


Sistemas de Recuperación de Información

Motor de
búsqueda:
Sherlock 

Autores

Laura Victoria Riera Pérez

Leandro Rodríguez Llosa

Marcos Manuel Tirador del Riego

Resumen A medida que aumenta el poder de cómputo y disminuye el costo de almacenamiento, la cantidad de datos que manejamos día a día crece exponencialmente. Pero sin una forma de recuperar la información y poder consultarla, la información que recopilamos se desperdicia. Con el inmenso crecimiento del internet, se convierte en un reto cada vez mayor el manejo de la información, su recuperación y la extracción de conocimiento de ella. Por esto, es de especial interés la creación de algoritmos que ayuden a su manipulación. En el presente trabajo se implementó un sistema de recuperación de información mediante el motor de búsqueda "Sherlock". Se utilizaron tres colecciones de textos y se implementaron tres modelos de recuperación de información: el booleano y el vectorial clásicos, y el fuzzy. Se realizó una evaluación de los resultados obtenidos en cada uno de los modelos comparándolos con los de las colecciones de prueba. Además se añadieron funcionalidades de retroalimentación, expansión de consultas y agrupamiento.

Palabras clave — recuperación de información (RI) · motor de búsqueda · modelo vectorial · modelo booleano · modelo fuzzy · retroalimentación de Rocchio · agrupamiento

Abstract. As computing power increases and the cost of storage decreases, the amount of day-to-day data we deal with is growing exponentially. However, without a way to retrieve the information and be able to query it, the data we collect most likely goes to waste. With the immense growth of the Internet, managing information, retrieving it, and extracting knowledge from it becomes an ever-increasing challenge. Therefore, it is of special interest the creation of algorithms that help its manipulation. In the present work, an information retrieval system was implemented through the "Sherlock" search engine. Three collections of texts were used and three information retrieval models were implemented: the classic Boolean and Vector models, and the Fuzzy model. An evaluation of the results obtained in each of the models was carried out, comparing them with those of the test collections. In addition, feedback, query expansion, and clustering features were added.

Keywords — information retrieval (IR) · search engine · vector model · boolean model · fuzzy model · Rocchio feedback · clustering

Índice general

1	Introducción	1
2	Diseño del sistema	1
2.1	Documentos	1
2.2	Normalización de un término	2
2.3	Corpus	2
2.4	Modelo base	2
3	Modelo Booleano	2
3.1	Descripción del modelo	3
3.2	Implementación	3
4	Modelo Vectorial	3
4.1	Implementación	5
4.2	Retroalimentación	6
5	Modelo Fuzzy	6
5.1	Descripción del modelo usado	7
5.2	Implementación	8
6	Evaluación de los modelos	8
6.1	Modelo Booleano	9
6.2	Modelo Vectorial	10
6.3	Modelo Fuzzy	11
7	Agrupamiento	12
7.1	K-means	13
7.2	Objetivo perseguido	13
7.3	Implementación	13
7.4	Resultados	14
8	Conclusiones y trabajo futuro	15

1 Introducción

Un sistema de información es “un conjunto de componentes interrelacionados que permiten capturar, procesar, almacenar y distribuir la información para apoyar la toma de decisiones y el control en una organización”.¹ Por otro lado, la recuperación de información se ocupa de la localización de materiales de naturaleza no estructurada en grandes repositorios de datos (generalmente documentos de texto). Se cree que la recuperación de información es la forma dominante de acceso a la información.

Actualmente, cientos de millones de personas utilizan la recuperación de información todos los días cuando usan motores de búsqueda web. El sistema de recuperación de información ayuda a los usuarios a encontrar la información que necesitan, notificando sobre la existencia y ubicación de documentos que pudieran constar de la información requerida.

Un modelo de Recuperación de Información selecciona y clasifica el documento que el usuario ha solicitado en forma de consulta. Los documentos y las consultas se representan de manera similar, de modo que la selección y clasificación de documentos se puede formalizar mediante una función de coincidencia que devuelve un valor de similitud para cada documento de la colección. Muchos de los sistemas de Recuperación de Información representan el contenido de los documentos mediante un conjunto de términos, pertenecientes a un vocabulario.

Los sistemas de recuperación de información son muy importantes para dar sentido a los datos. La información no es conocimiento sin sistemas de recuperación de información.

En el presente trabajo se implementó un sistema de recuperación de información mediante el motor de búsqueda web “Sherlock”. Se utilizaron tres colecciones de textos y se implementaron tres modelos de recuperación de información: el booleano y el vectorial clásicos, y el modelo difuso o fuzzy. Se realizó una evaluación de los resultados obtenidos en cada uno de los modelos comparándolos con los de las colecciones de prueba. Además se añadieron funcionalidades de retroalimentación, expansión de consultas y agrupamiento.

2 Diseño del sistema

2.1 Documentos

Para el desarrollo de este Sistema de Recuperación de la Información modelamos un documento como un objeto que tiene dos propiedades, un `doc_id` que identifica de manera única a un documento dentro del conjunto de documentos del dataset en cuestión; y un diccionario que posee el conjunto de términos indexados contra la frecuencia de los mismos en el documento.

Como el texto de un documento es incómodo de manipular por venir en forma de `string`, este se tokeniza y convierte en una lista de términos indexados

¹ C. Fleitas y M. Sánchez, Conferencia 1 de Sistemas de la Recuperación de la Información, Curso 2022, p. 12

normalizados. Esto se logra haciendo uso del paquete de Python `re` (referirse a [4]) que proporciona una colección de funciones que facilitan el trabajo con expresiones regulares.

2.2 Normalización de un término

Es importante destacar algunas asunciones que se tuvieron en cuenta en el proceso de tokenización. No se considera relevante la diferenciación entre una letra mayúscula y una minúscula, ya que, en la mayoría de los casos, el significado semántico que expresan es el mismo. Para reducir algunos errores ortográficos, y que esto no perjudique la recuperación de un documento importante, se considera que las tildes no diferencian una palabra de otra. Se conoce que esto último no es cierto en el español, pero por el momento se está trabajando con textos en inglés.

Resumiendo, se trata de llevar todos los términos a cadenas de caracteres que contienen letras minúsculas o números.

2.3 Corpus

Un corpus no es más que el conjunto de documentos de un dataset, tokenizados y normalizados.

2.4 Modelo base

Se define un modelo base como un concepto abstracto, que engloba los comportamientos comunes que debe tener cada modelo de recuperación de información.

Cada instancia de un modelo tiene un corpus asociado a este, sobre el cual se hacen las búsquedas. Se deja en manos del programador qué preprocesamientos hacer con el corpus, en dependencia del modelo que se esté implementando, cómo guardar y cargar estos preprocesamientos. Esto último se logra con el paquete de Python `dictdatabase` ([?]), que permite un manejo fácil con *JSON*.

3 Modelo Booleano

El modelo booleano clásico expuesto en [2, Sección 2.2.5] es un modelo simple de Recuperación de Información basado en teoría de conjuntos y álgebra booleana. Las consultas son expresiones booleanas que utilizan los operadores lógicos AND, OR y NOT, y sólo se recuperan los documentos que tengan coincidencias exactas a las mismas. Es un modelo eficiente y formal, de fácil comprensión e implementación, recomendado para el trabajo con expertos sobre un tema específico.

3.1 Descripción del modelo

Para el modelo booleano, los pesos de los términos indexados son binarios, es decir, $w_{ij} \in \{0, 1\}$.

Definición 1 [2] Sea q una consulta (expresión booleana convencional), q_{fnd} la forma normal disyuntiva de la consulta q . Además, sea q_{cc} cualquiera de los componentes conjuntivos de q_{fnd} . La similitud de un documento d_j con la consulta q se define como:

$$sim(d_j, q) = \begin{cases} 1, & \text{si } \exists q_{cc} (q_{cc} \in q_{fnd}) \wedge (\forall k_i, g_i(d_j) = g_i(q_{cc})) \\ 0, & \text{en otro caso} \end{cases} \quad (1)$$

Si $sim(d_j, q) = 1$, entonces el modelo booleano predice que el documento d_j es relevante a la consulta q (podría no ser). De lo contrario, la predicción es que el documento no es relevante y este no es devuelto.

3.2 Implementación

La consulta debe ser una expresión booleana convencional, donde se utilizan los símbolos $\&$, $|$ y \sim para representar las operaciones AND, OR y NOT respectivamente. En caso de que no aparezca ningún operador se asume que se desea que todas las palabras aparezcan en el documento y, por tanto, se toman con AND entre ellas.

Para procesar la consulta se eliminan de la misma todos los caracteres innecesarios, quedando solo letras, los símbolos de los operadores y paréntesis (para poder agrupar términos). Para convertir la consulta de string a expresión y hallar entonces su forma normal disyuntiva se utilizan los métodos `sympify` y `to_dfn` de la biblioteca `sympy`. Finalmente, se tendrá una lista en donde en cada posición se tiene otra lista con todos los términos de una componente conjuntiva.

Para hallar las coincidencias a documentos simplemente se recorre cada componente conjuntiva y se añaden a la respuesta los documentos que contengan a todos los términos de la misma con un score igual a 1.

4 Modelo Vectorial

En el modelo vectorial cada documento se representa como un vector de términos indexados, donde lo único que interesa saber es qué términos aparecen en el documento y qué pesos tiene cada uno de esos términos en el documento. Luego dada una consulta, que también se representa como un documento, hallar el conjunto de documentos relevantes se reduce a encontrar el conjunto de vectores documento más similares a la consulta.

Para entender mejor estas ideas, se necesitan conocer que es $TF * IDF$.

TF La frecuencia de un término en un documento se halla calculando la cantidad de ocurrencias del término en el documento dividido por la máxima cantidad de ocurrencias que obtiene algún término en ese documento. Esta idea de dividir por la máxima cantidad de ocurrencias pretende normalizar los valores de frecuencia para evitar que documentos muy extensos obtengan mayores valores de frecuencia que otros documentos más pequeños.

$$TF_{t_i, d_j} = \frac{freq(t_i, d_j)}{\max freq(t_k, d_j)} \quad (2)$$

t_i - i-ésimo término

d_j - j-ésimo documento

$freq(t_i, d_j)$ - cantidad de ocurrencias de t_i en d_j

Con TF solamente no basta para saber la relevancia que tiene un término en un documento porque las palabras muy comunes en un texto, como por ejemplo las preposiciones, aparecerían con un TF alto en todos los documentos, por tanto se les daría mayor relevancia a estas palabras, que no tienen mucha importancia, con a otras que quizás son palabras claves para la consulta. Idealmente se quisiera penalizar este tipo de palabras tan frecuentes en todos los textos, ese es el objetivo de *IDF*.

IDF La frecuencia inversa de un documento calcula la proporción de documentos en los que está un término contra la cantidad total de documentos. Luego se halla el inverso para obtener mayores valores mientras en menos documentos esté el término, bajo el supuesto de que mientras en menos documentos esté, mayor rareza tiene y por tanto más importante debe ser ese término si aparece en la consulta.

$$IDF_{t_i} = \log \frac{N}{n_i} \quad (3)$$

t_i - i-ésimo término

N - cantidad total de documentos

n_i - cantidad de documentos en los que aparece t_i

Teniendo estas dos medidas es posible formular el peso de un término en un documento, dado que se conoce la frecuencia de cada término en un documento y la frecuencia inversa de un término dentro del conjunto de documentos.

$$W_{t_i, d_j} = TF_{t_i, d_j} * IDF_{t_i} \quad (4)$$

En el caso del vector consulta, al calculo de los pesos del término se le adiciona una constante de suavizado para amortiguar la variación en los pesos de términos que ocurren poco y evitar grandes saltos entre la frecuencia de un término que aparece una vez y otro que aparece dos veces.

$$W_{t_i, q} = (a + (1 - a) * TF_{t_i, q}) * IDF_{t_i} \quad (5)$$

Por tanto si se quisiera hacer una consulta, idealmente se quiere recuperar documentos que tenga términos de la consulta que sean poco comunes dentro del conjunto de documentos, y a la vez tengan una frecuencia alta dentro de los documentos recuperados. Ahora la pregunta interesante sería cómo encontrar esos documentos que cumplen esto y cómo ordenarlos por su nivel de relevancia.

Similitud del coseno Para calcular la correlación entre el vector consulta y un vector documento se calcula el coseno del ángulo entre estos dos vectores, obteniendo valores cercanos a 1 mientras más cercanos sean, y cercanos a -1 mientras más distintos sean. Con esto se logra obtener una función de ranking, que le da mayor peso a documentos que poseen términos poco comunes dentro del conjunto de documentos y una alta frecuencia dentro del documento.

$$\text{sim}(\mathbf{d}_j, \mathbf{q}) = \frac{\mathbf{d}_j * \mathbf{q}}{\|\mathbf{d}_j\| * \|\mathbf{q}\|} \quad (6)$$

\mathbf{d}_j - vector del j-ésimo documento
 \mathbf{q} - vector consulta

Para encontrar una explicación más detallada de como funcionan estas ideas, se sugiere leer [1, epígrafe 6].

4.1 Implementación

Preprocesamiento Según la fórmula de similitud del coseno ([1, Ecuación (6.10)]), se puede notar que el aporte de un documento a la fórmula se mantiene invariante para todas las consultas, ya que este solo depende del vector que representa al documento, el cual es independiente de la consulta. Por tanto, como el corpus sobre el cuál se va a recuperar información se mantendrá estático, se calcula el peso de cada término en cada documento para así poder utilizarlo en cada consulta que se realice.

Para calcular el peso de cada término en cada documento según [1, Ecuación (2.3)], primero se necesita calcular las frecuencias normalizadas de los términos en cada documento (TF [2, Ecuación (2.1)]), y la frecuencia de ocurrencia de cada término dentro de todos los documentos del corpus (IDF [2, Ecuación(2.2)]).

Recuperación de documentos La primera fase es similar a la del preprocesamiento de los documentos del corpus. Lo primero que se hace es tokenizar y normalizar la consulta. Luego se hallan los TFs, y se calcula el peso de cada término en la consulta. Para el cálculo de los pesos de cada término se utiliza la medida de suavizado $\alpha = 0.4$ para amortizar la contribución de la frecuencia del término (ver [2, ecuación (2.4)]). Por último se halla la cercanía entre el vector consulta y cada vector documento, utilizando la similitud del coseno entre los vectores según [1, Ecuación 6.10], y se hace un ranking teniendo este valor calculado.

4.2 Retroalimentación

Con el objetivo de mejorar los resultados del sistema es bueno involucrar al usuario en el proceso de recuperación de información. Para ello el usuario debe emitir una clasificación de algunos documentos recuperados, en relevantes o no, dada la consulta realizada, y de esta forma poder descartar o incluir a otros documentos. El algoritmo utilizado para esto es el de Rocchio, a continuación se explicará brevemente.

Algoritmo de Rocchio Este consiste en acercar el vector que representa a la consulta con aquellos que representan a los documentos clasificados como relevantes por el usuario y alejarlo de los no relevantes. Así mediante este proceso se va modificando el vector de la consulta inicial, hasta encontrar un vector consulta que satisfaga las necesidades del usuario. El vector q_m de la consulta se transforma en el vector:

$$q_m = \alpha * q_0 + \frac{\beta}{\|D_r\|} * \sum_{d_i \in D_r} d_i - \frac{\gamma}{\|D_{nr}\|} * \sum_{d_j \in D_{nr}} d_j \quad (7)$$

q_m - vector consulta modificado

q_0 - vector consulta inicial

D_r - conjunto de documentos relevantes

D_{nr} - conjunto de documentos no relevantes

α - peso asignado al vector consulta inicial

β - peso asignado al conjunto de documentos clasificados como relevantes

γ - peso asignado al conjunto de documentos clasificados como no relevantes

Para obtener más información sobre este algoritmo se propone consultar a [1, epígrafe 9.1.1].

5 Modelo Fuzzy

En el modelo booleano se representan las consultas y documentos como conjuntos de palabras. Al determinar que un documento es relevante a una consulta si y solo si tiene una coincidencia exacta de las palabras en la consulta, solamente nos acercamos parcialmente al contenido semántico real de la información. Una alternativa sería definir un conjunto difuso por cada palabra de la consulta y determinar un grado de pertenencia de cada documento a este conjunto, para luego basado en esto, poder asignar un grado de relevancia de cada documento a la consulta dada. Esta es la idea básica detrás de distintos modelos de recuperación de la información basados en conjuntos difusos.

Se presentan a continuación algunos conceptos de la teoría de conjuntos difusos necesarios para entender el modelo implementado que se describe en esta sección.

Definición 2 ([2, Section 2.6.1]) *Un conjunto difuso A de un universo de discurso U está caracterizado por una función de membresía $\mu_A : U \rightarrow [0, 1]$ que asocia a cada elemento $u \in U$ un número $\mu_A(u)$ en el intervalo $[0, 1]$.*

Las tres operaciones más usadas de conjuntos difusos se definen a continuación.

Definición 3 ([2, Section 2.6.1]) *Sea U el universo de discurso, A y B dos conjuntos difusos de U y \bar{A} el complemento de A en U . Sea además un elemento $u \in U$. Entonces,*

$$\begin{aligned}\mu_{\bar{A}}(u) &= 1 - \mu_A(u) \\ \mu_{A \cup B}(u) &= \max(\mu_A(u), \mu_B(u)) \\ \mu_{A \cap B}(u) &= \min(\mu_A(u), \mu_B(u)).\end{aligned}$$

5.1 Descripción del modelo usado

El modelo fuzzy que se seleccionó para su implementación fue propuesto por Ogawa, Morita y Kobayashi en [5]. En este los autores se basan en el uso de la relación entre términos para expandir los términos de las consultas, de forma que se encuentren más documentos que sean relevantes a las necesidades del usuario. Para expandir las consultas se opta por el uso de la matriz de conexión de términos clave (keyword connection matrix). En esta matriz se encuentra la correlación normalizada entre cualesquiera dos términos k_i y k_j definida como

$$c_{i,j} = \frac{n_{i,j}}{n_i + n_j - n_{i,j}},$$

donde n_i (respectivamente n_j) es el número de documentos que contienen a k_i (respectivamente k_j) y $n_{i,j}$ es el número de documentos que los contienen a ambos. Esta correlación se basa en la idea de que si dos palabras están relacionadas aparecerán, con frecuencia, juntas en un mismo documento.

Se define entonces la pertenencia de un documento d_j al conjunto difuso relativo al término k_i como

$$\mu_{i,j} = 1 - \prod_{k_l \in d_j} (1 - c_{i,l}).$$

Como se puede ver, luego de expandir el producto de la derecha, esta fórmula lo que computa es cierta suma algebraica de la correlación entre k_l y todos los términos del documento d_j . Se observa que esta expresión no es exactamente la suma directa de los $c_{i,l}$ para cada k_l , sino que es una suma suavizada. La expresión resultante de expandir es similar a un principio de inclusión exclusión. Esto da la idea de que la relación entre d_j y k_i no es la suma de las correlaciones entre cada $k_l \in d_j$ y k_i , sino que tiene en cuenta la relación conjunta con k_i de los subconjuntos de términos de d_j , ya que puede que varios k_l diferentes estén muy relacionados entre sí, y relacionados con k_i , y su aporte a la suma este siendo sobrevalorado.

Aun no usándose la suma directa, se sigue cumpliendo que cuando un término $k_l \in d_j$ está muy relacionado con k_i (es decir $c_{i,l} \approx 1$), entonces $\mu_{i,j} \approx 1$, lo cual dice que d_j tiene un grado de pertenencia alto al conjunto difuso de k_i . Lo opuesto sucede cuando todos los términos $k_l \in d_j$ están vagamente relacionados con k_i , en cuyo caso $c_{i,l} \approx 0$. Por tanto se mantiene el objetivo de la relación, aun cuando la suma usada no es la usual.

Las consultas del usuario en este modelo vienen dadas de la misma forma que en el modelo booleano. Estamos hablando de una expresión lógica que se convierte luego a forma normal disyuntiva. Supóngase entonces que una consulta q se descompone en las n componentes conjuntivas cc_1, cc_2, \dots, cc_n . Sea $\mu_{cc_t,j}$ el grado de pertenencia del documento d_j al conjunto de documentos relevantes para la forma normal disyuntiva cc_t . Este se calcula como sigue

$$\mu_{cc_t,j} = \prod_{k_i \in cc_t} \mu'_{i,j},$$

donde $\mu'_{i,j} = 1 - \mu_{i,j}$ si k_i aparece negado en cc_t y $\mu'_{i,j} = \mu_{i,j}$ en otro caso. Entonces si se denota $\mu_{q,j}$ como el grado de pertenencia del documento d_j al conjunto de documentos relevantes a la consulta q , este se podría calcular como

$$\mu_{q,j} = 1 - \prod_{t=1}^n (1 - \mu_{cc_t,j}).$$

Según la Definición 3, el valor $\mu_{cc_t,j}$ (el grado de pertenencia a una conjunto difuso conjuntivo) y $\mu_{q,j}$ (el grado de pertenencia a un conjunto difuso disyuntivo) deberían ser calculados tomando el mínimo y el máximo de las variables que intervienen respectivamente. Sin embargo, se opta en este caso por usar producto y suma algebraica, respectivamente, para suavizar los resultados.

5.2 Implementación.

Para implementar el modelo descrito los autores del presente se basaron en el procesamiento de los documentos y consultas que se expusieron en el modelo booleano, ya que estas adoptan la misma forma. Se modifica entonces el método que computa el resultado de una consulta, asignando en esta ocasión una puntuación a cada documento d_j respecto a la consulta q , correspondiente a $\mu_{q,j}$. Los documentos serán recuperados estableciendo un orden según este valor.

Para calcular los valores en cada consulta de $\mu_{q,j}$ simplemente se computan las fórmulas descritas antes. La correlación entre términos, sin embargo, se calcula una sola vez para el total de pares de términos que aparecen en todos los documentos, y se guarda en el almacenamiento físico de la computadora. Esto se hace para no tener que recalcular todos estos valores más de una vez, ya que constituye un costo alto en tiempo.

6 Evaluación de los modelos

Se han presentado en este trabajo tres alternativas a modelos usados para implementar un sistema de recuperación de la información. Es importante entonces

contar con una forma de calcular la efectividad de cada uno de estos, de forma que podamos compararlos en sus distintas aplicaciones.

Para medir la efectividad de cualquier modelo se necesita de una colección de prueba, que no es más que un conjunto de documentos sobre el que se puede ejecutar el motor de búsqueda con el modelo en cuestión. Esta colección debe contar con un conjunto de necesidades de información, expresadas en forma de consulta, así como con una evaluación de cada documento de la colección según su relevancia respecto a cada consulta.² En el sistema implementado se usaron los siguientes tres colecciones de prueba: *Cranfield*, *Vaswani* y *Cord19/trec-covid/round1* presentes en la biblioteca de python `ir_datasets`.

Para evaluar los modelos se dividieron los grados de relevancia de los documentos en dos grupos, relevantes y no relevantes. Idealmente los conjuntos de prueba deberían contar por cada necesidad de información con una evaluación de relevancia para cada documento. Sin embargo, esto no sucede para las colecciones disponibles por lo que aquellos documentos que no han sido evaluados se considerarán como irrelevantes.

Las medidas que se usaron para evaluar los modelos fueron precisión (P), recobrado (R) y F_1 . Estas se definen como sigue

$$P = \frac{RR}{RR + RI},$$

$$R = \frac{RR}{RR + NR},$$

$$F_1 = \frac{2}{1/P + 1/R},$$

donde RR es la cantidad de documentos relevantes que fueron correctamente recuperados, RI los recuperados que no eran relevantes y NR los no recuperados que eran relevantes. Estas medidas se evalúan por cada consulta disponible en la colección y luego se procede a promediarlas para el total de consultas.

En el caso de los modelos por ranking, como el vectorial y el fuzzy, se procede a calcular los valores anteriormente mencionados tomando como recuperados los primeros k documentos del ranking. Entonces se procede a graficar la relación de precisión-recobrado para cada uno de estos valores de k .

6.1 Modelo Booleano

El modelo booleano puede ser complejo para usuarios inexpertos, debido a la dificultad de expresar las necesidades de información en la consulta como una expresión lógica. No ofrece un ranking, por lo que potencialmente los documentos mostrados al inicio no sean los de mayor interés para el usuario. No establece correlación entre términos por lo que puede devolver documentos que cumplan con la consulta, pero que no sean los más precisos para esta. Además, al todos los términos son igual de importantes, cuando en la práctica no es así, y solo

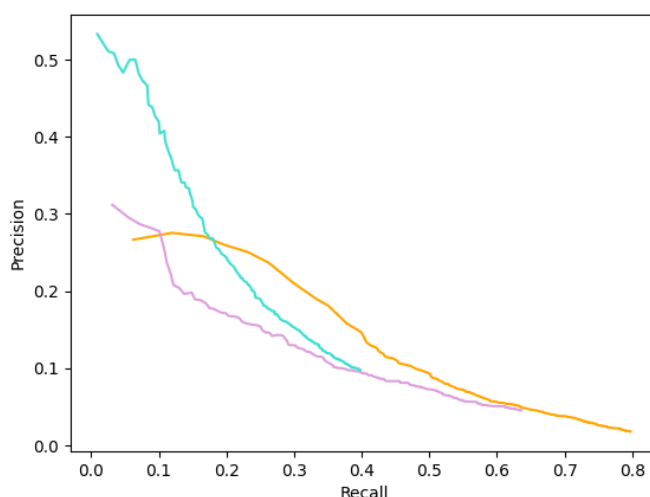
² Idea tomada de [1, Sección 8.1]

permitir coincidencias exactas, puede que no se devuelvan documentos en donde si bien, por ejemplo, falta un término, estos sean relevantes a la información requerida.

Las conjuntos de consultas de prueba tanto de *Cranfield*, como de *Vaswani*, como de *Cord19* no están pensadas para aplicarlas sobre un modelo booleano. Dado que las consultas evaluadas poseen forma oracional, si tan solo un término no aparece en el documento, el mismo no será devuelto. Es por esto que tanto precisión como recobrado en este modelo son prácticamente cero.

6.2 Modelo Vectorial

El modelo vectorial no se comporta bien en corpus de dominio específico. Tanto *Cranfield*, como *Vaswani* y *Cord19* son corpus de dominio específico, por lo que las evaluaciones no son muy acertadas.

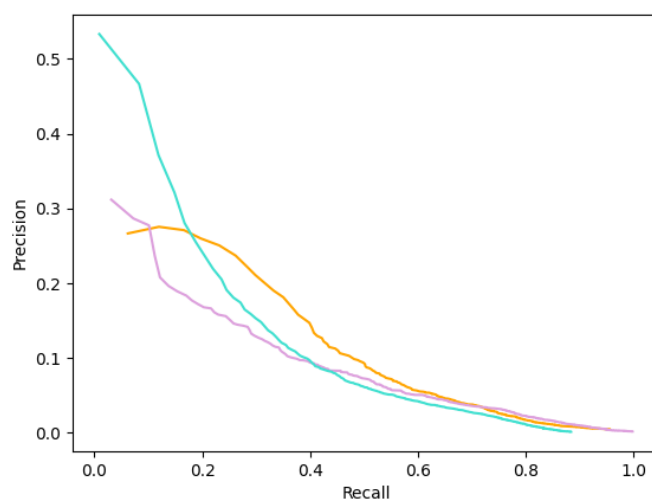


En esta gráfica se muestra hasta los primeros 300 documentos, como se comporta el modelo con los 3 corpus.

Cord19 comienza con una precisión bastante alta, puesto que es un corpus grande, y el modelo puede distinguir mejor entre palabras que sean comunes en muchos documentos y aquellas que no lo son. Esto como bien sabemos es una ventaja y desventaja a la vez, puesto que como tiene muchos documentos, deben haber una mayor cantidad de documentos relevantes y en las primeras iteraciones se logra recuperar menos, por lo que el recobrado da muy bajo. Esto a su vez afecta bastante a la medida F1. Pero a medida que va disminuyendo la precisión y aumentando el recobrado se logra un mejor equilibrio de ambos para $k = 71$, con $F1 = 0,2196$, $P = 0,2192$ y $R = 0,2201$.

Por otra parte, *Vaswani* y *Cranfield* se comportan bastante parecidos. La diferencia radica en que la precisión de *Vaswani* baja con mayor rapidez que la de *Cranfield* ya que es un corpus más grande y para subir el recobrado necesita de mayor cantidad de documentos a recolectar, y con este aumento de la cantidad de documentos disminuye la precisión porque este modelo como no tiene en cuenta la correlación entre los términos y en corpus de dominio específico esto influye bastante.

Los mejores resultados para *Cranfield* se obtienen en $k = 8$ con $F1 = 0,2472$, $P = 0,2111$ y $R = 0,2982$; y para *Vaswani* en $k = 29$ con $F1 = 0,1902$, $P = 0,1561$ y $R = 0,2435$.



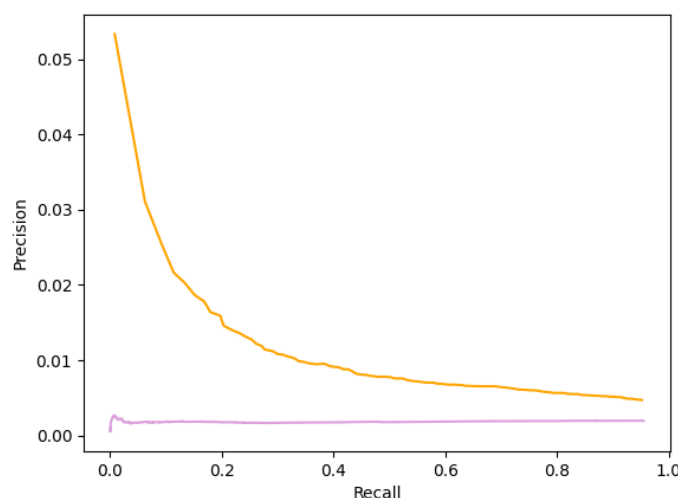
6.3 Modelo Fuzzy

Al ser un modelo que extiende al modelo booleano, aunque resuelve muchas de las dificultades del anterior, este aun acarrea otras de sus deficiencias. Una de estas principales dificultades es que el lenguaje de consultas es complejo para inexpertos.

Sin embargo, si se puede apreciar una ligera mejoría respecto al modelo booleano. Este resultado es de esperar ya que este modelo mejora algunas de las dificultades del anterior, como por ejemplo que en este caso la coincidencia de los documentos no tiene que ser exacta, se crea un ranking, se adiciona cierta semántica a los términos analizando cierta correlación entre los pares de ellos y además, de esto último se puede inferir que no todos los términos seguirán siendo igual de importantes.

En el caso del conjunto de datos *cord19* este no pudo usarse en el modelo fuzzy ya que la gran cantidad de documentos y términos que posee este es muy grande, lo que hace que calcular la correlación entre cada par de términos incurra

en un uso de recursos muy elevado. Si dicha correlación se precalcula, harían falta más de 20GB de RAM según los cálculos hechos (que no se presentan pues son aproximaciones poco formales). Si en cambio se hace en el momento de cada consulta solo para los términos de la consulta, ejecutar cada consulta tomaría más de 5 minutos. Además, el modelo fuzzy no ha sido extensamente probado en experimentos con colecciones grandes de documentos³.



7 Agrupamiento

Los algoritmos de agrupamiento, como su nombre lo indica, agrupan un conjunto de documentos en subconjuntos o clústeres. Los grupos formados deben tener un alto grado de asociación entre los documentos de un mismo grupo, es decir, deben ser lo más similares posibles, y un bajo grado entre miembros de diferentes grupos. Es la forma más común de *aprendizaje no supervisado*, es decir no hay ningún experto humano que haya asignado documentos a clases. Es la distribución y composición de los datos lo que determinará la pertenencia al clúster.⁴

Se presenta a continuación la hipótesis en que se basan los algoritmos de agrupamiento, que fue tomada de [1, Sección 16.1].

Hipótesis de agrupamiento: *Los documentos en el mismo grupo se comportan de manera similar con respecto a la relevancia para las necesidades de información.*

³ Esta idea se tomó de [2], página 38.

⁴ Las ideas de esta introducción a la sección fueron tomadas de [6] de los propios autores de este informe.

La hipótesis establece que si hay un documento de un grupo que es relevante a una solicitud de búsqueda, entonces es probable que otros documentos del mismo clúster también sean relevantes.

7.1 K-means

En este trabajo se optó por usar uno de los algoritmos de agrupamiento más importantes, conocido como K-means. Este algoritmo se ejecuta sobre un conjunto espacio de documentos representados como vectores dimensionales. El mismo fija inicialmente de forma aleatoria los centroides de los clústeres y en cada iteración los va moviendo de forma que se disminuya en cada paso la suma de los cuadrados de las distancias de cada documento a su clúster más cercano (RSS).

Se define el centroide de un clúster formalmente como

$$\vec{\mu}(w_k) = \frac{1}{|w|} \sum_{x \in w_k} \vec{x},$$

donde w_k es el clúster y \vec{x} es un vector que representa a un documento que pertenece a w . Se define RSS como

$$RSS = \sum_{i=1}^K \sum_{\vec{x} \in w_k} |\vec{x} - \vec{\mu}(w_k)|^2,$$

donde K es el número de clústeres.

7.2 Objetivo perseguido

Se decidió usar este algoritmo de agrupamiento para mejorar la recuperación de documentos en el modelo vectorial. La idea perseguida es que según la hipótesis de agrupamiento se espera que documentos en el clúster más cercano al vector de la consulta debe contener con mayor probabilidad documentos similares a la consulta. Entonces reordenamos y reasignamos puntuaciones de relevancia a los documentos, de forma que sean más relevantes aquellos que pertenezcan a clústeres más cercanos a la consulta, manteniendo el orden relativo que tenían antes aquellos documentos en el mismo clúster. Además, podemos presentar al usuario los documentos agrupados en los clústeres, siendo más fácil escanear algunos grupos coherentes que muchos documentos individuales. Esto es particularmente útil si un término de búsqueda tiene diferentes significados.

7.3 Implementación

Para poner en práctica la idea perseguida, se implementó un nuevo modelo de recuperación de la información que se sustenta en el modelo vectorial antes explicado. Este modifica el método de recuperación de documentos a una consulta

para añadir el criterio por clústeres descrito. Los autores se auxiliaron de la biblioteca de python *sk-learn*.

Una de las decisiones de implementación que se tuvo que tomar fue la elección de un número k de clústeres adecuados. Se implementó primero el conocido método del codo que toma la decisión en función de la gráfica de los valores de RSS para cada elección de k considerada. Sin embargo este método no arrojó ninguna luz en el conjunto de datos de *Cranfield* el cual se usó para implementar este modelo. Entonces se optó por una decisión en función de minimizar los RSS, pero penalizando el número de clústeres usados. La expresión que se utilizó fue la siguiente

$$k = \arg \min_{k \in \mathbb{N}} (RSS_k + \lambda * k),$$

donde λ se tomo como 0,08 multiplicado por la cantidad de documentos del corpus. Según esta expresión un buen número de clústeres (inferior a 10 porque para valores mucho más grandes el algoritmo tomaba mucho tiempo en ejecutarse) es $k = 5$, y este es el utilizado.

Para determinar la nueva puntuación de cada documento respecto a una consulta se toma

$$score_d = \frac{min_distance - 1 + oldscore}{|\vec{q} - \vec{\mu}_d|},$$

donde \vec{q} es el vector consulta, $\vec{\mu}$ es el centroide al que pertenece el documento d y

$$min_distance = \min_d (|\vec{q} - \vec{\mu}_d|).$$

De esta forma siempre obtienen mejor puntuación los documentos de los clústeres más cercanos a la consulta y se mantiene el orden relativo que tenían los documentos de un mismo clúster según el modelo vectorial.

7.4 Resultados

En su aplicación en los conjuntos de datos de *Cranfield* y *Vaswani*, y su evaluación, este modelo tuvo resultados muy similares a los del modelo vectorial. Por tanto, una conclusión a la que arribamos es que la aplicación de K-means en la forma descrita no constituye una mejora del modelo vectorial. Estos resultados son coherentes con la idea intuitiva de que si los documentos recuperados con el nuevo modelo son todos de un mismo clúster, entonces son cercanos entre sí, y por tanto cercanos al vector consulta, por lo que deberían ser recuperados también en el modelo vectorial clásico.

Sin embargo, se tienen otras ideas de como aplicar este algoritmo al mismo modelo, sobre la base del mismo código, que se seguirán probando en busca de mejorar los resultados.

8 Conclusiones y trabajo futuro

Los datos son uno de los recursos más valiosos en la actualidad y son los sistemas de recuperación de información los que nos permiten organizarlos, recuperarlos y extraer conocimiento de los mismos. En este trabajo se implementó un sistema de recuperación de información mediante un motor de búsqueda, con los modelos booleano, vectorial y fuzzy; y tres corpus. Se evaluaron cada uno de estos modelos, siendo el vectorial el de mejor desempeño. Además, se implementó un algoritmo de agrupamiento para mejorar el modelo vectorial y se utilizó también para mostrar los resultados de búsqueda. Además, se implementó la retroalimentación de Rocchio.

Referencias

1. Maning C. D.: *An Introduction To Information Retrieval* (2009).
2. Ricardo Baeza-Yates: *Modern Information Retrieval* (1999).
3. Documentación oficial: <https://ir-datasets.com/index.html>
4. Documentación oficial: <https://docs.python.org/3/library/re.html>
5. Y. Ogawa, T. Morita y K. Kobayashi. A fuzzy document retrieval system using the keyword connection matrix and a learning method. *Fuzzy Sets and Systems*, 39:163-179, 1991.
6. Laura Riera Pérez y Marcos Tirador del Riego. Aplicaciones del agrupamiento y de la clasificación en la recuperación de información en la Web.