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
5 Evaluación del sistema 7
6 Ventajas y desventajas del sistema desarrollado 7
7 Recomendaciones y Conclusiones 8
8 Referencias 8

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 díficil 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 sitema, 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.

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:

$$tf_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}} \tag{1}$$

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.

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:

$$idf = \log \frac{N}{n_i} \tag{2}$$

donde `N` es la cantidad total de documentos y n_i la cantidad de documentos en los que aparece el término t_i.

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:

$$w_{i,j} = tf_{i,j} \times idf_i \tag{3}$$

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.

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:

$$sim(d_j, q) = \frac{\overrightarrow{d_j} * \overrightarrow{q}}{|\overrightarrow{d_j}| * |\overrightarrow{q}|} \tag{4}$$

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 $\overrightarrow{d_j} * \overrightarrow{q}$ 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 $\overrightarrow{d_j} * \overrightarrow{q}$. Luego, con las funciones dj y q se calculan $|\overrightarrow{d_j}|$ y $|\overrightarrow{q}|$, respectivamente. Por último, se retorna $\frac{\overrightarrow{d_j} * \overrightarrow{q}}{|\overrightarrow{d_j}| * |\overrightarrow{q}|}$.

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 implementción 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 implemtación 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.

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.

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.

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

6 Ventajas y desventajas del sistema desarrollado

El proyecto desarrollado indica ser bastante bueno, pues como se vio en la sección anterior da buenas respuestas. 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 sitema 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 desición 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 Bordignon(2007)

[2]. Reducir el número de palabras de un texto: lematización y radicalización(stemming) con python. (medium.com)

[3]. Técnicas avanzadas de recuperación de información.(tecnicasrecuperacioninformacion.blogspot.com)

[4]. Cómo obtener sinónimos/antónimos de nltk wordnet en python. (es.acervolima.com)

[5]. Primeros pasos en PyQt5 y QtDesigner: Programas gráficos con python. (medium.com)

[6]. Conferencias de Sistemas de Recuperación de Información.