Universidad de los Andes
Facultad de ingeniería
Tarea 01

Procesamiento de lenguaje natural

Tarea - 01

Presentado por:
Juan David García Hernández
Nicolás Rocha Pacheco
César Daniel Garrido Urbano

Presentado a:
Rubén Francisco Manrique Pirmanrique

Tabla de contenido

Introducción

De acuerdo con [1] Recuperación de Información, o IR por sus siglas en inglés hace referencia a encontrar material que satisfaga una necesidad de información. Usualmente, este material son documentos de texto y se almacena en computadores. Hasta hace algunos años esta actividad se limitaba a algunas profesiones específicas. No obstante, el *boom* del internet ha ocasionado que la mayoría de estas búsquedas se hagan a través de este, sea mediante motores de búsqueda o correo electrónico.

Para IR se han propuesto diferentes métodos que permiten solucionar el problema desde diferentes ángulos, dentro de los cuales se encuentran búsqueda binaria, el retorna todos los documentos que cumplan cierta condición, como que contenga todos los términos buscados o al menos uno de ellos. Otra técnica es búsqueda clasificada, el cual retorna ordenadamente los documentos considerados relevantes. En el presente documento se explican algunas técnicas implementadas por los autores y una comparación de los resultados obtenidos por cada una de estas.

Dataset

El *dataset* utilizado para el desarrollo de esta comparación consiste de un total de 331 documentos y 35 consultas (*queries*) con sus respectivas etiquetas. Los documentos y las consultas están almacenados en formato `.naf`, por lo que se utiliza una librería de lectura de archivos `.xml` para ello. Por su parte, las etiquetas vienen almacenadas en archivos `.tsv`, los cuales son un documentos con valores separados por 't'. Para la lectura de estos se utiliza la librería *pandas*.

Cada documento tiene un encabeza en donde se encuentra el título asociado al texto contenido, un identificador (*id*) y una dirección de donde fue obtenido. Por último se encuentra el texto correspondiente. Cabe resaltar que en el texto se encuentra incluido el título del documento, razón por la cual no se incluye de manera adicional. Las consultas están organizadas de manera similar, pero sin título y con los términos clave en el lugar donde estaría el texto.

Las etiquetas de cada una de las consultas indican los documentos que resultan relevantes para esa consulta y un puntaje de 2 a 5 clasificando la relevancia del documento.

Preprocesamiento

Antes de implementar las distintas estrategias de recuperación de información, se realizo un preprocesamiento igual a todo el *dataset*. Con este proceso se busca

representar tanto los *queries* como los documentos de forma numérica de manera tal que dicha representación ayude a resolver el problema de IR. Para esto se le aplicaron distintos procesamientos estándar, los cuales se presentan de forma resumida en la figura 1. Todo el código correspondiente a esta sección se encuentra en el cuaderno `HW01_Utilities.ipynb`.

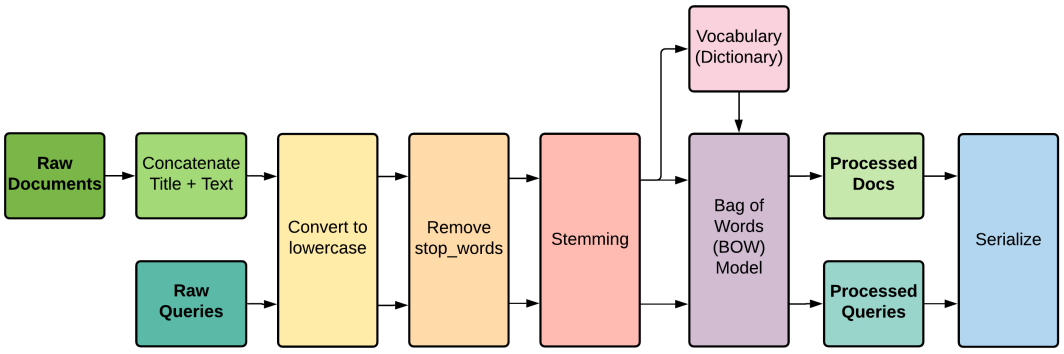

Figura 1: Etapas de procesamiento aplicadas al texto para resolver el problema de IR

A grande rasgos lo que se hace es tomar el texto crudo (para los documentos se concatena el titulo con su cuerpo) y se convierten todos los caracteres a minúscula. Posteriormente, se remueven las *stop words* (palabras comunes para el idioma inglés) y se reducen todas las palabras a su raíz con *stemming*, todo esto con la librería `gensim.parsing` y las funciones de `remove_stopwords` y `PorterStemer`, respectivamente. Vale la pena aclarar, que el proceso de *stemming* no reduce las palabras a su raíz semántica sino con una serie de reglas establecidas para el idioma. Finalmente, con los términos (únicamente de el corpus de documentos) se construye el vocabulario (o *dictionary*) y a partir de este los modelos de Bag of words (BOW), tanto para los documentos como para los *queries*. Por último, estos se serializan para poder utilizarse en los distintos cuadernos donde se implementan las distintas estrategias de recuperación de información.

Organización del contenido

Por facilidad, el desarrollo del presente proyecto se llevó acabo utilizando un repositorio *online* de GitHub, disponible en el en `https://github.com/ISIS4221-JCN/HW01`.

El contenido del repositorio se explica a continuación:

- **doc/:** Carpeta con los archivos necesarios para el informe escrito.
- **resources/:** Carpeta que contiene las instrucciones del proyecto y los archivos relacionados al *vocabulario* y *corpus*.
- **results/:** Carpeta con los archivos resultantes solicitados en cada una de las estrategias e imágenes de comparación.
- **scripts/:** Clase con el código relacionado a las métricas para importarlo con mayor facilidad en los *notebooks*.
- **HW01_*.ipynb:** *Notebooks* con la implementación de cada uno de los puntos.
- **HW01_Results.ipynb:** *Notebook* utilizado para consolidar todos los resultados y obtener gráficas de análisis.
- **HW01_Utilities.ipynb:** *Notebook* que contiene el preprocesamiento descrito en la sección previa.

1. Métricas de evaluación para Recuperación de Información

Antes de realizar una implementación de un sistema de búsqueda de información es necesario definir métricas de evaluación. Las métricas de evaluación van a permitir comparar cuantitativamente los diferentes sistemas e implementaciones. A lo largo de este documento se usarán siete métricas diferentes: precisión, precisión en K, recall en K, precisión-promedio, MAP, DCG y NDCG. A lo largo de esta sección se introducen las métricas, su interpretación y la metodología usada para su cálculo.

1.1. Precisión

La precisión es una métrica que permite evaluar qué porcentaje del resultado de una consulta es relevante para la búsqueda. Teniendo en cuenta que en el problema de la búsqueda de información retorna conjuntos de elementos discretos, la precisión (P) se puede definir como se presenta en la ecuación (1). En dicha ecuación se evalúa la cardinalidad de la intersección del conjunto de elementos retornados por el sistema de búsqueda de información y el conjunto de los documentos que efectivamente son relevantes a la búsqueda. Este valor es normalizado a partir de la cardinalidad del conjunto de documentos retornados.

$$P = \frac{|RET \cap REL|}{|RET|} \tag{1}$$

1.2. Precisión en K

Una alternativa de la precisión es evaluarla sobre un subconjunto de los documentos relevantes y los documentos recuperados. Al aplicar esta métrica lo que se hace es evaluar la precisión considerando una cantidad determinada de documentos. La cantidad de documentos suele ser denominada con la letra K. Esta métrica suele ser usada en conjuntos de datos donde no es factible conocer el conjunto de documentos relevantes en su totalidad. La ecuación 2 presenta cómo se calcula esta métrica.

$$P@K = \frac{|RET \cap REL|}{K} \tag{2}$$

1.3. Recall en K

El recall es una métrica complementaria a la precisión en la cual se calcula el porcentaje de documentos recuperados que son relevantes en relación a los documentos relevantes. De forma similar a la precisión en K, es posible definir un recall sobre un subconjunto de elementos cuya cardinalidad se denomina igualmente con la letra K. No obstante, para el cálculo de esta métrica adicionalmente se tiene en cuenta el número total de documentos relevantes. Es decir, sobre el conjunto de documentos recuperados se extrae un subconjunto con K elementos y sobre dicho subconjunto se evalúa qué porcentaje es relevante en relación al número total de elementos relevantes.

$$R@K = \frac{|RET@K \cup REL|}{|REL|} \tag{3}$$

1.4. Precisión-Promedio

Esta métrica combina la precisión y el recall para una búsqueda de información clasificada por orden. En esta métrica se calcula el promedio de la precisión sobre los documentos recuperados que son relevantes a la búsqueda. Esto permite obtener una noción de qué tan bien clasificados están los documentos recuperados por el sistema de búsqueda de información. Para calcular la precisión-promedio se realizan los siguientes pasos:

1. Se itera sobre cada documento recuperado del conjunto correspondiente.
2. Si el documento del rango K es relevante a la búsqueda, es decir, cuando el recall en ese rango aumenta, se calcula la precisión en dicho K.
3. Una vez se han evaluado todos los documentos relevantes, lo cual implica un recall del 100 % se promedian las precisiones.

1.5. MAP

La media de precisión promedio (MAP, *Mean Average Precision*) es una métrica que permite evaluar el sistema como un global. En otras palabras, mientras que la precisión promedio es una métrica para una búsqueda en particular, la MAP es una métrica para un conjunto de búsquedas. El proceso para calcular esta MAP consiste en calcular la precisión-promedio para varias búsquedas y luego calcular su media aritmética.

1.6. DCG

La ganancia descontada acumulada (DCG, *Discounted Cumulative Gain*) es una métrica que permite evaluar búsquedas de documentos relevantes que son clasificados y categorizados según su relevancia. Esta métrica permite evaluar que los K documentos mejor clasificados son recuperados le sean útiles al usuario que lleva a cabo la búsqueda. La ecuación (**??**) presenta la expresión matemática para calcular esta métrica.

$$DCG@K = \sum_{i=1}^{K} \frac{REL_i}{log_2(max(i,2))} \tag{4}$$

1.7. NDCG

Es una variación de la DCG en la cuál se normaliza el valor sobre el mejor ordenamiento posible de una búsqueda en particular. En otras palabras, esta métrica resulta del cociente entre el DCG calculado en un rango K y el DCG del mejor ordenamiento para dicho rango K. Para obtener el mejor ordenamiento es necesario organizar los documentos según su relevancia de mayor a menor.

2. Estrategias de recuperación de información

A continuación, se explica la implementación de cuatro técnicas diferentes y se comparan sus resultados.

2.1. Búsqueda binaria (BS)

La búsqueda binaria es un algoritmo en que se utilizan operaciones lógicas como AND y OR con el fin de encontrar documentos que contienen los términos solicitados en la consulta, asumiendo que ello indica que son relevantes para dicha necesidad de información. Para llevar a cabo la búsqueda de manera eficiente, se crea una matriz

binaria indicando los términos que contiene cada documento, dando a la técnica el nombre de *búsqueda binaria*. A continuación, se explica detalladamente el proceso que sigue la técnica.

2.1.1. Funcionamiento de BS

Una vez se conoce de cada documento los términos que contiene y su frecuencia, se procede a construir la matriz binaria. Las filas de esta matriz corresponden a los documentos que se consideran en la búsqueda y las filas corresponden a los términos existentes en el diccionario. Siendo así, los registros de cada columna que corresponden a uno indican que ese término existe en el documento indicado por la columna. Cabe resaltar que la frecuencia no es considerada en esta técnica. En la figura 2 se muestra un ejemplo de cómo se vería una matriz binaria.

	Documento 1	Documento 2	Documento 3	Documento 4		Documento n
Término 1	1	1	1	0	...	1
Término 2	0	1	0	1	...	0
Término 3	1	0	1	1	...	1
Término 4	0	0	1	0	...	0
Término 5	1	1	0	1	...	0
Término 6	0	1	1	1	...	1
Término 7	0	1	1	1	...	0
⋮	⋮	⋮	⋮	⋮		⋮
Término m	1	1	0	1	...	0

Figura 2: Matriz binaria ejemplo

Tras recibir la consulta (palabras claves que caracterizan la necesidad de información) se procede a construir un vector del tamaño del vocabulario que indica, al igual que la matriz, qué términos incluye la consulta, generando así un vector binario disperso. En la figura 3 se muestra cómo sería el vector de la consulta.

En el caso específico del *dataset* trabajado la matriz tiene dimensión de 17.365 filas, 331 columnas, dado que se tiene un vocabulario de 17.365 términos.

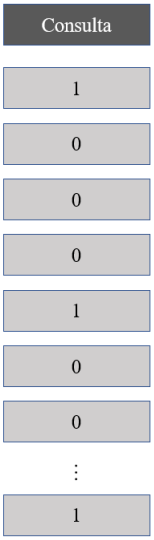

Figura 3: Vector binario de la consulta

Con el fin de almacenar de manera eficiente una matriz de tamaño (5'747.815) se procede a utilizar el tipo de dato que menor espacio de memoria ocupa. Al momento de crear la matriz se indica el tipo de dato a utilizar, el cual corresponde a `numpy.bool_`. Para guardar y abrir la matriz se utilizan las funciones estándar de numpy (`load` y `save`).

Ahora bien, una vez se tiene el vector que representa la consulta y la matriz que almacena toda la información de términos por documento, se utiliza una operación lógica que permita seleccionar los documentos que contienen uno o todos los términos consultados. En el primer caso se utiliza la operación lógica OR, lo cual indica que se extraerán los documentos que tienen al menos uno de los términos buscados; en caso contrario, se utiliza AND para obtener los documentos que tienen todos y cada uno de los términos buscados.

Como es de esperarse es poco común que un documento contenga exactamente todos los términos al igual que es muy probable que algún documento tenga al menos uno de los términos, ocasionando el fenómeno conocido como *feast and famine*. Este indica que las consultas binarias tienen la desventaja de devolver muy pocos o ningún documento (*famine*) en el caso de AND o devolver demasiados documentos (*feast*) al usar OR.

2.1.2. Implementación

En el *notebook* `HW01_2.ipynb` se presenta la implementación de la técnica descrita previamente.

En primer lugar se procede con la construcción de la matriz. Para ello, después de importar los *corpus* tanto de vocabulario como de consultas, se recorre cada uno de los documentos insertando un '1' en los registros correspondientes de la matriz. Dichos registros son las coordenadas de la matriz en donde el número de documento corresponde al que se está recorriendo y el término a aquel encontrado en el documento.

Posteriormente, se realiza un proceso similar para la vectorización de la consulta. Se crea un vector, igualmente binario, del tamaño del vocabulario. En este vector se almacenan unos en cada uno de los términos que aparece en la consulta hecha.

Teniendo esta matriz y el vector de consulta se procede a multiplicar por elemento cada vector columna de la matriz, correspondiente a un documento, con el vector asociado a la consulta. El resultado tendrá '1' en todos aquellos términos que aparezcan tanto en la consulta como en el documento. Es en este punto en donde aplica la operación lógica AND u OR. Si la operación utilizada es conjunción el documento será retornado como relevante en caso de que contenga todas y cada una de las palabras de la consulta. Por otra parte, en caso de que se utilice la disyunción, todos los documentos que contengan uno o más de los términos. En el caso de la operación OR se utiliza la función `any`, en el primer caso con el fin de saber si algún término de los buscados está en el documento. En el caso de la operación AND se revisa que el total de términos encontrados sea igual al total buscados.

La función previamente descrita retorna una lista con los índices de documentos considerados como relevantes. Finalmente, se recorre todo el listado de consultas y se almacenan todos los documentos devueltos por la función para cada consulta. Los resultados obtenidos se analizarán en la sección correspondiente.

2.2. Búsqueda binaria usando índice invertido (BSII)

La búsqueda binaria con índice invertido (BSII, *Binary Search with Inverted Index*) es una evolución de la búsqueda binaria presentada previamente. Como su nombre lo indica, esta aproximación al problema de búsqueda de información utiliza un índice invertido: una estructura de datos donde se relaciona cada término del vocabulario con los documentos donde está presente. En este sentido, cada término va a estar vinculado con lo que se conoce como una lista de posteo (*posting list*)

donde cada ítem de dicha lista es el identificador de un documento donde aparece el término. Con el fin de optimizar el proceso de recuperación de la información se suele incluir la frecuencia relativa de cada término dentro del índice invertido. La figura 4 presenta una representación de un índice invertido, donde se pueden evidenciar los términos, la frecuencia relativa y la lista de posteo.

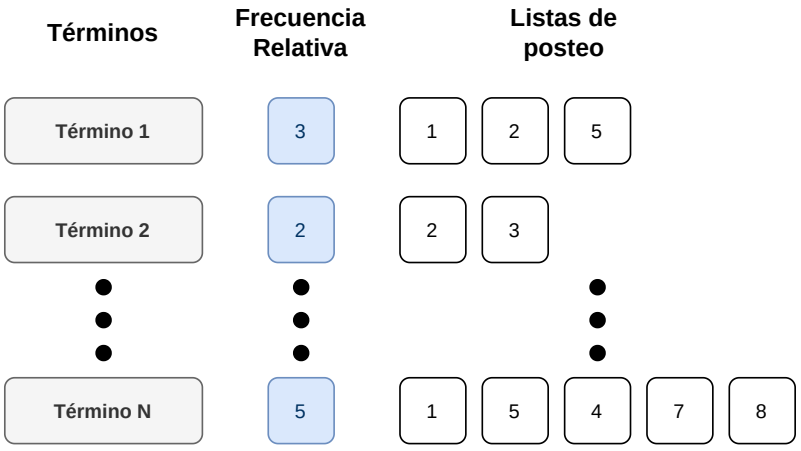

Figura 4: Representación de un índice invertido. Se puede evidenciar una serie de términos al cual se asocia su frecuencia relativa y una lista de posteo con los identificadores de los documentos donde aparecen.

2.2.1. Construcción del Índice Invertido

Según [1] existen cuatro estrategias principales para construir eficientemente un índice invertido de búsqueda binaria (BSII, *Binary Search Inverted Index*. La selección de una es estas estrategias depende de las características del hardware sobre el cuál vaya a ser implementada, especialmente en lo que se refiere al número de máquinas que van a ejecutar el algoritmo.

Vale la pena mencionar que algunos de los valores presentados en [1] incluyen valores y consideraciones propias de sistemas típicos de 2007. Es necesario tener en cuenta que desde 2007 hasta la actualidad (2021) se han presentado avances de hardware que pueden afectar los algoritmos de construcción del índice invertido y búsqueda de información.

Las estrategias para construir el índice invertido son: indexado basado en ordenamiento en bloques (BSBI, *Blocked Sort-Based Indexing*), indexado en memoria de un solo paso (SPIMI, *Single-Pass In-Memory Indexing*), indexado distribuido e indexado dinámico. Tanto el BSBI como el SPIMI están contemplados como estrategias para ser usadas en una máquina, mientras que el indexado distribuido y el indexado dinámico suelen ser usados en grupos de máquinas.

Dado que la solución del enunciado debe ser realizada exclusivamente en una máquina, debido al tamaño de la base de datos y la disponibilidad de recursos, solo serán revisadas las estrategias de BSBI y SPIMI. Las dos estrategias están contempladas para realizar un único paso por los datos para construir el índice.

Indexado Basado en Ordenamiento en Bloques (BSBI) Esta estrategia es utilizada para construir el índice invertido dividiendo la colección en bloques de igual tamaño que son analizadas en memoria. Para cada bloque se realiza el siguiente proceso:

1. Se recupera el siguiente bloque de la colección.
2. El bloque es procesado para extraer parejas de identificador del término e identificador del documento (*termID* - *docID*).
3. Se invierten el bloque para construir un resultado intermedio del índice invertido. El resultado de este proceso es que para un identificador de un término se tendrá una lista con los identificadores de los documentos en los que aparece.
4. Los resultados intermedios para cada bloque son almacenados temporalmente en el almacenamiento secundario.
5. Una vez se han invertido todos los bloques de la colección, los resultados intermedios se unen en lo que será el índice invertido.

La idea de dividir la colección en bloques de igual tamaño tiene el propósito de realizar lecturas contiguas en el almacenamiento secundario. De esta manera, los tiempos de lectura son inferiores a lo que sería una serie lecturas aleatorias en el disco.

Un factor crítico de esta estrategia es el procesamiento del texto para la extracción de las parejas de identificadores de términos y documentos. La estructura de datos que realiza la relación entre término y su identificador podría llegar a ocupar una gran cantidad de espacio de memoria para colecciones lo suficientemente grandes. Esta situación se corrige con la estrategia SPIMI.

Indexado en Memoria de un Solo Paso (SPIMI) La estrategia SPIMI busca construir el índice invertido a partir de los términos directamente. El objetivo de SPIMI consiste en crear un diccionario de forma dinámica conforme se van leyendo las parejas de término e identificador del documento. De forma general, el algoritmo de SPIMI realiza el siguiente proceso:

1. Se crea un archivo de salida y un diccionario. El archivo de salida será usado para almacenar los resultados intermedios. En este caso, el resultado intermedio será el diccionario.

2. Se va iterando sobre cada pareja de término e identificador del documento. Si el término está en el diccionario, se recupera la su entrada y se agrega el nuevo identificador de documento. Se lo contrario, se agrega la entrada en el documento.

3. Cuando la memoria se llena, se organizan los términos del diccionario con un orden lexicográfico y se escriben en el almacenamiento secundario.

4. Una vez se han procesado todas las parejas, se realiza la unión de los diccionarios para obtener el índice invertido.

Una de las ganancias que se obtiene al utilizar la estrategia SPIMI en comparación al BSBI es que no es necesario ordenar las parejas de términos e identificadores de documentos durante la construcción del índice. Otro beneficio consiste en que, como se indicó previamente, SPIMI no requiere de una estructura de datos que asocie un término con su identificador, por lo que la memoria principal puede ser usada eficientemente.

2.2.2. Gestión de Recursos y Acceso a Memoria

Dado que un índice invertido puede ser una estructura que ocupe una gran cantidad de espacio de almacenamiento, es necesario establecer estrategias con el fin de que dicha estructura sea accesible de forma eficiente y que requiera menos almacenamiento. En primer lugar, es necesario prevenir que el índice va a ser guardado en el almacenamiento secundario ya que es probable que ocupe más espacio del provisto por la memoria principal. En este orden de ideas deben solucionarse dos situaciones: el índice debe ser accedido en el almacenamiento secundario de forma rápida y debe ocupar el menor espacio posible.

Para disminuir el tiempo de acceso al índice es posible hacer uso de la jerarquía de memoria y el principio de localidad para cachear (*caching*) las listas de posteo en la memoria principal. De esta manera se reduce el tiempo de acceso al índice para los términos que se usan con mayor frecuencia. Por otra parte, se pueden utilizar algoritmos de compresión y descompresión con el fin de disminuir el espacio de almacenamiento requerido para el índice. El uso de dichos algoritmos tiene dos beneficios: reduce el espacio utilizado en el almacenamiento secundario y permite incluir más términos en la memoria principal.

Es necesario considerar que un índice está compuesto por el vocabulario y las listas de posteo, donde el primero ocupa un espacio de almacenamiento considerablemente menor al segundo. En este contexto, es necesario establecer estrategias

diferentes de compresión dependiendo de si el elemento a comprimir es el vocabulario o las listas de posteo. Para comprimir el diccionario se pude recurrir a codificarlo como una cadena de caracteres donde se incluya el término, su frecuencia relativa y un puntero a su lista de posteo. Esta estrategia de compresión puede se complementada con un agrupamiento de los términos en bloques de un tamaño definido. Al implementar estas dos estrategias de compresión es posible reducir la cantidad de almacenamiento requerido por el diccionario. Sin embargo, el tiempo de acceso para este puede verse afectado, especialmente al implementar el agrupamiento.

El caso de las listas de posteo tiene mayor incidencia en el tamaño del diccionario dado que corresponden a la mayor parte de los datos que deben ser almacenados. En primer lugar, es necesario tener en cuenta que las listas de posteo se pueden definir como el identificador de un documento inicial y una serie de incrementos relativos a este. En este sentido, es posible codificar dichos incrementos como una serie de valores de tamaño dinámico en contraposición a valores de tamaño fijo. De esta manera se puede garantizar que el almacenamiento empleado para codificar los incrementos sean usados eficientemente. Existen dos estrategias para codificar dichos incrementos de forma dinámica: codificar a nivel de bytes y a nivel de bits.

2.2.3. Solución de Consultas al Índice Invertido

Una consulta al índice invertido permite realizar búsquedas binarias usando los operadores lógicos AND, OR y NOT. Por una parte se tiene que dos términos con conjuntivos si se relacionan mediante un operador AND y disjuntos si se relacionan con un operador OR. Con el fin de resolver una consulta al índice invertido es necesario recuperar las listas de posteo de los dos términos y realizar la intersección de las dos listas según el operador lógico usado. Naturalmente, en caso que algún término sea modificado mediante el operador NOT, es necesario establecer el complemento de la lista de posteo de dicho término. Una vez se ha realizado la intersección de las dos listas se puede retornar el resultado o continuar con los demás términos de la consulta.

Un aspecto clave de la consulta a un índice invertido es la intersección entre dos listas de posteo. En [1] se presenta un algoritmo de intersección para dos listas de posteo con un operador de conjunción (AND). Sin embargo, este debe modificarse para que pueda incluir una intersección disjunta (OR) y un operador de modificación (NOT). El listado 1 presenta el algoritmo para realizar la intersección de dos listas de posteo con operadores conjuntivos, disjuntos y de modificación.

```
INTERSECT(p1, p2, op, mod1, mod2, n):
    if mod1 == NOT do
        p1 = complement(p1, n)

    if mod2 == NOT do
        p2 = complement(p2, n)
```

```
if length(p2) < length(p1) do
    p1, p2 = p2, p1

answers = []
indexes = [0, 0]
finish = false

while not finish do
    posting1 = p1[indexes[0]]
    posting2 = p2[indexes[1]]

    if operator == AND do
        if posting1 == posting2 do
            append(answers, posting1)

        if posting1 < posting2 do
            increment(indexes[0])
        else do
            increment(indexes[0])

        if indexes[0] == length(p1) and indexes[1] == length(p2) do
            finish = true

    else do
        if indexes[0] != length(p1) do
            if posting1 < posting2 do
                append(answers, posting1)
                increment(indexes[0])

            else if posting1 == posting2 do
                append(answers, posting1)
                increment(indexes[0])
                increment(indexes[1])

            else do
                append(answers, posting2)
                increment(indexes[1])

        else do
            append(answers, posting2)
            increment(indexes[1])

        if indexes[1] == length(p2) do
            finish = true
```

Listing 1: Algoritmo para la intersección de listas de posteo con operadores conjuntivos, disjuntos y de modificación.

En la ejecución del algoritmo presentado en el listado 1 se realizan los siguientes pasos:

- Se realiza la modificación (`mod1`, `mod2`) de las listas de posteo (`p1` y `p2`) dependiendo del número de documentos presentes en la colección (`n`). Se asume que los identificadores de los documentos hacen parte de un conjunto de enteros que van desde el número uno hasta `n`.

- Se organizan las listas de posteo de modo tal que `p1` apunte a la lista de menor tamaño y `p2` apunte a la lista de mayor tamaño.
- Se crea un arreglo vacío para almacenar las respuestas y otro con dos enteros que almacenan los índices para las listas de posteo.
- Se itera mientras que no se haya alcanzado la condición de parada. En primer lugar se recuperan los elementos de las listas de posteo según los índices almacenados en el arreglo correspondiente. Dependiendo del operador (`op`) se realizan las siguientes acciones dentro del ciclo:
    - Si se trata de una intersección conjuntiva se revisa si los dos elementos son iguales. En caso que sean iguales se agregan al arreglo de respuestas. Se incrementa el índice del elemento que corresponda al menor valor. Cuando los dos índices alcancen la longitud de la lista se activa la condición de parada.
    - Si se trata de una intersección disjunta se revisa si el primer índice no haya alcanzado la longitud de la primera lista de posteo. Si el elemento de la primera lista es inferior al elemento de la segunda, el primer elemento es agregado al arreglo de respuestas y el índice de la primera lista incrementado. En caso tal que los dos elementos sean iguales, uno de estos se agrega a la lista de respuestas y los dos índices se incrementan. Finalmente, si el elemento de la segunda lista es mayor, este es agregado a las respuestas y su índice es aumentado. En caso tal que el primer índice haya alcanzado la longitud de su lista correspondiente, se agregan los elementos restantes de la segunda lista al arreglo de respuestas. Cuando se han agregado todos los elementos de la segunda lista se agrega la condición de parada.

Si una consulta contiene más de dos términos es necesario establecer una estrategia para optimizar la recuperación de las listas de posteo y su intersección. Una de las estrategias con mayor adopción consiste en organizar la intersección de las listas según la frecuencia relativa de los términos. No obstante, esta estrategia de implementación es válida para las intersecciones conjuntivas. Es necesario modificar la estrategia levemente con el fin de incluir tanto las operaciones disjuntas. En este orden de ideas, una estrategia sería resolver las operaciones disjuntas en orden incremental antes de resolver las operaciones disjuntas.

2.2.4. Implementación

Teniendo en cuenta que la implementación se haría en Python y que esta sería ejecutada en una máquina, se podría partir de la estrategia SPIMI para construir el

índice invertido. Esta decisión se toma en base a las siguientes consideraciones:

- SPIMI no requiere un mayor control sobre la memoria principal como sí lo requiere BSBI. Dado que Python no ofrece métodos para realizar este control, dado que es un lenguaje de alto nivel, se favorece la implementación de SPIMI sobre BSBI.
- Python ofrece diccionarios de forma nativa. Estos diccionarios pueden contener como llave (*key*) la cadena de texto del término en cuestión y como valor (*value*) una lista con los identificadores del documento.
- Con la implementación de listas nativas de Python no es necesario alocar memoria directamente desde el algoritmo. Python hace esto sin que el programador deba indicarlo explícitamente.

Aprovechando dicha estrategia se tiene que el conjunto de datos es almacenado enteramente en la memoria, por lo que no es necesario implementar una estrategia de lectura en el almacenamiento secundario. En este sentido, la implementación lee los documentos del conjunto de datos de modo tal que se cree una pareja con el término y el identificador del documento. Inicialmente se tiene un diccionario cuya llave (*key*) es el término y el valor (*value*) corresponde a una lista donde se van agregando los identificadores de los documentos. Una vez se han procesado todos los documentos se convierte dicho diccionario en una matriz de Numpy que es almacenada en disco para su uso posterior.

2.3. Recuperación clasificada básica (RRI)

Las dos técnicas de IR presentadas anteriormente consideran únicamente una selección binaria de documentos (estos son o no son relevantes para el *query*). Esta no suele ser una buena estrategia por lo que es difícil para los usuarios construir *queries* booleanos, se sufre del problema de *feast or famine* (muchos o muy pocos resultados) y cuando son muchos todos los documentos tienen un mismo nivel de relevancia. Con esto en mente, con la técnica de recuperación clasificada (*ranked retrieval*), lo que se desea es no solo encontrar los documentos relevantes para el *query* sino también poder ordenarlos de alguna forma.

Para esta implementación básica de *ranked retrieval* se consideran dos factores: la frecuencia de un termino en el documento (cuantificado por el *term frequency* o tf) y la rareza de la palabra entre los documentos (cuantificado por el *inverse document frecuency* o *idf*). Por un lado, se asume que, entre más veces estén los términos de un *query* en un documento, este es más relevante. Y, por el otro lado, se asume que, la rareza de los términos en la colección los hace más importantes

(son más informativos). De esta manera, se utilizan estos dos conceptos para dar un puntaje (*score*) a cada uno de los documentos de la colección sobre los términos de un *query*.

Ahora bien, la importancia de la cantidad de términos en un documento (*term frecuency*) no necesariamente es lineal (si un término del *query* aparece 10 veces en un documento, este no es necesariamente 10 veces más importante que un documento que solo lo tiene 1 vez). Por esta razón, típicamente se utiliza la frecuencia logarítmica para pesar este concepto:

$$w_{t,d} = \begin{cases} 1 + log_{10}(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Por su parte, la cantidad de documentos en los que aparece un termino (*document frecuency* o *df*) es una medida inversa de la rareza, por lo que se utiliza su forma invertida *inverse document frecuency* o *idf*). A esta medida, de forma similar a la anterior, se le aplica la función logaritmo para reducir el efecto que tiene:

$$idf_t = log_{10}\left(\frac{N}{df_t}\right) \tag{6}$$

Así las cosas, para obtener el peso de cada término para cada documento simplemente se multiplican los dos pesos explicados en (5) y (6). COn esto se obtiene:

$$w_{t,d} = tf * idf_{t,d} = log(1 + tf_{t,d}) * log_{10}\left(\frac{N}{df_t}\right) \tag{7}$$

Donde:

- w es el peso resultante.
- t es el índice del término.
- d es el índice del documento.
- $tf_{t,d}$ es la frecuencia del término t en el documento d.
- df_t es la frecuencia de documento. Corresponde al número de documentos que contienen el término t.
- idf es la frecuencia de documento invertida. Corresponde al total inverso de documentos que contienen el término t al menos una vez.
- N: Tamaño total del corpus.

Finalmente, para obtener el puntaje (o *score*) de cada documento para un *query* dado, lo único que se debe hacer es sumar estos pesos sobre todos los términos de dicho *query*.

$$score(q, d) = \sum_{t \in q \cap d} tf * idf_{t,d} \tag{8}$$

2.3.1. Implementación

La implementación de esta estrategia de recuperación de información se encuentra en el cuaderno `HW01_4.ipynb`. Para esta se decidió construir la matriz de tf a partir del corpus de documentos, en donde se utiliza la expresión (5). Y, de igual forma, se construyó el vector de *idf*, con la expresión (6) para cada termino del vocabulario. A partir de estas dos expresiones se crea la matriz de pesos *tf-idf* con la que se procede a realizar los *queries* del dataset.

Para esto, se construye la función *basic_ranked_retrieval(tfidf_matrix, query)*, la cual recibe como parámetros la matriz con los pesos del corpus *tfidf* y el query correspondiente, y retorna los documentos documentos relevantes de cada query. Para esto se tienen en cuenta únicamente los documentos con un puntaje (*score(q,d)*) mayor a 0. Estos se exportan al archivo `RRI-queries_results`, con el mismo formato del archivo de etiquetas para su posterior evaluación.

2.4. Recuperación clasificada y vectorización de documentos

La técnica de recuperación clasificada con vectorización de documentos se basa en asignar una calificación a la similaridad a dos vectores, uno de los cuales corresponde a un documento y el otro a la consulta. La similaridad entre los dos vectores se obtiene con base en la similaridad el coseno, pues otro tipo de distancias como Euclidiana o Manhattan no permitirían obtener un valor adecuando. La similaridad del coseno está dada por la ecuación 9.

$$cos(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}} \tag{9}$$

Allí q hace referencia a la consulta, por su inicial en ingles (*query*) y d hace referencia al documento con el que se está comparando la consulta. $|V|$ hace referencia al tamaño del vocabulario, el cual, en este caso es de m.

El proceso de vectorización de cada documento se obtiene con base en el modelo *tf-idf*. Este indica que cada documento o consulta se representa con un vector con tamaño del vocabulario en donde cada registro indica un peso asignado a un término específico en relación a ese documento. Cada peso se obtiene con base en la ecuación 7, explicada en detalle en la sección anterior.

Como retorno se organiza los documentos según la similaridad que presentan. Podría tenerse en cuenta un umbral para descartar documentos con baja similaridad.

2.4.1. Implementación

Como se solicita en las instrucciones, se inicia con la implementación de una función que permita obtener el vector que representa cada documento con base en el peso $w_{t,d}$. Para ello, se inicia creando una matriz en donde se registra qué términos contiene cada documento con el fin de obtener un vector *idf*, el cual se multiplica con el logaritmo de la suma de 1 con la frecuencia del término asociado en el documento correspondiente.

Esta función se utiliza para calcular la similaridad del coseno entre el vector retornado para la consulta y para el documento. Finalmente, se organizan los valores retornados para cada documento para seleccionar aquellos que resultan relevantes.

2.5. GENSIM Corpus y modelo tf.idf

Con la librería de `Gensim` se puede implementar la misma técnica de recuperación clasificada a través de la vectorización de documentos (del literal anterior) de forma sencilla. Para esto se parte del modelo de bolsa de palabras (BOW) construido en el procesamiento inicial. Posteriormente, se crea un modelo de Tf-idf (`TfidfModel()`) con el corpus de documentos y se transforma el modelo para evaluar la similaridad de *queries*, creando un matriz de similaridad ((`MatrixSimilarity()`) con el modelo de Tf-idf del corpus de documentos. Este proceso lo puede encontrar en el cuaderno `HW01_6.ipybn`.

Ahora bien, con esta matriz de similaridad y el modelo de Tfidf se transforman cada una de las *queries* y se realiza la recuperación de la información a partir de su similaridad de coseno (*cosine similarity score)*, ordenando los resultados de mayor a menor similaridad. De igual forma, se tienen en cuenta únicamente los documentos con un puntaje (*score*) mayor a 0. Estos se exportan al archivo `GENSIM-queries_results`, con el mismo formato del archivo de etiquetas para su posterior evaluación (véase siguiente sección).

3. Análisis de resultados

3.1. Comparación de estrategias binarias: BS y BSII

Para la comparación de los resultados obtenidos con cada una de las estrategias binarias se registran las métricas de precisión, *recall* y *f1-score*. Por claridad se indica que en las gráficas siguientes 5 a 7 las etiquetas son:

- **BS_and:** Búsqueda binaria con conjunción.
- **BS_or:** Búsqueda binaria con disyunción.
- **BS_II_and:** Búsqueda binaria con índice invertido con conjunción.
- **BS_II_or:** Búsqueda binaria con índice invertido con disyunción.

La precisión (véase figura 5) indica qué porcentaje de los documentos recuperados se consideran relevantes de acuerdo con la etiqueta de la consulta. Como se esperaba los resultados de las estrategias con conjunción (AND) son mejores que los de disyunción (OR). Esto debido a que hay una mayor probabilidad de que si el documento contienen todos los términos de la consulta sea relevante, mientras que si contiene solo uno de ellos, la probabilidad es muchísimo menor.

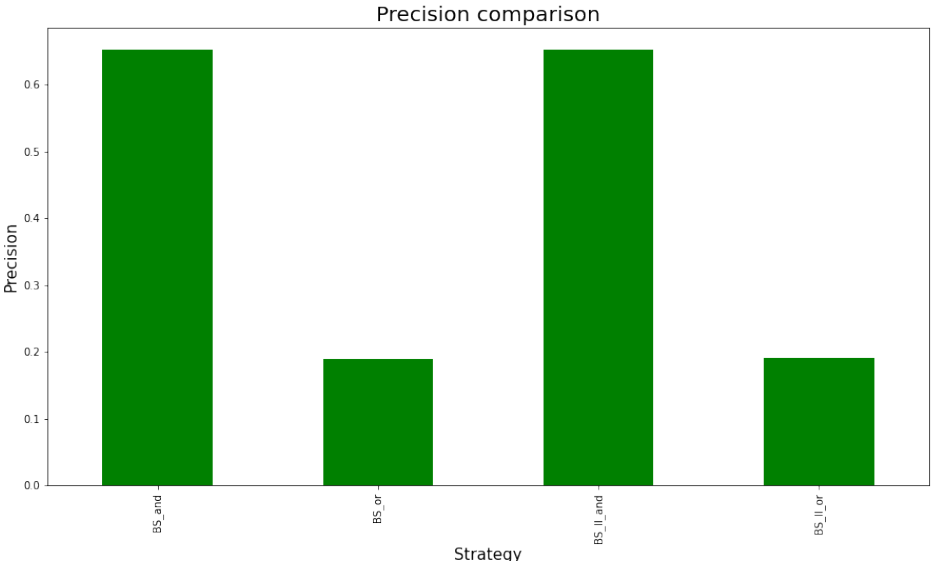

Figura 5: Precisión de cada estrategia.

El *recall* (véase figura 6), por su parte evalúa qué porcentaje de los documentos relevantes fueron recuperados. Con esto en mente, se espera que el resultado de esta métrica con las estrategias de disyunción sea muy buena, mientras que en las estrategias de conjunción resulta mala. Esto se debe a que existe la posibilidad de que un documento que no contiene todos los términos de la búsqueda sea relevante, por ejemplos sinónimos. Esto ocasiona que la estrategia de conjunción devuelva un valor mucho menor, pues retorna un porcentaje bajo del total de documentos que debería retornar.

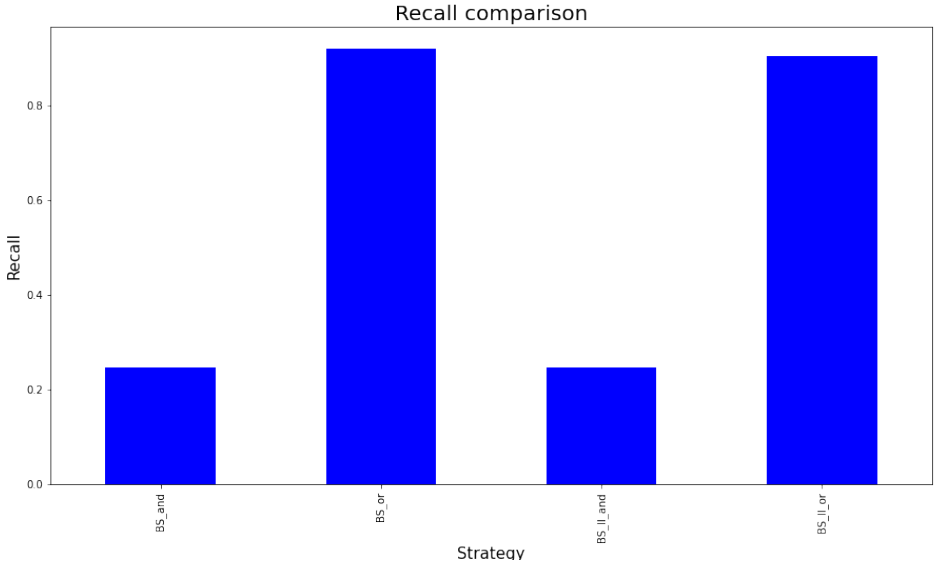

Figura 6: *Recall* de cada estrategia.

Como se puede observar, las gráficas anteriores evidencian que bajo una de la luz de la métrica de precisión, las estrategias de AND retornan valores muchísimo más altos que los de OR, mientras que para *recall* el resultado de las estrategias disyuntivas son muchísimo mejores. Es por esto que se recurre a una métrica que permita ponderar el resultado de precisión y *recall*. F1-score (véase figura 7) es una métrica que pondera estos resultados. Se calcula como se muestra en la ecuación 10.

$$R = \frac{2PR}{P + R} \tag{10}$$

El resultado es ideal cuando es 1, pues significa que ambas métricas son perfectas. No obstante, es difícil de obtener. El resultado obtenido con las implementaciones actuales indica que el desempeño de las estrategias AND es mejor que el de las

OR. Sin embargo, puede resultar conveniente ponderar de manera diferente el peso dado a cada métrica, en caso tal que resulte preferible obtener más documentos relacionados y no únicamente de los que se tenga total certeza de que son relevantes.

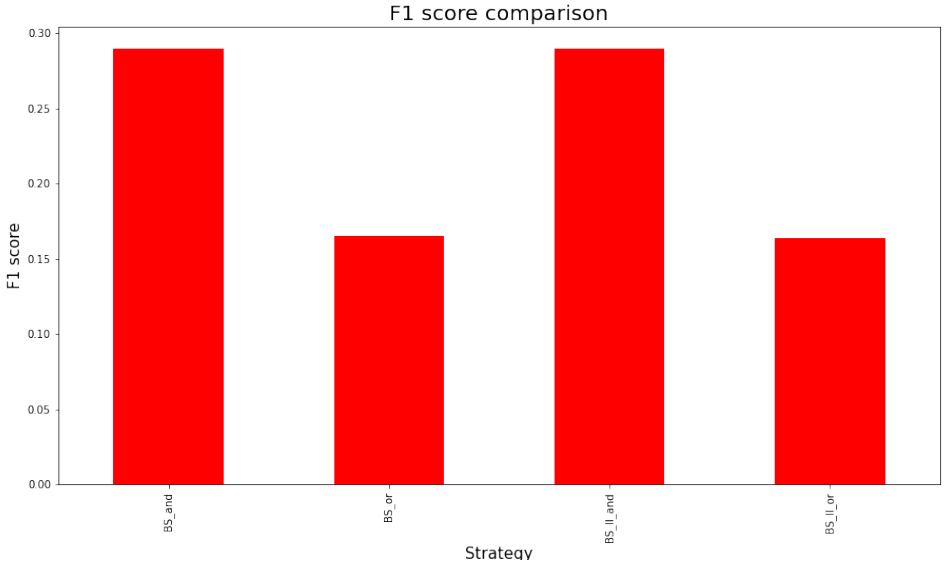

Figura 7: F1-score de cada estrategia.

Finalmente, se destaca que los resultados de búsqueda binaria normal y búsqueda binaria con índice invertido son los mismos (para AND y para OR). La única diferencia está en el tiempo de ejecución de cada una, pues el índice invertido da mucha resulta mucho más eficiente. Los tiempos de ejecución para cada estrategia fueron:

- **BS_OR:** 1.315 s
- **BS_AND:** 1.289 s
- **BSII_OR:** 0.00497 s
- **BS_AND:** 0.00185 s

Como es de esperarse, la estrategia de búsqueda binaria con índice invertido es mucho más rápida que la búsqueda binaria con el índice sin invertir. Esto se debe a que en vez de operar con una matriz sobre la cual se debe buscar en qué documento se presenta un término, el índice invertido puede identificar los documentos relevantes directamente a partir del término.

Teniendo en cuenta las dos estrategias de búsqueda binaria analizadas con sus dos posibles operadores binarios se estableció que hay cuatro términos en los que la búsqueda que requiere el mayor consumo computacional cuando se hacen con la búsqueda binaria. Estas búsquedas corresponden a: *q18*, *q27*, *q40* y *q42*. Las cuatro búsquedas indicadas anteriormente pueden ser consideradas las más caras dado que son las que más términos tienen para la búsqueda. Por otra parte, se considera que la búsqueda binaria es la estrategia con mayor consumo computacional dado que debe recorrer una matriz de tamaño fijo en todas las búsquedas. Sin embargo, no hay una distinción del operador binario que sea utilizado porque la intersección de dos arreglos de tamaño fijo consume los mismos recursos sin importar qué operación binaria se lleve a cabo.

En caso tal que el número de documentos relevantes aumente se presentarían un aumento de costo computacional para las dos estrategias, solo que con condiciones diferentes. Por una parte, la búsqueda binaria debe recorrer todas las posiciones de una matriz por lo que si el tamaño de esta aumenta, el recorrido también lo hará. En este orden de ideas, el consumo computacional para la búsqueda binaria aumentará inevitablemente. Por otra parte, en el caso del índice invertido el aumento del consumo computacional se hará en la medida que los nuevos documentos sean relevantes para los términos. En este caso se podría dar la situación en la que una consulta utiliza términos que no son incluidos en los nuevos documentos, por lo que el consumo computacional para dicha consulta no aumentará. Sin embargo, lo más probable es que los nuevos documentos aumenten la cardinalidad de las listas de posteo, lo cual aumenta a su vez el consumo computacional y el tiempo de respuesta del sistema.

Estrategia	**x100**	**x1000**	**x10000**
BS_OR	131.5	1315	13150
BS_AND	128.9	1289	12890
BSII_OR	0.01491	0.01988	0.02485
BSII_AND	0.00555	0.0074	0.00925

Cuadro 1: Estimación de tiempos de ejecución para las consultas con un aumento del tamaño de la colección.

A partir de los tiempos de ejecución presentados previamente, la tabla 1 estima nuevos tiempos en función del número de veces en que aumente el tamaño de la colección. Se tiene que un aumento en la búsqueda binaria implicaría que se deben analizar más documentos en la matriz que compone el índice. En este orden de ideas, se espera que haya un aumento lineal del tiempo de ejecución para esta estrategia. En el caso del índice invertido, se tiene que un aumento de la colección no implica un aumento en las listas de posteo. Sin embargo, se espera que estas aumenten de

tamaño aunque no de forma lineal. Para modelar un aumento de estas, lo cual va a afectar el tiempo de ejecución, se incluye un aumento en un factor logarítmico.

Para incluir nuevos documentos en el índice de una búsqueda binaria se deben ejecutar los siguientes pasos:

1. Agregar filas al índice con el fin de asociarlos a los nuevos documentos.
2. Llenar las filas de los nuevos documentos con unos o ceros dependiendo de la presencia o no, respectivamente, de los términos en dichos documentos.

Vale la pena mencionar que en caso tal que los documentos nuevos contengan un vocabulario considerablemente diferente al actual, será necesario agregar columnas al índice para considera los nuevos términos. Será necesario recorrer toda la colección con el fin de incluir los términos a los documentos antiguos en caso de ser necesario. Cuando se agregan nuevos documentos a un índice invertido es necesario realizar las siguientes acciones:

1. Organizar los términos presentes en los nuevos documentos con un criterio lexicográfico.
2. Iterar sobre los documentos que van a ser agregados.
3. Para cada término de un documento, ir agregando el identificador a la lista de posteo correspondiente.

En caso tal que se deba agregar vocabulario al índice invertido es necesario incluir el término según el ordenamiento lexicográfico. De igual manera, se deben recorrer todos los documentos de la colección con el fin de garantizar que el índice invertido es construido apropiadamente.

3.2. Comparación de estrategias clasificadas (*ranked*): RRI, RRDV, GENSIM

Para evaluar el desempeño de las estrategias de recuperación de la información clasificadas *(ranked)*, se realizó el calculo de las métricas `P@M`, `R@M`, `NDGC@M` y `MAP` para cada uno de los 35 *queries* en el dataset, donde M representa el número de documentos relevantes en la etiqueta. Para esto se hizo uso de las funciones desarrolladas en la primera parte de este trabajo, las cuales se encuentran en el archivo `metrics.py`. Como resultado de este proceso se obtuvieron las figuras 8 - 10 y el cuadro 2 que resume los resultados promedio de todas las estrategias implementadas.

En términos generales, se puede observar que los resultados de todas las estrategias *ranked* son bastante buenos. Salvo en dos ocasiones, todas las estrategias son capaces de recuperar al menos un documento relevante para el *dataset* de 35 *queries*. De igual forma, en varios casos se obtienen valores de *precisión*, *recall* y de *NDGC* de 1, lo que representa una recuperación perfecta (se retronaron todos los documentos relevantes en el orden correcto). Vale la pena aclarar que en la mayoría de los casos el resultado de las estrategias de IR, recuperaba muchos más documentos de los que el archivo de etiquetas sugería como relevantes. No obstante, al estar estos resultados ordenados y al hacer la evaluación del desempeño hasta M (salvo por el MAP), se obtiene un desempeño adecuado.

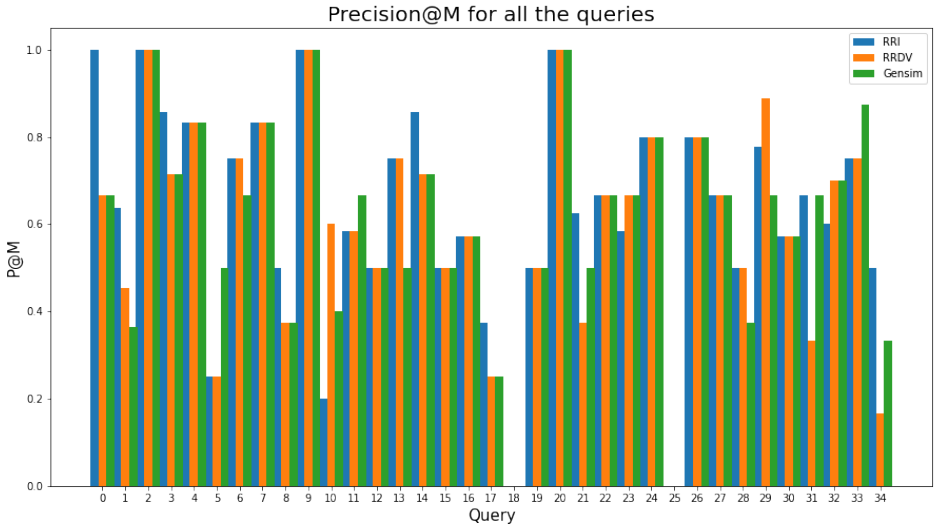

Figura 8: Resultados de P@M para todas las *queries* del data set con las estrategias clasificadas (*ranked*)

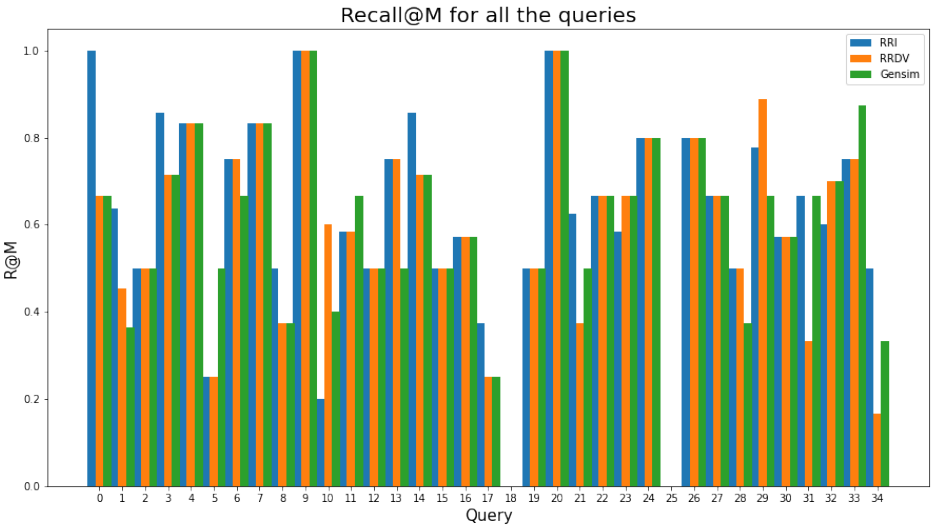

Figura 9: Resultados de R@M para todas las *queries* del data set con las estrategias clasificadas (*ranked*)

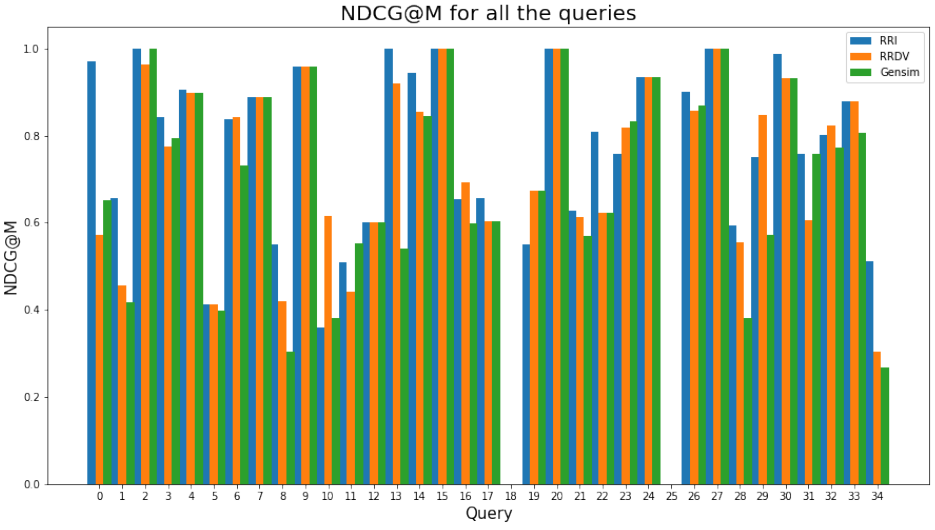

Figura 10: Resultados de NDCG@M para todas las *queries* del data set con las estrategias clasificadas (*ranked*)

Al momento de comparar el desempeños de las estrategias, se puede observar

que, en promedio, RRI presenta mejores resultados que RRDV y Gensim en todas las métricas. Aunque en general los resultados de las 3 estrategias son bastante comparables y sus resultados no difieren en mucho. Adicionalmente, se puede observar que RRDV y Gensim tienen desempeños aun más cercanos, en cuanto a la precisión estos son casi iguales, en cuanto al Recall Gensim es ligeramente mejor y en cuanto al NDCG y al MAP la implementación de RRDV es mejor.

Se podría pensar que al aplicar la misma estrategia sobre el mismo *dataset* los resultados deberían ser los mismos. No obstante, existen distintas variaciones que se le pueden aplicar a esta estrategia, la cual cambia sus resultados. Para este caso en especifico, la librería de Gensim, muestra en su documentación distintas formas de pesar las medidas de tf, df y realizar la normalización de documentos. Al utilizar la configuración por defecto, se toma como medida de tf el conteo crudo *(raw frequency)* de los términos en el documento, el idf y una normalización de documentos con similaridad coseno. Por su parte la implementación de RRDV realizada es muy similar, pero incorpora un peso logarítmico sobre el tf (véase ecuación 7). Así las cosas, se podría decir que el efecto de suavizar la frecuencia del tf con la función logaritmo, brinda un beneficio en cuanto al orden de los documentos, pues se obtiene un NDGC@M y un MAP significativamente mayores; pero con una ligera reducción en el desempeño del *Recall* y una precisión prácticamente igual.

Estrategia	**Mean P@M**	**Mean R@M**	**Mean NDCG@M**	**MAP**
RRI	0.6287	0.6144	0.7317	0.7340
RRDV	0.5923	0.5780	0.6967	0.6972
Gensim	0.5955	0.5812	0.6618	0.6592

Cuadro 2: Resumen de resultados de desempeño para las estrategias clasificadas (*ranked*).

Ahora bien, además de su desempeño, las estrategias de RRI y RRDV tienen otras diferencias que vale la pena mencionar. Por un lado, RRDV hace uso del concepto de documentos como vectores, en donde tanto el corpus de documentos, como los *queries* (y en general cualquier agrupación de palabras) son tratados como vectores de un modelo de bolsa de palabras (BOW Model). Por el otro lado, RRI utiliza los pesos de *tf-idf* para comparar puntajes (*scores*) entre documentos a partir de un *query*. Con esto en mente, el enfoque que propone RRDV puede ser ventajoso por su flexibilidad, pues se puede hacer una comparación de similaridad entre cualquier texto. En este caso, los *queries* y en los documentos se piensan todos en un mismo espacio vectorial, con todas las ventajas que eso conlleva: como la capacidad de crear normas y distancias especificas para la tarea deseada.

No obstante, en términos de costo computacional, el enfoque de RRDV parece ser el más costoso. Esto se debe a que típicamente los modelos de bolsa de palabras (BOW) tienen vectores con una alta dimensionalidad (tamaño del vocabulario) y son muy dispersos. Adicionalmente, para realizar la tarea de IR se debe completar un paso adicional de similaridad, el cual conlleva a realizar un producto punto entre dos vectores de este tipo. Al contrastar esto con el enfoque de RRI, en donde solo se debe hacer la suma de los pesos de *tf-idf* (valores ya guardados) sobre los distintos términos del *query*, los cuales además tienden a ser pocos, da una idea de porque este enfoque puede ser mucho más eficiente.

A nivel empírico este razonamiento concuerda con las implementaciones realizadas y se podría probar de forma experimental. Sin embargo, y aunque se cumple de forma evidente en las estrategias implementadas desde cero, el desempeño de Gensim es bastante eficiente. Esto lleva a pensar que existen muchas formas de mejorar la implementación en cuanto a costo computacional y también a que existe un gran número de factores que pueden alterar dicha eficiencia, como el tamaño del vocabulario, el tamaño de los *queries*, el tamaño de los documentos, entre otras cosas.

Finalmente, en el caso de que 100 nuevos documentos se incorporen a la colección los cambios requeridos pueden llegar a ser ligeramente distintos a los de una búsqueda binaria. Además del respectivo procesamiento inicial que requiere cada uno de los nuevos documentos (véase figura 1) es necesario tener en cuenta algunos detalles adicionales. Por un lado, no basta con agregar los documentos bien sea a la matriz de pesos de *tf-idf* o al índice invertido que guarda el tf, sino que es necesario recalcular el *df* de cada uno de los términos, pues el número de documentos en los que ahora aparece cada termino puede aumentar. No obstante, el tf no será algo que requiera ser recalculado, pues esta medida depende únicamente de cada documento, y solo deberá ser calculada para los nuevos documentos. Ahora bien, esto sería bajo el supuesto de que el vocabulario actual (de alrededor de 18000 palabras para el *dataset* en cuestión) sirva para los 100 nuevos documentos. En caso de que estos nuevos documentos fueran de un dominio totalmente distinto, y el vocabulario actual no respondiera a esto, sería necesario volver a construir el vocabulario y realizar nuevamente todo el proceso para la construcción de las estrategias de IR.

Referencias

[1] Cristopher D. Manning, Prabhakar Ragavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. 2009.