Proyecto de Programación I.
Facultad de Matemática y Computación.
Universidad de La Habana. Curso 2022.

Miguel Alejandro Yáñez Martínez
Grupo C112

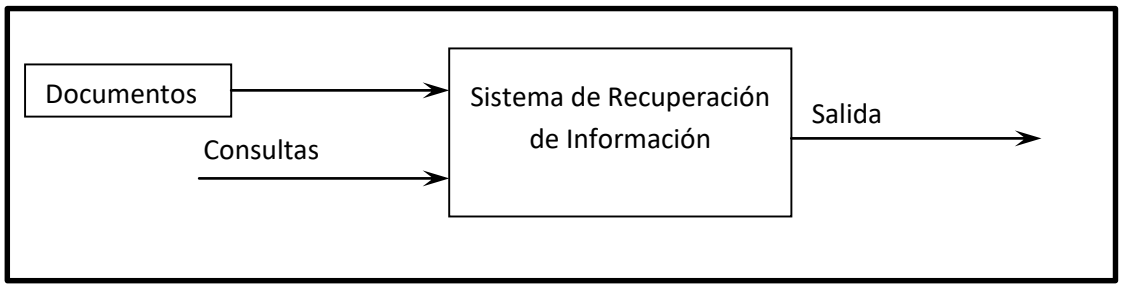
Introducción

La Recuperación de Información tiene sus orígenes en las bibliotecas y centros de documentación en los que se requerían búsquedas bibliográficas de libros y artículos de revista. El objetivo principal de cualquier centro de documentación es satisfacer las necesidades reales y potenciales de información de todos los usuarios, proporcionándoles la información veraz, pertinente, justo a tiempo y al menor coste.

En este proyecto desarrollaremos el Sistema de Recuperación de Información (SRI) Moogle!, aplicación totalmente original cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos.

La Recuperación de Información.

El objetivo de la recuperación de información es, dada una necesidad de información y un conjunto de documentos, ordenar los documentos de más a menos relevantes para esa necesidad y presentarlos al usuario. Para ilustrar el problema nos ayudamos de la figura siguiente:



Sistema de recuperación de información.

En esa figura se ha supuesto que el sistema de recuperación de información es una caja negra que acepta documentos y consultas, y que obtiene una salida, que es el conjunto de documentos que satisfacen la consulta. El problema principal en este sistema es obtener representaciones homogéneas de documentos y consultas, y procesar convenientemente esas representaciones para obtener la salida.

En el proceso de recuperación de información se suelen distinguir las siguientes etapas:

1. Obtener representación de los documentos. Generalmente los documentos se presentan utilizando un conjunto más o menos grande de términos índice. La elección de dichos términos es el proceso más complicado.
2. Identificar la necesidad informativa del usuario. Se trata de obtener la representación de esa necesidad, y plasmarla formalmente en una consulta acorde con el sistema de recuperación.
3. Búsqueda de documentos que satisfagan la consulta. Consiste en comparar las representaciones de documentos y la representación de la necesidad informativa para seleccionar los documentos pertinentes.
4. Obtención de resultados y presentación al usuario.

Descripción del Proyecto

La búsqueda en nuestro proyecto es lo más inteligente posible. No nos limitamos a los documentos donde aparece exactamente la frase introducida por el usuario, sino que el usuario puede buscar no solo una palabra sino en general una frase cualquiera.

Si no aparecen todas las palabras de la frase en un documento, pero al menos aparecen algunas, este documento también es devuelto, pero con un peso menor mientras menos palabras aparezcan.

El orden en que aparezcan en el documento los términos de la consulta en general no importa, ni siquiera que aparezcan en lugares totalmente diferentes del documento.

Si en diferentes documentos aparecen la misma cantidad de palabras de la consulta, pero uno de ellos contiene una palabra más rara porque aparece en menos documentos, el documento con palabras más raras tiene un peso más alto, porque es una respuesta más específica.

De la misma forma, si un documento tiene más términos de la consulta que otro, tiene un peso más alto, a menos que sean términos menos relevantes.

Las palabras excesivamente comunes como las preposiciones, conjunciones, etc. y aquellas que se repiten muchas veces, son ignoradas por completo ya que aparecerán en la inmensa mayoría de los documentos. Las mismas se computan de los documentos.

Operadores

El proyecto cuenta con varios operadores para mejorar la búsqueda del usuario:

- Exclusión, identificado con un ! delante de una palabra, indica que la palabra no debe aparecer en ningún documento devuelto.
- Inclusión, identificado con un ^ delante de una palabra, indica que la palabra debe aparecer en todos los documentos devueltos.

- Mayor Relevancia, identificado por varios * delante de una palabra, indica que la palabra es más relevante que las demás palabras de la consulta tantas veces como * tenga delante de ella.
- Cercanía identificado con un ~ entre las palabras indica que los documentos en los que aparecen estas dos palabras más cerca tendrán mayor peso.

Sugerencia

Para brindar una mayor exactitud en la búsqueda, el proyecto cuenta con un corrector de palabras, el cual se encarga de dar una sugerencia al usuario en caso de que la búsqueda no coincida con los datos almacenados.

Ideas extras

Con el fin de que nuestra aplicación sea mucho mejor, implementamos en la búsqueda que si las palabras exactas no aparecen, pero aparecen palabras derivadas de la misma raíz, también devuelve esos documentos, pero con un peso menor.

También, si aparecen palabras relacionadas aunque no tengan la misma raíz, devolvemos esos documentos pero con menor peso que si apareciera la palabra exacta o una de la misma raíz.

Resultados de la Búsqueda

Una vez calculados los pesos de los documentos, se le muestra al usuario una lista de los mismos ordenados de mayor a menor peso, con el título, el peso y el fragmento donde se encontraron las palabras buscadas por este.

Algoritmos de Búsqueda. El Modelo Vectorial

La búsqueda está basada en el modelo vectorial de recuperación de la información SRI, utilizando el TF-IDF (frecuencia de término - frecuencia inversa de documento), el cual determina la relevancia de una palabra asociada a un documento en una determinada colección, sumado a la Similitud del Coseno, método mediante el cual se asigna un peso a cada documento y se establece un ranking de resultados para el usuario.

Fue definido por Salton [Salton, 1968] hace ya bastantes años, y es ampliamente usado en operaciones de recuperación de información. En el modelo vectorial se intenta recoger la relación de cada documento Di, de una colección de N documentos, con el conjunto de las m características de la colección. Formalmente un documento puede considerarse como un vector que expresa la relación del documento con cada una de esas características.

Si se consideran los términos como características definitorias del documento, el proceso que debe seguir el sistema pasa primero por seleccionar aquellos

términos útiles que permitan discriminar unos documentos de otros. En este punto debemos señalar que no todas las palabras contribuyen con la misma importancia en la caracterización del documento. Desde el punto de vista de la recuperación de información existen palabras casi vacías de contenido semántico, como los artículos, preposiciones o conjunciones, que son poco útiles en el proceso.

Pero también son poco importantes aquellas palabras que por su alta frecuencia de aparición en toda la colección de documentos pierden su poder de discriminación. Todas ellas forman parte del conjunto de palabras vacías, que se eliminan en el proceso de indexación. Además de la eliminación de palabras vacías, en el proceso se pueden incluir aplicaciones léxicas como lematización o extracción de raíces.

Una vez seleccionado el conjunto de términos caracterizadores de la colección de documentos, es necesario obtener el valor de cada elemento del vector del documento.

El proceso realizado para los documentos también puede aplicarse a las consultas. Una consulta realizada en lenguaje natural está formada por términos, y, por tanto, puede verse como un documento más. Así el mecanismo de obtención de pesos también se aplica a las consultas, para de esta manera poder disponer de representaciones homogéneas de consultas y documentos, que posibiliten obtener el grado de similitud entre ambas representaciones.

La resolución de la consulta consiste en un proceso de establecer el grado de semejanza entre el vector consulta y el vector de cada uno de los documentos. Para una consulta determinada, cada documento arrojará un grado de similitud determinado; aquéllos cuyo grado de similitud sea más elevado se ajustarán mejor a las necesidades expresadas en la consulta, desde el punto de vista del sistema de recuperación de información.

El modo más simple de calcular la similitud entre una consulta y un documento, utilizando el modelo vectorial, es realizar el producto escalar de los vectores que los representan. El índice de similitud (al cual llamaremos informalmente peso del documento) más utilizado es el coseno del ángulo formado por ambos vectores, el cual es utilizado en nuestro proyecto.

Ejecutando el proyecto

Los documentos en los que quiere desarrollar la búsqueda se deben colocar en la carpeta Content, en formato texto plano. En nuestro caso hemos utilizado a manera de prueba los artículos periodísticos escritos por nuestro Héroe Nacional José Martí.

Este proyecto está desarrollado para la versión objetivo de .NET Core 6.0. Para ejecutarlo debe ir a la ruta en la que está ubicado el proyecto y ejecutar en la terminal de Linux:

make dev

Si está en Windows, debe poder hacer lo mismo desde la terminal del WSL (Windows Subsystem for Linux), en caso contrario puede ejecutar:

dotnet watch run --project MoogleServer

Implementación MoogleEngine

Estructura de la biblioteca de clases MoogleEngine.

Clases Auxiliares

Estas clases brindan funcionalidades al proyecto, encapsulando procesos necesarios para las clases principales, evitando repetir código en cada una de ellas.

Clase Utiles.

Esta clase es la encargada de brindar al resto de las clases propiedades y métodos que son utilizados de forma indistinta. En ella se definen los caminos de los archivos de los documentos, del archivo de sinónimos, métodos básicos del procesamiento de los textos y de cálculos, tipos de datos nuevos, etc.

Clase Sinonimos.

Esta clase se utiliza para buscar los sinónimos de una palabra. Tenemos el archivo sinónimos.txt que contiene en cada una de sus líneas la lista de palabras separadas por comas que son sinónimos entre sí. El método BuscarSinonimos busca los sinónimos de una palabra. Tiene como entrada la palabra a la que se le quiere buscar sus sinónimos y como salida la lista con los sinónimos de la palabra.

Clase ListadoPalabrasVacias.

Esta clase busca las palabras vacías en los documentos creando los valores de la propiedad PalabrasVacias de la clase Utiles. Brinda el método CreaListado() que devuelve como salida el listado de las palabras vacías que aparecen en los documentos de la colección.

Debido a que en la consulta pueden aparecer palabras vacías como lo artículos, las preposiciones, las conjunciones, etc., que no necesariamente aparecen todas en los documentos, comenzamos el proceso adicionando a nuestra lista los artículos, las preposiciones, las conjunciones, los pronombres y los adverbios del español.

Después creamos una lista de las palabras que aparecen en los documentos y de las veces que aparece cada una. Tomando en cuenta las veces que aparece cada palabra, ordenamos este listado de menor a mayor y buscamos la mayor cantidad posible de palabras que aparezcan en los documentos y que la suma de sus apariciones sea menor que la suma de las apariciones de las palabras que más aparecen. Esto nos brinda un punto de corte para diferenciar las palabras vacías de las palabras buenas.

Las palabras que no estén dentro de las palabras buenas y que serán también las que más veces aparecen, las tomamos como palabras vacías.

Clase Lematizador.

Esta clase es la encargada de sacar las raíces de las palabras para buscar los lemas. Es una variación del algoritmo de Porter para el español, publicada en jesusutrera.com/articles/article01.html. La misma se ha modificado para adaptarla a nuestro programa.

El algoritmo de Porter [Porter, 1980], se basa en una serie de reglas condición / acción. Tiene la finalidad de obtener la raíz de una palabra (lexema) y consta de una serie de pasos, en cada uno de los cuales se aplican sucesivamente una serie de reglas (reglas de paso) que van suprimiendo sufijos o prefijos de la palabra hasta que nos quedamos sólo con el lexema final.

Nuestra clase consta con el método Lematizar, el que recibe como entrada la palabra de la que se quiere obtener la raíz y devuelve como salida la raíz de la palabra. Es utilizada por las clases Documento y Consulta.

Clases Principales

Clase Documento

Esta clase es la encargada de gestionar todo lo relativo a los documentos sobre los que se realizaran las consultas. Al llamar a su constructor se le pasa el camino de uno de los documentos contenidos en la colección de la carpeta Content, el cual es cargado a la memoria y procesado. Al procesar el documento se eliminan las palabras vacías, se extraen y normalizan los términos y se crean los lemas de estos.

Almacena el texto del documento, los términos que aparecen en el mismo y los lemas de los mismos. Brinda métodos para obtener una lista de las posiciones donde aparece un término en el documento, la cantidad de veces que aparece, si existe un término en el documento y la distancia mínima en el documento entre dos términos. También tiene métodos para obtener la cantidad de veces que aparece un lema y si existe un lema en el documento. El método ExtraerOracion extrae del texto del documento una oración con la mayor cantidad de palabras posibles de una lista que le pasamos (términos que

aparecen en la consulta) y que se utiliza para buscar el fragmento del texto que se devuelve junto con los resultados de la búsqueda.

Clase Consulta

Esta clase es la encargada de gestionar todo lo relativo a las consultas. Al llamar a su constructor se le pasa la consulta, de la que se eliminan las palabras vacías, se normalizan los términos y se procesan los operadores.

Almacena el texto de la consulta y los términos que aparecen en la misma. Brinda métodos para obtener los términos que aparecen en la consulta y la cantidad de veces que aparece cada uno, los términos obligatorios (operador ^), los términos prohibidos (operador !), los términos importantes (operador *) y los términos cercanos (operador ~).

Clase Núcleo

Esta clase es el núcleo del proyecto. Realiza el proceso de la búsqueda de la consulta en los documentos de la colección y calcula los pesos de cada uno.

Al llamar a su constructor se crea la lista de documentos, los cuales se procesan y quedan listos para las consultas. También se crea la lista de palabras vacías y se inicia el mecanismo para buscar los sinónimos. Este método es invocado una vez cuando se inicia la aplicación y deja ordenada toda la información para cuando se realicen las consultas.

El método RealizarConsulta es el utilizado para realizar una consulta al grupo de documentos de la colección. Se le pasa como parámetro la consulta que se va a realizar y retorna un arreglo de SearchItem con los resultados de la consulta realizada. En la propiedad Sugerencia de esta clase se devuelve una sugerencia al usuario en caso de que la búsqueda no coincida con los datos almacenados.

Al realizar el usuario una consulta, el proceso realizado es el siguiente:

1. Del total de documentos de la colección, buscamos los documentos que tengan al menos un término de la consulta. Si no tienen ningún término, el peso del documento será 0, por lo que no nos interesan ya que no aportan ninguna información.
2. Los documentos obtenidos en el paso anterior son filtrados para eliminar del grupo aquellos que tengan términos prohibidos o no tengan términos obligatorios (según los operadores ! o ^ que aparezcan en la consulta).
3. Se crea el vector TF de la consulta.
4. Se crea la matriz con los vectores TF de los documentos.
5. Realizamos la revisión de la matriz TF de los documentos antes de pasarla al cálculo de los pesos.
6. Se realiza en cálculo de los pesos de los documentos utilizando el coseno del ángulo formado por el vector de la consulta y el vector de cada uno de los documentos.

7. Se ordenan los documentos de mayor a menor de acuerdo a los pesos de cada uno.
8. Se procede a crear el arreglo de SearchItem, incluyendo fragmento del texto que se devuelve junto con los resultados de la búsqueda.
9. Se devuelven los resultados.

En el paso 5 se realiza la revisión de la matriz TF de los documentos antes de pasarla al cálculo de los pesos, debido a que si hay alguna columna en cero significa que el término no se encontró en ningún documento y esto haría una división por 0 que invalidaría el cálculo.

La revisión se realiza en cuatro pasos que son los siguientes:

Paso 1: buscamos las columnas que están en cero y si hay alguna buscamos si la palabra está mal escrita y nos da el sistema una sugerencia. Para buscar la sugerencia buscamos en todas las palabras de los documentos aquella con un porciento de similitud mayor con respecto a nuestra palabra. Si hay una sugerencia, se devuelve una sugerencia al usuario y calculamos la frecuencia del nuevo término en los documentos, actualizando la columna de la matriz TF.

Paso 2: buscamos las columnas que están en cero y si hay alguna buscamos el lema de la palabra. Calculamos la frecuencia del lema en los documentos, actualizando la columna de la matriz TF. La cantidad de veces que aparece el lema la multiplicamos por 2, para que sea menor el peso del lema que el del término original.

Paso 3: buscamos las columnas que están en cero y si hay alguna buscamos los sinónimos de la palabra. Calculamos la frecuencia de los sinónimos en los documentos actualizando la columna de la matriz TF. La cantidad de veces que aparecen los sinónimos la multiplicamos por 3, para que sea menor el peso del sinónimo que el del lema y el del término original.

Paso 4: buscamos las columnas que están en cero y si hay alguna la eliminamos del vector TF de la consulta y de la matriz de vectores TF de los documentos.

En el paso 1, para buscar la sugerencia buscamos en todas las palabras de los documentos aquella con un porciento de similitud mayor con respecto a nuestra palabra. Este porciento de similitud lo calculamos utilizando la distancia de edición entre dos palabras según el algoritmo de Damerau-Levenshtein. Se le llama distancia de edición al número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación, bien una inserción, eliminación, sustitución o transposición de dos caracteres.

El código de este método lo tomamos de https://www.csharpstar.com/csharp-string-distance-algorithm/