Sistemas de Recuperación de Información

Arnel Sánchez Rodríguez
Samuel Efraín Pupo Wong

Grupo C312

Universidad de la Habana

1. Introducción

La idea del uso de computadoras para la búsqueda de información se popularizó a raíz de un artículo *As We May Think* de Vannevar Bush en el año 1945. Los primeros sistemas automatizados de recuperación de la información fueron presentados durante la década de 1950 a 1960. En esta etapa se describe una máquina denominada UNIVAC capaz de buscar referencias de texto asociadas a un código temático. También se comenzó el uso de las listas de términos para indexar los objetos, el empleo de ranking en los documentos y la retroalimentación para la relevancia. Ya en este momento estaban consolidadas las computadoras como las herramientas definitivas para la búsqueda.

Durante 1970 se realizaron pruebas en un grupo de textos como la colección *Cranfield* para un gran número de distintas técnicas cuyo rendimiento fue bueno. Comenzaron a utilizarse los sistemas de recuperación a larga escala, como el Sistema de Diálogo *Lockheed*. Asimismo, se introdujo la idea de la frecuencia inversa en documentos

En la década de 1980, una preocupación de la comunidad académica era el tamaño de las colecciones documentos que se estaban empleando para las pruebas, pues se presumía eran muy pequeños respecto a los reales. Se creó una organización y anualmente se reunían para, cooperativamente, construir colecciones de prueba más grandes. El trabajo con estos nuevos conjuntos de datos mostró que las funciones existentes de ponderación y ranking no eran ideales. Además, se confirmó la idea de que diferentes colecciones requerirían diferentes acercamientos.

En 1992, el Departamento de Defensa de los Estados Unidos, en conjunto con el Instituto Nacional de Estándares y Tecnología (NIST por su sigla en inglés), patrocinaron la Conferencia de Recuperación de Texto (TREC) como parte del programa TIPSTER. Esto proveyó ayuda desde la comunidad de recuperación de la información al suministrar la infraestructura necesaria para la evaluación de metodologías de recuperación de texto en una colección a larga escala.

Desde los inicios de la *World Wide Web*, a finales de 1990, se hizo necesaria la adopción de motores de búsqueda que respondieran de manera eficiente a los requerimientos de los usuarios. Los desarrolladores rápidamente se dieron cuenta de que podían emplear los enlaces a los sitios web para construir robots que recorrieran y recopilaran la mayoría de las páginas en la Internet, automatizando la adquisición de contenidos.

Las aplicaciones de búsqueda y el campo de la recuperación de información continúan cambiando mientras que la computación evoluciona. Ejemplo de esto es el rápido crecimiento de los dispositivos móviles y las redes sociales. Nuevas investigaciones en una variedad de áreas como el etiquetado de usuarios, la recuperación de conversaciones, el filtrado y la recomendación comienzan a proporcionar herramientas efectivas para el manejo de la información personal y social.

Los buscadores, tales como **Google**, **Yahoo Search**, **Bing**, **Baidu** y **DuckDuckGo**, son algunas de las aplicaciones más populares para la recuperación de información.

2. Diseño del Sistema

Un Sistema de Recuperación de Información (IRS, por sus siglas en inglés) se compone por el cuádruplo $[D, Q, F, R(q_j, d_j)]$, donde:

- D es un conjunto de representaciones lógicas de los documentos que conforman un *corpus*.
- Q es un conjunto compuesto por representaciones lógicas de las consultas que el usuario realiza al sistema.
- F es un *framework* para modelar las representaciones de los documentos, consultas y sus relaciones.
- R es una función de *ranking* que asocia un número real con una consulta q_j que pertenece a Q y una representación de documento d_j que pertenece a D. La evaluación de esta función establece un cierto orden de relevancia entre los documentos de acuerdo con la consulta.

El modelo clásico de búsqueda comprende, en primer lugar, al usuario con un requerimiento informacional, que se transforma en una consulta. Luego, esta es representada de una manera lógica y procesada en un motor de búsqueda que examina una colección de documentos, denominada *Corpus*. De aquí se seleccionan los documentos relevantes, pudiendo darles una ordenación, según su importancia, y se presentan al usuario. En algunos sistemas se realiza luego un proceso de retroalimentación en el que se trata de refinar la respuesta con el objetivo de que sea más relevante respecto a la necesidad del usuario.

Analizaremos a continuación nuestro sistema en su totalidad: las consultas, la composición del motor de búsqueda y la clasificación de los documentos según su relevancia. En nuestro caso, el sistema va enfocado a la búsqueda de documentos, los cuales tienen varias características, siendo las dos más útiles: el título y el cuerpo.

2.1. Motor de Búsqueda

El motor de búsqueda implementa un modelo de recuperación de información, en este se realiza el preprocesado de los documentos para su representación interna. También es el encargado de construir el ranking de relevancia de los documentos del *corpus*, acorde a la consulta de entrada.

Preprocesamiento: Para representar los documentos, es necesario realizar un preprocesado. En nuestro sistema empleamos recursos del procesamiento de lenguaje natural para modificar los documentos y luego poder vectorizar. Las siguientes transformaciones son realizadas para cada documento del *corpus*:

1) *Tokenización* del documento: Primeramente, se crea la representación básica de un documento: la división en tokens; de tal forma que podamos realizar procesamientos más avanzados sobre el texto. Desde esta etapa del preprocesamiento podemos llevar a minúscula todos los caracteres de las palabras, eliminar símbolos y signos de puntuación.

2) Eliminación de *stop words*: Los *stop words* son palabras vacías de significado, las cuales pueden servir de nexo entre entidades o funcionan como modificadores. Si incluimos estas palabras en la representación del documento se puede afectar precisión del modelo, debido a la alta frecuencia que poseen estas palabras en los textos.
3) *Stemming*: Este método busca relacionar palabras con igual significado pero que difieren en cuanto a la escritura debido a que presentan prefijos o sufijos. En este caso no usaremos *lemmatizing*.

Estas técnicas son las empleadas por el sistema para obtener un conjunto de textos formados por entidades con un significado y peso correcto para empezar a construir el modelo, pero antes de definir el modelo es necesario definir cómo serán estas entidades y qué representación interna tendrán. Empleando los tokens como entidad del sistema, se construye una colección con el siguiente formato:

```
corpus: {
           [id: n,
           'title': Tokenize(doc_title),
           'body': Tokenize(doc_body),
           ],
           ...
        }
```

Vectorización: Una vez que tenemos los tokens es necesario realizar una vectorización, obteniendo así una entrada correcta para el modelo, emplearemos para ello TF-IDF. Para calcular los valores de frecuencia se utilizarán el título y el cuerpo del documento juntos, sin realizar distinción, por lo que el título podrá ser una oración más del texto.

Term Frequency–Inverse Document Frequency (TF-IDF): Es una técnica para cuantificar términos en documentos, generalmente calculamos un peso para cada término, cuyo significado es la importancia de este en el documento y en el *corpus*. Su idea se basa en que la ocurrencia de un término en una colección de documentos es inversamente proporcional a su importancia en la recuperación. Dicho método es ampliamente utilizado en recuperación de información y minería de textos. Se representa por $w(t_i, d_j, D)$, donde t_i es un término del documento d_j.

La definición de TF-IDF se compone de dos conceptos fundamentales en el procesamiento de los documentos:

TF (Frecuencia de Término): Mide la frecuencia de una palabra en un documento. Esto depende en gran medida de la longitud de dicho documento y de la generalidad de esta palabra. Debido la variación de la longitud, contar simplemente las palabras podría dar prioridad a documentos de mayor tamaño, por esta razón se divide la cantidad de ocurrencias por el número total de palabras $N(d_j)$. TF es individual para cada documento y palabra, por lo tanto, podemos formularlo de la siguiente manera:

$$ft(t_i, d_j) = \frac{C(t_i, d_j)}{N(d_j)}$$

Donde $C(t_i, d_j)$ representar la cantidad de ocurrencias del término en el documento. Cada componente tendrá el valor de frecuencia asociado para esa palabra, el cual estará en el intervalo $[0, 1]$. Sin embargo, esta ponderación beneficia a palabras comunes que se repiten en los documentos.

DF (Frecuencia de Documento): Mide la importancia del término t_i en todo el conjunto de *corpus*. Es el recuento de apariciones de este en el conjunto de documentos D, es decir, el número de documentos en los que está presente la palabra. Se considera una ocurrencia si el término consta en el documento al menos una vez, ignorando el número de veces que está presente.

IDF (Frecuencia Inversa de Documento): Es la inversa de DF, que mide la informatividad del término t_i. Este será muy bajo para las palabras más frecuentes y mayor para las menos comunes. Como es posible que la DF sea 0, entonces es necesario realizar algunas modificaciones en la división de la inversa. Además, si el *corpus* es grande se podría obtener un valor alto, que no es de utilidad, por lo que se empleará una función de crecimiento más lento para suavizar la IDF.

$$idf(t_i, D) = \frac{log|D|}{df(t_i, D) + 1}$$

Combinando ambas ecuaciones (TF-IDF):

$$w(t_i, d_j, D) = ft(t_i, d_j) * idf(t_i, D)$$

$$w(t_i, d_j, D) = \frac{C(t_i, d_j)}{N(d_j)} * \frac{log|D|}{df(t_i, D) + 1}$$

Anteriormente se expuso que no existe distinción entre el título y el cuerpo del documento, puesto que el primero se analiza como una oración más del texto. Por tanto, no hay diferencias en el cálculo de la ponderación de los términos de ambos.

Para vectorizar el documento emplearemos el siguiente concepto:

Bag of Words: Consiste en tomar todas las palabras del *corpus* que constituyen el vocabulario (V) y formar una colección donde cada palabra tiene asociado un índice, de esta forma se obtiene un vector $\overrightarrow{d_j}$ de dimensión $1 * |V|$ asociado al documento d_j, donde en cada componente se encuentra el valor de TF-IDF ligado a la palabra.

El Bag of Words puede considerarse como un *mapping* donde podemos comprobar en que posición de una lista formada por todas las palabras se encuentra la que estamos buscando, empleando esta idea es posible vectorizar cada documento.

2.2. Consultas

Se conoce que en el sistema existe un conjunto Q, compuesto por las consultas realizadas por el usuario. Una consulta q no es más que una oración o pequeño párrafo empleado para realizar una búsqueda. Para poder determinar cuáles documentos son relevantes para esa consulta, es necesario establecer un orden de estos, acorde a la similitud que exista entre ellos. Por lo tanto, debemos convertir la consulta a un vector empleando el mismo proceso descrito en el motor de búsqueda.

Se usa el mismo vocabulario V utilizado para la representación del *corpus*, y se procede a crear el vector $\overrightarrow{q_j}$, en caso de que existan palabras en q_j que no se encuentren en el vocabulario, serán ignoradas. Para crear el vector de consulta usando TF-IDF, se agrega un parámetro de suavizado a (en nuestro caso $a = 0.5$), el cual permite amortiguar la frecuencia, evitando que aparezcan grandes saltos en esta o se repita el mismo valor de forma consecutiva. El cálculo de los pesos para los términos de la consulta sería del modo:

$$w(t_i, q_j, D) = \left(a + (1-a) * ft(t_i, q_j)\right) * idf(t_i, D)$$

$$w(t_i, q_j, D) = \left(a + (1-a) * \frac{C(t_i, q_j)}{N(q_j)}\right) * \frac{log|D|}{df(t_i, D) + 1}$$

2.3. Función de *Ranking*

Ahora es necesario relacionar la consulta con el *corpus*, para determinar el valor de relevancia que tienen los documentos con respecto a esta. Para ello debemos definir la función $R(q_i, d_j)$ de la cual expondremos a continuación dos enfoques:

Puntuación Coincidente: Una forma sencilla de calcular la similitud es representar el vector de consulta $\overrightarrow{q_i}$ de forma *booleana*, asignando 1 para los términos que aparecen en la consulta y 0 en caso contrario. Luego, para cada documento sumamos los valores de TF-IDF que tienen 1 en el vector consulta, obteniéndose de esta forma el valor de similitud para realizar la ordenación. Esta operación solamente realiza un producto escalar de los vectores.

$$R(q_i, d_j, D) = \overrightarrow{q_i} * \overrightarrow{d_j} = \sum_{k=0}^{|V|} \left(w(t_k, d_j, D) * w(t_k, q_i, D)\right)$$

De esta forma, se obtienen mayores valores de relevancia para documentos con valores de frecuencia altos para los términos que componen la consulta. Al representar el vector consulta solamente con 0 y 1, no se aprovecha la relación de cercanía espacial que podría existir entre los vectores de consulta y documento.

Similitud del Coseno: Se utilizan el valor de TF-IDF asociado al término en la consulta y el coseno del ángulo comprendido entre los vectores consulta $\overrightarrow{q_i}$ y documento $\overrightarrow{d_j}$ para determinar la similitud:

$$R(q_i, d_j, D) = \frac{\overrightarrow{q_i} * \overrightarrow{d_j}}{|\overrightarrow{q_i}| * |\overrightarrow{d_j}|} = \frac{\sum_{k=0}^{|V|} \left(w(t_k, d_j, D) * w(t_k, q_i, D)\right)}{\sqrt{\sum_{k=0}^{|V|} w^2(t_k, d_j, D)} * \sqrt{\sum_{k=0}^{|V|} w^2(t_k, q_i, D)}}$$

En este caso, es posible apreciar cómo se busca establecer una relación entre los valores de frecuencia de la consulta y el documento. De esta forma, mientras más cerca estén los vectores de consulta y documento, más relevante es ese documento para el usuario que realiza la petición. Por esta razón, se implementó este segundo enfoque, pues se entiende que es superior al primero (visto anteriormente).

***Output* del Sistema:** Una vez se tiene la función de *ranking*, se calculan los valores correspondientes para la consulta, ordenando de forma descendente aquellos que son mayores que 0. Se devuelve el título del documento y su dirección física.

Nota: En los casos prueba, el Sistema devuelve solamente el índice del documento, puesto que la dirección física de todos es la misma.

2.4. Expansión de consultas

Una variante para lograr la retroalimentación del SRI es adecuar mejor la consulta al conjunto de documentos relevantes esperado. Esto se puede lograr expandiendo el conjunto de términos de la consulta. En nuestro caso, lo hicimos a través de un análisis global.

De esta manera, por cada término t en la consulta, esta puede ser expandida automáticamente con sinónimos y palabras relacionadas con t, utilizando un tesauro. Se incrementa así el recobrado sin la intervención del usuario, pues este modelo muestra sugerencias extras que permiten formular una nueva consulta.

3. Detalles de la Implementación

La implementación del Sistema se realizó en **C#**, este cuenta con herramientas para el procesamiento de lenguaje natural y puede ser empleado para realizar trabajos de manejo de datos y recuperación de información.

El código del proyecto está dividido en varios archivos **.cs**, siendo el que inicia la ejecución **Program.cs**. Desde este se hace una llamada a la clase estática **Corpus**, la cual recibe la dirección de los archivos a analizar y un *bool* que indica si se está trabajando con archivos reales o colecciones de prueba.

En la sección de producción del método **CorpusLoad**, de **Corpus.cs**, se cargan los archivos del directorio configurado, se leen todos estos y se les hace *tokenización* y eliminación de *stop words*, utilizando la librería oficial **Microsoft ML**. Luego, se hace *stemming* a la lista de tokens de cada documento, para lo cual contamos con dos librerías: una es **Annytab** y la otra es una implementación de **Porter**; pudiendo seleccionarse cual se usará comentando el código de una y descomentando el de la otra.

Por último, se ejecuta la vectorización aplicando las fórmulas antes expuestas. Por cada archivo en cuestión, se crea una instancia de la clase **File** y se le asignan los parámetros de la vectorización que solo dependen de sí mismo. Luego el **CorpusLoad** termina de ejecutar la vectorización con los parámetros que dependen de toda la colección.

Para el caso de la sección de pruebas, la única diferencia está en que la forma de leer los documentos, puesto que todos vienen comprimidos en un único archivo.

El resultado final es almacenado en una lista de **Files** (instancias de la clase File) y se puede acceder a esta a través del método **GetFiles**, con el fin de realizar las consultas deseadas a los archivos ya procesados.

Las consultas son recibidas en **SearchEngine.cs** y estas son clasificadas en dos conjuntos, del mismo modo que los archivos: modo de producción o modo de prueba. Se cuenta con una instancia de la interfaz **ISearchEngine** que implementa la clase **SearchEngine**, la cual contiene un método **Search** que recibe un *string* con la cadena a buscar y un *bool* que indica en qué modo se está trabajando. En ese método se inicia un cronómetro para calcular el tiempo que demora la consulta y se crea una instancia de la clase **Query**.

En **Query**, se vectoriza la consulta de la forma señalada con anterioridad, pues en su constructor se *tokeniza*, se eliminan los *stop words* y se le hace *stemming*. Luego, se utiliza un método privado para actualizar su vector. Con dicho vector actualizado, en la clase **SearchEngine** se hace llamada al método **Ranking**, que lo calcula y devuelve, ordenado en orden descendente, los archivos de acuerdo con su similitud, siempre que su valor sea mayor que 0.

En la función **Ranking**, cuando se crea la instancia de la clase **SearchResult**, se utiliza el método **GetSuggestions** que utiliza un tesauro de sinónimos a través de la librería **WeCantSpell.Hunspell** para hacer sugerencias de nuevas búsquedas al usuario mediante la expansión de consultas. De esta manera,

se devuelve un **SearchResult**, el cual posee el tiempo de respuesta de la consulta, la lista de resultados y de sugerencias.

La lista de documentos obtenidos está compuesta por elementos de la clase **SearchObjectResult**, cuyas propiedades son **Name** y **Location**. Cabe destacar que, en entorno de producción, estas propiedades responden al nombre y ubicación física de cada documento recuperado; en caso contrario, será solamente su id, puesto que las colecciones de prueba poseen su *corpus* en un único archivo.

El sistema posee una comunicación mediante un ***Endpoint*** del tipo **POST** creado en el archivo **SearchController.cs**, en el cual se pueden realizar las consultas de manera manual a través de la URL: *https://localhost:7290/swagger/index.html*. Esta dirección permite hacer comprobaciones al Sistema tanto con documentos reales como en modo de prueba.

Del mismo modo, es posible hacer consultas a documentos de manera automática, mediante peticiones en formato **json**, a través de la URL: *https://localhost:7290/api/Search*; o a colecciones en el ambiente de prueba: *https://localhost:7290/api/Search/test*

3.1. Interfaz Visual

Para hacer más cómoda la realización de consultas, se implementó una interfaz visual utilizando la librería **React** de JavaScript, tomando una plantilla de Google como base. Su funcionamiento está ligado al ***Endpoint*** antes mencionado. Por esta razón, para su correcto funcionamiento el **Backend** del proyecto debe estarse ejecutando previamente, pues este recibe cada consulta hecha en el **Frontend**, la analiza y devuelve los resultados obtenidos. La comunicación se hace de manera bilateral en formato **json**.

Esta interfaz visual es muy intuitiva, puesto que cuenta con una pantalla principal con un campo de texto donde se introduce la búsqueda deseada y un botón para iniciarla. Luego, los resultados de la búsqueda son mostrados en una nueva pantalla, de manera tal que se pueden visualizar los títulos de los documentos obtenidos y un *link* que permite abrirlos en caso de desearse.

En esta pantalla también se cuenta con un campo de texto que permite ejecutar una nueva búsqueda y un botón que retorna a la pantalla principal. Es posible observar el total de resultados obtenidos y el tiempo en que se obtuvo la respuesta. Del mismo modo, se muestra una lista de posibles sugerencias afines con los términos empleados.

La dirección inicial del navegador (pantalla principal) es: *http://localhost:3000*

Las direcciones de búsqueda son: *http://localhost:3000/search=termino*

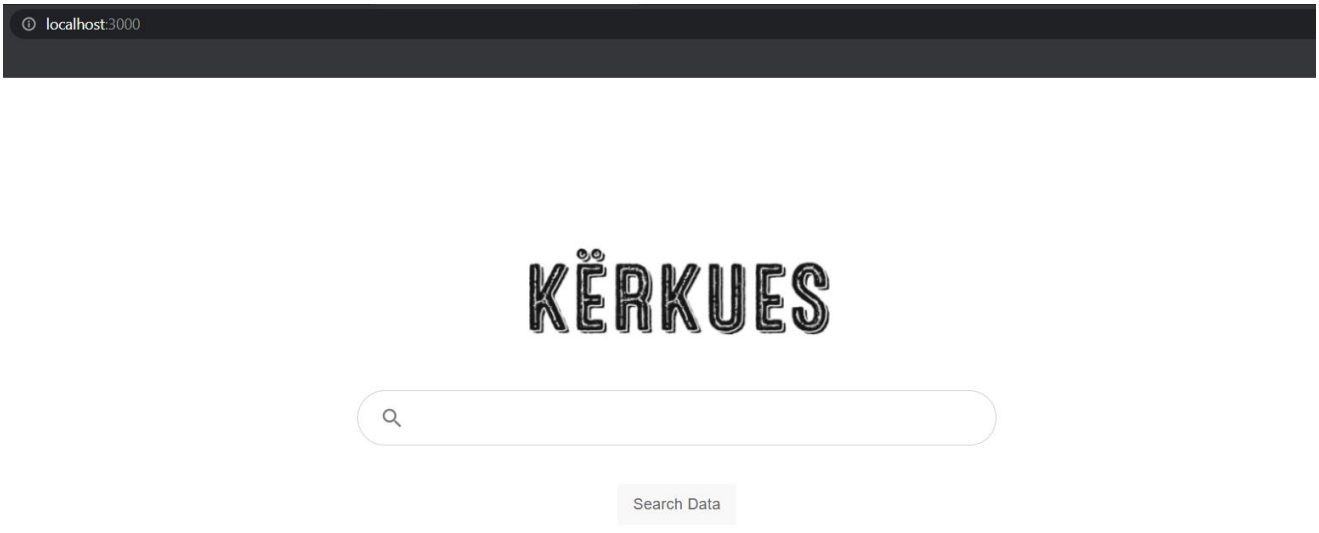


Imagen 3.1.1: Pantalla principal

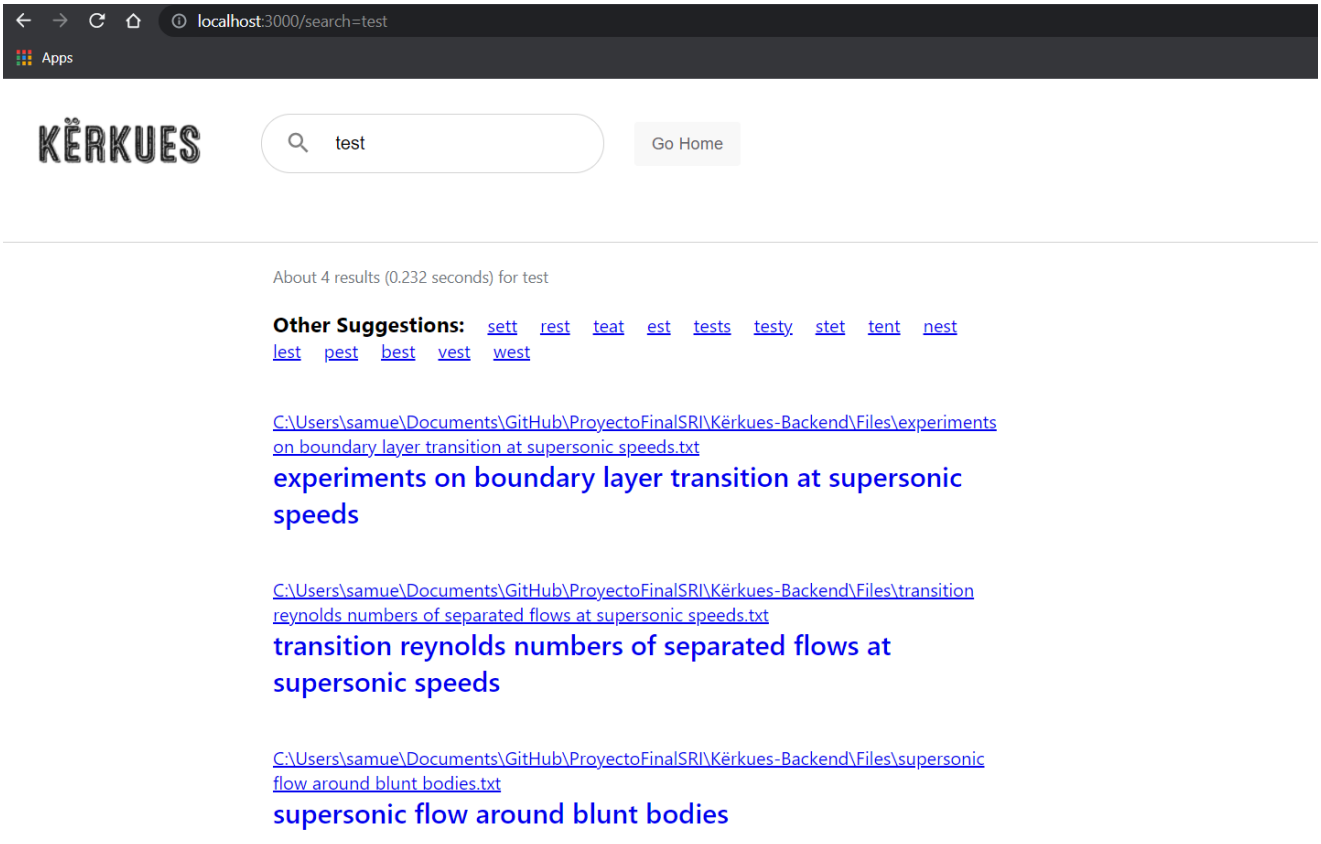


Imagen 3.1.2: Pantalla de búsqueda

Dado que los navegadores no permiten abrir archivos locales de manera automática (por cuestiones de seguridad), es necesario crear un servidor local donde estos sean accesibles. Se propone utilizar el *Add* de Google Chrome: ***Web Server for Chrome***.

En el botón ***Choose Folder*** debe seleccionarse la carpeta en la cual están los archivos utilizados en el **Backend**, y el puerto a utilizar debe ser el **8080**. La URL del servidor debe ser: *http://127.0.0.1:8080*

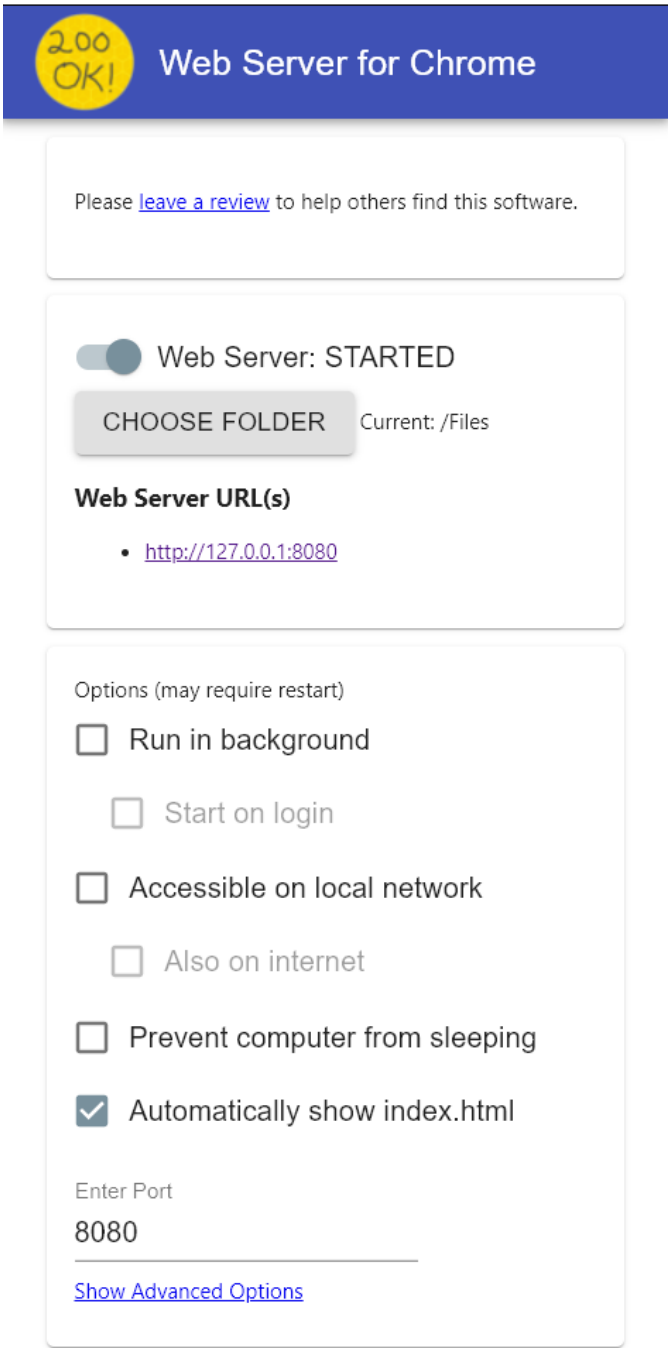


Imagen 3.1.3: Servidor local

4. Evaluación del Sistema

La evaluación del sistema se hace utilizando colecciones de prueba, las cuales en este caso serán tres: *Cranfield*, *Medline* y *ADI*. Para ello se definirán cuatro conjuntos fundamentales, los cuáles intervendrán en la mayoría de las medidas de evaluación:

Documentos	Relevantes	Irrelevantes
Recuperados	RR	RI
No Recuperados	NR	NI

Los documentos recuperados irrelevantes (RI) pueden ser considerados como falsos positivos, pues se recuperaron, pero no eran interesantes para el usuario. De manera análoga, los documentos no recuperados relevantes (NR) son los falsos negativos.

Empleando los conjuntos anteriores, se definen las siguientes medidas de evaluación:

Precisión: Fracción de los documentos recuperados que son relevantes

$$P = \frac{|RR|}{|RR \cup RI|}$$

Sin embargo, en el campo de la recuperación de información no solo interesa qué tan precisa es la respuesta, también que incluya a la mayor cantidad posible de documentos relevantes, por lo tanto la medida de precisión por sí sola no es suficiente en el caso de un modelo de recuperación de información.

Recobrado: Fracción de los documentos relevantes que fueron recuperados

$$P = \frac{|RR|}{|RR \cup NR|}$$

Este es fundamental en los procesos de recuperación de información. Al contrario de la precisión, el recobrado aumenta a medida que incorporamos más documentos a la respuesta, pues es cada vez más probable que los elementos del conjunto de documentos relevantes estén contenidos en nuestra respuesta.

Es importante notar que las dos metidas (P y R) se compensan (no son inversamente proporcionales). La precisión tiende a decrecer cuando la cantidad de documentos recuperados aumenta y siempre es posible obtener $R = 1$, si recuperamos todos los documentos para la consulta, pero en este caso obtendríamos una precisión baja. La solución ideal plantea un alto valor de recobrado tolerando la menor cantidad posible de falsos positivos, es decir, de documentos recuperados irrelevantes.

Medida F_1: Medida que armoniza Precisión y Recobrado teniéndolos en cuenta a ambos

$$F_1 = \frac{2 * P * R}{P + R}$$

La medida $\boldsymbol{F_1}$ toma un valor alto cuando la precisión y el recobrado son altos, por lo que puede verse como el intento de encontrar la mejor relación entre P y R, donde ambos tienen igual importancia

Proporción de Fallo: Tiene en cuenta la cantidad de elementos irrelevantes

$$Fallout = \frac{|RI|}{|RI \cup NI|}$$

Las medidas básicas de evaluación están diseñadas para modelos que no hacen *ranking* de los documentos en la respuesta. Pero en nuestro caso también analizaremos las medidas para una cantidad fija de documentos dada su ordenación, es decir, tomando los n documentos de mayor relevancia.

5. Resultados Obtenidos

Empleando las medidas expuestas sobre las colecciones de prueba antes mencionadas, se puede evaluar qué tan eficiente es el SRI implementado. Con tal fin, cada colección posee un conjunto de consultas y los resultados esperados para cada una.

El **Tester** utilizado para la evaluación del sistema se hizo en **Python**. Este permite, seleccionando una de dichas colecciones, realizar la búsqueda de las consultas predefinidas sobre el *corpus* procesado.

Primeramente, lee el archivo que posee la lista de consultas a procesar y el que contiene los resultados esperados para cada una de estas. Luego, teniendo esta información, se comunica con el **Endpoint** del **Backend** mediante peticiones en formato **json**, a través de la URL: *https://localhost:7290/api/Search/test*; comparando los datos obtenidos con los esperados, evaluándolos.

Este proceso muestra los resultados individuales por *query* y luego, al finalizar, una evaluación general y el gráfico P/R correspondiente. De esta manera, se pueden observar el comportamiento de las medidas calculadas tanto individualmente como el promedio general; así como también la relación existente entre la precisión y el recobrado.

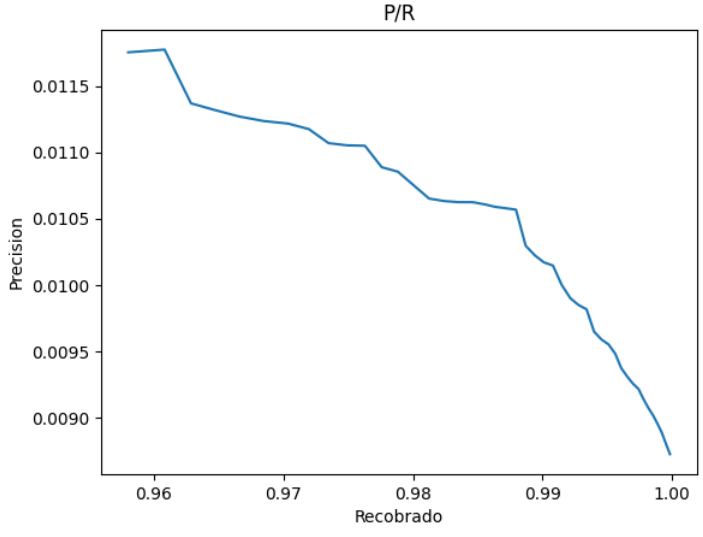
Nótese que se computa el análisis para todo el conjunto de documentos obtenidos como resultado de una consulta, pero también para los n primeros del *ranking* ($n = 10$ en el caso de las imágenes). Esto permite verificar qué tan buenos son los documentos a los cuales el SRI les otorga mayor valor de relevancia.

```
Consulta_1: "what similarity laws must be obeyed when constructing aeroelastic models of heated high speed aircraft"
Precision: 0.033
Recobrado: 0.8966
Medida-F1: 0.0636
Fallo:     0.5558
Precision_10: 0.6
Recobrado_10: 0.2069
Medida-F1_10: 0.3077
Fallo_10:     0.0029
```

Imagen 5.3: Resultados para la Consulta 1 de Cranfield

```
Evaluacion General:
Ningun documento recuperado:           0 (0.0%)
Ningun documento relevante recuperado: 2 (0.89%)
Precision (promedio): 0.01
Recobrado (promedio): 0.9543
Medida-F1 (promedio): 0.0198
Fallo     (promedio): 0.6256
Precision_10 (promedio): 0.2911
Recobrado_10 (promedio): 0.4204
Medida-F1_10 (promedio): 0.3213
Fallo_10     (promedio): 0.0051
```

Imagen 5.2: Resultados generales para Cranfield



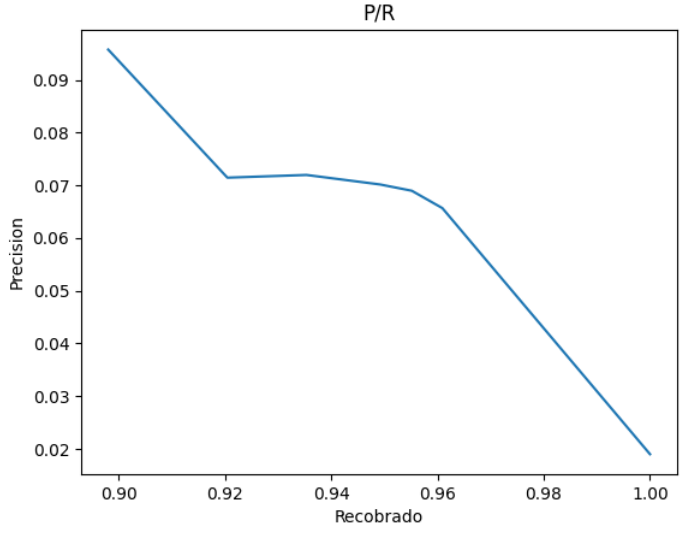
Gráfica 5.1: P/R para Cranfield

```
Consulta_13: "bacillus subtilis phages and genetics, with particular reference to transduction"
Precision: 0.1481
Recobrado: 0.9524
Medida-F1: 0.2564
Fallo:     0.0834
Precision_10: 0.9
Recobrado_10: 0.4286
Medida-F1_10: 0.5806
Fallo_10:     0.0007
```

Imagen 5.3: Resultados para la Consulta 13 de Medline

```
Evaluacion General:
Ningun documento recuperado:           0 (0.0%)
Ningun documento relevante recuperado: 0 (0.0%)
Precision (promedio): 0.0891
Recobrado (promedio): 0.8976
Medida-F1 (promedio): 0.1374
Fallo     (promedio): 0.2689
Precision_10 (promedio): 0.6433
Recobrado_10 (promedio): 0.311
Medida-F1_10 (promedio): 0.405
Fallo_10     (promedio): 0.0026
```

Imagen 5.4: Resultados generales para Medline



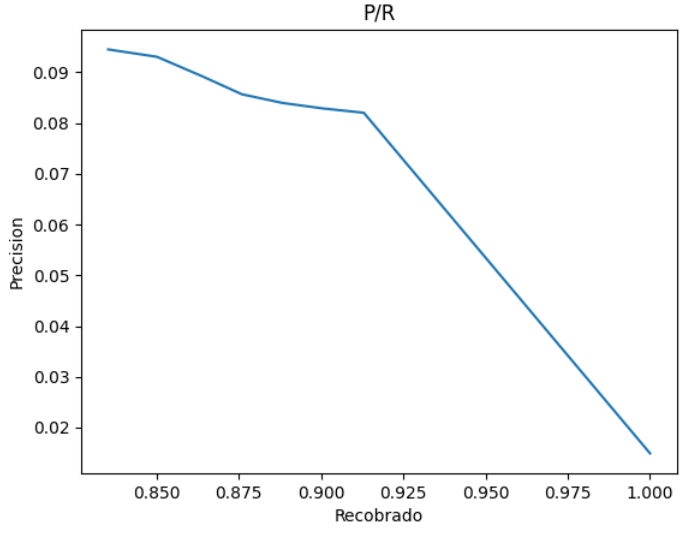
Gráfica 5.2: P/R para Medline

```
Consulta_35: "Government supported agencies and projects dealing with information dissemination"
Precision: 0.1458
Recobrado: 0.875
Medida-F1: 0.25
Fallo:     0.0295
Precision_10: 0.1
Recobrado_10: 0.125
Medida-F1_10: 0.1111
Fallo_10:     0.0065
```

Imagen 5.5: Resultados para la Consulta 35 de ADI

```
Evaluacion General:
Ningun documento recuperado:           0 (0.0%)
Ningun documento relevante recuperado: 1 (2.86%)
Precision (promedio): 0.0777
Recobrado (promedio): 0.8301
Medida-F1 (promedio): 0.1379
Fallo     (promedio): 0.0334
Precision_10 (promedio): 0.1971
Recobrado_10 (promedio): 0.4938
Medida-F1_10 (promedio): 0.2585
Fallo_10     (promedio): 0.0058
```

Imagen 5.6: Resultados generales para ADI



Gráfica 5.3: P/R para ADI

Se puede afirmar que los resultados son aceptables, tomando en cuenta que se empleó un modelo vectorial usando TF-IDF con algunas modificaciones. El recobrado promedio es alto, aunque la precisión es baja, esto se debe a que se devuelve un alto número de documentos ante cada consulta, muchos de los cuales no son catalogados como relevantes.

Sin embargo, cuando se analizan las medidas para el *ranking* (en este caso, los 10 documentos catalogados con mayor relevancia), los valores de la precisión y el recobrado son más cercanos, elevando la medida F_1; así como también el fallo es mucho menor.

Cabe destacar que los resultados obtenidos con las colecciones *Medline* y *ADI* son mejores que los obtenidos en *Cranfield*, lo cual se debe a las dimensiones de estas y de sus textos, y al número de consultas realizadas en cada caso.

6. Recomendaciones

Analizando las herramientas empleadas y los resultados obtenidos, se pueden plantear algunas modificaciones y proponer el uso de otros modelos, con el fin de lograr mejoras en el SRI.

Es importante señalar que la representación vectorial considera a los términos indexados como independientes, por lo que no toma en cuenta ni la posición, ni las relaciones entre estos. Podría realizarse un enfoque empleando modelos de Semántica Latente para agregar cierto grado de interpretación a las consultas realizadas. Adicionalmente, también se podrían agregar elementos de similitud semántica para una mejor expansión de las consultas realizadas por el usuario.

Añadir un modelo basado en entrenamiento, tales como Regresión Logística, Árboles de Decisión o Redes Neuronales, puede ayudar construir una representación más expresiva del *corpus* y de esta forma obtener mejores resultados.

Por último, si en lugar de palabras se emplearan *n-grams* o subsecuencias máximas, se contaría con entidades más completas. Sin embargo, esto conllevaría a que tenga que ser modificada la representación interna, debido a que puede aumentar considerablemente la dimensión de los vectores. Por ello sería necesario buscar estructuras de datos de almacenamiento más sólidas para una mejor representación del *corpus*.

7. Referencias

1. Conferencias de Sistemas de Recuperación de Información (Curso 2021-2022). Facultad de Matemática y Computación, Universidad de la Habana.
2. Manning, C. D. (2009). An Introduction to Information Retrieval. Cambridge UP.
3. Baeza-Yates, R. (2002). Modern Information Retrieval.
4. Baeza-Yates, R. (s.f.). Information Retrieval: Data Structures & Algorithms.
5. https://www.microsoft.ml/ : *Tokenización* del documento y eliminación de los *stop words*
6. https://github.com/annytab/a-stemmer : Annytab.Stemmer
7. http://snowball.tartarus.org/algorithms/english/stemmer.html : Algoritmo de Porter
8. https://github.com/titoBouzout/Dictionaries : Diccionarios
9. http://ir.dcs.gla.ac.uk/resources/test_collections/ : Colecciones de prueba