

Cerca i Anàlisi de la Informació

Carlos Arbonés and Juan P. Zaldivar

GCED, UPC.

Apunts.

Contents

1	Introduction	4
1.1	Information Retrieval	4
1.1.1	Information Retrieval vs Database Queries	4
1.1.2	User Expectations	4
1.1.3	Information Retrieval Models	4
1.1.4	The Information Retrieval Process	4
1.2	Preprocessing	5
1.2.1	Tokenization	5
1.2.2	Enriching	6
1.2.3	Lemmatizing and Stemming	6
1.3	Text Laws	6
1.3.1	Zipf's Law	6
1.3.2	Heaps' Law	7
2	Two Information Retrieval Models	8
2.1	Boolean Model of Information Retrieval	8
2.1.1	Queries	8
2.1.2	Phrase Queries	8
2.2	Vector Space Model of Information Retrieval	9
2.2.1	Weights assignation	9
2.2.2	Comparison of documents - Cosine Similarity	10
2.2.3	Query Answering	10
2.3	Evaluation	10
2.3.1	Recall and Precision	10
2.3.2	How many documents to show?	11
2.3.3	Other measures of effectiveness	12
2.3.4	Relevance Feedback	13
3	Implementation	15
3.1	Central Data Structure	15
3.1.1	Postings	15
3.2	Implementation of the Boolean Model	15
3.3	Query Optimization	16
3.4	Sublinear time intersection	17
3.4.1	Binary Search	17
3.4.2	Skip pointers	17
3.5	Implementation of the Vector Model	17
3.6	Index compression	18
3.6.1	Frecuency compression	18
3.6.2	Docid compression	19
3.7	Getting fast the top r results	20
3.8	How to build the Index (offline)	21
3.8.1	More efficient	22
6	Network Analysis	23
6.1	Examples of networks	23
6.1.1	Social Networks	23
6.1.2	Information Networks	23
6.1.3	Technological networks	23
6.1.4	Biological networks	23

6.1.5	Representing networks	23
6.2	Small World phenomenon I	24
6.2.1	The (basic) random graph model	24
6.3	Small World phenomenon II	25
6.3.1	Global clustering coefficient	25
6.3.2	Local clustering coefficient	25
6.3.3	Watts-Strogatz Model	26
6.4	Degree Distribution	27
6.5	Scale-free/Scale-invariant Networks	28
6.5.1	Preferential Attachment Model	28
6.6	Centrality	29
6.6.1	Degree centrality	29
6.6.2	Closeness centrality	30
6.6.3	Betweenness centrality	30
6.7	Communities	31
6.7.1	Community Quality Criteria	32
6.7.2	Hierarchical clustering	33
6.7.3	Definition of Modularity	34
6.8	Spreading in networks	36
6.8.1	Other models of infection	37
6.8.2	Two algorithmic problems - combinatorial optimization	38
7	Hashing	39
7.1	Hashing functions	39
7.2	Local Sensitive Hashing (LSH)	39
7.2.1	Locality sensitive hashing functions	39
7.2.2	Cómo utilizar LSH para encontrar el vecino más cercano	40
7.2.3	Hashing Family functions	40
7.2.4	Similarity search	43
7.3	Consistent Hashing	43
A	Session 1: Introduction. Preprocessing. Text Statistics Exercise List, Fall 2023	45
A.1	Exercise 1	45
A.2	Exercise 2	45
A.3	Exercise 3	46
A.4	Exercise 4	47
A.5	Exercise 5	47
A.6	Exercise 6	48
B	Session 2: Models	49
B.1	Exercise 1	50
B.2	Exercise 2	50
B.3	Exercise 3	51
B.4	Exercise 4	52
B.5	Exercise 5	52
B.6	Exercise 6	53
B.7	Exercise 7	54
B.8	Exercise 8	55
C	Session 3: Implementation Exercise List, Fall 2023	57
C.1	Exercise 9	57

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 almacenamos. 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.

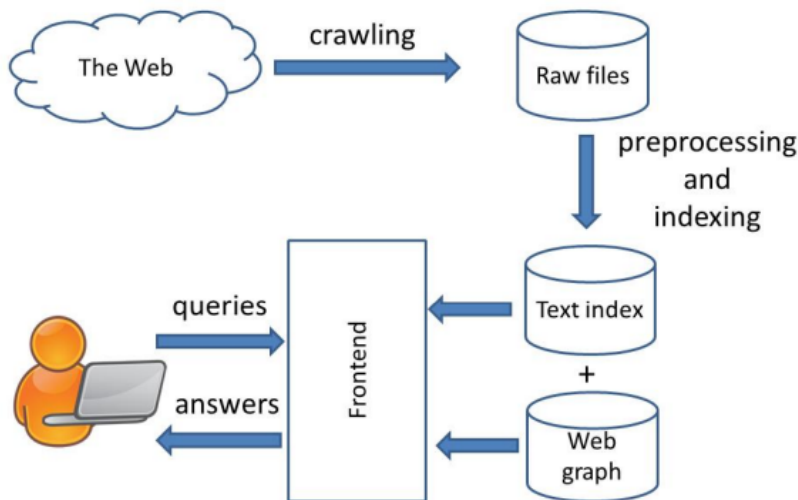


Fig. 1.1: The Information Retrieval Process

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 puntuación 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*), categorías (*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 lingüísticas como *be*, *am*, *are*, *is* → *be*, *gave* → *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 frecuente 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.

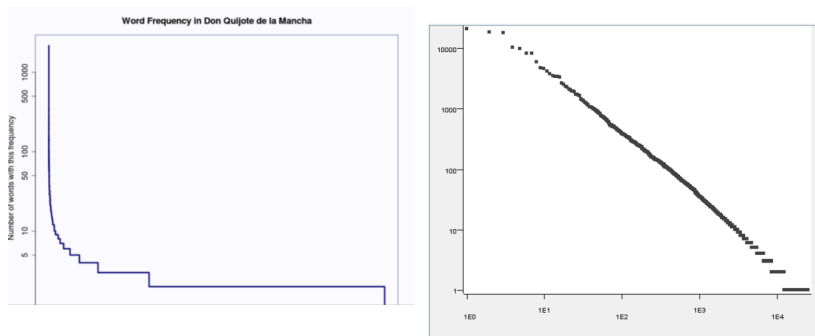


Fig. 1.2: A la izquierda la frecuencia de las palabras en el libro *Don Quijote*, a la derecha el mismo plot con log-log scale

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 es que cada vez se observan menos palabras nuevas.

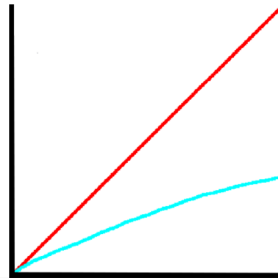


Fig. 1.3: La **línea roja** indica el número de palabras leídas y la **línea azul** el número de palabras distintas. Al principio la azul crece muy rápido y va disminuyendo la pendiente.

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 línea 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 varía 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 matrices 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 hicieramos 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 $\tau = \{t_1, \dots, t_T\}$ un documento es solo un **subconjunto** de τ . Cada documento se puede visualizar como un **vector de bits** de longitud T , $d = (d_1, \dots, d_T)$ 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 queries**:

- OR, AND
- t_1 BUTNOT t_2

Solo se permiten este tipo de queries y no por ejemplo NOT porque en la mayoría 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 sino 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 *stop 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**, $d = (w_1, \dots, w_T)$ 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 documentos **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 $d = [w_{d,1}, \dots, w_{d,i}, \dots, w_{d,T}]$ 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 algún 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 **similitud 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 similar**, que apuntan al mismo sitio y que están utilizando el mismo vocabulario.

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

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 similitud se usa en el momento de contestar una pregunta. Cogemos 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 similitud** 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 similitud** 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.

		<i>Answered</i>	
		relevant	not relevant
<i>Reality</i>	relevant	tp	fn
	not relevant	fp	tn

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

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.

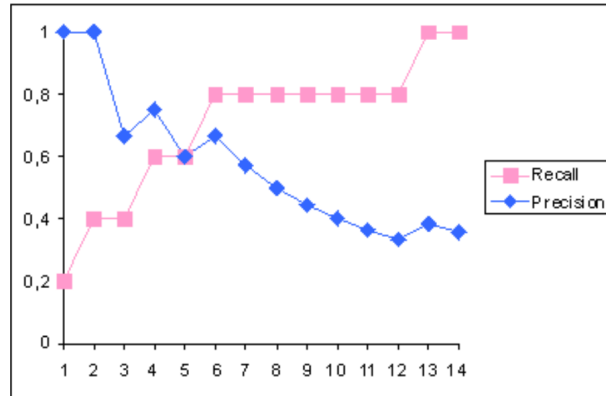


Fig. 2.2: *Rank-recall and rank-precision plots*

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.

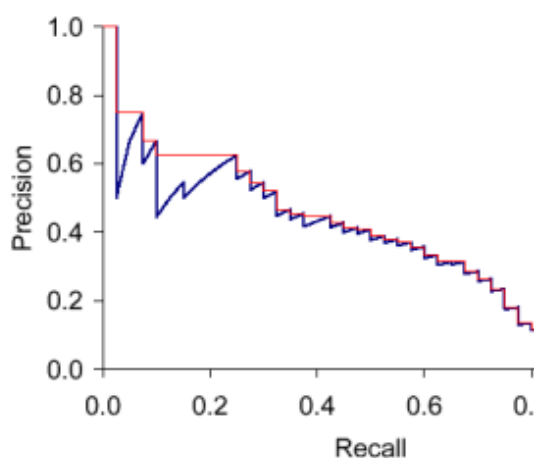


Fig. 2.3: *Recall vs Precision plot*

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**

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

- 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 obtiene su respuesta, 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 , se 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:

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

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

Conjunctive 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**.

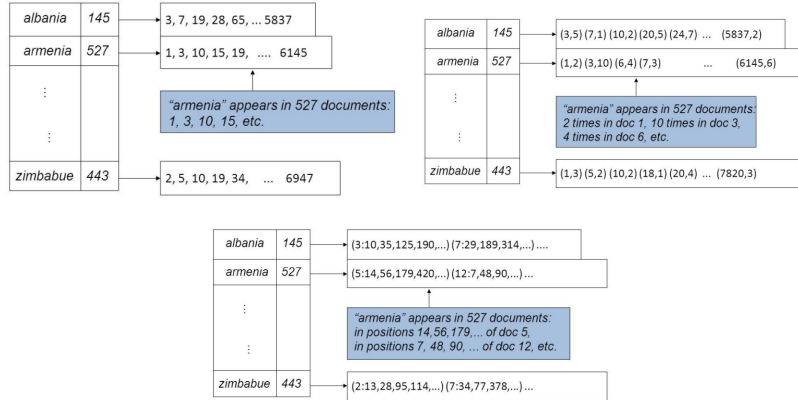


Fig. 3.1: Tres tipos de variantes del inverted file. En la variante 1 se guarda el índice de documento en los que aparece la palabra. En la variante dos también se contienen las frecuencias correspondientes a cada documento que contiene la palabra. En la variante 3 se añade la posición de la palabra 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 límites 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.

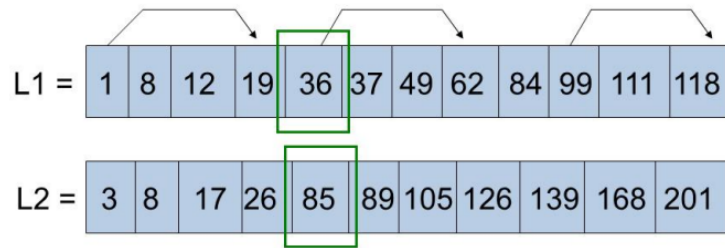
- **Tiempo:** longitud de la lista más corta $L_1 \times$ logaritmo de la lista más larga L_2 .

$$\text{if } |L_1| \ll |L_2| \implies |L_1| \cdot \log(|L_2|) < |L_1| + |L_2|$$

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
     $sim(d, q) += tf(d, w) \cdot idf(w) \cdot (1)$ 
  end for
  normalize  $sim(d, q)$  by  $|d| \cdot |q|$ 
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
     $sim(d, q) += tf(d, w) \cdot idf(w) \cdot (1)$ 
  end for
end for
for each  $d$  seen do
  normalize  $sim(d, q)$  by  $|d| \cdot |q|$ 
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 Frequency 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.

$$\text{unary}(15) = 111111111111111, |\text{unary}(x)| = x$$

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.

$$\begin{aligned} &[(id_1, f_1), (id_2, f_2), \dots, (id_n, f_n)] \\ &\quad id_1 > \dots > id_n \\ &[(id_1, f_1), (id_2 - id_1, f_2), \dots, (id_n - id_{n-1}, f_n)] \end{aligned}$$

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 $w = \text{binary}(x)$ 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.

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

$$EG(20) = [\text{len}(\text{bin}(20)) | \text{bin}(20)] = [00001 | 10100] = [000010100]$$

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 $\text{sim}(d, q) > \text{sim}_{\min}$ para un usuario que normalmente solo quiere los r -primeros ($r \ll R$). 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  $[d_1, \dots, d_r]$  in a minheap
for  $i = r + 1$  to  $R$  do
     $\text{min\_val} = \text{sim}(d, q)$  for  $d = \text{top of the heap}$ 
    if  $\text{sim}(d_i, q) > \text{min\_val}$  then
        Replace the smallest element in the heap with  $d_i$ 
        Reorganize the heap
    end if
end for

```

- $L = [d_1, \dots, d_R]$, 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 está 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 $r \ll R$.

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.

6 Network Analysis

La web puede verse como una red compleja, con enlaces entre páginas. El tipo de redes no es realmente relevante cuando se considera el gráfico desde un punto de vista matemático.

6.1 Examples of networks

6.1.1 Social Networks

Las aristas son interacciones entre personas. Puede haber simetrías entre las conexiones. Los nodos son personas.

6.1.2 Information Networks

Los nodos son una especie de recurso que contiene información. En el caso de la web son las páginas que contienen información. Las redes de citas son algo del tipo, donde los nodos son artículos y los enlaces son artículos que hacen referencia a otros.

6.1.3 Technological networks

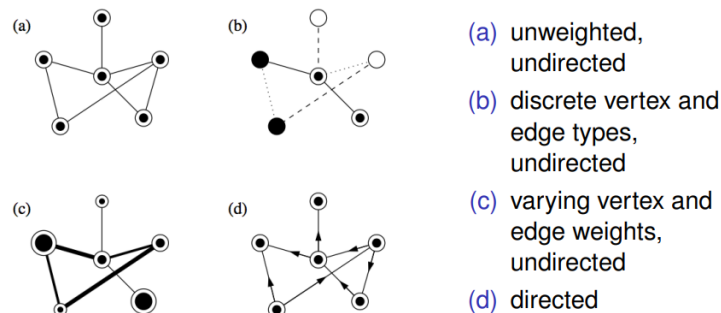
Son más de ingeniería, por ejemplo el mapa del metro. Nodos donde están las estaciones y luego ponemos una conexión si hay un tren que va de una estación a otra,

6.1.4 Biological networks

Esas son grafos que representan algún proceso biológico. Donde los nodos son genes, proteínas, etc. y todos ellos están relacionados por algún proceso que los afecta de forma conjunta.

6.1.5 Representing networks

Hay diferentes tipos de redes en los que puede haber pesos en las aristas al igual que las direcciones de las aristas. También se puede diferenciar entre diferentes tipos de nodos.



Hay tres propiedades comunes:

1. Existen caminos muy cortos entre la mayoría de los pares de nodos.
2. Un amigo de un amigo frecuentemente es también un amigo.
3. *Degree distribution* sigue una *power law*. Grado es el número de conexiones de un nodo.

Las propiedades 1 y 2 juntas se denominan propiedades de **small world**.

6.2 Small World phenomenon I

El fenómeno del small-world en el análisis de redes se refiere a la observación de que la mayoría de los nodos dentro de una red están relativamente estrechamente conectados a través de caminos cortos, incluso dentro de redes grandes y complejas.

Consideramos primero $d_{i,j}$ como la longitud del camino más corto de i a j . El **diámetro** (es decir, la distancia máxima del camino más corto) se define como:

$$d = \max_{i,j} d_{i,j}$$

En el análisis de redes, el diámetro de una red se refiere al camino corto más largo entre dos nodos cualesquiera dentro de esa red. Representa la distancia máxima entre cualquier par de nodos y proporciona una medida de qué tan lejos están los nodos más distantes entre sí. Si queremos medir globalmente de un grafo cual es el peor caso cogeremos la distancia máxima entre parejas de nodos. El diámetro es por analogía, si ponemos a todos en un círculo, la distancia máxima entre elementos es el diámetro.

En vez del diámetro que se trata de un máximo podemos coger el promedio. A veces se coge la media armónica de estas medidas. En casos en que tenemos grafos no conexos solo podríamos considerar parejas de nodos que están en la misma componente conexas, ya que sino la distancia es infinita y el sumatorio sería infinito.

$$l = \frac{2}{n \cdot (n+1)} \sum_{i>j} d_{ij}$$

Otro método más robusto de calcular el diámetro es no considerar el máximo sino la distancia que se encuentra en el percentil 95% de la distribución de distancias, lo que se llama **effective diameter**.

$$d \text{ s.t } 95\% \text{ of } d_{ij} \text{ are } \leq d$$

6.2.1 The (basic) random graph model

El modelo básico $G_{n,p}$ **Erdos-Renyi** es un gráfico con n nodos y cada arista está presente con probabilidad p . Lo que significa que generamos una arista entre dos nodos con probabilidad p independientemente de las otras aristas. Si fijamos la n lo que estamos haciendo es determinar esta distribución de probabilidad sobre grafos con esta cardinalidad. Lanzamos $\binom{n}{2}$ monedas y si nos sale cara pintamos la arista y sino no la pintamos. Si p es muy cercano a 0 observaremos que no tenemos muchas aristas, si p está cercano a 1 estaremos cerca de un grafo completo.

$$P(\#aristas) \approx Binom\left(\binom{n}{2}, p\right)$$

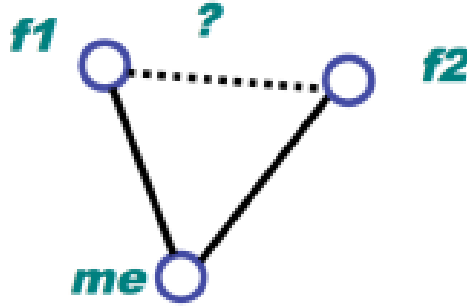
$$P(degree) \approx Binom(n-1, p)$$

En ER, **el diámetro de la red es $\log n$ con alta probabilidad**. Cogemos un nodo cualquiera dentro de la red generada aleatoriamente y miraremos sus vecinos. Con un salto cuantas conexiones podemos encontrar. Como sabemos que el grado de un nodo sigue una distribución binomial con una media de $(n-1) \cdot p$. La gran mayoría de nodos tendrán grado z que sería como está esperanza del grado de un nodo determinado.

Entonces a distancia l estamos conectando con z^l nodos. Vemos que con una distancia de $\frac{\log n}{\log z}$ realmente estamos empezando de un nodo aleatorio y llegando a n nodos. Asintóticamente si empezamos con cualquier nodo, en $\log n$ pasos nos encontramos con todos. Si queremos alcanzar n nodos, necesitamos tener $l = \log n$ ($n = z^l \rightarrow \log n = l \log z \rightarrow l = \frac{\log n}{\log z}$).

6.3 Small World phenomenon II

Para comprobar si dos amigos que tengo en común también lo son, en la mayoría de los casos definimos el **coeficiente de agrupamiento** o la **transitividad** como medida de probabilidad de que dos de mis amigos también sean amigos.



6.3.1 Global clustering coefficient

El coeficiente de agrupamiento global se calcula promediando los coeficientes de agrupamiento local de todos los nodos dentro de la red.

Una vez que se calculan los coeficientes de agrupación local de todos los nodos, el coeficiente de agrupación global se calcula como el promedio de estos coeficientes de agrupación local en todos los nodos de la red.

$$C = \frac{1}{n} \sum_{i=1}^n C_i$$

6.3.2 Local clustering coefficient

Para cada vértice i , sea n_i el número de vecinos de i . Sea C_i la fracción de pares de vecinos que están conectados entre sí (es decir, el número de aristas entre vecinos de i dividido por el número de posibles aristas entre vecinos de i). Entonces el coeficiente de agrupamiento local de i se define como:

$$C_i = \frac{\text{number of edges between neighbors of } i}{\frac{1}{2}n_i(n_i - 1)}$$

Donde $\frac{1}{2}n_i(n_i - 1)$ es el máximo número de posibles conexiones entre los vecinos de i . Si i tiene n_i vecinos entonces el máximo de aristas es

$$\binom{n_i}{2} = \frac{n_i!}{2!(n_i - 2)!} = \frac{n_i \cdot (n_i - 1) \cdot (n_i - 2)!}{2 \cdot (n_i - 2)!} = \frac{1}{2}n_i(n_i - 1)$$

Y C es el promedio de C_i sobre todos los nodos i de la red. Es difícil determinar la interpretabilidad de C , puesto que no hay referentes con los que se pueda comparar para un cierto valor de nodos. En su lugar se puede compara al C de un grafo similar al de la vida real pero creado de forma random (este grafo tendrá las mismas dimensiones que el grafo real).

En las redes ER, $C = p$ ya que cada arista está presente con probabilidad p independientemente de las otras conexiones. Para saber si la red real tiene un coeficiente global de clustering elevado, comparamos su C obtenida de la red real y la comparamos con p de un grafo random comparable.

Suponemos que tenemos una red real G con n, m nodos y aristas respectivamente. Su C_G será un numero determinado, para saber si es grande o no calcularemos que sería la p para un grafo similar generado de manera aleatoria. p debería ser igual a la densidad de \hat{p} (que se divide por el numero máximo de aristas posibles).

$$\hat{p} = \frac{|E|}{\binom{n}{2}}$$

Si quisiéramos generar un grafo similar al real (con el mismo valor de los parámetros) tendríamos que ajustar p a su valor estimado. Y si C_G es mucho mayor que el valor \hat{p} , podemos decir que G tiene transitividad alta. En la mayoría de los casos reales, $C_G \gg \hat{p}$. Si son similares, vemos que C_G no muestra transitividad alta.

► where p is estimated as $\hat{p} := \frac{|E|}{n(n-1)/2}$

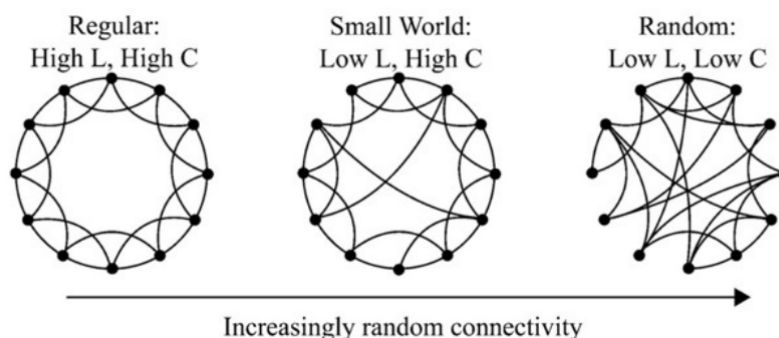
```
def is_clustered(G):
    C = clustering_coefficient(G)
    n = num_nodes(G)
    m = num_edges(G)
    p_est = m / math.comb(n, 2)

    return C >> p_est
```

Una forma de verlo es que se esta reestructurando el grafo original en uno con las conectividades de las aristas de una forma aleatoria.

6.3.3 Watts-Strogatz Model

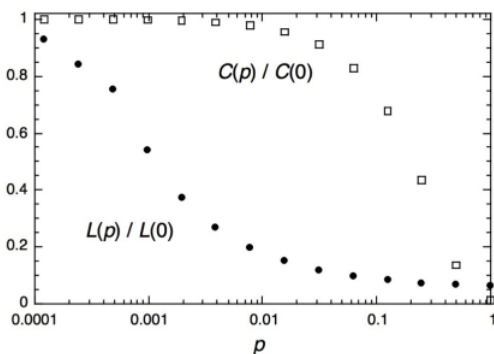
Otros tipos de grafos sintéticos si que presentan un valor de global clustering alto, no como es el caso de los grafos ER. El modelo Watts-Strogatz es un modelo de generación de gráficos aleatorios que produce gráficos con propiedades de small-world, incluidas longitudes de ruta promedio cortas y alta agrupación.



L se refiere a la distancia media mínima de máxima longitud. Se tiene que encontrar el valor de p adecuado para estar en el caso del centro. El caso de la derecha es en el que nos encontrábamos antes con los grafos ER.

El modelo se construye de la siguiente manera:

1. Comience con un anillo de n nodos, donde cada nodo está conectado a sus k vecinos más cercanos.
2. Para cada nodo i , y para cada arista (i, j) , vuelva a cablear la arista con probabilidad p . El recableado se realiza reemplazando (i, j) con (i, l) , donde l se elige uniformemente al azar de todos los nodos posibles evitando los self-loops y la duplicación de enlaces.

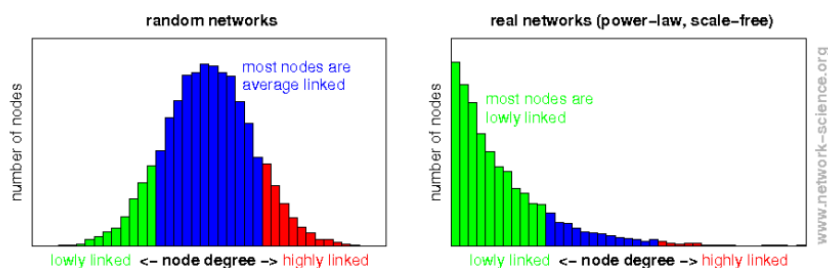


Notese que la L sufre un decremento mayor en función de p en comparación de la C . (La escala de p esta en escala logarítmica). EL valor $p = 0.01$ muestra un buen ajuste entre ambas propiedades ya que queremos un *cluster coefficient* alto y un valor de *diámetro* bajo.

6.4 Degree Distribution

La distribución de grados de la mayoría de las redes del mundo real sigue una ley de potencia, lo que significa que la probabilidad de que un nodo elegido al azar tenga un grado k es proporcional a $k^{-\alpha}$.

Histogram of nr of nodes having a particular degree



f_k = fraction of nodes of degree k

Esta distribución de **heavy tail** introduce la existencia de **hubs**, nodos con un grado muy alto, lo cual es muy poco probable en los grafos sintéticos.

En la distribución de los grafos random, el grado de los nodos se distribuyen según una Binomial $B(n, p)$ y para un valor de n grande se obtiene una distribución similar a la Normal. Lo que buscamos es una red artificial que presente la misma distribución en el grado de los nodos que un grafo real.

6.5 Scale-free/Scale-invariant Networks

Las redes que cumplen la distribución de grados como una función exponencial se llaman **Scale-free/Scale-invariant**. Las redes generadas de manera aleatoria no entran en este tipo de redes ya que como hemos visto no cumplen con esta distribución.

- Si escalamos el grado de los nodos de la red; $D(\lambda x) = f(\lambda)D(x)$ y entonces la forma de la distribución no cambia.
- Las redes ER random no son *scale-free*, la probabilidad de nodos con grado muy grande disminuye de manera exponencial (no *hubs*).

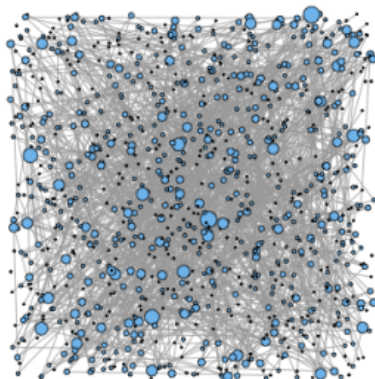
6.5.1 Preferential Attachment Model

Estos modelos siguen una dinámica de *los ricos se hacen más ricos* y tienen la característica de que son **Scale-free**. En lugar de tener a todos los nodos esparcidos al principio y conectarlos mediante conexiones, se empieza a construir añadiendo nodos secuencialmente a un grafo inicial. Las conexiones que se añadan seguirán al dinámica que se ha mencionado (entre más conexiones tenga un nodo, mas propenso sera para recibir una nueva conexión).

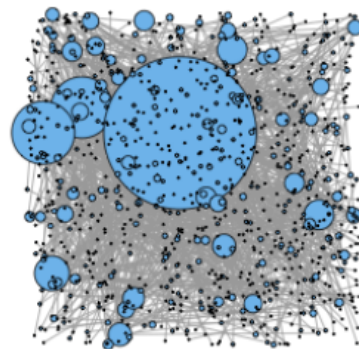
- Es modelo creciente, que crece con el tiempo al añadir nodos secuencialmente.
 - Nuevos nodos que se añadan, preferirán añadirse a nodos **well-connected**.
1. Empezamos con un subgrafo inicial donde n es el numero total de nodos.
 2. Cada nodo que se añade lo hace con m aristas.
 3. La probabilidad de conectar a un nodo ya existente i es proporcional a su grado. Como mayor grado mayor probabilidad.
 4. Resulta en una *power-law degree distribution* con exponente $\alpha = 3$.

El tamaño de los nodos es proporcional a su grado en la imagen debajo.

Experiment with 1000 nodes, 999 edges ($m_0 = 1$ in BA model).



random



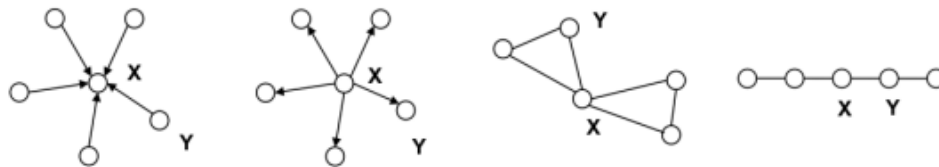
preferential attachment

6.6 Centrality

La centralidad está relacionada con la importancia de los nodos en una red, dependiendo de su estructura. Un nodo puede ocupar una posición muy conectada o influyente. Aprovechamos la posición de los nodos, ya que estar en una posición de influencia significa que el nodo es más importante.

- Un nodo central es **importante** y/o **poderoso**.
- Un nodo central ocupa una posición **influyente** en la red.
- Un nodo central tiene una posición **ventajosa** en la red.

En este caso, X es más influyente ya que recibe más información de sus vecinos. Esto también está relacionado con el *in-degree*. X es más central porque puede llegar a más nodos con sus conexiones. La idea de que X sea más central que Y se debe a que X está en el cuello de botella entre las dos partes de la red y es crucial para transmitir procesos de una parte de la red a otra. En el ejemplo de la derecha, X es más importante porque está más cerca de todos los nodos de la red.



6.6.1 Degree centrality

Definimos la importancia del nodo como el número de vecinos directos que tiene sin mas consideraciones de los nodos vecinos. Normalmente, estos valores se normalizan para que estén en el rango de 0 a 1.



Si lo importante es llegar rápidamente a otros nodos, la centralidad de grado no es la métrica adecuada, por ejemplo.

6.6.2 Closeness centrality

Tiene que ver con cuán cerca estás de cualquier otro nodo en la red. Se define como el inverso de la distancia media a todos los demás nodos. *Si estas mas cerca de otros nodos en promedio, eres mas central.*

$$\text{closeness_centrality}(i) = \left(\frac{\sum_{j \neq i} d(i, j)}{n - 1} \right)^{-1} = \frac{n - 1}{\sum_{j \neq i} d(i, j)}$$



Aquí, lo que importa es estar cerca de todos los demás, es decir, poder ser fácilmente alcanzable o tener el poder de llegar rápidamente a otros.

6.6.3 Betweenness centrality

Lo que captura es centralidad al evaluar cuán crucial es la posición de un nodo conectando diferentes partes de la red. Se refiere a que un nodo es un cuello de botella (o parte de el) que permite conectar diferentes componentes.

$$\text{betweenness_centrality}(i) = \sum_{j < k} \frac{g_{jk}(i)}{g_{jk}}$$

donde

- g_{jk} es el número de caminos más cortos entre los nodos j e k .
- $g_{jk}(i)$ es el número de caminos más cortos que pasan por i ; solo consideramos aquellos que pasan explícitamente por el nodo i .

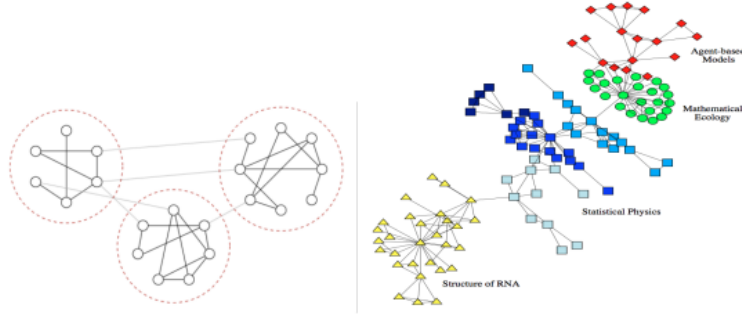
Tiene sentido que, para calcularlo, vayamos para cada pareja de nodos i , y cada pareja de nodos contribuye a la centralidad de i con este ratio de la fórmula. Lo que queremos calcular es cuán crucial es i para conectar componentes.

Podemos normalizarlo para que este entre 0 y 1.

$$\text{norm_betweenness_centrality}(i) = \frac{\text{betweenness_centrality}(i)}{\binom{n-1}{2}}$$

6.7 Communities

Alunos nodos se han encerclado, lo que es común en cada comunidad es que hay mas componentes conectados dentro de los componentes y hay pocas aristas conectando las diferentes comunidades. Lo que vemos a la derecha son los colores dependiendo de las comunidades con un algoritmo automático donde los nodos dentro de una comunidad están mas relacionados entre ellos. A posteriori se puede mirar y hacer un diagnostico si hay unas características comunes que comparten los nodos dentro de la misma comunidad.



La definición general de comunidades no existe (como sucede en el caso de clustering y se busca maximizar la noción de comunidad), pero hay un consenso que consiste en:

- *A community should be densely connected*; es decir que debemos encontrar muchas conexiones dentro de cada comunidad.
- Debe ser esparsa para el exterior, pocas conexiones con el exterior. Si tuviéramos conexiones densas en el exterior, probablemente esas conexiones deberían formar parte de la comunidad.
- *Members of a community should be more similar among themselves than with the rest*. Típicamente las comunidades se forman de nodos que son similares en algún sentido.

Existen varias definiciones de que significa ser una comunidad. Dado un grafo $G = (V, E)$ con $|V| = n$ nodos y $|E| = m$ aristas. Sea C un subconjunto de nodos (cluster o comunidad) de tamaño $|C| = n_c$. Entonces:

- **Intra-cluster density:** Es la densidad cuando miramos dentro de la comunidad (termino relativo). Contamos cuantas aristas hay dentro de la comunidad y lo dividimos por el numero máximo de aristas que podemos tener. Será 1 si todas las conexiones internas son presentes i 0 sino (normalizado). Debería estar cercano a 1 si C realmente es una comunidad.

$$\delta_{int}(C) = \frac{\text{nr. internal edges of } C}{n_c(n_c - 1)/2}$$

- **Inter-cluster density:** (densidad relativa) Numero de conexiones que van desde nuestra comunidad hasta el exterior. Dividimos por el numero máximo posible de conexiones con nodos de fuera de la comunidad. Esto numero debería estar cercano de 0 si realmente tenemos una comunidad real.

$$\delta_{ext}(C) = \frac{\text{nr. internal edges of } C}{n_c(n - n_c)}$$

Esperamos que sea menor que la interna y cerca a 0. Con pocas aristas cruzando la frontera de la comunidad.

Una comunidad debería tener $\delta_{int}(C) > \delta(G)$ donde $\delta(G)$ es la densidad de aristas media de todo el grafo.

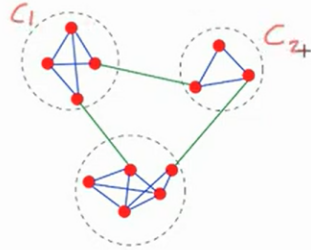
$$\delta(G) = \frac{\text{nr. edges in } G}{n(n-1)/2}$$

La mayoría de algoritmos tienen un trade-off entre gran $\delta_{int}(C)$ y pequeñas $\delta_{ext}(C)$. Buscamos C con estas características. Por ejemplo podríamos intentar optimizar (maximizar)

$$\sum_C \delta_{int}(C) - \delta_{ext}(C)$$

para todas las comunidades C . Pero este problema es NP-hard y es un problema muy complicado que la mayoría de algoritmos no encontraran óptimos globales pero si heurísticas para conseguir buenas soluciones. Es un problema muy similar a los problemas que se encuentran en algoritmos de clusterización.

- m_c es el numero de aristas dentro del cluster $C = \{(u, v) | u, v \in C\}$
- f_c es el numero de aristas en la frontera entre el cluster i los demás nodos. $C = \{(u, v) | u \in C, v \notin C\}$



- $n_{c_1} = 4, m_{c_1} = 5, f_{c_1} = 2$
- $n_{c_2} = 3, m_{c_2} = 3, f_{c_2} = 2$
- $n_{c_3} = 5, m_{c_3} = 8, f_{c_3} = 2$

6.7.1 Community Quality Criteria

- **Conductance:** Es la fracción de aristas que se extienden fuera de la comunidad. En el numerador, consideramos las aristas que cruzan las fronteras de la comunidad. La normalización difiere en este caso, ya que nos enfocamos en todas las aristas que tienen al menos un extremo dentro de la comunidad. Es importante multiplicar por 2 debido a que cada arista tiene dos extremos. El objetivo es minimizar la conductancia, ya que en comunidades densas, se espera que un menor número de aristas se conecten fuera de la comunidad.

$$\frac{f_c}{2m_c + f_c}$$

- **Expansion:** Numero de aristas por nodo que salen de la comunidad. Se busca minimizar. Otro tipo de normalización.

$$\frac{f_c}{n_c}$$

- **Internal density:** a.k.a. “intra-cluster density”. Se busca maximizar.

$$\frac{m_c}{n_c(n_c - 1)/2}$$

- **Cut ratio:** a.k.a. “inter-cluster density”, que porcentaje de las aristas que cruzan la frontera están presentes. Se busca minimizar.

$$\frac{f_c}{n_c(n - n_c)}$$

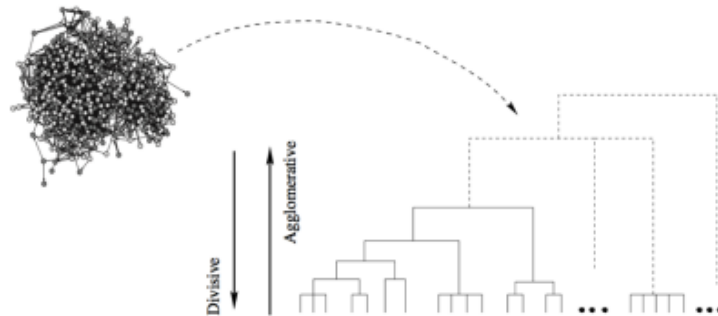
- **Modularity:** Diferencia entre el número de aristas en C y el numero esperado de aristas $E[m_c]$ de un grafo random (la noción de *random graph* es un parámetro). Alta modularity representa la presencia de aristas internas con respecto con un modelo de grafo random.

$$\frac{1}{4m}(m_c - E[m_c])$$

Mientras que el resto de métricas son puramente combinatorias, pero no hay noción de comparación de numero que se obtiene con respecto al esperado.

6.7.2 Hierarchical clustering

Queremos encontrar el dendograma a partir de la bola de nodos que tenemos. Lo que veremos es que iremos de un desorden (bola de nodos) hacia el dendograma donde los nodos están en las hojas de manera que podemos cortar para formar comunidades.



En niveles interiores, tenemos más comunidades con un nivel de resolución mayor. Dependiendo de como se construyan estos dendogramas, se pueden construir de cada nodo siendo su propio cluster y combinarlas, o empezar con una sola comunidad e ir las dividiendo. Suele ser útil cuando el grafo de entrada ya presenta noción de clustering de forma jerárquica.

Agglomerative Hierarchical Clustering

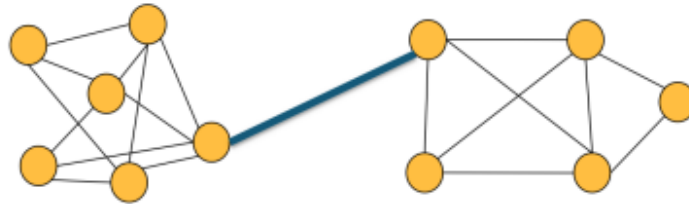
- Medida de similitud entre nodos. (Jaccard, Cosine, Hamming)
- Similitud entre los conjuntos de nodos. (Single, Complete, Average linkage)

Modelo iterativo:

1. Assign each node to its own cluster
2. Find the cluster pair with highest similarity and join them together into a cluster
3. Compute new similarities between new joined cluster and others
4. Go to step 2 until all nodes form a single cluster

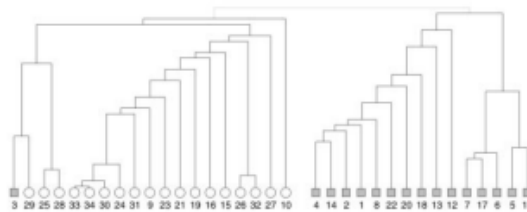
Divisive Hierarchical Clustering

The Girvan-Newman algorithm: Lo interesante es que usa Edge betweenness para decidir como dividir las comunidades existentes. Empezamos con el grafo siendo una comunidad, y a partir de allí vamos partiendo aristas (transformando la betweenness de un nodo a betweenness de aristas) para crear comunidades separadas. La idea es que este cuello de botella esta dividiendo comunidades. La betweenness de un nodo de dentro de una comunidad no sera alta ya que tendrá muchos nodos alrededor.



1. Compute betweenness for all edges in the network
2. Remove the edge with highest betweenness
3. Go to step 1 until no edges left

Result is a dendrogram



6.7.3 Definition of Modularity

Incorpora la idea de comparar la estructura de nuestro grafo (y su descomposición de los nodos dentro de las comunidades) con lo que esperaríamos en un **random graph** usando **null models**.

La modularity $Q(\cdot)$ de una partición de comunidades c_1, \dots, c_K depende del **null model** que se definan:

$$Q(c_1, \dots, c_K) = \frac{1}{2m} \sum_k (m_k - E[m_k]), \quad m = |E|$$

Lo que hacemos es hacer el *average* de comparar las aristas que tenemos en la comunidad con el número de aristas esperadas según la estadística de la comunidad c_k del **null model**. Si esto es positivo es que tenemos mas aristas de las que esperaríamos, por lo que tenemos una estructura de comunidad fuerte. Si tenemos pocas aristas esto puede ser negativo porque significa que tenemos una partición donde hay poca evidencia que realmente tenemos comunidades.

Para un null model ER: $E[m_k] = p \binom{|c_k|}{2}$ aunque comúnmente se emplea el **Modelo de Configuración** debido a que proporciona un grafo aleatorio con una distribución de grados idéntica a la del grafo original, capturando así de manera más fiel las características del grafo. En este modelo, la probabilidad de que dos nodos i y j estén conectados se define como $E[i \text{ and } j \text{ are connected}] = \frac{\deg(i) \cdot \deg(j)}{2m}$.

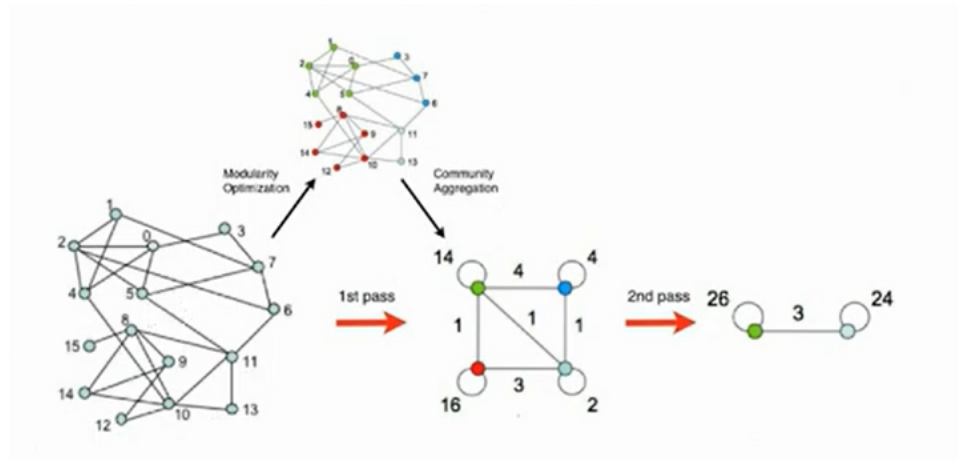
Analizaremos todas las comunidades, pero en lugar de contar aristas, examinaremos cada par de nodos dentro de la comunidad observada. Registraremos A_{ij} como 1 si la arista está presente y 0 si no lo está. En el contexto de una comunidad real, esperamos encontrar numerosas aristas internas que contribuyan positivamente a la modularidad; sin embargo, si la contribución es negativa, es probable que no se trate de una comunidad.

La expresión $\frac{\deg(i) \cdot \deg(j)}{2m}$ representa la probabilidad de que los nodos i y j estén conectados en caso de reconectar aleatoriamente todas las aristas de la comunidad. Básicamente, en un grafo aleatorio que conserva la misma distribución de grados que el grafo original, si i tiene $\deg(i)$ posibles conexiones y j tiene $\deg(j)$ extremos potenciales, calculamos la probabilidad de que i y j estén conectados al reorganizar i de manera aleatoria con el resto de los nodos. Cuanto mayor sea el grado de ambos nodos, mayores serán las posibilidades de reorganizarlos de esta manera

$$Q = \frac{1}{2m} \sum_k \sum_{i \neq j \in V_i, j \in c_k} \left(A_{ij} - \frac{\deg(i) \cdot \deg(j)}{2m} \right)$$

Louvain Method

Heurística que optimiza la modularity de forma eficiente. Procedimiento iterativo *bottom-up*. Se comienza con una comunidad para cada nodo y se van uniendo de forma aglomerativa.



Comenzamos con un grafo y de sus particiones se construyen supergrafos, donde los supernodos son las particiones del grafo anterior. Se repite hasta encontrar un óptimo local.

1. Particionamos la red de una forma greedy, optimizando el criterio de modularity.
2. Las particiones, se convierten en nodos, super nodos y super aristas (pueden haber self-loops) de un super grafo.

La parte difícil es encontrar la partición de forma greedy.

1. Se asigna primero una comunidad para cada nodo.
2. Para cada nodo i :
 - Para cada vecino j de i , se remueve i de su comunidad y se coloca en la comunidad de j .
 - De forma greedy, se decide colocar a i en la comunidad que resulte en la máxima ganancia de modularity.
3. Repetir hasta que no se pueda hacer ninguna mejora.

Observaciones

- La mayoría del tiempo gastado en el algoritmo es en el primer *greedy loop*, aproximadamente el 95% del tiempo.
- Para grafos con estructura conocida el tiempo de ejecución es conocido y es del orden $O(n \log(\text{avg degree})) = O(n \log n)$
- El output es una jerarquía.
- Louvian también puede ser usado con pesos.

6.8 Spreading in networks

Cuando no tenemos toma de decisiones, no hay estrategia. Es aleatorio. Lo que asumimos es que tenemos un grupo de personas y en algún momento una persona se infecta (hay un momento 0) que hace que se pueda contagiar a otras personas. Esta persona va por el mundo y se va encontrando gente; cada vez que tiene un contacto con una persona, por probabilidad (hay un parámetro q , que normalmente se llama **infection rate**), existe la probabilidad de que, si hay contacto, haya infección.

Veremos que en función de este q y de la tipología de la red donde modelamos los contactos (red social), miraremos cuándo este paciente 0 contagia a todo el mundo (el contagio con pandemia) o simplemente contagia a unos cuantos y la gente se recupera.

Escogemos como red de contacto para modelar una Erdős/Rényi-like para que la red sea fácil de modelar. Cada nodo es igual que el resto y no hay comunidades. Todo un poco más aleatorio y homogéneo. Evidentemente, no es realista, porque no tiene comunidades y en la realidad sí que ocurre. Se tiene que simular la propagación y ver cómo cambian las cosas sobre la red.

Los nodos tienen, en promedio, un grado $d = pn$. Dependiendo del parámetro de la red, se asume que todos los nodos tienen un grado muy similar a d , con una distribución cercana a la normal. Ver personas con un grado mucho más pequeño o mucho más grande es muy complicado. Se trata de una simplificación. En redes reales, esto no es cierto; nos encontraríamos con otra distribución con grados especialmente grandes/pequeños.

p_l = Probability that at least one node at depth l is infected

$$\lim_{l \rightarrow \infty} p_l = \begin{cases} 0 & \text{if the epidemics dies out,} \\ 1 & \text{if the epidemics remains.} \end{cases}$$

- $p_0 = 1$ por hipótesis
- Ningún nodo infectado en el nivel 1 con probabilidad $(1 - q)^d$, suponiendo independencia de contagios.
- $p_1 = 1 - (1 - q)^d$ Mirar la probabilidad complementaria, es decir, 1 - la probabilidad que ninguno este infectado.
- De forma inductiva, en el nivel l , la probabilidad de que haya llegado la infección es $p_l = 1 - (1 - qp_{l-1})^d$.

$$\lim_{t \rightarrow \infty} p_t = \begin{cases} 0 & \text{if } q \cdot d < 1, \\ 1 & \text{if } q \cdot d \geq 1. \end{cases}$$

El contagio o **reproduction number** $R_0 = nd$ depende del tipo de red que usemos. Para prevenir la infección y su propagación, se intenta reducir R .

- If $R_0 < 1$, infection dies out.
- If $R_0 > 1$, epidemic never dies and spreads exponentially.

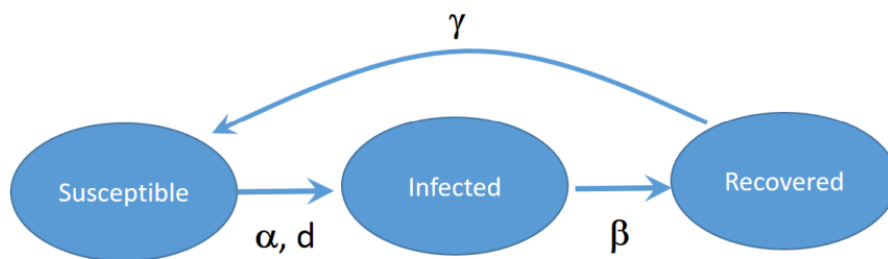
En la realidad no todos los nodos tiene el mismo grado. Hay nodos con un grado mucho mayor **super-spreaders**. EL caso medio es un fenomeno que no desaparece pero tampoco se convierte en pandemia. Este comportamiento

- Simula el proceso con redes de tipo scale-free. Con comunidades.
- La infección no muere, sino que es de forma exponencial.

6.8.1 Other models of infection

Enfoques basados en cadenas de Markov o ecuaciones diferenciales se utilizan para modelar el comportamiento dinámico de poblaciones en sistemas SIR. El modelo SIR clasifica a las personas en tres estados: Susceptibles, Infectados o Recuperados. Los individuos susceptibles pueden contraer la infección con una tasa α (tasa de contagio) y un grado d (promedio de conexiones). Los susceptibles se transforman en infectados, y aquellos que sobreviven pueden adquirir inmunidad, ya sea de forma permanente o temporal. Se puede extender el modelo para considerar nacimientos y defunciones, lo que agrega complejidad a la dinámica de la población.

Podemos representar los estados por los que pasa una persona en este modelo. Inicialmente, una persona está en el estado de "sano pero susceptible" con parámetros α y d , denotando la probabilidad de contraer la infección. Los individuos pueden pasar de susceptible a infectado y también de recuperado a susceptible con una tasa γ . Este enfoque permite simular cómo evolucionan los estados de las personas en el tiempo, proporcionando una representación matemática de la propagación de la infección en la población. Inmunidad se representa con $\alpha = 0$.



6.8.2 Two algorithmic problems - combinatorial optimization

Dada una red, información sobre los mecanismos de propagación y un presupuesto k (numero de nodos en los que se puede hacer una accion):

- **Cómo detectar brotes pronto:** Elegir un subconjunto S de nodos, $|S| \leq k$, "observadores", que minimice el tiempo de detección de una cascada grande.
- **Cómo maximizar la propagación:** Elegir un subconjunto S de nodos, $|S| \leq k$, "influyentes" o "semillas", de manera que las cascadas iniciadas desde ellos se propaguen al máximo (se vuelvan virales).

7 Hashing

7.1 Hashing functions

Una función de hash $h : X \Rightarrow Y$ distribuye elementos de X de manera aleatoria entre los elementos de Y . Sea H un conjunto de funciones de hash, la aleatoriedad se encuentra en escoger una función de H .

El objetivo es evitar colisiones, ya que estas lo que provocan es que las estructuras de datos que tenemos vayan más lentas. En nuestro caso, lo que buscamos **es que haya colisiones solo entre elementos similares, pero no entre elementos no semejantes**.

- **Perfect Hashing**: Mapeo de 1 a 1, por lo que evita colisiones. No nos interesa.
- **Universal Hashing**: Se limita la probabilidad de colisión de objetos. Queremos que la probabilidad de colisión sea $1/n$, lo más parecido a aleatorio.
- **Locality sensitive hashing (lsh)**: Es el modelo que usaremos. Queremos una colisión (su imagen respecto la función de hash es la misma) si son suficientemente parecidos, y no queremos que colisionen si no se parecen. .

7.2 Local Sensitive Hashing (LSH)

Local Sensitive Hashing (LSH) es una técnica utilizada en el campo de la recuperación de información y la minería de datos para aproximarse a la similitud entre conjuntos de datos. La idea principal detrás de LSH es mapear elementos similares a la misma "ubicación" o "vecindad" en un espacio hash, de modo que elementos similares tengan una alta probabilidad de colisionar (ser asignados al mismo valor hash).

Sirve para encontrar dos objetos que son muy similares. Puede ser útil cuando queremos detectar plagio entre documentos con plagio un poco disfrazado, es decir, copias que no son exactamente iguales.

Hace que la detección de duplicados sea más rápida. Si queremos verificar que un objeto es un duplicado de otro original en una base de datos grande, la comparación de documentos con una función de similitud mediante fuerza bruta tendría un costo lineal con el número de elementos de la base de datos y no escalaría bien.

LSH resuelve esto en tiempo sublineal, en tiempo logarítmico posible gracias a las **tablas de hash**. Los falsos negativos podrían ocurrir al usar LSH, cosa que no pasaría con la fuerza bruta.

7.2.1 Locality sensitive hashing functions

Normalmente, al considerar funciones de hash, no nos limitamos a una sola función. Las probabilidades se calculan según la función de hash seleccionada de una familia de funciones F . Elegimos una función de manera aleatoria dentro de esta familia y calculamos las probabilidades a partir de ella.

Una familia F es considerada una buena función de hash $(s, c \cdot s, p_1, p_2)$ si, para cualquier x e y , se cumple que:

$$\begin{cases} \text{If } s(x, y) \geq s, & \text{then } P[h(x) = h(y)] \geq p_1 \\ \text{If } s(x, y) \leq c \cdot s, & \text{then } P[h(x) = h(y)] \leq p_2 \end{cases}$$

Ser **similar** implica que la similitud entre dos objetos (asumiendo que tenemos una función de similitud como cosine similarity, inverso de Hamming, Jaccard, entre otros) es mayor que s (un parámetro proporcionado). En este caso, la probabilidad de colisión es lo suficientemente alta, definida por p_1 , que podría ser, por ejemplo, 0.8.

Por otro lado, la **probabilidad de colisión debe ser baja si los objetos no son similares**. $c \cdot s$ es el límite superior de similitud para considerar que son similares. Si la similitud es menor que este valor, los consideramos no similares, y por lo tanto, queremos que la probabilidad de colisión sea lo suficientemente pequeña, determinada por p_2 .

7.2.2 Cómo utilizar LSH para encontrar el vecino más cercano

Para abordar el problema de encontrar duplicados o casi duplicados, seguiremos estos pasos.

- **Preprocesamiento:** No se realiza en tiempo real. Realizamos una indexación previa para facilitar la búsqueda rápida de duplicados (vecino más cercano) cuando se presenta una consulta. Durante el tiempo de preprocesamiento, seleccionamos de manera aleatoria una función de entre la familia apropiada F y la almacenamos. Calculamos la función de hash para todos los objetos (por ejemplo, para todos los documentos de nuestro corpus) y la guardamos. Calculamos $h(x)$ para todos los objetos x en el conjunto de datos.
- **Cuando llega una nueva consulta q :** Queremos detectar si es un plagio de los que ya tenemos. Calcularemos $h(q)$ y examinaremos secuencialmente entre las colisiones del documento q . Lo único que necesitamos detectar son posibles plagios entre las colisiones. Esta verificación se realiza sobre un subconjunto mucho más pequeño que tomar todo el conjunto de datos.

Por ejemplo, puede ser útil cuando quiero encontrar los elementos más cercanos a mi consulta cuando hay muchos elementos en la base de datos.

Tablas de hash: Método de indexación de los elementos. Es una estructura de datos que implementa una colección de pares clave-valor. La principal idea detrás de una tabla de hash es utilizar una función de hash para convertir las claves en índices de la tabla, donde se almacenan los valores correspondientes. Esto permite un acceso rápido y eficiente a los valores asociados con una clave dada. Se hace de manera online.

7.2.3 Hashing Family functions

Ejemplo con vectores de bits longitud d . La similitud entre los vectores es el inverso de la distancia de Hamming. Normalizada para que esté entre 0 y 1.

$$s(x, y) = 1 - \frac{d(x, y)}{d}$$

Se proponen las funciones de hashing como proyecciones. La familia de funciones

$$F = \{f_i \mid i \in [d]\} \text{ donde } f_i(x) = x_i$$

retornan el elemento i -ésimo del vector de bits. Entonces la probabilidad de la colisión es la similitud (proporción de posiciones del vector que coinciden).

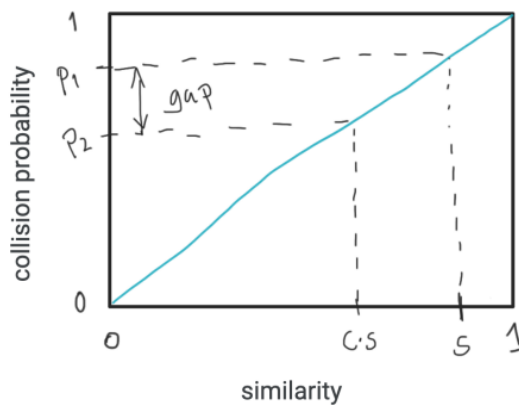
$$P[h(x) = h(y)] = s(x, y)$$

Esto es porque $s(x, y)$ es la proporción de las posiciones que coinciden los dos vectores x y y . Si escogemos una posición de forma aleatoria será justo esta probabilidad. (Escogemos la función de hash de manera aleatoria con probabilidad $\frac{1}{d}$). Entonces F es (s, cs, s, cs) -sensitive (con $c < 1$ para que $s > cs$) ya que la probabilidad de colisión es exactamente la similitud por lo que creamos una familia con buenas propiedades.

Distinguimos las parejas que tienen similitud más que s . Para parejas (vectores) similares, la probabilidad de colisión es mayor o igual que p_1 . El **gap** (diferencia entre p_1 y p_2 se buscaría ampliar.

Queremos que, para aquellas parejas cuya similitud sea mayor que s , la probabilidad de colisión sea muy alta, ya que, al realizar la búsqueda, deseamos que se encuentren juntas. Sin embargo, no queremos que todas las parejas colisionen, por lo que preferimos que p_2 sea lo más bajo posible. Idealmente,

The “gap” $p_1 - p_2$ in this case is $s - sc$



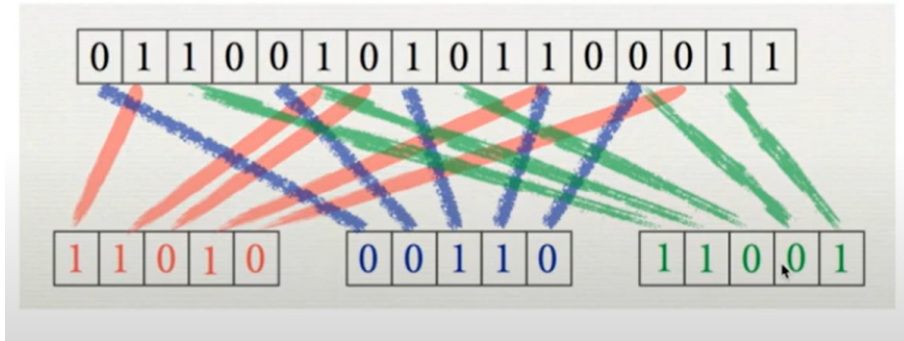
Se puede ampliar aumentando el valor de p_1 o disminuyendo la probabilidad de p_2 .

- Queremos que para dos documentos que son diferentes (en el contexto de vectores de bits), la probabilidad de colisión sea mucho más baja. En lugar de seleccionar solo 1 bit en la función de hashing, cogeremos k bits. Ahora la probabilidad de que dos documentos diferentes coincidan en las k posiciones es más difícil de conseguir. Esto hace que la probabilidad de los objetos por debajo del umbral $c \cdot s$ realmente sea más pequeño.
 - La probabilidad de colisión de objetos similares disminuye; s^k , pero disminuye aún más la probabilidad de colisión de objetos no similares; $(c \cdot s)^k$
- Repitiendo este paso (m veces, se usan m funciones), se evita que elementos que sí tienen elementos en común no caigan en la misma celda. Cuando se busquen duplicados, no solo se buscarán en una tabla de hash sino en m , para que la probabilidad de p_1 incremente (no sea tan pequeña).

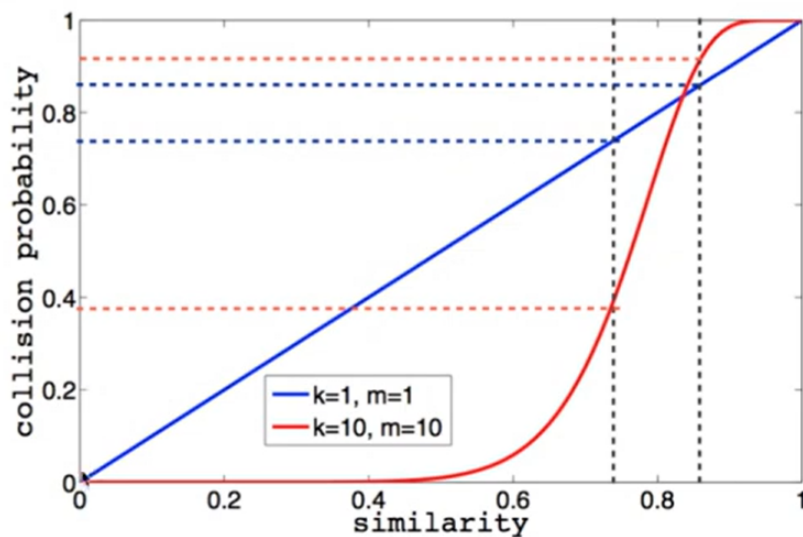
Por lo tanto, la k lo que hace es evitar que por mala suerte objetos poco similares sean mapeados al mismo cajón (colisionen). Pero como esto también pasa con objetos que son similares (en menos medida) lo que hacemos es repetir el proceso. Con estas repeticiones estamos cambiando nuestra familia modificando nuestras probabilidades de la siguiente forma:

$$(s, c, 1 - (1 - s^k)^m, 1 - (1 - (cs)^k)^m)$$

A string of d bits gets hashed to a string of km bits.
Here, $k = 5, m = 3$



Aquí creamos $m = 3$ funciones de hash diferentes donde cada uno consiste en proyecciones de tamaño $k = 5$. Este vector iría a 3 cajones correspondientes a las proyecciones, cada una a su respectiva tabla de hash (hay m tablas).



En esta imagen se ve lo que estamos consiguiendo. La línea azul es nuestro esquema original, con $k = 1$ y $m = 1$, es decir solo escogemos 1 bit para proyectar y solo tenemos 1 tablas de hash. Entonces la probabilidad de colisión es proporcional a la similitud.

Con $k = 10$ y $m = 10$ (10 proyecciones y 10 tablas de hash) lo que hacemos es variar esta probabilidad de colisión como se ve en la línea roja. Estamos amplificando el gap.

7.2.4 Similarity search

Preprocesamiento:

- **Entrada:** Conjunto de objetos X
- Para $i = 1..m$
 - Para cada $x \in X$
 - * Apilar k funciones de hash y formar $x_i = (h_1(x), \dots, h_k(x))$
 - * Almacenar x en el cajón dado por x_i

En el momento de la consulta:

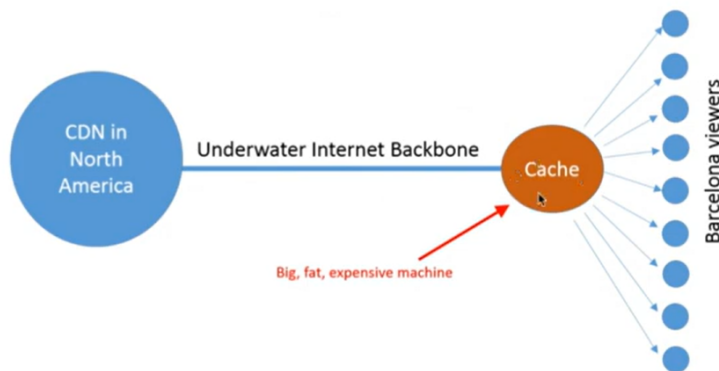
- **Entrada:** Objeto de consulta q
- $Z = \emptyset$
- Para $i = 1..m$
 - Apilar k funciones de hash y formar $q_i = (h_1(q), \dots, h_k(q))$
 - $Z_i = \{\text{objetos encontrados en el cubo } q_i\}$
 - $Z = Z \cup Z_i$
- Devolver todos los $z \in Z$ tal que $s(q, z) \geq s$

Otro caso con vectores de enteros. Se hace una especie de reducción. Se hace una representación unaria de los elementos con longitud del elemento máximo. La gracia de la representación es que la diferencia se convierte en la distancia de Hamming y se puede usar el mismo esquema de familias que en el caso anterior.

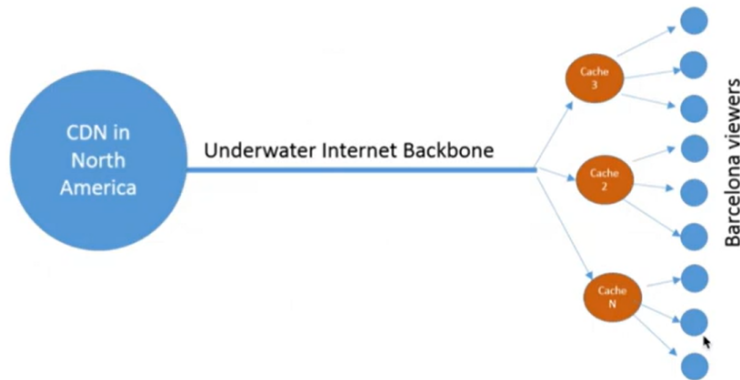
La idea general es buscar una familia de funciones de hash para que se asemejen (idealmente igual) a la función de similitud (también escogida). Después se juega con los parámetros k y m para amplificar el gap.

7.3 Consistent Hashing

Intentan resolver un problema totalmente diferente. El problema que se busca resolver es la distribución de contenido para que los usuarios accedan rápidamente y no vean caídas de réplicas.



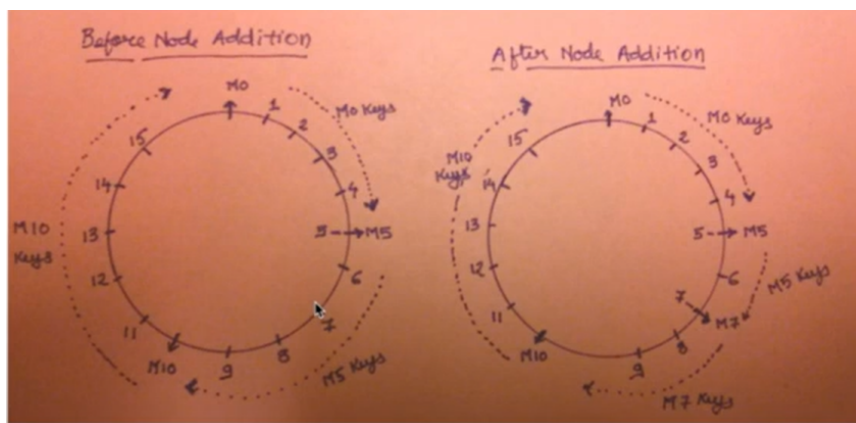
Se distribuyen en máquinas más pequeñas y manejables y los usuarios ya sabrán a dónde ir de las mini máquinas según sus intereses.



Como función de hash típica es el módulo N . El problema es que si el tráfico crece y se quiere poner una máquina nueva, se tendría que recolocar el contenido en las nuevas máquinas. Cambiar de posición entre las máquinas y este es el problema (comporta movimiento de datos).

La idea es tener dos tipos de funciones de hash. No solo la información se pasa a través de la función de hash h_I sino que las máquinas (id) se pasan también por la función de hash h_S .

La representación es mediante estos tipos de círculos, donde M_i es la codificación hash de los servidores, mientras que los números son la codificación de los identificadores de las páginas. Las páginas se asignan al servidor más cercano en sentido contrario a las agujas del reloj.



De modo que cuando se agrega un nuevo servidor, la codificación hash lo posiciona en un punto de la circunferencia y únicamente las de entre este nuevo servidor y el siguiente en sentido horario son las que se tienen que recolocar. En este caso de la imagen, el nuevo servidor se posiciona con la codificación hash en M_7 y las únicas páginas que se tienen que recolocar es la de identificador 6.

La implementación se hace por medio de un árbol binario. Ahora el tiempo de la función de hash depende de la profundidad de la profundidad del árbol binario.

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 Moonstone, 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 screw^d down so as to over-shadow her face. Instead of look^{ing} up at us in her usual straightforward way , she sat close at the table, and kept her eyes fix^{ed} obstinately on an open book.

"Officer," she said, "it is important to the inquiry you are conduct^{ing} to know beforehand if any person now in this house wish^{es} 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.

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

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 $frequency = c \cdot rank^{-\alpha}$.

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).

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

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 frequency 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 frequencies (generalizing Zipf's law) from Heap's law]

Calculamos primero la constante de proporcionalidad c

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

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 :

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

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:

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

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

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

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.

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

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

$$(\text{fire OR gold}) \text{ AND } (\text{truck OR NOT silver})$$

Do the same with the query

$$(\text{fire OR NOT silver}) \text{ AND } (\text{NOT truck OR NOT fire}).$$

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:

$T = \{\text{efficiency, implementation, particular, application, classification, methods, followed, pattern}\}$

- 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

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

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 $Q = \text{"computer software programmers"}$, 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: $q = (1, 1, 0, 0, 0, 1)$.

- D1 = "programmers write computer software code"

$$\begin{aligned} \text{freq} &= (1, 1, 0, 1, 0, 1) \\ v_{d1} &= (\log 10, \log 10, 0, \log 50, 0, \log 50) \\ \text{sim}(q, v_{d1}) &= 0.766 \end{aligned}$$

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

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

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

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

(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 serían 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 $d_1 = \text{"A B B A C C"}$ and $d_2 = \text{"D A D B B C C"}$ using tf-idf weighting and the cosine measure.

$$\begin{aligned}D &= 100,000 \\w_{d_1} &= (\log_2 10, \log_2 12.5, \log_2 20, 0) \\w_{d_2} &= \left(\frac{1}{2} \log_2 10, \log_2 12.5, \log_2 20, \log_2 \frac{100}{3}\right) \\sim(d_1, d_2) &= 0.736\end{aligned}$$

(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.
 $T = \{\text{Shared, Computer, Resources, Services, Digital, Components}\}$
 $\text{Doc3} = (1, 0, 0, 0, 1, 1)$
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.

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

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

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

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

$$w_{d4} = (\log_2 \frac{4}{3}, \log_2 \frac{4}{3}, 1, 0, 0, 1)$$

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

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

$$\text{sim}(q, d_4) = 0.6534$$

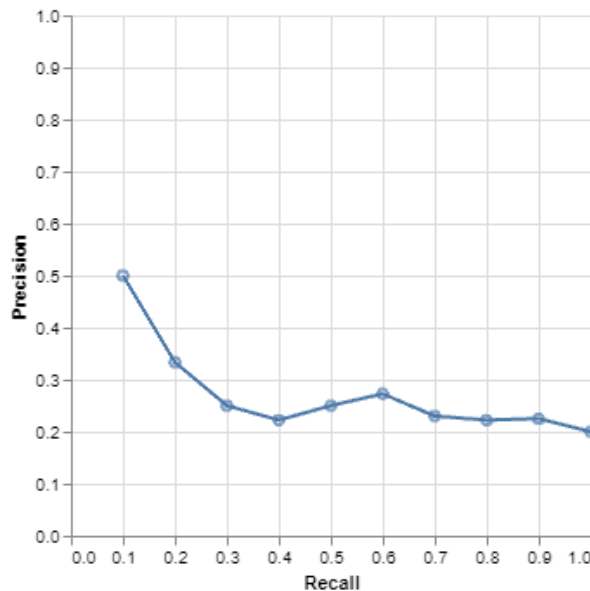
(Answer to 4: I get 0.6534)

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:

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

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

$$\begin{aligned} \alpha = 1/2, F &= \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = \frac{2}{1 + 3} = 0.5 \\ \alpha = 1/4, F &= \frac{1}{\frac{1/4}{\text{recall}} + \frac{3/4}{\text{precision}}} = \frac{1}{1/4 + 9/4} = 0.4 \\ \alpha = 3/4, F &= \frac{1}{\frac{3/4}{\text{recall}} + \frac{1/4}{\text{precision}}} = \frac{1}{3/4 + 3/4} = 2/3 = 0.66 \end{aligned}$$

En el 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 tamaño de la respuesta es de 39 documentos.

$$\begin{aligned} \alpha = 1/2, F &= \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = \frac{2}{4/3 + 39/15} \approx 0.51 \\ \alpha = 1/4, F &= \frac{1}{\frac{1/4}{\text{recall}} + \frac{3/4}{\text{precision}}} = \frac{1}{1/3 + \frac{3 \cdot 39}{4 \cdot 15}} \approx 0.438 \\ \alpha = 3/4, F &= \frac{1}{\frac{3/4}{\text{recall}} + \frac{1/4}{\text{precision}}} = \frac{1}{1 + 39/45} \approx 0.5357 \end{aligned}$$

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

- **Coverage**

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

- **Novelty**

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

- $|\text{relevant \& known}| = 10$, 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.

C.1 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 millones). 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 , sería 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 sería el siguiente.

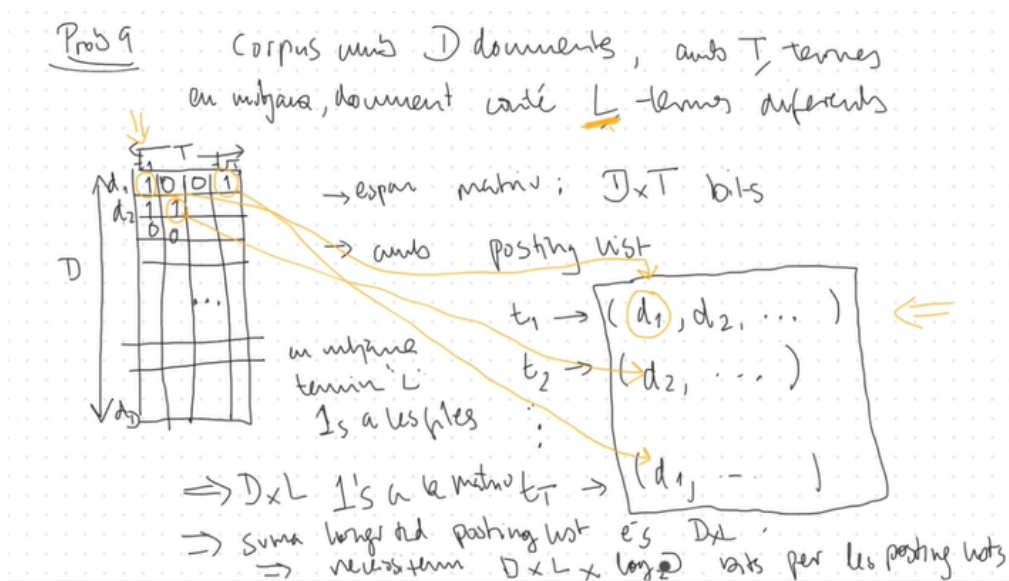
	t_1	t_2	t_3	\dots	t_T
d_1	1	0	1	\dots	0
d_2	0	1	0	\dots	1
d_3	1	0	0	\dots	1
d_4	0	1	1	\dots	0
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots
d_D	1	0	1	\dots	1

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 será 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. 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.

- Imagine now that we do want to keep the intradocument frequencies. Estimate the index size if we compress docids 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 $N = \sum_i \text{len}(\text{doc}_i)$.

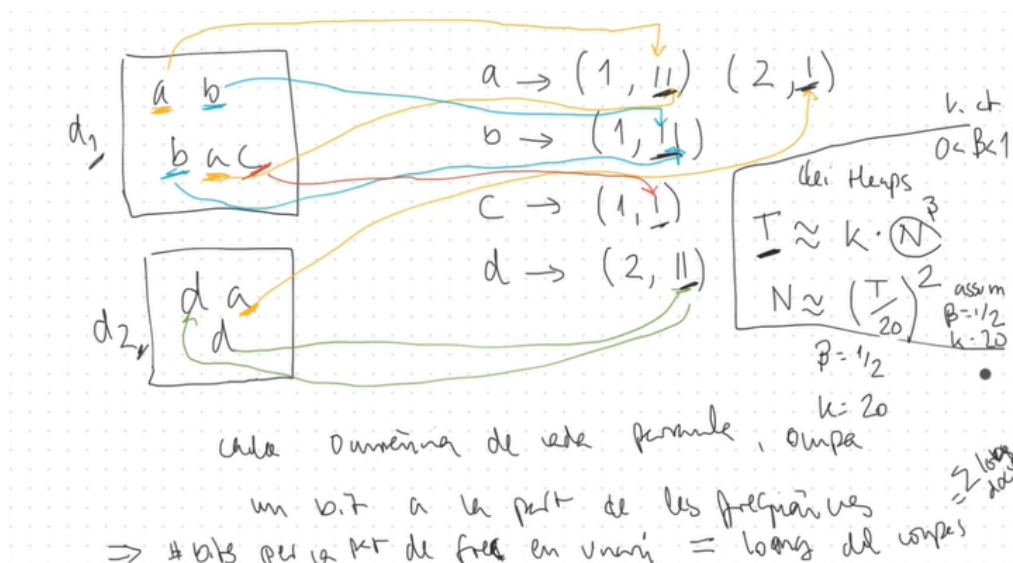
Entonces el número de bits necesarios sería: $L \times D \times 2 \log_2\left(\frac{T}{L}\right) + N$

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 $k = 20$.

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



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

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

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).