documento puede ser más o menos relevante frente a la consulta planteada:

> Necesito toda la información posible sobre combustibles diesel aplicados en la agricultura y en el transporte de pasajeros.

Como todavía los programas de búsqueda (llamados *motores de búsqueda*) no pueden analizar ni comprender el significado de un texto en forma automática, las técnicas algorítmicas que se usan para responder a una consulta buscan una aproximación a la mejor respuesta posible. En ese sentido, el problema de la *RI* se puede definir de la siguiente forma:

> Dada una necesidad de información (consulta + perfil del usuario + ...) y un conjunto de documentos; ordenar los documentos de más a menos relevantes para esa necesidad y presentar un subconjunto de los más relevantes.

2.] Modelos matemáticos: el Modelo Booleano.

Para enfrentar el problema de desarrollar un motor de búsqueda, se tienen dos grandes etapas:

i.) Definir un modelo matemático que permita calcular la relevancia de un documento frente a una consulta.

ii.) Diseñar las estructuras de datos y los algoritmos que implementen ese modelo en la forma más eficiente.

Se han planteado muchos modelos matemáticos basados en distintas ideas para el cálculo de relevancias. Los más clásicos son tres: el *modelo booleano*, el *modelo vectorial* y el *modelo probabilístico*:

- **Modelo booleano**: fue el primero que se planteó y el más simple y directo (y quizás por ello es también el más popular, aunque no el más eficiente). Es común comenzar estudiando este modelo en sus rasgos generales antes de saltar a modelos mas complejos.
- **Modelo vectorial**: es el más usado en la actualidad (y por mucho, más eficiente que el booleano) y es también el que expondremos en nuestro desarrollo.
- **Modelo probabilístico**: se basa en calcular la probabilidad de que un documento sea relevante y parte desde una base matemática completamente distinta al vectorial (sin embargo, la comunidad de RI lo considera inferior al vectorial, aunque algunas de sus extensiones algorítmicas son eficientes)

Recordemos que se usan modelos matemáticos (antes de comenzar a programar...) para intentar *calcular la relevancia* de un documento frente a una consulta. La idea principal del *modelo booleano* es simple: *la relevancia de un documento es binaria*. Esto significa que un documento es relevante o no lo es... Si la consulta consiste sólo de una palabra (como suele pasar en muchos casos) entonces en el *modelo booleano* un documento es relevante sí y sólo si contiene la palabra buscada. Si la consulta contiene varias palabras, se usan conectores lógicos *and*, *or*, y similares. Así, si la consulta incluye palabras encadenadas con *and*, entonces el documento es relevante si contiene a todas las palabras de la consulta. Si la consulta se arma con conectores *or*, entonces el documento es relevante si contiene alguna/s de las palabras. Y valen las combinaciones... Ejemplos de consultas en el modelo booleano:

De una sola palabra:	Agricultura
Usando **and**:	Diesel **and** agricultura **and** transporte público
Usando **or** y combinaciones:	Diesel **and** (agricultura **or** campo) **and** transporte
Otros:	Diesel **but not** alconafta

Este modelo, como dijimos, fue el primero que se planteó y es muy popular todavía, aunque tiene muchos problemas. Podemos ver fácilmente cuáles son los principales inconvenientes:

1. No es capaz de discriminar entre documentos “mejores” y “peores” (no puede decidir si un documento es más relevante que otro): da igual si un documento contiene una o cien veces las palabras de la consulta.
2. Cuando se evalúa una consulta con *or*, da lo mismo si el documento cumple una o varias de las proposiciones del *or* (de nuevo, esto no permite decidir si un documento es mejor que otro...)
3. No considera un *calce parcial* de un documento cuando se usan conectores *and*: si se usa *and* y al menos una proposición es falsa, el documento se descarta. Pero si una sola era falsa (o unas pocas lo eran) y todas las otras eran

verdaderas, el documento podría haber sido tomado como bueno (y sin embargo fue rechazado...)

4. Ni siquiera permite ordenar los resultados: los documentos considerados relevantes tienen el mismo valor ("*son relevantes*") y no hay forma de ranquearlos para ayudar al usuario a navegar desde el "mejor" hacia el "peor"...
5. Como si esto fuera poco, ¡el usuario promedio no lo entiende! Por ejemplo, si la necesidad fuera tal como: "*quiero todo sobre los combustibles diesel y la agricultura*" es notable que muchos usuarios novatos escribirán una consulta como:

diesel **and** agricultura

y eso es un error... el motor ignorará documentos muy buenos que hablen sólo de una de las dos palabras... La consulta debió ser:

diesel **or** agricultura

Los motivos por los cuales el modelo booleano es todavía (a pesar de todo) popular, son básicamente los siguientes:

- Es una de las primeras ideas que suelen surgir, y es una idea sencilla de entender, modelar e implementar.
- Muchos de los primeros sistemas de RI se basaron en él.
- Los sistemas de bases de datos relacionales lo usan mucho para manejar campos de texto.
- En algunos casos, si es usado por expertos, puede ser adecuado.
- Combinado con otros modelos puede ser valioso para cumplir alguna tarea de filtrado previo (por ejemplo, para excluir documentos rápidamente)

3.] Análisis general del Modelo Vectorial.

Sea **D** el conjunto de **N** documentos sobre los que se desea buscar (la *base de datos documental*) y denotemos como $\mathbf{d_i}$ al documento número **i**:

$$D = \{ d_1, d_2, d_3, \ldots, d_i, \ldots, d_N \}$$

De entre todos los **N** documentos se selecciona un conjunto de **k** *términos* o palabras que sean útiles para discriminar o comparar documentos. En principio, no toda palabra contenida en un documento es buena para discriminar: los artículos, las preposiciones, las conjunciones, etc., son tan comunes en todos los documentos que difícilmente una de ellas pueda usarse para determinar que un documento es mejor que otro. A estas palabras se las llama "*palabras vacías*" o "*stop words*" y es común que los motores de búsqueda simplemente las ignoren. Sin embargo, en muchos sistemas modernos se toman *todas* las palabras de *todos* los documentos (incluidas las stop words). Tenemos así, un conjunto **T** con las **k** palabras distintas $\mathbf{t_i}$ de todos los documentos:

$$T = \{ t_1, t_2, t_3, \ldots, t_i, \ldots t_k \}$$

La idea principal del modelo vectorial es que cada documento $\mathbf{d_i}$ se modela o representa como un *vector* en un espacio vectorial *k-dimensional* (una *coordenada* por cada una de las **k** palabras posibles que existen en **T**). Cada *coordenada* es un número $w(\mathbf{t_r}, \mathbf{d_i})$ que indica el *peso* o valor que el término $\mathbf{t_r}$ tiene para el documento $\mathbf{d_i}$:

$$\vec{d_i} = \mathbf{d_i} = (w(\mathbf{t_1}, \mathbf{d_i}), w(\mathbf{t_2}, \mathbf{d_i}), \ldots, w(\mathbf{t_k}, \mathbf{d_i}))$$

Esta es la idea base. A partir de allí, existen muchos planteos matemáticos para calcular el peso de un término para un documento, y ese el punto crucial: si el cálculo de los pesos es acertado, se podrá discriminar con relativa eficiencia a los mejores documentos de los peores. Una de las fórmulas de cálculo de pesos más usada es la siguiente:

$$w(t_r, d_i) = w_{r,i} = \frac{tf_{r,i} * idf_r}{|\vec{d_i}|} = \frac{tf_{r,i} * \log\left(\frac{N}{n_r}\right)}{\sqrt{\sum_{s=1}^{k}\left(tf_{s,i} * \log\left(\frac{N}{n_s}\right)\right)^2}}$$

donde:

$tf_{r,i}$ Es la frecuencia del término **t_r** (*term frequency*). Es la cantidad de veces que **t_r** aparece en el documento **d_i**.

N Es la cantidad total de documentos de la base.

n_r Cantidad de documentos en los que aparece el término **t_r**. Por lo mismo, **n_s** es la cantidad de documentos en los que aparece el término **t_s**.

idf_r Llamamos *frecuencia inversa* del término **t_r** a este factor. Como se puede ver, se calcula como el logaritmo del cociente entre **N** y **n_r**.

Las ideas básicas de esta fórmula se pueden resumir así:

- ✓ Si un término aparece muchas veces en un documento, entonces podemos empezar suponiendo que ese término es importante para ese documento (por lo tanto el valor **tf** crece)
- ✓ Pero si el mismo término aparece también en muchos documentos, entonces no es un término útil para distinguir a un documento de los otros: no es útil para discriminar qué tan bueno es un documento (**idf** decrece: el valor **n_r** tiende a **N**, con lo que el cociente **N / n_r** tiende a uno; y el logaritmo de ese cociente entonces tiende a cero...) Piense en una palabra como "*la*": en un documento aparecerá cientos de veces (y podemos pensar que para ese documento es una muy buena palabra...) pero también aparecerá en casi todos los otros documentos de la base... por lo que esa palabra no puede usarse para valorar si un documento es mejor que otro.
- ✓ Como se ve, empezar midiendo cuántas veces un término aparece en un documento es una idea intuitivamente buena (de hecho, podría pensarse en una fórmula trivial de cálculo de peso en la que el peso sea directamente el **tf** del término...) Pero hay que *ajustar* ese cálculo disminuyendo el peso si la palabra es muy común en todos los otros documentos. Para eso se usa el *idf*, y el "truco" de calcularlo como el logaritmo de un cociente que tiende a uno si el término es común en la base, es un recurso muy usado como factor de ajuste.
- ✓ En la fórmula usada, se divide por el módulo normalizado de los vectores (la raíz que figura en el denominador) para no favorecer a documentos más largos al discriminar. Otra vez, la división por el módulo del vector es un factor de ajuste.
- ✓ Lo que se busca es determinar cuánto ayuda un término a distinguir la relevancia de un documento entre todos los otros.

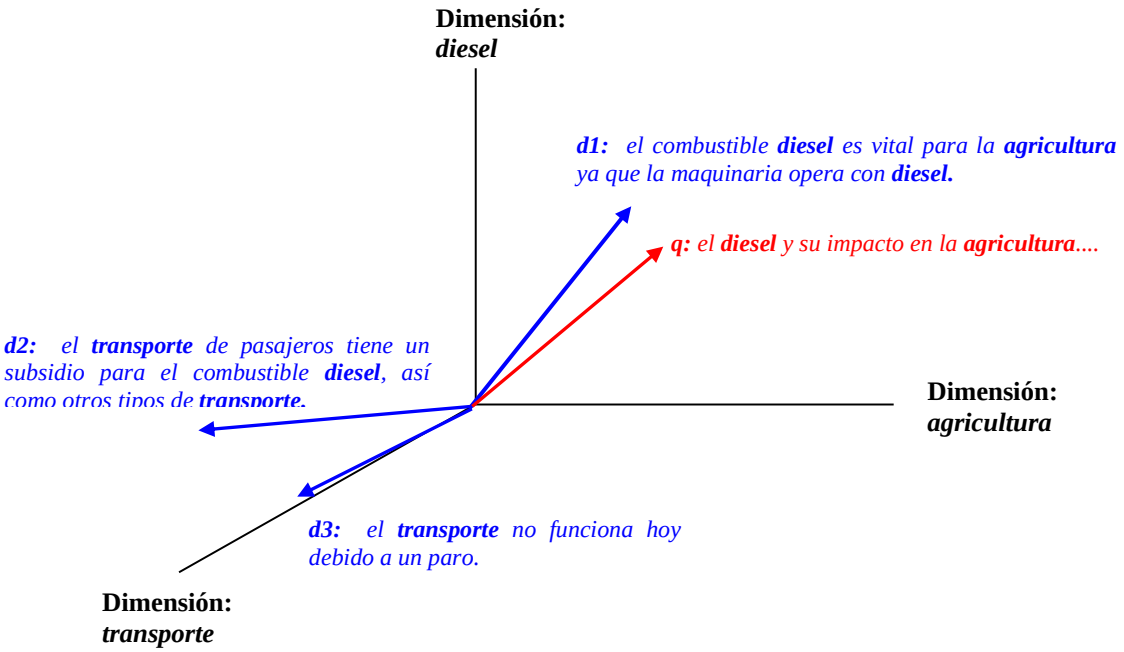
Si bien estamos hablando de documentos representados como vectores de **k** "coordenadas" y por lo tanto no es posible pensar en dibujarlos como se haría con vectores en un espacio geométrico de tres o dos dimensiones, podemos intentar sin embargo una *aproximación* gráfica para mostrar las ideas que siguen. Supongamos que el conjunto D de documentos sólo contenía estos tres simples documentos:

d1 = "El combustible diesel es vital para la agricultura"

d2 = "El transporte de pasajeros tiene un subsidio para el combustible diesel"

d3 = "El transporte no funciona hoy".

Si asumimos que los términos relevantes en el modelo serán sólo las palabras "*diesel*", "*transporte*" y "*agricultura*" (ignoraremos las demás para simplificar el ejemplo) tenemos entonces que cada documento se modelará como un vector de tres "coordenadas": cada una será el peso que estas tres palabras tienen en cada documento. Si cada palabra es una "coordenada" podemos hacer que cada una coincida con un eje de un sistema de coordenadas, y mostrar a cada documento en ese gráfico... Esto es una metáfora más que un gráfico técnicamente correcto, pero servirá para dar la idea:



Y si ahora se desea realizar una consulta tal como:

q = "*El diesel y su impacto en la historia de la agricultura*"

entonces se toma a la propia consulta **q** como un documento y se modela ella misma como un vector (ver gráfica). El tema crucial es ahora: ¿cómo comparar dos documentos? ¿Cómo determinar cuáles de los documentos de la base son más parecidos (o "similares") al documento **q** que representa a la consulta?

Si los documentos son vectores, se puede medir la similitud *sim(di, dj)* entre dos documentos **di** y **dj** midiendo *el ángulo* que forman los vectores que los representan. En la gráfica, es fácil ver que **q** "se parece" más a **d1** que a los otros dos documentos: el ángulo entre **d1** y **q** es más bajo que el ángulo entre **q** y los otros dos... En la práctica, la medición de ese ángulo se puede hacer en forma indirecta, calculando el coseno de este (y designamos a ese valor como la "*distancia coseno*" entre los documentos):

$$sim(di, dj) = \frac{\overrightarrow{di} \bullet \overrightarrow{dj}}{\|\overrightarrow{di}\| \, \|\overrightarrow{dj}\|} = \frac{\sum_{r=1}^{k} (w_{r,i} * w_{r,j})}{\|\overrightarrow{di}\| \|\overrightarrow{dj}\|}$$

El cálculo del coseno se hace mediante el *producto escalar* entre los vectores, dividiendo por el producto de los módulos (notar que el módulo de cada vector ya está incluido en el cálculo del peso, y por eso no se muestra el cociente en la gráfica). Si el ángulo entre dos vectores es cero (los vectores están "encimados" y si representan documentos entonces ¡se parecen mucho!) el coseno será igual a uno. Y si dos vectores son perpendiculares (no se parecen mucho si son documentos: comparten pocos términos) el coseno será igual a cero...

Por lo tanto, la similitud entre dos documentos será un valor entre cero y uno: mientras más cerca de cero esté ese valor, menos se parecen los documentos, y viceversa: mientras más cerca de uno esté, más parecidos son los documentos.

4.] Implementación del Modelo Vectorial.

Una vez analizadas las ideas del modelo matemático, la tarea es implementarlo mediante estructuras de datos y algoritmos en forma eficiente. En la práctica, la estructura más usada se conoce como "*estructura de índices invertidos*" y consta de los siguientes elementos:

- El **Vocabulario**: el conjunto T de todos los términos distintos de la base.
- El **Posteo**: para cada término en T, la lista de documentos donde aparece ese término

Ejemplo: sean los siguientes documentos:

d1 = "El combustible diesel es vital para la agricultura"

d2 = "El transporte de pasajeros tiene un subsidio para el combustible diesel"

d3 = "El transporte no funciona hoy".

d4 = "Hay transportes y transportes..."

d5 = "El diesel venezolano es de menor calidad que el diesel argentino"

Las estructuras podrían verse así (obviando algunas palabras irrelevantes):

Vocabulario	**Posteo**
combustible	d1, d2, d5
diesel	d1, d2, d5
maquinaria	d1
agricultura	d1
transporte	d2, d3, d4
paro	d3
venezolano	d5
calidad	d5
argentino	d5

En la implementación deben tenerse en cuenta las siguientes consideraciones:

a) El *vocabulario* normalmente puede implementarse *en memoria*: diversos estudios y leyes empíricas muestran que la gente tiende a usar las mismas palabras en los documentos. Por ejemplo, un conjunto de 1 Gb de texto puede aportar unos 5 Mb de palabras distintas. Por lo tanto, se puede pensar en implementar el vocabulario como una tabla de búsqueda rápida (*hashing*).

b) Las *listas de posteo* consumen mucho espacio, y *se implementan en disco* (normalmente todas juntas en el mismo archivo). Cada entrada del vocabulario debe disponer de un campo que indique en qué lugar del archivo de posteo empieza su propia lista.

c) Para permitir un ordenamiento de los documentos relevantes frente a una consulta, se recomienda que cada entrada del posteo almacene el **tf** del término referido.

d) En cada entrada del vocabulario, almacenar el **idf** (en la práctica se almacena el valor $\mathbf{n_r}$)

e) También es conveniente almacenar el máximo **tf** de cada término en el vocabulario.

f) Y finalmente, almacenar la lista de posteo de cada término en orden decreciente de **tf**.

Con estos elementos, la gráfica para los documentos de ejemplo vistos antes puede replantearse así:

Vocabulario		***Posteo***
combustible	**[n_r=3 max tf=1]**	d1 **(tf=1)**, d2**(tf=1),** d5**(tf=1)**
diesel	**[n_r=3, max tf=2]**	d5**(tf=2)**, d1**(tf=2)**, d2**(tf=1)**
maquinaria	**[n_r=1, max tf=1]**	d1**(tf=1)**
agricultura	**[n_r=1, max tf=1]**	d1**(tf=1)**
transporte	**[n_r=3, max tf=2]**	d4**(tf=2)**, d2**(tf=2)**, d3**(tf=1)**
paro	**[n_r=1, max tf=1]**	d3**(tf=1)**
venezolano	**[n_r=1, max tf=1]**	d5**(tf=1)**
calidad	**[n_r=1, max tf=1]**	d5**(tf=1)**
argentino	**[n_r=1, max tf=1]**	d5**(tf=1)**

Con estas estructuras armadas (es decir, habiendo armado los *índices*), una consulta se puede resolver de varias formas. La forma básica requiere de las siguientes consideraciones:

- Una consulta **q** puede tener varios términos, y nos interesan los **R** documentos más relevantes.
- Las listas de posteo de cada término están almacenadas en orden decreciente de **tf**. La idea: mantener un ranking de los **R** documentos **di** con mayor ***sim(di, q)***.
- Se comienza con el término de la consulta que tenga el mayor **idf** (o sea, la lista de posteo más corta. En nuestro caso, comenzamos por el término con menor $\mathbf{n_r}$) y traemos de su lista de posteo los **R** primeros documentos. Si no se llega a reunir **R** documentos, se continúa con el segundo término de mayor **idf**, y así sucesivamente.
- Aún cuando ya se tengan **R** documentos candidatos, se siguen recorriendo las listas de los términos, de mayor a menor **idf** (por ejemplo, tomando hasta *R* candidatos de cada lista).
- Notar que como el **tf** en cada lista de posteo va decreciendo, en algún momento podemos cortar el recorrido de alguna lista, pues los candidatos ya no van a entrar al ranking de los **R** mejores. Por otra parte, como el máximo **tf** está almacenado en

cada término del vocabulario, es posible eliminar términos completos sin siquiera ir al disco una vez para mirar su lista de posteo.

- Para armar el ranking, los documentos se van manteniendo en el orden en que ingresan, pero si al chequear la lista de otro término el mismo documento aparece otra vez, se puede hacer que suba en el ranking general.
- De todos modos, como la lista de documentos recuperados debe ser ordenada de mayor a menor relevancia frente a la consulta **q**, es necesario entonces que cada documento **di** sea ponderado con un valor o número que de alguna manera mida su relevancia frente a la consulta. En principio, puede usarse la fórmula de la distancia coseno para medir el ángulo entre cada documento recuperado y la consulta, pero para simplificar los cálculos y evitar pérdida de tiempo, se puede hacer una calificación local basada en la expresión $tf_{r,i} * idf_r = tf_{r,i} * log(N/n_r)$ (esto es, el cálculo del peso de un término *r* para el documento *i*, pero sin dividir por el producto de sus módulos). Se califica de esta forma cada término de la consulta en cada documento recuperado, se suman esas calificaciones, y el valor de la suma se propone como valor de ranking del documento.

La forma de realizar una consulta y obtener el ranking puede ser muy diferente en cada motor: de hecho muchos de los motores comerciales actuales utilizan variantes en la forma de organizar los índices y luego en la forma de obtener los rankings, y muchas de esas estrategias no son de dominio público... Aquí hemos indicado la forma más elemental de comenzar a plantear el motor y la forma de hacer el ranqueo. Hemos agregado (al final de esta Ficha) un *esquema de pseudocódigo* para el proceso general y más básico de respuesta a una consulta.

Debe notarse un detalle significativo: el modelo vectorial *está implícito* en las estructuras de datos mostradas aquí y en los algoritmos usados para rescatar documentos de ellas. La forma de organizar el vocabulario y las listas de posteo incluye los factores de las fórmulas de peso y cálculo de coseno, distribuidos de forma tal que en la práctica es suficiente con eso... no debería el programador tener que calcular efectivamente los pesos de cada término sobre cada documento, ni guardar esos pesos en ninguna parte, sino simplemente aplicar una consulta en la forma explicada a página anterior en esta Ficha.

5.] Esquema de pseudocódigo: Proceso (básico) de respuesta a una consulta.

Proceso: Respuesta a una Consulta

Entradas:

1. El conjunto *D* con *N* documentos d_i.
2. El índice invertido ya creado, conteniendo:
   a.) Vocabulario: El conjunto *T* con los *k* términos t_k contenidos en los *N* documentos.
   b.) Posteo: La lista de posteo P_k de cada término t_k.
3. La cantidad *R* de documentos que se desea recuperar.
4. La consulta *q* realizada por el usuario.

Salidas:

1. Una lista *LD*, con los *R* (o más) documentos relevantes para *q*, ordenada de mayor a menor relevancia.

Algoritmo:

1. Sea LD = [ ] (lista vacía).

2. Mientras queden términos en *q* sin procesar:
    a. En *q* tomar el término t_k que tenga menor longitud de P_k (es decir, el que tenga el mayor *idf*).
    b. En la lista P_k de t_k, repetir *R* veces:
        i. Si el documento actual d_i no está en *LD* agregar en *LD* el documento actual d_i, junto con su índice *ir* de relevancia (valor inicial *ir* = 0). Mantener *LD* ordenada por *ir*.
        ii. Sumar en el *ir* del documento d_i el valor tf_k * *idf* (la frecuencia del término t_k en el documento d_i por la frecuencia inversa del término t_k).
    c. Tanto si se recuperaron *R* documentos en el paso anterior, como si no, volver al ciclo del paso 2, y continuar con el siguiente término.
3. Mostrar los primeros *R* documentos de *LD* (incluyendo el índice *ir* de cada uno si se trata de un prototipo).