Cerca i Anàlisi de la Informació

Carlos Arbonés and Juan P. Zaldivar

GCED, UPC.

Apunts.

Contents

1 Introduction

1.1 Information Retrieval

1.1.1 Information Retrieval vs Database Queries

Para obtener unos datos hace falta tenerlos, pero no solo eso. También debemos saber donde los almazenamos. Las *queries* de bases de datos se centran en las tuplas y en el esquema de la BD.
En IR:

- Podemos **no saber donde** se encuentra la información
- Podemos **no saber si** esta información existe
- **No tenemos un esquema** como en las BD relacionales
- Podemos **no saber exactamente qué** información queremos (de manera precisa)

1.1.2 User Expectations

A menudo **no sabemos lo que queremos preguntar exactamente**, por lo tanto el concepto de *relevancia* es importante y está lejos de ser no trivial. En IR queremos resultados que sean **relevantes** para la *query* y con suerte el sistema devolverá la información que es más relevante para el usuario.

1.1.3 Information Retrieval Models

Un **modelo de IR** se caracteriza por:

- La noción de **documento**. Se trata de una pieza de información, una abstracción de documentos reales.
- La noción de una **query** admisible. Se necesita un lenguaje para las *queries*, una secuencia de palabras clave o tokens.
- La noción de **relevancia**. Se trata de una función de la pareja *documento-query* y nos dice qué tan relevante es ese documento para esa *query*. Se debería devolver el documento con mayor relevancia. El rango de valores puede ser *booleano, rango, valores reales...*

1.1.4 The Information Retrieval Process

Existe el **proceso offline** que consiste en *crawling, preprocessing* y *indexing*. Crawling es el mecanismo que trata de explorar la *web* y guardar el contenido de estas en la BD con documentos *raw*. Desde los ficheros *raw* hacemos el preprocesado, seleccionar lo que es más importante e indexar la información encontrada para estructurala de manera eficiente. El **proceso offline** tiene como objetivo preparar las estructuras de datos para que el **proceso online** sea más rápido. Puede permitirse largas computaciones y debe producir un *output* razonablemente compacto (estructura de datos). La **eficiencia es la clave**, sea lo que sea que usamos para acceder las estructuras debe ser muy rápido.

Por otro lado, también tenemos el **proceso online**. Consiste en obtener las **queries**, extraer **documentos relevantes**, **rank** los documentos por relevancia y **formatear la respuesta** para devolverla al usuario. El objetivo es una reacción instantánea. El proceso se puede observar en la Figura 1.1.

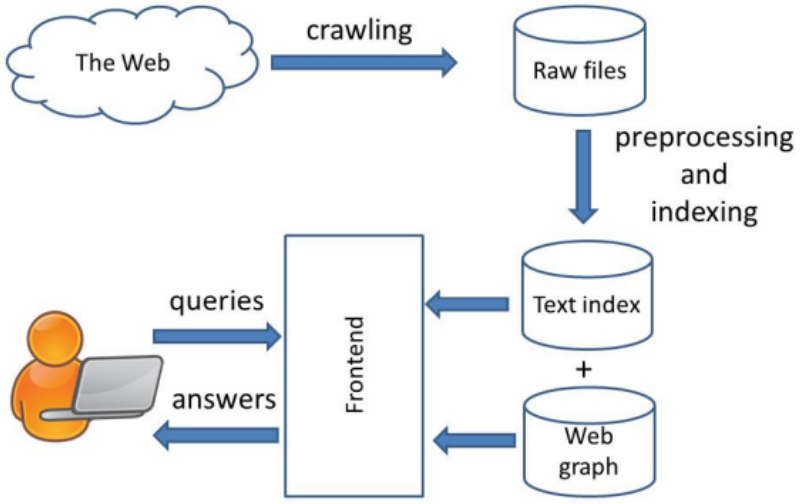


Fig. 1.1: *The Information Retrieval Process*

1.2 Preprocessing

Las potenciales acciones que se pueden realizar son:

- **Parsing:** se trata de hacer un barrido de la secuencia de símbolos de entrada que nos permita sacar alguna estructura que no nos interese. Por ejemplo, si estamos mirando una *web*, nos interesara sacar los *tags*. Para **extraer la estructura** necesitamos saber que tipo de documento estamos leyendo ya que será diferente en cada caso.
- **Tokenization:** a partir de nuestra secuencia de símbolos lo que haremos es descomponer esta secuencia en unidades individuales (palabras). Lo más fácil será coger los espacios para determinar las palabras y los signos de puntuación, pero habrán errores asociados.
- **Enriching:** añadir información adicional a los tokens, lo que hará más fácil el proceso de recuperación de la información.
- **Normalización:** mediante *Lemmatization* o *Stemming*, reducir las palabras (mapeo) a la raíz.

1.2.1 Tokenization

Lo más común es juntar caracteres consecutivos formando *palabras*, usamos los espacios y los signos de puntación para marcar las fronteras. Aunque parece fácil aparecen errores como *IP's* y numero de teléfono, conjuntos como *R+D*, *753 B.C.* También hay problemas con los guiones, los quitamos y unimos las palabras o los dejamos? Un paso más adelante es la *Named Entity Recognition*, conceptos como *The president of the United States* o *June 6th, 1944*.

Además existe el **Case folding**, pasar todo a minúscula para que las búsquedas sean independientes de como esté escrito. Pero *Usa* a *usa* o *Windows* a windows.

Hay palabras de aparecen en todos los documentos y que por lo tanto no ayudan. El **Stopword removal** elimina palabras como preposiciones, artículos, verbos muy comunes, etc. Puede reducir el texto hasta un 40%. Pero también se pueden quitar palabras que no se deberían y puede generar problemas por lo que la tendencia actual es dejar todo y filtrar documentos por relevancia.

1.2.2 Enriching

Cada término se asocia con información adicional que puede ser útil para conseguir los documentos "buenos". Por ejemplo el uso de sinónimos (*gun* a *weapon*), palabras relacionadas (*laptop* a *portable computer*), categorias (*fencing* a *sports*), ...

1.2.3 Lemmatizing and Stemming

Stemming consiste en quitar los sufijos de las palabras (por ejemplo de *swim, swimming, swimmer, swimmed* a *swim*), mientras que **Lemmatizing** consiste en reducir las palabras a sus raíces linguísticas como *be, am, are, is* $\rightarrow$ *be*, *gave* $\rightarrow$ give, ... La primera es mas sencilla y rápida, aunque imposible en algunos idiomas. La segunda es más lenta pero más precisa.

1.3 Text Laws

Tiene que ver con la frecuencia en la que aparecen las palabras en un texto, las preposiciones por ejemplo (*stepwords*) son mucho más frecuentes que las demás, pero cuánto de más? Cuánto más frequente es la primera palabra más frecuente respecto la segunda, y respecto la tercera? La ley de *Zipf* nos da cómo están distribuidas las frecuencias de las palabras.

También veremos con la ley de *Heaps* que determina, dentro de un texto con muchas palabras, cuántas palabras diferentes encontramos.

1.3.1 Zipf's Law

Si dibujamos el plot de las palabras más frecuentes y su frecuencia lo que encontramos es una **curva que decrece más lentamente que una Gaussiana**, que tiene **heavy tails**, si sumas el área de esa zona tiene un peso no nulo.

Para definir de forma más matemática la distribución usamos la ecuación de *Zipf-Mandelbrot*. La distribución se define como

$$fr(i) \approx \frac{c}{(i+b)^a}$$

donde la variable i **es el rango de los elementos**, el elemento más frecuente tendrá rango igual a 1, el segundo elemento más frecuente igual a 2, etc. En un texto como por ejemplo *Don Quijote* contaremos cada palabra cuántas veces aparece, ordenaremos por esa frecuencia con la palabra más frecuente en la primera posición con $i = 1$.

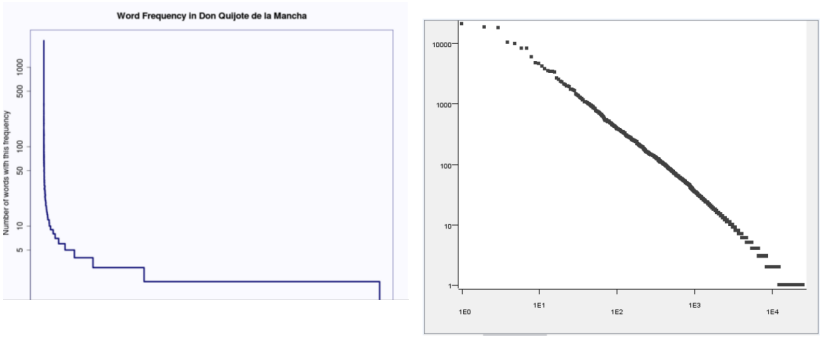


Fig. 1.2: *A la izquierda la frecuencia de las palabras en el libro Don Quijote, a la derecha el mismo plot con log-log scale*

a, b i c son constantes que determinaran la forma exacta la distribución y que tendremos que ajustar a los datos empíricos que observaremos y que dependerán de cada texto. Para obtener la a podemos calcular la pendiente de la recta en log-log plot.

Para detectar *power laws* lo que se hace es ordenar de más frecuente a menos y hacer el plot de frecuencia vs rango. Para detectar a ojo lo que haremos es hacer el plot con escala logarítmica y ajustar una recta (con regresión lineal).

1.3.2 Heaps' Law

Para calcular el tamaño de un vocabulario en función de la longitud del texto que estamos analizando. Si tenemos textos más largos lo normal es que usen más palabras, pero lo que se observa que cada vez se observan menos palabras nuevas.

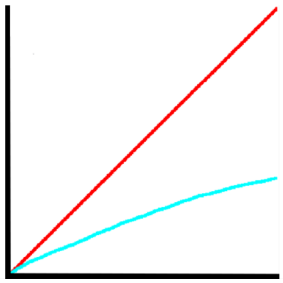


Fig. 1.3: *La **linea roja** indica el número de palabras leídas y la **linea azul** el número de palabras distintas. Al principio la azul crece muy rápido y va disminuyendo la pendiente.*

El número de palabras diferentes se puede describir con un polinomio de grado menor que 1, por ejemplo $\log(x)$ o $\sqrt{x}$, lo que significa que crece a una velocidad menor que lineal. Si hacemos el *log-log plot* la linea azul se convierte en una recta.

Para un texto de N palabras, le diremos a d el número de palabras diferentes encontradas. Si hacemos el *log-log plot* obtenemos que

$$\log d = \log k + \beta \cdot \log N, \text{ that is } d = k \cdot N^{\beta}$$

El valor de β varia según el idioma y el tipo de texto. Al tener un **vocabulario finito** implica que para N grande no habrá más crecimiento, se estabilizará.

2 Two Information Retrieval Models

El **modelo booleano** es un sistema que no considera las matices de frecuencia, un documento es totalmente relevante o no lo es. El **modelo vectorial** si que se permiten una serie de pesos que nos permiten más flexibilidad. Esto nos dejará dar un ranking a la respuesta.

Los dos modelos son modelos ***Bag of Words***, que significa que el orden en el que aparecen las palabras no es relevante. Si hiceramos un *shuffle* de todas las palabras del documento la representación seria la misma. Este sistema de representar un documento seria un conjunto de parejas *(palabra, frecuencia)* dónde vemos cada palabra cuantas veces aparece. El modelo booleano le da igual la frecuencia, solo mira si la palabra aparece o no (0,1).

En la matriz de frecuencias, las filas son los documentos y las columnas son los todos los términos. Los términos son **ordenados** de manera alfabética, lo que nos ayudará más tarde. En cada posición, se indica el número de coincidencias en las que aparecen los términos en el documento.

2.1 Boolean Model of Information Retrieval

En representación *sparse*, se crea un vector de parejas para cada documento en el que cada pareja se almacena la palabra que aparece y su número de ocurrencias en el documento.

Un documento está completamente identificado por el conjunto de términos que aparecen. Para un conjunto de elementos $\tau = \{t_1, \ldots, t_T\}$ un documento es solo un **subconjunto** de τ. Cada documento se puede visualizar como un **vector de bits** de longitud T, $d = (d_1, \ldots, d_T)$ donde:

- $d_i = 1$ si y solo si t_i aparece en el documento
- $d_i = 0$ si y solo si t_i no aparece en el documento

2.1.1 Queries

Un **atomic query** seria un **solo término**. La noción de relevancia de una query es que los documentos relevantes son los que contienen la palabra de la *query*. Se pueden **combinar querys**:

- OR, AND
- t_1 BUTNOT t_2

Solo se permiten este tipo de quieries y no por ejemplo NOT porque en la mayoria de las consultas el resultado concurriria en un gran numero de documentos que coinciden con la query.

2.1.2 Phrase Queries

Para hacer más flexible el modelo con *frases hechas*. No solo queremos que el modelo busque que las dos palabras esten en el documento sinó que además una esté detrás de la otra, teniendo en cuenta la adyacencia de las palabras. Esto es también interesante en nombres propios.

Tenemos que enriquezer el modelo ya que el **Bag of Words** pierde el orden, tenemos que implementarlo encima del modelo. Para hacer que el sistema ejecute este tipo de *queries* se puede implementar lo siguiente:

- Ejecutar la query **como una conjunción** y lo que hacemos después es escanear la lista de resultados y **verificar** que una palabra salga detrás de la otra. Puede ser

muy costoso (requiere escanear a través de la lista de resultados) y lento si existen falsos positivos.
- Hacer el *index* más sofisticado, guardar no solo si sale la palabra sino también la posición en la que sale.
- Guardarse directamente el índice de las palabras que aparecen juntas muchas veces que las demás (*interesting pairs*).

2.2 Vector Space Model of Information Retrieval

El orden de las palabras sigue siendo irrelevante (***Bag of Words***) pero la **frecuencia es relevante**, nos guardaremos el número de veces que aparecen. **No todas las palabras son igualmente importantes**, por ejemplo las *step words* no son demasiado importantes ya que todos los documentos las tienen, como **mas discriminativa** sea una palabra **mayor peso** debe tener.

Los documentos continuaran siendo vectores pero en vez de vectores de bits, **vectores de floats**, $d = (w_1, \ldots, w_T)$ donde w_i es el **peso** de la palabra t_i en d.

Ahora los documentos son vectores en $\mathbb{R}^T$. La colección de documntos **conceptualmente** se convierte en una **matriz** de *términos* $\times$ *documentos*.

2.2.1 Weights assignation

Se usará el esquema ***tf-idf*** que se basa en dos principios:
1. Como más frecuente sea t en d mayor debe ser el peso.
2. Como más frecuente sea t en la **colección de documentos**, menos discrimina entre los documentos y menor el peso debe ser en todos los documentos.

Un documento es un vector de pesos $d = [w_{d,1}, \ldots, w_{d,i}, \ldots, w_{d,T}]$ donde cada **peso** es un producto de dos términos

$$w_{d,i} = tf_{d,i} \cdot idf_i$$

donde $tf_{d,i}$ es la **frecuencia del término** i en el documento d "*normalizado*" y se puede calcular como

$$tf_{d,i} = \frac{f_{d,i}}{\max_j f_{d,j}}$$

donde $f_{d,j}$ es la frecuencia de t_j en d y la **frecuencia inversa del documento** idf_i

$$idf_i = \log_2 \frac{D}{df_i}$$

con D = número de documentos y df_i = número de documentos que contienen el término t_i. Se puede observar que cuando un término aparece en todos los documentos $D = df_i$ y por lo tanto el peso de ese término será 0.

Hay formas de modificar este esquema:
- Como en general tratamos documentos que tienen algún **tipo de estructura** (HTML) podemos aumentar los pesos de palabras que por ejemplo aparecen en negrita o que aparecen en el título.
- **Corrección de Laplace**. Es equivalente a tener un documento extra con todas las palabras. Si algun término no aparece en nuestro corpus hay veces que no tiene sentido que tenga un peso nulo. Se suaviza de la siguiente forma

$$idf_i = \log_2 \frac{D+1}{df_i + 1}$$

2.2.2 Comparison of documents - Cosine Similarity

Todos los documentos son representados como vectores. Para ver la similitud entre documentos se usa la **similaridad del coseno**. Lo que estamos haciendo es: si tenemos dos documentos y queremos verificar si se parecen o no lo que hacemos es comprobar **si tienen una dirección similiar**, que apuntan al mismo sitio y que están utilizando el mismo vocabulario.

$$sim(d1, d2) = \frac{d1 \cdot d2}{|d1||d2|} = \frac{d1}{|d1|} \cdot \frac{d2}{|d2|}$$

Como los pesos son todos **no negativos**, esta similitud está siempre en el rango $[0, 1]$. En el caso de tener similitud 0, los dos vectores son ortogonales.

2.2.3 Query Answering

Esta noción de similaridad se usa en el momento de contestar una pregunta. Cogeremos la *query* del usuario teniendo previamente ya calculada la matriz del *corpus* y la transformaremos en un vector de pesos. Normalmente la transformaremos en un vector de bits, donde aparecerá 1 en las palabras de la *query*. Si el usuario pone dos términos significa que buscaremos documentos dónde tengan ambos importancia.

Lo que haremos es **calcular la similaridad** del coseno entre cada uno de nuestros documentos y la query que nos da el usuario. La respuesta que se dará será una **lista de los documentos ordenados por similaridad** en orden decreciente.

2.3 Evaluation

Saber si un sistema es bueno o malo es muy importante. Un **sistema bueno** es una que dada una query nos da los documentos realmente más importantes, que la respuesta del sistema sea la más alineada posible con la realidad.

Empezaremos con el **modelo Booleano** donde

- $\mathcal{D}$: conjunto de todos los documentos (*corpus*)
- $\mathcal{A}$: conjunto de documentos que el sistema devuelve para una *query* fijada.
- $\mathcal{R}$: documentos relevantes, son los que al usuario les gustaría tener de respuesta. En la práctica no se conocen (ni si quiera el usuario de la query).

2.3.1 Recall and Precision

Tenemos dos medidas conocidas:

- **Recall**: capacidad que tiene el sistema de realmente devolver los documentos que son relevantes. De entre todas las respuestas relevantes cuáles forman parte de la respuesta.

$$\frac{|\mathcal{R} \cap \mathcal{A}|}{|\mathcal{R}|}, \ \Pr(d \in \mathcal{A} | d \in \mathcal{R})$$

- **Precisión**: De los documentos que nos da el sistema cuántos hay que sean relevantes. De todas mis respuestas cuáles son relevantes.

$$\frac{|\mathcal{R} \cap \mathcal{A}|}{|\mathcal{A}|}, \ \Pr(d \in \mathcal{R} | d \in \mathcal{A})$$

Es difícil obtener índices altos en las dos medidas. un sistema con **recall perfecto** es aquel que devuelve todos los documentos, pero a costa de que la precisión sea muy baja. Para obtener **precisión perfecta** podemos no enviar ninguna respuesta, pero a costa

del recall. Se tiene que intentar que las **dos medidas sean grandes**. Dependiendo del caso alomejor nos interesa más una medida que la otra.

Un mayor conjunto de respuestas normalmente resulta en una mejora del recall pero no de la precisión y viceversa con un conjunto de respuestas menor.

		Answered	
		relevant	not relevant
Reality	relevant	tp	fn
	not relevant	fp	tn

- $|\mathcal{R}| = tp + fn$
- $|\mathcal{A}| = tp + fp$
- $|\mathcal{R} \cap \mathcal{A}| = tp$

- Recall = $\frac{|\mathcal{R} \cap \mathcal{A}|}{|\mathcal{R}|} = \frac{tp}{tp+fn}$
- Precision = $\frac{|\mathcal{R} \cap \mathcal{A}|}{|\mathcal{A}|} = \frac{tp}{tp+fp}$

Fig. 2.1: *Confusion Matrix*

2.3.2 How many documents to show?

Una respuesta muy extensa puede que no sea interesante al usuario. Respuestas muy largas tienden a exhibir **baja precisión**. Respuestas muy cortas suelen exhibir **bajo recall**. Analizaremos la precisión y el recall como funciones del número de documentos k entregados como respuesta.

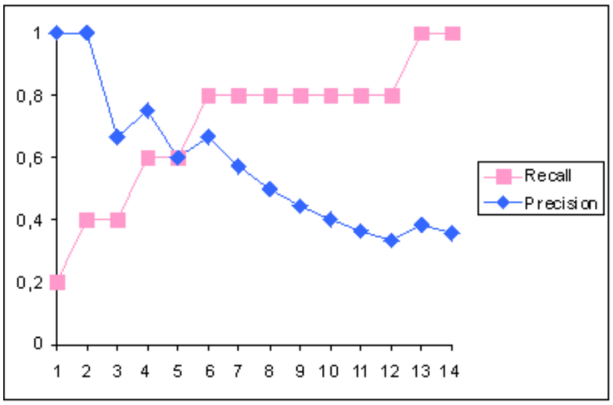


Fig. 2.2: *Rank-recall and rank-precision plots*

La longitud de la respuesta en el eje horizontal. Entre mas documentos se añaden, el recall solo tiende a aumentar. En el primer documento como respuesta se obtiene un 20% de recall, lo que indica que se ha seleccionado un documento relevante de entre 5 posibles y la precisión es del 100%. La precisión siempre baja ya que a más documentos que se entregan, mayor es la posibilidad de equivocarse, es decir, que el algún documento no sea relevante.

El **plot ideal** es aquel que pone todos los documentos relevantes en las primeras posiciones. Como más tarde en crecer la curva del recall es que los relevantes están más

atrás. La curva de la precisión debería de ser 1 hasta que se acaben todos los documentos relevantes, que entonces empezaría a bajar.

Precision and Recall curve

Lo que se hace es poner la precisión en función del recall. Por ejemplo cuál es la precisión hasta que encontramos el 10% de los documentos relevantes? Lo más normal es que para llegar al 100% de documentos relevantes (recall máximo) nos tendremos que tragar muchos que no lo son y la precisión será baja. Un sistema muy bueno encontrará todos los relevantes en las primeras posiciones y por lo tanto la curva será recta (será un rectángulo). Los peores sistemas estarán muy debajo de este rectángulo perfecto.

La curva roja de la Figura 2.3 es la precisión interpolada, se usa para que no aparezcan estos picos negativos. Se coge el valor más grande tirando hacia la derecha.

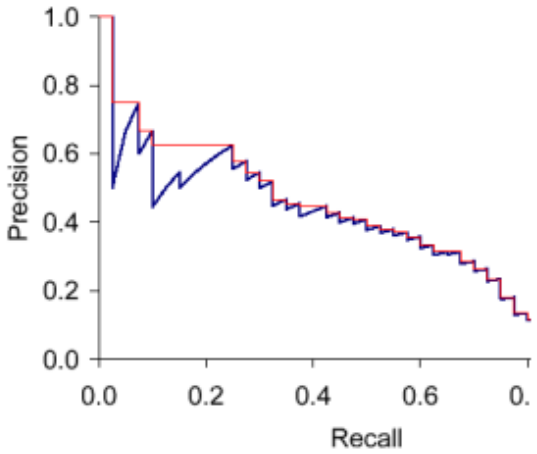


Fig. 2.3: *Recall vs Precision plot*

2.3.3 Other measures of effectiveness

Podemos recurrir a otras medidas para calcular la efectividad de nuestras respuestas:

- **AUC (Area Under Curve)**: Usamos el área debajo la curva de la Figura 2.3 para comparar sistemas Un sistema que tiene un mayor recall será mejor que uno que tiene picos de bajadas.
- **F-measure**

$$\frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}}$$

- α-**F-measure**

$$\frac{1}{\frac{\alpha}{\text{recall}} + \frac{1-\alpha}{\text{precision}}}$$

Hay otras medidas que van un poco más lejos de evaluar los sistemas solo en base a la relevancia. Tienen que ver con un sistema capaz de darnos documentos que nosotros no conocíamos a priori. Un sistema bueno nos da información que es nueva para nosotros, que nos sorprenda.

- **Coverage**

$$\frac{|relevant \ \& \ known \ \& \ retrieved|}{|relevant \ \& \ known|}$$

- **Novelty**

$$\frac{|relevant \ \& \ retrieved \ \& \ unknown|}{|relevant \ \& \ retrieved|}$$

2.3.4 Relevance Feedback

Formas que tiene el sistema de refinar las respuestas que da al usuario. Se implementa mediante el **ciclo de relevancia del usuario**:

1. El usuario da una query q
2. Se obtienen documentos relevantes (deberían) para q
3. Se muestran al usuario los k documentos más relevantes
4. El usuario marca cuáles son y no son relevantes de todos estos documentos.
5. Las respuestas del usuario son utilizados para **refinar** la consulta original q y dar una nueva respuesta.
6. Si deseado, volver a **2.**

Una vez el usuario obtenie su repuesta, este da **feedback** para poder refinar la consulta. Hay muchas maneras de implementar esto, una de ellas es la **Rocchio's rule**.

Rocchio's rule

Una forma de **refinar** la query en base al **feedback** de un usuario. Los inputs que se necesitan son:

- **Primera query del usuario** q, ve interpreta como un vector. Vector de bits con 1 en las palabras que ha puesto el usuario.
- **Vector** que obtenemos de **promediar** todos los documentos que el usuario selecciona como **relevantes** de la respuesta que hemos dado.

$$\frac{1}{|R|} \sum_{d \in R} d$$

- **Vector** que obtenemos de **promediar** todos los documentos que el usuario selecciona como **NO relevantes** de la respuesta que hemos dado.

$$\frac{1}{|NR|} \sum_{d \in NR} d$$

- α es el grado de confianza que tenemos en la query original (si α es mas grande en relación con los otros dos parámetros, la query original no sufrirá un gran cambio). β es el peso de la información positiva (términos que no aparecen en la query pero si aparecen en los documentos relevantes). γ es el peso de la información negativa. Normalmente $\gamma = 0$

$$\alpha > \beta > \gamma \geq 0$$

La idea es dar más peso a aquellas términos que aparecen en los documentos relevantes y quitar peso (restar) a aquellos términos que salen en documentos no relevantes. **Todos los vectores q i d's tienen que estar normalizados (L2)**.

$$q^{'} = \alpha \cdot q + \beta \cdot \frac{1}{|R|} \sum_{d \in R} d - \gamma \cdot \frac{1}{|NR|} \sum_{d \in NR} d$$

En la practica normalmente no se usa demasiado porque lo que se observa es que lo que suele mejorar el primer ciclo que se implementa ayuda al recall, pero las siguientes

rondas no se observa una mejora significativa. Como la precisión se prioriza mas que el recall, en este contexto puede no ser productivo.

Para implementar esto tenemos que preguntar al usuario y en el entorno de sitios web se asume que el usuario quiere respuestas muy rápidas.

Query expansion

El relevance feedback se puede entender como una **query expansion**. Lo que estamos haciendo es dada una query original añadir términos no 0, la estamos enriqueciendo que podrían estar relacionados con los términos de la query original.

Ej:

$$q = [1, 0, 0, 1, 0, 0, 1] \rightarrow q' = [x_1, x_2, x_3, x_4, 0, x_5, x_6],\ x_i > 0$$

Pseudorelevance feedback

En la práctica esto no se realiza. Lo que queremos evitar es preguntar demasiadas cosas al usuario. La paciencia del usuario es primordial. Una forma de implementar esto es con la pseudorelevance feedback. Lo que hacemos es en vez de esperar a que el usuario marque los documentos relevantes, se **asumen** que los primeros k documentos son realmente relevantes (como si los marcara el usuario), e implementar la **Regla de Rocchio** sobre esta información. Hay que parar cuando el sistema devuelva los mismos k documentos que la ronda anterior.

Otras fuentes alternativas de feedback son:

- Los links que el usuario hace click
- El tiempo que se mira un item determinado
- El historial de búsqueda de usuario

3 Implementation

Como hacer el query answering eficientemente, no importa que el sistema sea rápido para cuando venga el usuario.

Un mal algoritmo seria:

Algorithm 1 Sequential Search

```
input query q
for every document d in database do
    if d matches q then
        add its docid to list L
    end if
end for
output list L (perhaps sorted in some way)
```

El problema esta en que en la búsqueda secuencial, el tiempo de búsqueda es lineal al número de documentos guardados (cada vez que se hace una query tenemos que recorrer toda la BD). Implica que contestar cualquier pregunta tomaría mucho tiempo al haber muchos documentos. Intentamos que el tiempo de respuesta no dependa del corpus, sino del tamaño de la respuesta (pues esta siempre se tiene que devolver al usuario).

3.1 Central Data Structure

El **Documento/Índice invertido** es como un vector/diccionario que "cabe" en la memoria (con largo tiempo de construcción durante el preprocesado). En lugar de una colección de los documentos con las palabras que contienen accederemos a los documentos a través de las palabras.

Tenemos un índice (ver Figura 3.1) para cada palabra que nos lleva a los documentos dónde aparece. Este índice nos permite encontrar en tiempo constante la lista de documentos que contienen la palabra y que tal vez son relevantes para el usuario.

Obviamente, la creación de este índice nos costará un tiempo, pero esto es tiempo de preprocesado, no se construye cuando el usuario hace la query (tiempo offline).

3.1.1 Postings

Asignamos un *document identifier (**docid**)* a cada documento. Los archivos invertidos están conformados por **postings**. Para cada palabra indexada se le asigna una **posting list** con los identificadores de los documentos donde aparece y más información adicional. Normalmente el diccionario cabe en memoria RAM. Lo que no cabe son los *postings lists* (se tienen que guardar en disco). Normalmente están comprimidas, de manera que el paso de disco a memoria es mucho más rápido.

3.2 Implementation of the Boolean Model

Conjuctive query: a AND b

Esperamos encontrar todos los documentos que contengan ambas palabras. Se obtienen las **posting list** de ambas palabras y se hace la intersección (en caso de que las listas estén ordenadas, se pueden aplicar algoritmos para reducir el tiempo). El tiempo sera proporcional a la suma de las longitudes de las **posting lists**.

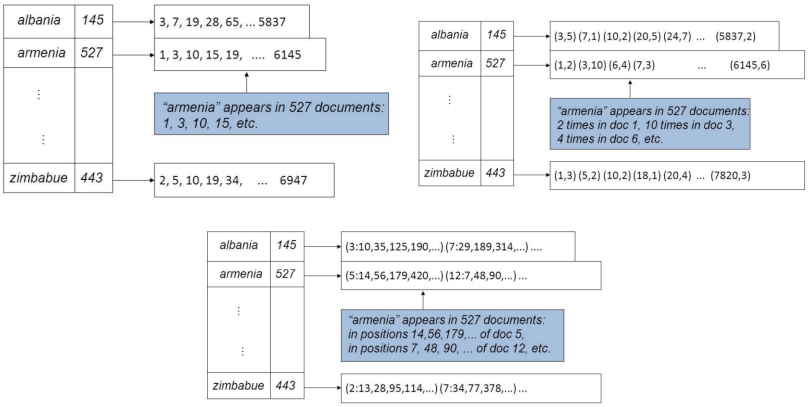


Fig. 3.1: *Tres tipos de variantes del inverted file. En la variante 1 se guarda el índice de documento en los que aparece la palabra. En la variante dos también se contienen las frecuencias correspondientes a cada documento que contiene la palabra. En la variante 3 se añade la posición de la palabra en los distintos documentos. Con esta información se puede reescribir todo el corpus.*

3.3 Query Optimization

Hay un problema de optimización para hacer mejor la query, pues hay formas equivalentes de escribir la query que pueden influenciar el tiempo de búsqueda. Es difícil escoger la mejor opción porque carecemos de información como la longitud de las listas, los documentos, etc. Se usaran aproximaciones o limites superiores/inferiores para las la optimización.

$$|L_1 \cap L_2| \leq \min(|L_1|, |L_2|)$$
$$|L_1 \cup L_2| = |L_1| + |L_2| - |L_1 \cap L_2| \leq |L_1| + |L_2|$$

Este proceso de buscar un plan de evaluación adecuado se llama *query optimization*. Usaremos leyes de álgebra booleana, algoritmos de intersección y unión y usaremos estructuras de datos más complicadas.

La intersección tiene un **coste lineal** respecto a la medida de las dos listas. Por lo que el **coste (número de comparaciones**) de hacer la intersección utilizando un escaneado secuencial es equivalente a la **suma del tamaño de las dos listas**.

$$Cost(A \cap B) = |A| + |B|$$

Una **heurística** para *queries* solo formadas por operadores AND es **hacer *intersects* desde las listas más pequeñas hacia a las más grandes**. De esta manera reducimos el coste general de la operación.

Instruction	Comparisons	Result $\leq$
1. $L_{b \cup c} = \text{union}(L_b, L_c)$	4.000+5.000 = 9.000	9.000
2. $L_{res} = \text{intersect}(L_a, L_{b \cup c})$	9.000 + 300 = 9.300	300
Total comparisons	9.000 + 9.300 = **18.300**	–

Como regla general intentaremos hacer **primero las intersecciones** y **después las uniones** para que estas ultimas tengan un coste menor.

3.4 Sublinear time intersection

3.4.1 Binary Search

Una alternativa es **Binary search**, que mejora el tiempo lineal de la búsqueda. Nos aprovechamos que las listas están la mayoría de veces ordenadas.

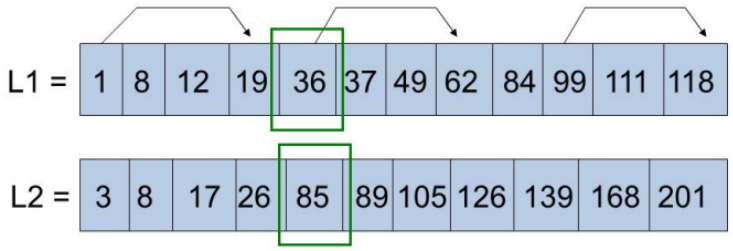
- **Tiempo**: longitud de la lista más corta L_1 × logaritmo de la lista más larga L_2.

$$\text{if } |L1| \ll |L2| \implies |L1| \cdot \log(|L2|) < |L1| + |L2|$$

3.4.2 Skip pointers

Para hacer búsquedas secuenciales con **coste sublinear**. Utilizamos **Skip pointers**, que son las flechas que se usan para avanzar más rápidamente para ahorrar comparaciones (para vectores ordenados).

La idea es que cuando avanzamos usamos estos punteros para mirar si una posición más avanzada continua siendo más pequeña que la que estamos comparando, de manera que nos podemos saltar diferentes elementos del vector en algunos casos.



3.5 Implementation of the Vector Model

La similitud del coseno se usa como la medida de similitud. Si el usuario pide una query, el modelo tiene que regresar todos los documentos relevantes de manera rápida.

Los documentos relevantes son típicamente los cuales tienen una medida de similitud por encima de un cierto umbral sim_{min}. O simplemente fijamos la longitud de la respuesta hasta cierto punto r (r puede llegar a ser el número de todos los documentos del corpus). Obviamente la respuesta debe estar ordenada en orden de mayor a menor similitud.

Algorithm 2 Inefficient solution

```
for each d in D do
    sim(d, q) = 0
    get vector representing d
    for each w in q do
        sim(d, q) += tf(d, w) · idf(w) · (1)
    end for
    normalize sim(d, q) by |d| · |q|
end for
sort results by similarity
```

La solución obvia es recorrer todos los documentos, computar su similitud en un vector y ordenarlo para regresar los documentos. Ineficiente porque recorre todo el corpus, proporcional al tamaño del conjunto de documentos $|D|$.

En su lugar, sabiendo que:

- La mayoría de los documentos incluyen una proporción pequeña de los términos. Lo que significa que hay muchos ceros en las fila correspondiente al documento d_i en la matriz de pesos. Lo que implica que para hacer el producto escalar solo un par de columnas serán relevantes.
- Los queries son de tamaños relativamente pequeños. Contienen un número reducido de términos.
- Además de que el conjunto de respuesta será pequeño en proporción a la cantidad de documentos $|D|$.
- Además se debe tener en cuenta la estructura de documento invertido.

Se invertirán los loops del código anterior y se hará un loop (sobre las palabras de la query) que se ejecutará pocas veces porque tenemos un número pequeño de palabras y ahí se hará un segundo loop que recorra los documentos que se encuentran en la **posting list** de la palabra.

Algorithm 3 Inverted file implementation

```
for each w in q do
    L = posting list for w, from inverted file
    for each d in L do
        if d seen for first time then
            sim(d, q) = 0
        end if
        sim(d, q) += tf(d, w) · idf(w) · (1)
    end for
end for
for each d seen do
    normalize sim(d, q) by |d| · |q|
end for
sort results by similarity
```

Lo que equivaldría a recorrer la matriz de pesos por columnas/términos en lugar de filas/documentos.

3.6 Index compression

El mayor tiempo de respuesta se produce en la obtención de las **posting list** del disco a la RAM. Se necesita reducir la cantidad de bits que se transfieren (comprimirlos).

- Podemos guardar los **docid** ordenados en orden ascendente.
- Comprimir la frecuencia de los **docid**.
- No debemos usar los mismos bits para codificar diferentes números, números más grandes (e.g 1000) necesitarán más bits que más pequeños (e.g 1).

3.6.1 Frecuency compression

Se pueden comprimir mediante una representación **unary encoding**, pues se asume que las frecuencias adoptan valores relativamente pequeños (cosa que no es cierta para los **docid**). El **unary encoding** usa 1's (podemos imaginarlos como "palitos"). Números

más grandes utilizarán más 1's que números más pequeños (el 1 usa 1 palito, el 100 usa 100 palitos). Y como **la mayoría de frecuencias (por la ley de Zipf) son 1 y 2**, estas ocuparan 1.5, 2, etc. bits, que es mucho mejor que reservar 1 byte para cada frecuencia.

$$unary(15) = 111111111111111,\ |unary(x)| = x$$

El problema con esto es que queremos codificar listas de frecuencias, si las pusiéramos concatenadas no sabríamos donde partir y no podríamos decodificar. Para partir los números usaremos el 0 como separador, en la última posición en vez de usar un 1 usamos un 0.

$$[3, 2, 1, 5] \rightarrow 110\ 10\ 0\ 11110$$

Unary encoding sirve muy bien porque permite hacer estimaciones del espacio que tenemos que ocupar. Podemos decodificar de manera única.

3.6.2 Docid compression

Unary encoding no serviría en este caso, pues los docids pueden tener valores muy grandes

Gap compression: Comprimir la diferencia de los docid con respecto al previo, pues si están ordenados, los valores serán positivos.

$$[(id_1, f_1), (id_2, f_2), \ldots, (id_n, f_n)]$$
$$id_1 > \ldots > id_n$$
$$[(id_1, f_1), (id_2 - id_1, f_2), \ldots, (id_n - id_{n-1}, f_n)]$$

No hay garantía de que sean suficientemente pequeños como para la codificación unitaria. Los *gaps* entre documentos no están sesgados hacia 1 por lo que deberemos utilizar un código de longitud variable.

Elias-Gamma code: La codificación consta de unir dos partes. La segunda parte será el número en binario $w = binary(x)$ y la primera parte será la longitud que esta segunda parte $y = |w|$. Como todos los números en binaria empiezan por 1 (no tiene sentido tener 0's a la izquierda) nos podemos ahorrar 1 bit y solo utilizar $y-1$ para marcar la longitud.

$$EG(x) = \underbrace{00\ldots00}_{y-1} w$$

$$EG(20) = [len(bin(20))|bin(20)] = [00001|10100] = [000010100]$$

En este caso, al haber un uno en los dos extremos de la división, este se unifica en uno solo para ahorrar memoria. Es una codificación eficiente porque la longitud del código es $2\log_2(w) - 1$ (-1 por la unión del 1 de división). Además que la descodificación es única pues se sabe al principio de cada número su longitud y el inicio del siguiente número codificado.

Byte-wise (8) or nibble-wise (4): Estos esquemas hacen uso del **continuation bit**, que es guardar un bit (del inicio o final de los 8 o 4 bloques de bits) para identificar cuando empieza o acaba la codificación. Normalmente el 0 representa el final del byte y el 1 la continuación.

- Mejor uso de la CPU al leer bytes y no bits.
- No se crea tanto desperdicio (un bit por byte). En EG el gasto sería la longitud del número codificado.

3.7 Getting fast the top r results

En tiempo $\mathcal{O}(R \log R)$ con R siendo el número de documentos que cumplen $sim(d, q) > sim_{min}$ para un usuario que normalmente solo quiere los r-primeros ($r << R$). Si tenemos decenas de miles de documentos perderemos tiempo ordenando muchos documentos que el usuario finalmente ni verá.

Tendremos una estructura que nos guardará los mejores r documentos que tenemos hasta ahora. Cuando procesamos los siguientes documentos, si resulta que la similitud es más grande que alguno de los que tenemos lo remplazamos. Esto tiene coste lineal.

Algorithm 4 MinHeap Operations

```
Put [d_1, ..., d_r] in a minheap
for i = r + 1 to R do
    min_val = sim(d, q) for d = top of the heap
    if sim(d_i, q) > min_val then
        Replace the smallest element in the heap with d_i
        Reorganize the heap
    end if
end for
```

- $L = [d_1, \ldots, d_R]$, lista de documentos que superan el requisito del usuario (que tienen similitud mayor que un threshold)
- **MinHeap**: Una especie de árbol binario que nos permite identificar rápidamente el elemento mínimo. Es una estructura de datos que almacena un conjunto de elementos de manera pseudoordenada para que la consulta del elemento mínimo sea muy rápida, y la inserción o reemplazo de elementos también se realice con un costo rápido.
- **min_val** equivale a mirar la raíz del MinHeap.
- Después de cada iteración, el *heap* contiene los mejores r documentos de entre los i primeros.
- Si las similitudes en L están ordenadas de manera aleatoria (si L no esta ordenada en orden ascendente), el tiempo de ejecución esperado del algoritmo es de $O(R + r \cdot \ln(r) \cdot \ln(\frac{R}{r})) \approx O(R)$ if $r << R$.

3.8 How to build the Index (offline)

Si todo nos cupiera en memoria RAM, podríamos hacer la siguiente implementación. En RAM los accesos directos son muy rápidos.

Algorithm 5 RAM implementation

```
F = {}
for each doc in D do
    d = docid(doc)
    for each w in doc do
        if w not in F then
            F[w] = {}
        end if
        if d not in F[w] then
            F[w][d] = 0
        end if
        F[w][d] += 1
    end for
end for
```

Normalmente, estos índices no caben en memoria RAM y por lo tanto tenemos que interactuar con disco. Tenemos que tener mucho más cuidado porque las cosas son mucho mas lentas. La forma de actuar es la misma pero es importante que ahora procesemos los documentos en orden de *docid* ascendente.

Algorithm 6 Update Inverted File on Disk

```
Initialize F to empty on disk
for each docid in D in increasing order do
    for each word in D[docid] do
        Retrieve list L from F(word) on disk
        if (docid, c) is in L then
            Replace (docid, c) with (docid, c + 1)          ▷ In the disk list
        else
            Append (docid, 1) at the end of F(word) ▷ This keeps lists sorted by docid
        end if
    end for
end for
```

Como las palabras no están ordenadas en disco, cada vez que tomamos una palabra, y no lo hacemos de manera secuencial, implica un costo significativo. Para optimizar este proceso, no escribiremos constantemente en disco. En su lugar, guardaremos partes de las listas de índices de manera temporal en memoria y solo ocasionalmente realizaremos la escritura en disco de las partes más actualizadas (que hemos construido) de dichas listas de índices.

3.8.1 More efficient

1. **Inicializar el índice en disco como vacío.**
2. **Construir el índice en la RAM, utilizando hasta la memoria asignada** M.
3. **Cuando la RAM esté llena:**
   (a) Agregar cada lista en la RAM al final de la lista correspondiente en el disco.
   (b) Realizar escrituras secuenciales en disco (¡rápido!).
   (c) Limpiar el índice en la RAM.
   (d) Volver al paso 2 para procesar más documentos.

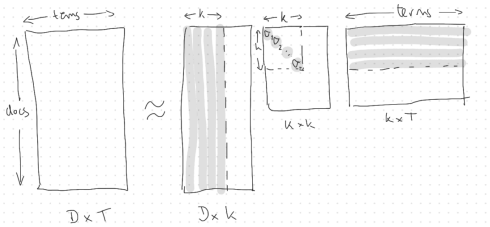
En vez de escribir cada vez en disco, como esto es muy lento, haremos todo lo que podamos en memoria RAM y una vez este llena haremos el traslado en disco de manera secuencial de cada posting list.

4 Topic Models

4.1 Latent Semantic Analysis (LSA)

In the context of Information Retrieval, this is also known as Latent Semantic Indexing (LSI).

LSA comienza con la representación completa de la matriz documento-término de un corpus (utilizando el esquema *tf-idf*, por ejemplo), que aquí llamaremos A. En este caso, los temas son incrustaciones lineales tanto de términos como de documentos, y estos se pueden encontrar a través de la descomposición de valores singulares (SVD). Gráficamente:



$$A = U \times S \times V^T$$

$$(D \times T) \approx (D \times k)(K \times k)(k \times T), \quad k << T$$

Aquí estamos utilizando SVD truncada para realizar una reducción de dimensionalidad a k temas, donde k es un hiperparámetro establecido externamente por el usuario. El teorema de SVD del álgebra lineal garantiza que $U_k S_k V_k^T$ es la mejor aproximación de rango k a nuestra matriz original documento-término A.

- U_k es la matriz *documento-topic*. k columnas ortogonales. Que tan fuerte esta relacionado el documento con el topic.
- V_k^T es la matriz *término-topic*. k filas ortogonales. Que tan fuerte esta relacionado el término con el topic.
- S_k es la matriz con los valores singulares, representa la importancia relativa de los topics.

Los topics son normalmente descubiertos por el SVD y pueden o no ser interpretables, pero por ejemplo se pueden intuir desde antes dependiendo del contexto de los documentos.

Si nos enfocamos en una celda específica $A_{i,t}$ de nuestra matriz A, que corresponde al peso del t-ésimo término para el i-ésimo documento en el corpus, podemos aproximarlo mediante un producto punto en el espacio de temas latentes:

4.2 Probabilistic LSA (pLSA)

4.2.1 Latent dirichlet Allocation (LDA)

Appendix A Session 1: Introduction. Preprocessing. Text Statistics Exercise List, Fall 2023

Basic comprehension questions. Check that you can answer them before proceeding.

1. Tell five Information Retrieval Systems you frequently use.
   The basic information retrieval tools include: bibliographies, catalogues, indexes, finding aids, registers, online databases, etc.
2. Tell the typical sequence of transformations we apply to a text while preprocessing and before adding to the index.
   Aplicamos parsing, tokenization, enriching and normalization.
3. Tell the difference between stemming and lemmatizing.
   Stemming quita los sufijos de las palabras y lemmatizing reduce las palabras a sus raíces lingüísticas.
4. Zipf's law tells the relation between X and Y. What are X and Y ?
   Relación entre el rango de los elementos (posición que ocupan ordenados por frecuencia) y su frecuencia (número de veces que salen en el texto).
5. Heaps' law tells the relation between X and Y. What are X and Y ?
   Relación entre la longitud de un texto y el número de palabras distintas que aparecen.

A.1 Exercise 1

Guess (without using any software) what a text preprocessor could give on this text if it performs stopword removal and stemming:

We found my lady with no light in the room but the reading-lamp. The shade was screwed down so as to over-shadow her face. Instead of looking up at us in her usual straightforward way, she sat close at the table, and kept her eyes fixed obstinately on an open book.
"Officer," she said, "it is important to the inquiry you are conducting to know beforehand if any person now in this house wishes to leave it?"

(William Wilkie Collins, The Mooonstone, Chapter 16)

Recordamos que *stopword removal* se trata de eliminar palabras de tipo preposiciones, artículos, verbos muy comunes, etc. Es decir, esas palabras que son comunes en todos los documentos y de manera muy recurrente. Por otro lado **Stemming** consiste en quitar los sufijos de las palabras. Dicho esto, el texto quedaría de la siguiente manera:

We found my lady with no light in the room but the reading-lamp. The shade was screwed down so as to over-shadow her face. Instead of looking up at us in her usual straightforward way , she sat close at the table, and kept her eyes fixed obstinately on an open book.
"Officer," she said, "it is important to the inquiry you are conducting to know beforehand if any person now in this house wishes to leave it?"

A.2 Exercise 2

Suppose that our document retrieval system lets us enter a query, which is a set of words, and returns the set of documents that contain all the words in the query.

Imagine that we configure the system in four different modes, and we ask four times the same query.

- Mode 1: We don't remove stopwords and we don't stem neither documents nor queries. Let A_1 be the set of returned documents.
- Mode 2: We don't remove stopwords, but we stem both documents and queries. Let A_2 be the set of returned documents.

- Mode 3: We remove stopwords, but don't stem. Let A_3 be the set of returned documents.
- Mode 4: We remove stopwords, and then we stem both documents and queries. Let A_4 be the set of returned documents.

What relations can you prove among A_1, A_2, A_3, and A_4 ? For example, is $A_1 = A_2$? Is A_2 a subset of A_4 ?, etc.

$$q = plays,\ d1 = children\ playing,\ d2 = the\ child\ plays$$

So ...

$$A_1 = \{d_2\},\ A_2 = \{d_1, d_2\} \rightarrow A_1 \subseteq A_2,\ A_3 \subseteq A_4$$

En el caso de que la query contenga solo stopwords, los modelos A_1 y A_3 no serán iguales (ni A_2 y A_4). Modelos A_2 y A_3 no son comparables.

A.3 Exercise 3

We have a document collection with a total of N word occurrences (N is large). We are told that it follows a Zipf's law of the form $frequence = c \cdot \text{rank}^{-\alpha}$.

1. What is c if $\alpha = 2$?
   Podemos utilizar el hecho que el número total de ocurrencias N debe de ser igual a la suma de todas las frecuencias para cada rango hasta un rango máximo que llamaremos R (número total de palabras distintas).

$$N = \sum_{r=1}^{R} f(r) = \sum_{r=1}^{R} \frac{c}{r^{\alpha}} = \sum_{r=1}^{R} \frac{c}{r^2} \implies c = \frac{N}{\sum_{r=1}^{R} \frac{1}{r^2}} = \frac{6N}{\pi^2}$$

   Donde utilizamos que $\sum_{r=1}^{R} \frac{1}{r^2} \approx \zeta(2) = \frac{\pi^2}{6}$ (función de Riemann).
2. And if $\alpha = 1$?
   Utilizando el apartado anterior podemos observar que

$$c = \frac{N}{\sum_{r=1}^{R} \frac{1}{r}} = \frac{N}{\gamma + \ln(R)}$$

   Donde utilizamos que $\sum_{r=1}^{R} \frac{1}{r} \approx \gamma + \ln(R)$ con $\gamma \approx 0.5772$ (constante de Euler).
3. Assume again $\alpha = 2$. What is the frequency of the most common term?

$$f_1 = c \cdot rank^{-2} = \frac{6N}{\pi^2} \cdot 1^{-2} = \frac{6N}{\pi^2} = 0.61 \cdot N$$

   La palabra más común es un 60% del texto, lo que es demasiado.
4. And what is the frequency of the 100th most frequent term?

$$f_{100} = c \cdot rank^{-2} = \frac{6N}{\pi^2} \cdot 100^{-2} = \frac{6N}{(100\pi)^2}$$

5. And (roughly) how many words have frequency 1?
   Escogemos un limite inferior y superior para los rangos de palabras con frec 1.

$$b = \max_i f(i) \geq 1 \iff \frac{c}{i^2} \geq 1 \iff i < \lfloor \sqrt{c} \rfloor$$

$$a = \min_i f(i) < 2 \iff \frac{c}{i^2} < 2 \iff i > \sqrt{\frac{c}{2}}$$

$$\#words = b - a = \sqrt{c} - \sqrt{\frac{c}{2}} = \sqrt{c} \cdot (1 - \sqrt{\frac{1}{2}}) = 0.23\sqrt{N}$$

A.4 Exercise 4

We have a document collection with a total of 10^6 term occurrences. Supposing that terms are distributed in the texts following a power law of the form

$$f_i \cong \frac{c}{(i+10)^2}$$

give estimates of

1. the number of occurrences of the most frequent term
   Tal y como hemos hecho en el ejercicio anterior lo primero que tenemos que hacer es encontrar el valor de la c.

   $$10^6 = \sum_{i=1}^{max\ rank} \frac{c}{(i+10)^2} \implies c = \frac{10^6}{\sum_{i=1}^{max\ rank} \frac{1}{(i+10)^2}} = \frac{10^6}{\sum_{j=11}^{max\ rank+10} \frac{1}{j^2}} \approx \frac{10^6}{0.095}$$

   Por lo tanto el número de ocurrencias del primer término es aproximadamente

   $$f_1 \approx \frac{c}{11^2} = \frac{10^6}{0.095 \cdot 11^2} \approx 86994$$

2. the number of occurrences of the 100-th most frequent term

   $$f_{100} \approx \frac{c}{110^2} = \frac{10^6}{0.095 \cdot 110^2} \approx 870$$

3. the number of words occurring more than 2 times

   $$3 = \frac{10^6}{0.095 \cdot (i+10)^2} \implies i^2 + 20i + 100 - \frac{10^6}{3 \cdot 0.095} \implies i = 1863$$

   Por lo tanto hay unas 1863 palabras con una frecuencia superior a 2 y que por lo tanto deberían aparecer más de dos veces.

Hint: $\sum_{i=11}^{\infty} \frac{1}{i^2} \cong 0.095$.

A.5 Exercise 5

We are given a random sample of 10,000 documents from a collection containing 1,000,000 documents. We count the different words in this sample, and we find 5,000. Supposing that the collection satisfies Heaps' law with exponent 0.5, give a reasoned estimate of the number of different words you expect to find in the whole collection.
Tenemos que

$$d = k \cdot N^{\beta}$$

Para calcular la k usamos los datos del enunciado

$$5000 = k \cdot 10000^{\frac{1}{2}} \implies k = 50$$

Entonces

$$d = k \cdot N^{\beta} = 50 \cdot 1000000^{\frac{1}{2}} = 50000$$

Esperamos encontrar 50000 palabras diferentes en toda la colección.

A.6 Exercise 6

Let us deduce Heaps' law from Zipf's law.

- Let a collection have N word occurrences, with the frequence f_i of the i-th most common word proportional to $i^{-\alpha}, \alpha > 1$.
- Figure out (from previous exercises) the proportionality constant.
- Estimate the rank i such that f_i is likely to be less than 1 .
- Explain why this should roughly be the number of distinct words we expect to see in the collection.
- Deduce that this number is $k \cdot N^{\beta}$. Tell the values of k and β as a function of α.

[Note: The given formulation of Zipf's law cannot, for obvious reasons, be taken too literally: If for some large i we have $c \cdot i^{-\alpha} = 0.03$, it makes no sense to say that the i th word appears 0.03 times in the collection. More abstractly, one could imagine texts generated by some random process which assigns probability $P(w)$ to the event that a random position in the text contains the word w. Then the word with rank 1 is the w with highest $P(w)$, etc. Zipf's law is a statement about the form of the probability distribution P. One can then compute rigorously the expected number of distinct words in a text of length N according to this probabilistic model. Let us just say that we this way we obtain the same β but a different k.]

[Note 2: It is also possible but a bit more involved to deduce a power law for word frequences (generalizing Zipf's law) from Heap's law]

Calculamos primero la constante de proporcionalidad c

$$N = \sum_{i=1}^{\text{max rank}} \frac{c}{i^{\alpha}} \implies c = \frac{N}{\sum_{i=1}^{\text{max rank}} \frac{1}{i^{\alpha}}}$$

Para considerar que una palabra es muy probable que su frecuencia f_i sea más pequeña que 1, consideraremos la i tq $f_i < 0.5$. $(i|f_i < 1)$ Para calcular esta i:

$$\frac{c}{i^{\alpha}} = 0.5 \implies i = (2c)^{\frac{1}{\alpha}}, \; i = c^{\frac{1}{\alpha}}$$

Este es más o menos el número de palabras distintas que encontraremos en la colección ya que el rango contiene palabras únicas y consideramos que las palabras con rango mayor a i no aparecen. A partir de aquí podemos deducir que:

$$(2c)^{\frac{1}{\alpha}} = \left(\frac{2}{\sum_{i=1}^{\text{max rank}}} \cdot N \right)^{\frac{1}{\alpha}} = k \cdot N^{\beta}$$

con $k = \frac{2}{\sum_{i=1}^{\text{max rank}}}$ y $\beta = \frac{1}{\alpha}$

con $k = \left(\frac{1}{\sum_{i=1}^{\text{max rank}}} \right)^{\frac{1}{\alpha}}$ y $\beta = \frac{1}{\alpha}$

Appendix B Session 2: Models

Basic comprehension questions. Check that you can answer them before proceeding.

1. True or false: The boolean model does not rank documents in the answer, while the vectorial model allows for ranking. True
2. Suppose you are given the frequency of every term in a given document. What other information do you need to compute its representation in tf-idf weights? Necesitamos también saber en cuantos documentos del corpus aparece (y el tamaño del corpus).
3. Hide the course slides. Write down the formula of the cosine measure of document similarity. Now look at the slides. Check your answer. Repeat until correct.

$$sim(d1, d2) = \frac{d1}{|d1|} \cdot \frac{d2}{|d2|}$$

4. Same for the tf-idf weight assignment scheme.

$$w_{d,i} = tf_{d,i} \cdot idf_i = \frac{f_{d,i}}{\max_j f_{d,j}} \cdot \log_2 \frac{D}{df_i}$$

5. Write down the definitions of recall, precision, coverage, and novelty. Explain them in words in a way that you think your classmates would understand.
   - **Recall**: de los documentos realmente relevantes, es el porcentaje que hay en mi respuesta.
   - **Precisión**: de todos los documentos que hay en mi respuesta, qué porcentaje son relevantes.
   - **Coverage**: capacidad de un sistema para extraer los documentos relevantes, cuántos de los documentos que sabemos que son relevantes hemos sacado en nuestra respuesta.
   - **Novelty**: capacidad de un sistema para extraer documentos relevantes desconocidos. De los documentos relevantes que hemos dado como respuesta, qué porcentaje eran desconocidos.
6. Explain to yourself how to compute a precision/recall graph.
   Tenemos un sistema de extracción de documentos, lo corremos y calculamos la precisión y el recall en función de número de documentos extraídos y de si estos son relevantes o no. Luego hacemos plot de estos puntos, ponemos la precisión en el eje y y el recall en el eje x.
7. True or false or criticize: To maximize user satisfaction, aim at a balance between recall and precision
   En general es cierto. En casos donde queramos extraer todos los documentos relevantes sin importarnos el tamaño de la respuesta priorizaríamos el recall. En el caso donde queremos extraer pocos documentos y que estos sean relevantes priorizaríamos la precisión. Pero en cualquier otro caso un balance entre las dos métricas nos asegurará una buena respuesta
8. Write down Rochio's formula for user relevance feedback.

$$q' = \alpha \cdot q + \beta \cdot \frac{1}{|R|} \sum_{d \in R} d - \gamma \cdot \frac{1}{|NR|} \sum_{d \in NR} d$$

B.1 Exercise 1

Consider the following documents:

D_1 : Shipment of gold damaged in a fire
D_2 : Delivery of silver arrived in a silver truck
D_3 : Shipment of gold arrived in a truck
and the following set of terms:

$$T = \{ \text{ fire, gold, silver, truck } \}.$$

Compute, using the boolean model, what documents satisfy the query and justify

(fire OR gold) AND (truck OR NOT silver)

Do the same with the query

(fire OR NOT silver) AND (NOT truck OR NOT fire).

Argue whether it is possible to rewrite these queries using only the operators AND, OR and BUTNOT in a logically equivalent way. This means that it must be equivalent for all possible document collections, not just this one.

Los documentos que satisfacen la primera query son D_1, D_3. D_1 tiene *fire* y no tiene *silver*. D_2 no tiene ni *fire* ni *gold*. D_3 tiene *gold* y *truck*.

Los que satisfacen la segunda son D_1, D_3, por razones parecidas.

Las queries si que pueden reescribir usando solo estos operadores. Usando la propiedad distributiva del operador OR sobre el operador AND para expandir la consulta original:

- (fire AND truck) OR (fire BUTNOT silver) OR (gold AND truck) OR (gold BUTNOT silver)
- (fire BUTNOT truck) OR (fire BUTNOT fire) OR (NOT silver BUTNOT truck) OR (NOT silver BUTNOT fire)

B.2 Exercise 2

Consider the following collection of five documents:

Doc1: we wish efficiency in the implementation for a particular application
Doc2: the classification methods are an application of Li's ideas
Doc3: the classification has not followed any implementation pattern
Doc4: we have to take care of the implementation time and implementation efficiency
Doc5: the efficiency is in terms of implementation methods and application methods

Assuming that every word with 6 or more letters is a term, and that terms are ordered in order of appearance,

1. Give the representation of each document in the boolean model.
   The terms are:
   T = {efficiency, implementation, particular, application, classification, methods, followed, pattern}
   - Doc1 = $(1, 1, 1, 1, 0, 0, 0, 0)$
   - Doc2 = $(0, 0, 0, 1, 1, 1, 0, 0)$
   - Doc3 = $(0, 1, 0, 0, 1, 0, 1, 1)$
   - Doc4 = $(1, 1, 0, 0, 0, 0, 0, 0)$
   - Doc5 = $(1, 1, 0, 1, 0, 1, 0, 0)$

2. Give the representation in the vector model using tf-idf weights of documents Doc1 and Doc5 Compute the similarity coefficient, using the cosine measure, among these two documents. (Answer to 2: I get 0.162.)

Tenemos que calcular la frecuencia de cada término en cada documento así como el número de veces que aparece en todo el corpus.

	efficiency	implementation	particular	application	classification	methods	followed	pattern
d1	1	1	1	1	0	0	0	0
d2	0	0	0	1	1	1	0	0
d3	0	1	0	0	1	0	1	1
d4	1	2	0	0	0	0	0	0
d5	1	1	0	1	0	2	0	0
df	3	4	1	3	2	3	1	1

	efficiency	implementation	particular	application	classification	methods	followed	pattern
d1	$\log_2 \frac{5}{3}$	$\log_2 \frac{5}{4}$	$\log_2 \frac{5}{1}$	$\log_2 \frac{5}{3}$	0	0	0	0
d5	$0.5 \cdot \log_2 \frac{5}{3}$	$0.5 \cdot \log_2 \frac{5}{4}$	0	$0.5 \cdot \log_2 \frac{5}{3}$	0	$\log_2 \frac{5}{3}$	0	0

$$sim(d1, d5) = \frac{d1}{|d1|} \cdot \frac{d5}{|d5|} = \frac{0.5949}{2.769 \cdot 1.48269} = 0.144$$

B.3 Exercise 3

We have indexed a collection of documents containing the terms of the following table; the second column indicates the percentage of documents in which each term appears.

Term	% docs
computer	10%
software	10%
bugs	5%
code	2%
developer	2%
programmers	2%

Given the query Q = "computer software programmers", compute the similarity between Q and the following documents, if we use tf-idf weights for the document, binary weights for the query, and the cosine measure. Determine their relative ranking:

The Q vector of weights would be: $q = (1, 1, 0, 0, 0, 0, 1)$.

- D1 = "programmers write computer software code"

$$freq = (1, 1, 0, 1, 0, 1)$$
$$v_{d1} = (\log 10, \log 10, 0, \log 50, 0, \log 50)$$
$$sim(q, v_{d1}) = 0.766$$

- D2 = "most software has bugs, but good software has less bugs than bad software"

$$freq = (0, 3, 2, 0, 0, 0)$$
$$v_{d2} = (0, \log 10, \frac{2}{3} \log 20, 0, 0, 0)$$
$$sim(q, v_{d1}) = 0.436$$

- D3 = "some bugs can be found only by executing the software, not by examining the source code"

$$freq = (0, 1, 1, 1, 0, 0)$$
$$v_{d3} = (0, \log 10, \log 20, \log 50, 0, 0)$$
$$sim(q, v_{d3}) = 0.244$$

(Answer: I get similarities 0.766, 0.436, 0.244.)

B.4 Exercise 4

Suppose that terms A, B, C, and D appear, respectively, in 10,000, 8,000, 5,000 , and 3,000 documents of a collection of 100,000.

1. Consider the boolean query (A and B) or (C and D). How large can the answer to this query be, in the worst case?

   En el peor de los casos devolverá los documentos que tengan tanto la A como la B, en este caso como máximo podrá haber 8000 documentos ya que la B solo aparece en estos. Además también nos puede devolver los que aparezca la C y la D, por la misma razón que antes en el peor de los casos en todos los documentos donde aparece la D también aparece la C y por lo tanto serian 3000 documentos. En el peor de los casos, en todos los documentos donde aparecen la A y B, no aparecerán la C y la D, y por lo tanto los conjuntos serán disjuntos.

$$8000 + 3000 = 11000$$

2. And for the query (A and B) or (A and D)? Think carefully.

   El término A solo aparece en 10000 documentos, por lo tanto como caso peor se devolverán 10000. Es el caso en el que donde aparece la B siempre aparece la A (8000 documentos), y donde aparece la D siempre A (3000). En 1000 de estos documentos aparecen tanto la A, B y D. Entiendo que 10000 es una cota inferior actualizada de la que se hubiese obtenido (11000) en el caso de seguir las cotas para las uniones/intersecciones.

3. Compute the similarity of the documents d1 = "A B B A C C" and d2 = "D A D B B C C" using tf-idf weighting and the cosine measure.

$$D = 100,000$$

$$w_{d_1} = (\log_2 10, \log_2 12.5, \log_2 20, 0)$$
$$w_{d_2} = (\frac{1}{2}\log_2 10, \log_2 12.5, \log_2 20, \log_2 \frac{100}{3})$$
$$sim(d1, d2) = 0.736$$

(Answers: 1) 11.000 2) 10.000 3) 0.736.)

B.5 Exercise 5

We have an indexed collection of one million documents that includes the following terms:

Term	# docs
computing	300,000
networks	200,000
computer	100,000
files	100,000
system	100,000
client	80,000
programs	80,000
transfer	50,000
agents	40,000
p2p	20,000
applications	10,000

1. Compute the similarity between the following documents D1 and D2 using tf-idf weights and the cosine measure:

D1 = "p2p programs help users sharing files, applications, other programs, etc. in computer networks"

D2 = "p2p networks contain programs, applications, and also files"

2. Assume we are using the cosine measure and tf-idf weights to compute document similarity. Give a document containing two different terms exactly that achieves maximum similarity with the following document

"p2p networks contain programs, applications, and also files"

Compute this similarity and justify that it is indeed maximum among documents with two terms.

(Answer to 1. 0.925.)

B.6 Exercise 6

Consider the following collection of four documents:

Doc1: Shared Computer Resources

Doc2: Computer Services

Doc3: Digital Shared Components

Doc4: Computer Resources Shared Components

Assuming each word is a term:

1. Write the boolean model representation of document Doc3.

   T = {Shared, Computer, Resources, Services, Digital, Components}

   Doc3 = $(1, 0, 0, 0, 1, 1)$

2. What documents are retrieved, with the boolean model, with the query "Computer BUTNOT Components" ? Doc1, Doc2

3. Compute the idf value of the terms "Computer" and "Components".

   El valor *idf* es la frecuencia inversa del documento, donde tenemos en cuenta el número de documentos donde sale cada palabra.

$$idf_{Computer} = \log_2 \frac{D}{df_i} = \log_2 \frac{4}{3} \approx 0.415$$

$$idf_{Components} = \log_2 \frac{D}{df_i} = \log_2 \frac{4}{2} = 1$$

4. Compute the vector model representation of Doc4 using tf-idf weights.

$$f = (1, 1, 1, 0, 0, 1)$$

$$w_{d_4} = (\log_2 \frac{4}{3}, \log_2 \frac{4}{3}, 1, 0, 0, 1)$$

5. Compute the similarity between the query "Computer Components" (with binary weights) and Doc4 (with tf-idf weights), with the cosine similarity measure.

$$q = (0, 1, 0, 0, 0, 1)$$

$$sim(q, d_4) = 0.6534$$

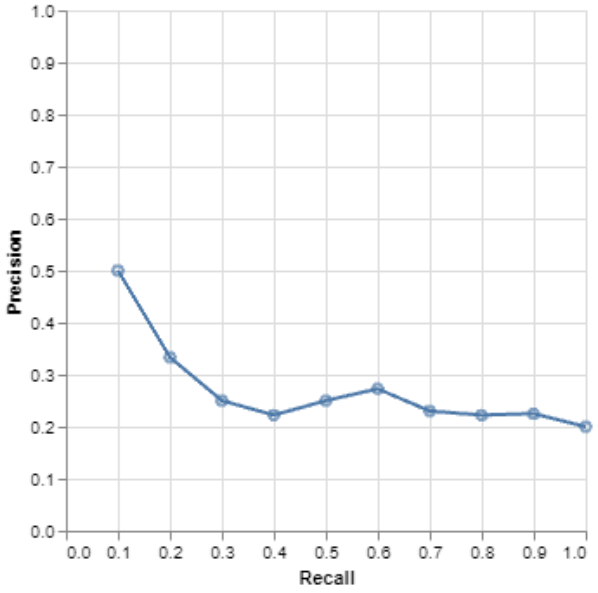
(Answer to 4: I get 0.6534)

B.7 Exercise 7

A user tells us that, after asking a query to our search system, she found 10 relevant documents in positions $2, 6, 12, 18, 20, 22, 30, 36, 40$, and 50 . Assuming there are no more relevant documents in the collection, draw a precision-recall graph of the answer at 10 recall levels. Make sure you give the table of numbers that you used to plot the graph.

Cada vez que encontramos un documento nuevo el recall sube 0.1 puntos, esto es porque tenemos 10 documentos relevantes en total y cada documento relevante representa el 10%. Calcularemos primero la precisión del sistema cada vez que se encuentra un documento relevante. Para calcularlo es depende la posición en que lo encontremos y el cuántos documentos relevantes hayamos encontrado ya.

- $P(d1) = \frac{1}{2} = 0.5$
- $P(d2) = \frac{2}{6} = \frac{1}{3}$
- $P(d3) = \frac{3}{12} = 0.25$
- $P(d4) = \frac{4}{18} = \frac{2}{9} \approx 0.22$
- $P(d5) = \frac{5}{20} = 0.25$
- $P(d6) = \frac{6}{22} = \frac{3}{11}$
- $P(d7) = \frac{7}{30} \approx 0.23$
- $P(d8) = \frac{8}{36} = \frac{2}{9} \approx 0.22$
- $P(d9) = \frac{9}{40} \approx 0.225$
- $P(d10) = \frac{10}{50} = 0.2$



B.8 Exercise 8

We have a document collection with 100 documents, identified by numbers 1... 100. Suppose that the relevant ones for a given query are those numbered 1 ...20.

Two information retrieval systems give as a result to the query the following answers:

$$\begin{aligned}
\text{S1} =&\{1, 2, 21, 22, 3, 23, 25, 4, 28, 5, 29, 30, 6, 7, 31, 32, 33, 40, 41, 42, 8, 43, 44,\\
&9, 45, 10, 50, 51, 11, 52, 53, 54, 12, 60, 62, 13, 63, 64, 14, 15, 16, 70, 78, 80, 17,\\
&81, 82, 83, 85, 18, 90, 19, 91, 92, 20, 93, 94, 95, 96, 98\},\\
\text{S2} =&\{25, 26, 1, 27, 28, 2, 3, 29, 30, 4, 35, 36, 5, 37, 6, 7, 8, 38, 9, 40, 10, 42, 11, 45, 46,\\
&12, 48, 50, 51, 13, 60, 61, 64, 14, 70, 72, 15, 78, 79, 90\}.
\end{aligned}$$

For this query and each of the two systems:

a) Compute the recall, precision, and F-measure (with $\alpha = 1/2, \alpha = 1/4$, and $\alpha = 3/4$).

Para el sistema 1 tenemos Recall máximo (= 1) ya que todos los documentos relevantes se encuentran en la respuesta. En cuanto a la precisión, la respuesta consta de un total de 60 documentos, por lo que Precision = $20/60 = 1/3$. Para la F-measure tenemos

$$\alpha = 1/2,\ F = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = \frac{2}{1+3} = 0.5$$

$$\alpha = 1/4,\ F = \frac{1}{\frac{1/4}{\text{recall}} + \frac{3/4}{\text{precision}}} = \frac{1}{1/4 + 9/4} = 0.4$$

$$\alpha = 3/4,\ F = \frac{1}{\frac{3/4}{\text{recall}} + \frac{1/4}{\text{precision}}} = \frac{1}{3/4 + 3/4} = 2/3 = 0.66$$

En en segundo sistema solo se devuelven 15 de los 20 documentos relevantes, por lo que el Recall = $15/20 = 3/4$. La Precision = $15/39$, ya que el tamño de la respuesta es de 39 documentos.

$$\alpha = 1/2,\ F = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = \frac{2}{4/3 + 39/15} \approx 0.51$$

$$\alpha = 1/4,\ F = \frac{1}{\frac{1/4}{\text{recall}} + \frac{3/4}{\text{precision}}} = \frac{1}{1/3 + \frac{3 \cdot 39}{4 \cdot 15}} \approx 0.438$$

$$\alpha = 3/4,\ F = \frac{1}{\frac{3/4}{\text{recall}} + \frac{1/4}{\text{precision}}} = \frac{1}{1 + 39/45} \approx 0.5357$$

b) Compute the novelty and coverage measures, assuming that the user already knew the documents with odd index and did not knew about those with even index.

- **Coverage**
$$\frac{|relevant\ \&\ known\ \&\ retrieved|}{|relevant\ \&\ known|}$$
- **Novelty**
$$\frac{|relevant\ \&\ retrieved\ \&\ unknown|}{|relevant\ \&\ retrieved|}$$
- $|relevant\ \&\ known| = 10$, que son los documentos con índice impar, que suponemos conocidos. Esto sirve para ambos sistemas.

Del Sistema 1:

- $|relevant \ \& \ known \ \& \ retrieved| = 10$, son los documentos relevantes e impares de la respuesta.
- $|relevant \ \& \ retrieved \ \& \ unknown| = 10$, son los documentos relevantes que no conocíamos (pares) presentes en nuestra respuesta.
- $|relevant \ \& \ retrieved| = 20$, son los documentos relevantes que aparecen en nuestra respuesta, en este caso son todos.

$$Coverage = \frac{10}{10} = 1$$
$$Novelty = \frac{10}{20} = 0.5$$

Del Sistema 2:

- $|relevant \ \& \ known \ \& \ retrieved| = 8$, son los documentos relevantes e impares de la respuesta.
- $|relevant \ \& \ retrieved \ \& \ unknown| = 7$, son los documentos relevantes que no conocíamos (pares) presentes en nuestra respuesta.
- $|relevant \ \& \ retrieved| = 15$, son los documentos relevantes que aparecen en nuestra respuesta, en este caso son todos.

$$Coverage = \frac{8}{10} = 0.8$$
$$Novelty = \frac{7}{15} = 0.46$$

Appendix C Session 3: Implementation Exercise List, Fall 2023

Check that you can answer them before proceeding. Not for credit.

1. Explain why the inverted index is adequate for retrieving documents matching a query.
2. Invent a small document collection (5-6 documents with 5-6 words each) and draw the inverted index it would produce. Do it for the variant that is required for the pure tf-idf representation, then for the variant required to keep positional information.
3. True or false: Query optimization is the process by which one finds the best queries for a given retrieval task.
4. What is the main reason for compressing the index?
5. (Looking at the course slides if you want) Write the self-delimiting unary encoding and Gamma codes of the numbers 17 and 23. Write the Gamma code of 787 .

C.1 Exercise 1

Consider the following fragment of an inverted index file containing, for each term, the docid's of the documents that contain it and in which positions. For example, document 2 contains "angels" in positions 36,174 , and so on.

angels

$$2 : 36, 174, 252, 651$$
$$4 : 12, 22, 102, 432$$
$$7 : 7, 17$$

fools

$$2 : 1, 17, 74, 222$$
$$4 : 8, 78, 108, 458$$
$$7 : 3, 13, 23, 193$$

fear
2: $87, 704, 722, 901$
4: $13, 43, 113, 433$
7: $18, 328, 528$
in
2: $7, 37, 76, 444, 851$
4: 11, 20, 110, 470, 500
7: $15, 25, 195$
rush
2: $2, 66, 75, 321, 702$
4: 9, 69, 149, 429, 569
7: 4, 194, 404
to
2: 47, 86, 234, 999
4: 14, 24, 774, 944
7: 19, 319, 599, 709
tread
2: $57, 94, 333$
4: $15, 35, 155$
7: 20,320

where
2: $67, 124, 393, 1001$
4: $11, 41, 101, 421, 431$
7: $16, 36, 736$
Which documents satisfy the query "fools rush in" AND "angels fear to tread'?

C.2 Exercise 2

Refer to the collection in Exercise 5 of Topic 2, about computers and networks.

1. Which processing order would you recommend for the query computer AND client AND applications?
2. Recommend a processing plan for the boolean query (computing AND programs) OR (p2p AND applications)
   OR (computing AND networks AND applications)

C.3 Exercise 3

In a set of 300,000 documents we have the following term frequencies for some of the existing terms:

Charles	Dickens	Leon	Tolstoi	Anton	Chejov
24.000	1.000	10.000	4.000	13.000	7.000

Propose an evaluation plan for the following query:
(Charles AND Dickens) AND ((Leon AND Tolstoi) OR (Anton AND Chejov))
in order to minimize the list processing time. Justify your answer.

C.4 Exercise 4

1. Given the posting list

$$[10, 1, 15, 3, 22, 2, 23, 4, 34, 1, 44, 1, 50, 2, 58, 8, 90, 1, 101, 1, 112, 2]$$

(which means that a term appears once in document 10, three times in document 15 , two times in document 22 , etc.), give the bit string that results of compressing it using self-delimiting unary for the frequencies and gap compression + Elias' Gamma code for the docid's. The first docid in the list (which does not have a gap) is encoded in Gamma code directly.

2. Perform the inverse process on the bit string

$$000010101100010001010001000100011011001000110$$

(Note: since there are dual usages of 0 and 1 in these codes, say that the unary self-delimiting code of 3 is 110 and the Elias Gamma code of 4 is 00100).

C.5 Exercise 5

We create an inverted index from a collection of 1 million documents. We only place 6 terms in the index, with the following frequencies:

Term	Documents
A	10,000
B	20,000
C	40,000
D	80,000
E	120,000
F	150,000

Compute the approximate number of comparison between documents needed in order to process the following queries without optimizations 1) in the worst case, and 2) assuming mutual independence between occurrences of terms within documents.

1. ((A and B) and C) and D) and E (Answer I get: 300.000)
2. A and (B and (C and (D and E))) (Answer I get: 410.000)
3. ((A and B) or (C and D)) or (E and F)
4. (A and B) or ((C and D) or (E and F))
5. (A and E) or (B and E)
6. (A and B) or E

C.6 Exercise 6

We have indexed a set of 10^7 documents. Knowing that terms A, B, C, D appear, respectively, in 2 million, 1 million, 800,000, and 20,000 documents, propose an efficient evaluation plan for the boolean query

$$(A \text{ and } B \text{ and } C \text{ }) \text{ or } (\text{ } A \text{ and } B \text{ and } D)$$
$$\text{or } (\text{ } A \text{ and } C \text{ and } D) \text{ or } (\text{ } B \text{ and } C \text{ and } D).$$

Express your plan as a sequence of list intersection and list union instructions. Justify your answer. You do not need to compute the expected cost of your plan.

C.7 Exercise 7

1. You want to compress a sequence of positive natural numbers. Say when you would prefer unary self-delimiting code over Elias' Gamma code, or vice-versa. Give a criterion as precise as possible. 2. If we use Elias' Gamma code in gap compression, what is the largest gap that can be encoded using 1 byte?
2. Give the variable-length encoding of the following posting list:

$$((1,3)(7,1)(19,5)(35,4)(52,2))$$

C.8 Exercise 8

We have a collection of 10^8 documents. The average document length is 10,000 characters, and the average word length in the documents is 7 characters.

1. Suppose that the collection satisfies Heaps' law in the form $10N^{0.5}$. Estimate the number of different words that you expect to find in the collection.
2. We create an inverted index containing docid's only. Estimate the average length of the posting lists. Hint: Estimate first the number of distinct words per document then the number of total entries in the posting lists, then this.
3. Estimate the average gap in posting lists.
4. We use gap compression + Elias Gamma code to encode the posting lists. Estimate the number of bits that the index will use.

(Short answers - look only at the order of magnitude 1) $\simeq$ 3.5 million words 2) $\simeq 10^4$ entries per list on average 3) also $\simeq 10^4$ 4) about 1 Terabit = 125 Gbytes.)

If for some of these items you need the result from a previous item that you could not solve, make a reasonable guess and specify clearly "Suppose that the result of item $x-1$ is...".

C.9 Exercise 9

Consider a collection of D documents with an average of L different terms per document, and a total of T different terms among all documents.

1. Suppose that we do not need to keep intradocument frequencies, only whether each terms appear or not in each document, and that we do not use any compression mechanism. How much memory is needed to keep the term-document incidences in a full $D \times T$ matrix form? And as posting lists?

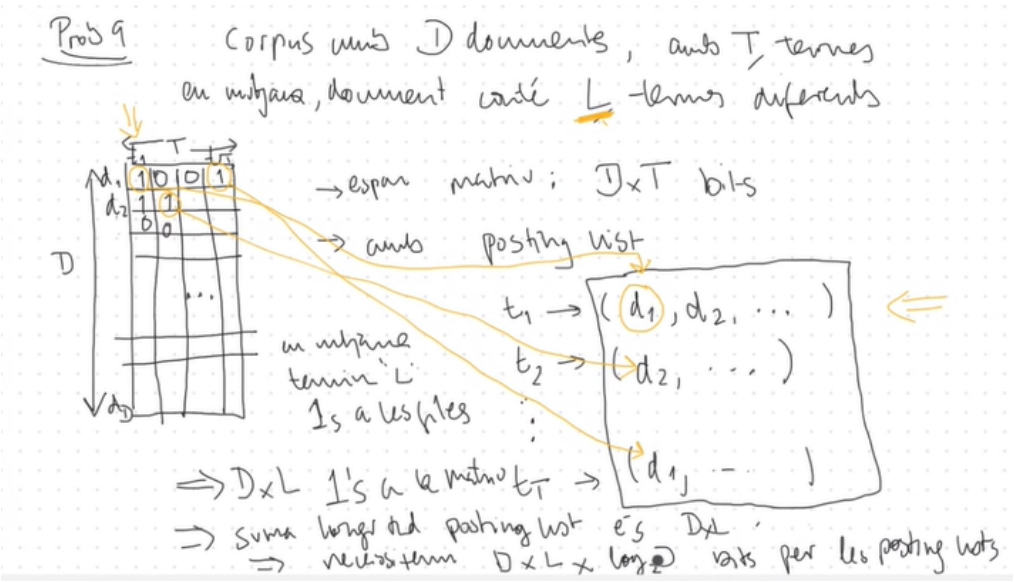
   Colección de D documentos (D grande e.g 100 miliones). De media cada document tiene L palabras diferentes $L = \frac{1}{D}\sum_i \sum_j M_{ij}$, algunos contendrán más palabras diferentes que otros. El nombre total de términos dentro de todo el corpus es T, seria la medida del vocabulario que tiene nuestro corpus.

   1. Como nos dice que no hace falta guardar la frecuencia de las palabras en cada documento solo nos interesa la información booleana, es decir, si ese termino aparece o no. Un ejemplo seria el siguiente.

	t_1	t_2	t_3	...	t_T
d_1	1	0	1	...	0
d_2	0	1	0	...	1
d_3	1	0	0	...	1
d_4	0	1	1	...	0
⋮	⋮	⋮	⋮	⋱	⋮
d_D	1	0	1	...	1

   La memoria que se necesita para guardar la matriz es de $D \times T$ bits. Aunque la matriz solo tiene $D \times L$ 1's.

   El índice invertido para cada término tenemos la lista de documentos que lo contiene. En este caso no hacemos ningún tipo de compresión. La memoria que se necesitará sera básicamente el espacio que ocupará las listas de identificación de documentos. De media el número de 1's en cada fila del documento será L. Por lo la suma de la longitudes de las **posting lists** es de $D \times L$. Necesitaremos $D \times L$ identificadores, si utilizamos un esquema binario ya que no usamos compresión cada identificador necesita $\log_2 D$. Por lo tanto total en bits que necesitamos es de $D \times L \times \log_2 D$



2. 2. Estimate the size of the index (as a function of D, L, T) if we compress it with gap compression and Elias' Gamma code.

   Ahora nos dicen que en vez de guardar las listas de esta manera sin comprimir, lo que haremos es hacer uso de la compresión Gap y la Elias Gamma. Nos interesaría que los documentos estuvieran seguidos para que el gap sea mínimo, pero no se puede saber. Para hacerlo, asumiremos que los 1's están distribuidos de manera equidistante. Si cogiéramos una columna cualquiera de la matriz (una palabra), asumiremos que los 1's están a la misma distancia, es decir, que los documentos están equidistantes. El peor caso para nuestro algoritmo sería que los documentos estuvieran muy alejados, pero estudiaremos el caso medio.

   En promedio, tenemos L 1's por fila, pero por columna no lo sabemos. Para estimarlo, dado que en total hay $D \times L$, si dividimos por el número de columnas, tendremos que la longitud media de las posting lists será de $\frac{D \times L}{T}$.

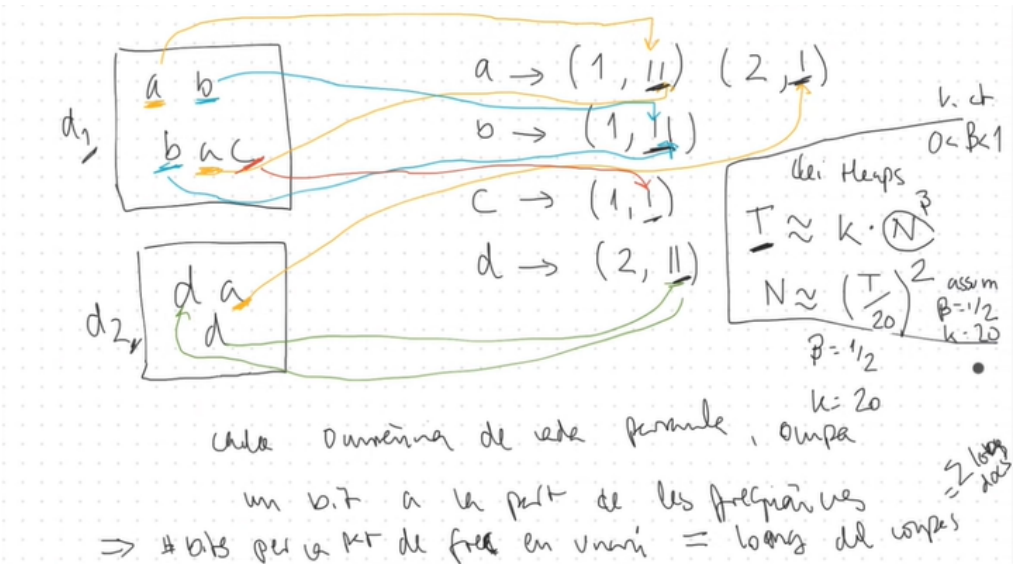
   Como tenemos un total de D documentos, hacemos grupos de $\frac{D \times L}{T}$ y, por lo tanto, el tamaño de los gaps será de $\frac{T}{L}$. Dado que para codificar x con Elias Gamma se usan $2\log_2(x)$ bits, necesitamos $D \times L \times 2\log_2\left(\frac{T}{L}\right)$ bits en este caso.

3. Imagine now that we do want to keep the intradocument frequencies. Estimate the index size if we compress docid's as in the previous question and the frequencies using self-delimiting unary.

   Para codificar un dado número x se necesitan x bits. Hay que darse cuenta que cada término en el corpus hace que añadamos un "palito" en las frecuencias correspondientes. Cada ocurrencia de cada palabra ocupa 1 bit en la parte de las frecuencias.

   Podemos estimar el número de bits que ocupa la parte de frecuencias calculando la longitud del corpus (cantidad de palabras) contando repeticiones. Como longitud del corpus es básicamente $N = \sum_i \text{len}(doc_i)$.

   Entonces el número de bits necesarios sería: $L \times D \times 2\log_2\left(\frac{T}{L}\right) + N$



No tenemos la longitud total del corpus, ya que solo nos dicen el número de palabras únicas. Pero podemos usar la ley de *Heaps* que nos relaciona el número de palabras diferentes con el número de ocurrencias totales de las palabras.

$$T \approx k \cdot N^{\beta}$$

donde $0 < \beta < 1$ y k es una constante. En este caso tenemos T y queremos saber N. Asumiremos por toda la puta cara (constantes que se suelen encontrar en textos reales) que $\beta = \frac{1}{2}$ y que $k = 20$.

$$N \approx (\frac{T}{k})^{\frac{1}{\beta}} = \frac{T^2}{400}$$

Finalmente, tenemos que el espacio para guardar las **posting lists** haciendo servir `EG + GAP + frecuencias unitarias` equivale a

$$D \times L \times 2\log_2(\frac{T}{L}) + \frac{T^2}{400}$$

If you need to make assumptions or use reasonable approximations, state them clearly. Partial answers: 1) $T \times D$ bits and $\log D \times D \times L$ bits 2) $D \times L \times 2\log_2 \frac{T}{L}$ bits 3) Assuming Heaps' law e.g. $20\sqrt{N}$, then $T^2/400$ plus the result in 2).