

**Carrera de Ciencia de la Computación**  
**Universidad de la Habana**

---

Proyecto Final de  
Sistemas de Recuperación de  
Información

Versión 1.0

---

- **Alejandro Escobar Giraudy C312**
- **Airelys Collazo Perez C312**

## Table of Contents

1	Abstracto .....	3
2	Introducción .....	3
3	Diseño del sistema .....	3
3.1	¿Qué modelo es conveniente usar? .....	3
3.2	¿Qué otras herramientas se pueden usar? .....	4
4	Implementación del sistema .....	4
4.1	Procesamiento y representación de documentos .....	4
4.2	Procesamiento y representación de consultas .....	6
4.3	Similitud .....	6
4.4	Clustering .....	7
4.5	Interfaz de usuario .....	7
4.6	Manual de usuario .....	7
5	Evaluación del sistema .....	10
5.1	Medidas usadas .....	10
5.2	Análisis de los resultados .....	12
6	Ventajas y desventajas del sistema desarrollado .....	14
7	Recomendaciones y Conclusiones .....	14
8	Referencias .....	15

## 1 Abstracto

Hoy vivimos en una era tecnológica, donde la mayoría de la información y de los documentos son digitales y se encuentran a nuestro alcance; pero, cómo encontrar un documento específico entre tanta información.

Cada día en algún lugar surge nueva información, con el decursar del tiempo esa información crece y crece, cuando alguien va a consultar un documento se le hace muy difícil encontrarlo; imaginemos qué tan difícil es encontrar un artículo entre millones y millones que están al alcance. Los sistemas de recuperación de información surgieron para facilitarnos la vida, ellos nos ayudarán a encontrar lo que buscamos de una manera mucho más rápida y efectiva.

## 2 Introducción

“La recuperación de información intenta resolver el problema de encontrar y rankear documentos relevantes que satisfagan la necesidad de información de un usuario, expresada en un determinado lenguaje de consulta”. [1]

Un proceso de recuperación de información comienza cuando un usuario hace una consulta al sistema. Una consulta constituye la necesidad de una información por parte del usuario. En la recuperación de información varios documentos pueden ser respuesta a una consulta con diferentes grados de relevancia. La mayoría de los sistemas computan un ranking para saber que tan bien el documento responde a la consulta, ordenando los documentos de acuerdo a su valor de ranking.

Este proyecto tiene como objetivo desarrollar un Sistema de Recuperación de Información que sea capaz de procesar determinadas colecciones. En las secciones siguientes se estará explicando el diseño y desarrollo de dicho sistema, se hará una evaluación del mismo teniendo en cuenta las métricas objetivas y subjetivas, haciendo uso de dos colecciones de prueba Cranfield y Medline. Por último, se realizará un análisis de las ventajas y desventajas del sistema, se plantearán algunas recomendaciones y las conclusiones.

## 3 Diseño del sistema

### 3.1 ¿Qué modelo es conveniente usar?

Para la realización del proyecto se escogió el modelo vectorial, pues, este se basa en la similitud que hay entre la consulta y los documentos. Comparando las ventajas y desventajas estudiadas se decidió que este era un buen modelo, aunque asume una independencia entre los términos, el modelo **tf-idf** mejora el

rendimiento y además se ordenan los documentos de acuerdo al grado de similitud con la consulta. Además, investigando un poco se vio que este era un modelo muy usado en diferentes sistemas de recuperación de información.[2]

### 3.2 ¿Qué otras herramientas se pueden usar?

- Los diccionarios son una herramienta muy útil para guardar información, es por ello que este proyecto hace uso de los mismos para guardar e intercambiar información, es una de las herramientas principales usadas en el sistema, son convenientes para guardar y acceder de una manera sencilla al *tf*, *idf* y *peso*(se explicarán más adelante).
- Una forma sencilla de expandir una consulta es procesarla también con los sinónimos de los términos de la misma.
- Puede ser usado algún algoritmo de clustering para agrupar los documentos.
- Una interfaz visual haría más fácil y sencillo el trabajo del usuario con el sistema.

## 4 Implementación del sistema

### 4.1 Procesamiento y representación de documentos

El procesamiento y representación de documentos se realizó en **model.py** en la función *documents*. Primeramente, se obtendrán los documentos de la colección(tienen que estar en formato .txt, pues es el que lee el proyecto), una vez obtenidos, se pasará a hacer el procesamiento textual de cada documento, guardando en una lista los términos obtenidos por cada uno.

El procesamiento textual se puede encontrar en **text\_processing.py**, donde se hace uso de la biblioteca *nltk*, donde, el proceso de tokenización se llevará a cabo, a partir, de *word\_tokenize*, en el cual, cada palabra se convierte en un token; se continua eliminando las stopwords, las palabras que no aportan ningún significado, a través de *corpus.stopwords* y con *SnowballStemmer* y *.stem* se realiza el proceso de lemmatizing, o sea, se reducen las palabras a su raíz gramatical.

Luego del procesamiento textual se vuelve a la función *documents*, donde se construirá un diccionario, en el cual las llaves serán los términos y los valores serán la frecuencia de ocurrencia del término  $t_i$  dentro de todos los documentos de la colección(*idf*) y todos los documentos en los que se encuentra el término(cada documento a su vez guardará también el valor del *tf* y *peso(w)*)

correspondiente a ese documento con ese término), inicialmente los valores de **idf** y peso serán 0 y posteriormente se calcularán y actualizarán, para la construcción de este diccionario se llama a la función *add\_terms* que se encuentra en **utils\_model.py**.

Cabe aclarar cómo se calculó el valor de **tf** en *add\_terms*. Es importante recordar que:

$$tf_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}} \quad (1)$$

donde  $freq_{i,j}$  es la frecuencia del término  $t_i$  en el documento  $d_j$  y el máximo se calcula sobre todos los términos del documento  $d_j$ . [2]

Para calcular la frecuencia de un término en el documento se usó *FreqDist* de *nltk*, luego se busca cuál de los términos tiene la máxima frecuencia, y posteriormente, se calcula el **tf** de cada término( ubicándolo en el diccionario antes mencionado) dividiendo la frecuencia de cada término entre la máxima frecuencia.

Regresando a la función *documents* se manda a actualizar los valores de **idf** y peso(pues, se habían inicializado en el diccionario con 0), a través de las funciones *add\_idfs* y *add\_w*, respectivamente, que se encuentran en **utils\_model.py**.

Se debe recordar que:

$$idf = \log \frac{N}{n_i} \quad (2)$$

donde  $N$  es la cantidad total de documentos y  $n_i$  la cantidad de documentos en los que aparece el término  $t_i$ . [2]

Como en el diccionario se tiene guardado como valor los documentos en los que el término aparece, buscando len de estos documentos se tiene la cantidad de documentos en los que aparece, y conociendo ya la cantidad de documentos de la colección, es fácil calcular el idf para cada término, sería el logaritmo de la cantidad de documentos de la colección entre el len de los documentos en los que se encuentra el término.

Recordar también que:

$$w_{i,j} = tf_{i,j} \times idf_i \quad (3)$$

Conociendo ya el valor del *tf* y del *idf* de cada término por cada documento calculamos el peso en *add\_w*, actualizándolo en el diccionario.[2]

Por último, la función *documents* guarda en una variable global *dictionary* el diccionario construido, esto se hace haciendo uso de la función *\_create\_* que se encuentra en **utils.py**.

## 4.2 Procesamiento y representación de consultas

Para el procesamiento y representación de consultas, primeramente, como forma de expansión de consultas, en **expansion\_query.py** se buscan los posibles sinónimos de los diferentes términos de la consulta, a través, de *nltk.corpus.wordnet.synsets*, posteriormente, se mandan a procesar todos los términos originales de la consulta junto con sus sinónimos.

En **model.py** en la función *query* se realizará el procesamiento y representación de la consultas. Al igual que en los documentos se realiza el procesamiento textual con **text\_processing.py**, igualmente, con *add\_terms* construimos un diccionario en el que las llaves son los términos y los valores serán el *idf* y todos los documentos en los que se encuentra el término(en este caso no son los documentos, sino, solamente la consulta). Igualmente se le actualizan los valores de *tf*, *idf* y *w*.

## 4.3 Similitud

Recordar que:

$$sim(d_j, q) = \frac{\vec{d_j} * \vec{q}}{|\vec{d_j}| * |\vec{q}|} \quad (4)$$

Luego de todo el procesamiento y representación de las consultas, se pasa a calcular la similitud entre los documentos y la consulta, ahí seguido en la función *query* de **model.py**, se llama a la función *sim* que se encuentra en **utils\_model.py**.

En la función *sim*, primeramente se pasa a buscar los documentos que contienen términos de la consulta, con ayuda de la función *documents\_with\_query* que su implementación se encuentra en el mismo **utils\_model.py**, esto se realiza porque cuando se pase a calcular la  $\vec{d_j} * \vec{q}$  solo tendrán incidencia aquellos términos presentes tanto en la consulta como en el documento, pues, los pesos se multiplican y se suman sucesivamente, por ejemplo, un término de la consulta que no se encuentre en el documento tendrá peso 0 en ese documento, pues, su frecuencia ahí sería 0. Teniendo los documentos que contienen términos de la

consulta se procede a calcular  $\vec{d}_j * \vec{q}$ . Luego, con las funciones  $dj$  y  $q$  se calculan  $|\vec{d}_j|$  y  $|\vec{q}|$ , respectivamente. Por último, se retorna  $\frac{\vec{d}_j * \vec{q}}{|\vec{d}_j| * |\vec{q}|}$ .

#### 4.4 Clustering

También se decidió implementar un algoritmo de clustering: *kmeans*, utilizando la biblioteca *sklearn.cluster.KMeans*. Primeramente, se le indican cuántos clústeres se quieren formar que en este caso son 4. Luego se le pasa al algoritmo la matriz de pesos de los términos de los documentos, haciendo uso de *sklearn.feature\_extraction.text.TfidfVectorizer* y *fit\_transform*. De esta forma se tienen agrupados los documentos en 4 clústeres, la implementación se encuentra en **cluster.py**.

#### 4.5 Interfaz de usuario

En esta sección se estará explicando el funcionamiento de la interfaz de usuario implementada, no se hará un análisis de la implementación de la misma puesto que no es objetivo del proyecto, solo se dirá que su implementación se encuentra en **user\_interfaz.py** y que se hizo uso de la biblioteca *PyQt5* para ello, para hacer uso de esta interfaz debe ejecutarse el archivo antes mencionado.[3]

#### 4.6 Manual de usuario

Cuando se inicia la aplicación, esta luce como se observaba en la Fig. 1.

Si se da clic en los 3 puntos que se encuentran en la sección Documents, se da la opción de seleccionar la dirección de la colección a procesar. Una vez seleccionada, saldría en esta sección un cartel que dice “Processing Documents”, mientras se estén procesando los documentos, como se muestra en la Fig. 2.

Una vez procesados los documentos, en la línea donde se seleccionó la dirección aparecerá esta, luego en el recuadro “Documents”, en la primera columna aparecerá el clúster al que pertenece el documento y en la segunda columna el documento. Esto se muestra en la Fig. 3.

Luego para hacer una consulta, en la sección “Query” en la línea de texto se introduce la consulta que se desea hacer y se da clic en el botón “Search”. Luego de procesar dicha consulta en la sección “Results” se muestran los documentos que fueron recuperados, en la primera columna el clúster al que pertenece el documento y en la segunda el documento. Esto se puede observar en la Fig. 4.

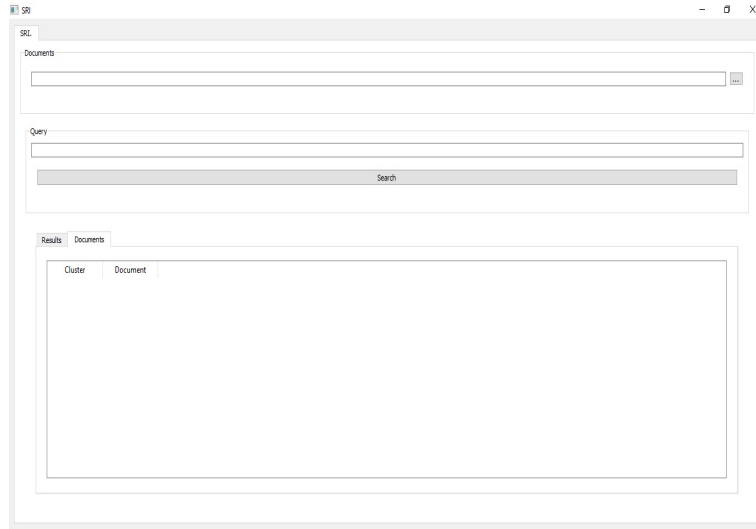


Fig. 1.

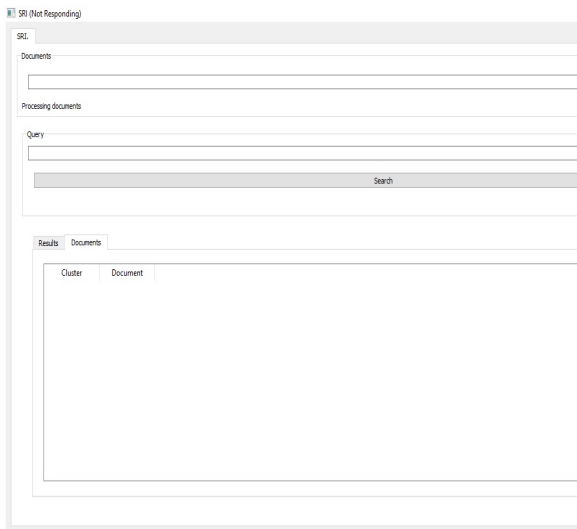


Fig. 2.



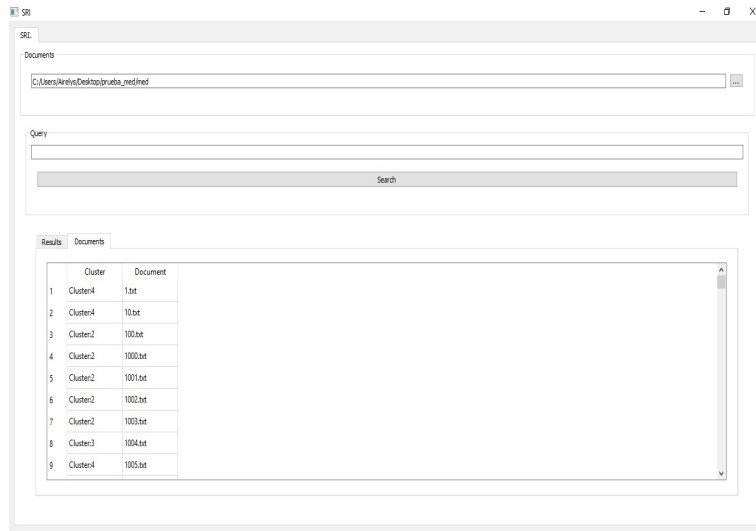


Fig. 3.

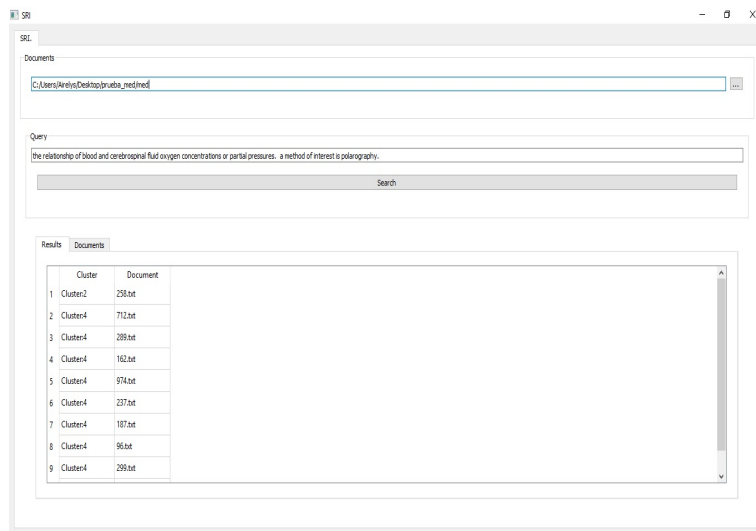


Fig. 4.

## 5 Evaluación del sistema

Para la evaluación del sistema se usaron las colecciones Cranfield y Medline, las cuales se dejan junto con este reporte y el código fuente del sistema. Las colecciones se tienen en *.txt* pues es el formato que recibe el proyecto, cada documento de la colección está en un *.txt* distinto. Las colecciones no están cargadas para que se vea el procesamiento de documentos, el cual se demora un poco, pues son algo extensas las colecciones.

### 5.1 Medidas usadas

Para evaluar el sistema se implementaron un conjunto de medidas que los evalúa de acuerdo a los documentos que recupera y a los que no. La implementación de estas medidas se encuentra en `evaluation_measures.py` y la aplicación de estas se encuentra en `evaluation.py`.

Antes de continuar es importante conocer algunos conjuntos:

- **REL** conjunto de documentos relevantes
- **REC** conjunto de documentos recuperados
- **RR** conjunto de documentos relevantes recuperados
- **NN** conjunto de documentos no relevantes no recuperados
- **RI** conjunto de documentos recuperados irrelevantes
- **NR** conjunto de documentos no recuperados relevantes

**Precisión:** evalúa la consulta de acuerdo a los documentos relevantes que se recuperan.[2]

$$P = \frac{|RR|}{|RR \cup RI|} = \frac{|RR|}{|REC|} \quad (5)$$

Después de tener los documentos que fueron recuperados en la consulta, se buscan cuáles son relevantes con ayuda de la información brindada por la colección, y teniendo esto, se aplica la fórmula. Este proceso se realizó para cada consulta de la colección y los resultados obtenidos fueron promediados.

**Recobrado:** fracción de los documentos relevantes que fueron recuperados.[2]

$$R = \frac{|RR|}{|RR \cup NR|} = \frac{|RR|}{|REL|} \quad (6)$$

Con ayuda de la información que brinda la colección se buscan los documentos relevantes para la consulta, teniendo ya los documentos recuperados se va

analizando de estos cuáles son relevantes e igualmente se aplica la fórmula. El proceso se realiza para cada consulta de la colección y los resultados obtenidos se promedian.

**Medida F:** enfatiza la precisión sobre el recobrado o viceversa.[2]

$$F = \frac{(1 + \beta^2)PR}{\beta^2 P + R} \quad (7)$$

$\beta = 1$ : mismo peso para la precisión y recobrado  $\beta > 1$ : mayor peso para la precisión  $\beta < 1$ : mayor peso para el recobrado

Para calcularla se usaron las dos medidas anteriores y se aplicó la fórmula. Igualmente, el proceso se realiza para cada consulta de la colección y los resultados obtenidos se promedian.

**Medida  $F_1$ :** armoniza precisión y recobrado teniendo en cuenta a ambos.[2]

$$F_1 = \frac{2PR}{P + R} \quad (8)$$

Para su cálculo se usó el cálculo de la precisión y el recobrado y se aplicó la fórmula. Igualmente, el proceso se realiza para cada consulta de la colección y los resultados obtenidos se promedian.

**R-Precisión:** es el ranking de documentos relevantes a una consulta para la cual existen R documentos relevantes.[2]

$$P_R = \frac{|RR|_R}{|RR \cup RI|_R} = \frac{|RR|_R}{|REC|_R} \quad (9)$$

Para su cálculo se seleccionaron los primeros R documentos recuperados y de estos se vieron cuáles son relevantes, de acuerdo a la información que brinda la colección y se aplica la fórmula. El proceso se realiza para cada consulta de la colección y los resultados obtenidos se promedian.

**Proporción de fallo:** tiene en cuenta la cantidad de documentos irrelevantes y el ranking.[2]

$$Fallout = \frac{|RI|}{|RI \cup NI|} = \frac{|RI|}{|I|} \quad (10)$$

Con la diferencia de los documentos recuperados y de los recuperados relevantes, se tiene el conjunto de los documentos recuperados irrelevantes; con la diferencia del total de documentos y los documentos relevantes se tiene el conjunto de los documentos irrelevantes; luego, se aplica la fórmula y se repite el

proceso para cada consulta de la colección y los resultados se promedian.

## 5.2 Análisis de los resultados

Los análisis de los resultados se encuentran en **tevaluation.py** donde en la línea 8 se especifica la dirección de la colección de los documentos, en la línea 18 el documento con las consultas a realizar, en la línea 45 la dirección del archivo con las relaciones de relevancia y en la línea 12 el umbral de similitud.

Para el análisis de los resultados se utilizó la colección Cranfield que cuenta con 1400 documentos y 225 consultas.[4]

Resultados promedios para un umbral de similitud de 0.09:

- Precisión: 0.10901124131401935
- Recobrado: 0.6467905483092768
- Medida  $F$   $\beta = 0$ : 0.10901124131401935
- Medida  $F_1$ : 0.1694418791771936
- Medida  $F$   $\beta = 2$ : 0.27972846259960443
- $R - precision_10$ : 0.2564444444444453
- Proporción de fallo: 0.038643443412114016

Se observa que el recobrado promedio es alto, un resultado bueno para esta medida, sin embargo, la precisión es bastante baja, aunque la  $R - precision_10$  es un poco más alta que la última. La proporción de fallo tiene un bajo valor.

Resultados promedios para un umbral de similitud de 0.13:

- Precisión: 0.08264034598980041
- Recobrado: 0.17700588420630056
- Medida  $F$   $\beta = 0$ : 0.08264034598980041
- Medida  $F_1$ : 0.09549277220095578
- Medida  $F$   $\beta = 2$ : 0.12221300248442155
- $R - precision_10$ : 0.08533333333333333
- Proporción de fallo: 0.012351378630590288

Con un umbral de similitud de 0.13 se observa que disminuyó considerablemente tanto la precisión como el recobrado. Los resultados son malos para este umbral.

Resultados promedios para un umbral de similitud de 0.11:

- Precisión: 0.2008825095059054
- Recobrado: 0.5171964188598902

- Medida  $F \beta = 0$ : 0.2008825095059054
- Medida  $F_1$ : 0.24577953319539336
- Medida  $F \beta = 2$ : 0.3341486091074317
- $R - precision_10$ : 0.2537777777777794
- Proporción de fallo: 0.015071520080302098

Si se fija el umbral de similitud en 0.11 se observa un recobrado bueno, la precisión, aunque es un poco baja aún, es la más alta obtenida. Los mejores valores para la Medida  $F$  y la Medida  $F_1$  son los obtenidos con este umbral.

Para el análisis de los resultados también se utilizó la colección Medline que cuenta con 1033 documentos y 30 consultas.[4]

Resultados promedios para un umbral de similitud de 0.12:

- Precisión: 0.5360821109288602
- Recobrado: 0.3127791713751234
- Medida  $F \beta = 0$ : 0.5360821109288602
- Medida  $F_1$ : 0.361086724788996
- Medida  $F \beta = 2$ : 0.32718245210365
- $R - precision_10$ : 0.4433333333333333
- Proporción de fallo: 0.0037987334206573643

En este caso, la precisión es alta, sin embargo el recobrado es un poco bajo, aunque la  $R - precision_10$  es buena. La proporción de fallo es pequeña.

Resultados promedios para un umbral de similitud de 0.1:

- Precisión: 0.5700169377692599
- Recobrado: 0.43652582642712573
- Medida  $F \beta = 0$ : 0.5700169377692599
- Medida  $F_1$ : 0.4501325525357391
- Medida  $F \beta = 2$ : 0.43590182433222535
- $R - precision_10$ : 0.56
- Proporción de fallo: 0.0053463336700038805

Ahora, se observa una mejoría en el recobrado y la precisión sigue buena, la  $R - precision_10$  también aumentó.

Resultados promedios para un umbral de similitud de 0.09:

- Precisión: 0.5359934670090849
- Recobrado: 0.5021524639279512
- Medida  $F \beta = 0$ : 0.5359934670090849
- Medida  $F_1$ : 0.47893542575549086

- Medida F  $\beta = 2$ : 0.4853989717474617
- $R - precision_10$ : 0.6000000000000001
- Proporción de fallo: 0.007766367574079214

Si se fija el umbral de similitud de 0.09 se observa una compensación de la precisión y recobrado, donde la Medida F con  $\beta = 0$  y  $\beta = 2$  y la Medida  $F_1$  la manifiestan. La  $R - precision_10$  es muy buena.

## 6 Ventajas y desventajas del sistema desarrollado

El proyecto desarrollado indica ser aceptable, pues como se vio en la sección anterior da respuestas bastante buenas. Las métricas que se usan para evaluar los sistemas de recuperación de información dieron buenos resultados. Se tiene implementada una forma de expansión de consulta y un algoritmo de clustering. Se tiene una interfaz visual que es amigable para el usuario. Además, si se desea añadir alguna nueva funcionalidad no se pasaría mucho trabajo porque está hecho y pensado para que futuros autores puedan trabajar en él fácilmente.

Aunque el sistema da buenos resultados, la parte de procesar los documentos se demora un poco, sobre todo para colecciones muy grandes y con documentos extensos. Otra desventaja es que solo lee documentos en formato *.txt*. Además, el sistema solo puede ser usado localmente, no en la web.

## 7 Recomendaciones y Conclusiones

Además de las técnicas utilizadas en este proyecto se considera que pueden añadirse otras que mejoren mucho más la propuesta. Primeramente, el método de leer documentos se puede ampliar más para que se lean otros formatos como *.pdf*, *.docx*, *.html*, etc; pues el de este proyecto solo lee *.txt*. También puede proponerse algún mecanismo de optimización para que se realice un poco más rápido el procesamiento de documentos, pues como ya se mencionó se demora un poco en procesarlos. Se pueden añadir otras formas de expandir las consultas como los tesauros. Además, se puede añadir crawling como forma de recuperación en la web.

Siempre los proyectos se pueden mejorar un poco más, siempre queda algo por hacer que lo mejore.

Desarrollar un Sistema de Recuperación de información es algo trabajoso y que lleva estudio. Hay que intentar responderle al usuario de la mejor manera posible, o sea, se deben satisfacer sus peticiones. Cada decisión que se tome es pensando si dará mejores resultados. Son varias fases por las que hay que pasar para mostrar buenas respuestas y siempre intentando hacer un poquito más.

## 8 Referencias

- [1]. Tolosa y Bordinon(2007)
- [2] Conferencias de Sistemas de Recuperación de Información.
- [3] Primeros pasos en PyQt5 y QtDesigner: Programas gráficos con python.  
(medium.com)
- [4] [http://ir.dcs.gla.ac.uk/resources/test\\_collections/](http://ir.dcs.gla.ac.uk/resources/test_collections/)