Informe Escrito

Proyecto de Programación I. Facultad de Matemática y Computación. Universidad de La Habana
Estudiante: Johnangel Crespo Leal
Grupo: C112

-Primeras Ideas

Durante el desarrollo del proyecto, se crearon tres clases nuevas:

- Metodos
- TrabajoSinQuery
- Snippet

Y se modificó una clase ya existente:

- Moogle

-Acerca de las clases

Moogle

Solo contiene un método llamado "Query" el cual se explicará en la siguiente línea:
En este método básicamente se realiza el funcionamiento principal del motor de búsqueda, recibe la query que el usuario insertó y devuelve una variable del tipo "SearchResult" la cual contiene "items" y "suggestion". La variable "ítems" es una variable de tipo "SearchItem" la cual contiene "title" (es el título de cada documento), "snippet" (fragmento del texto que contiene alguna relación con la query) y "score" (nivel de relevancia de cada documento de acuerdo a la query insertada por el usuario). Por ahora la variable "suggestion" aún no posee un algoritmo que cumpla con la sugerencia, por lo tanto solamente devuelve la misma query insertada por el usuario de momento.
Al entrar la query del usuario se definen dentro del método varias variables que serán útiles para el resultado final (pueden ver cada variable y lo que guarda en los comentarios que hay justo al lado de cada variable en el método dentro del código).
A partir de aquí se comienza de verdad a trabajar con la query, se define una variable string llamada "NormQuery" que contiene a la propia query pero ya normalizada (los métodos que se usen en esta clase pero no sean de esta se explicarán en sus respectivas clases). Después se define un array de double llamado "SimCos" con los documentos que son relevantes con respecto a la query insertada (usando el cálculo de similitud del coseno el cual se explicará en la clase Metodos). Posteriormente se define una lista de tuplas llamada "DocOrdenados" que contiene los "score" en orden descendente de aquellos documentos que posean alguna relación con la query insertada por el usuario. Posteriormente se crea un array de tipo "SearchItem" llamado "items" del tamaño de la cantidad de los documentos que coinciden con la query, si la query llega vacía, se reconfigurará el tamaño de "items" a 1 y solo se devolverá al usuario un mensaje de advertencia, en cualquier otro caso se rellenará el array "items" con cada documento que coincide de alguna forma con la query (se pone su título, una porción del texto y un score).

TrabajoSinQuery

En esta clase básicamente se encuentra la mayoría de los métodos que se aplican sin necesidad de usar la query. A continuación se explicará cada método que se encuentre en esta clase:

TF()

- Recibe como parámetro un string el cual debe de ser un documento, una vez insertado el documento se separa el mismo en palabras y se calcula el TF a cada palabra en el documento, una vez calculado se devuelve un diccionario de la forma Dictionary<string,double> el cual contiene en su claves las palabras que se encuentran en el texto y en su valor su respectivo TF. (Para calcular el TF de una palabra del

documento se realiza dividendo la frecuencia de dicha palabra en el documento entre el total de palabras del documento.)

TotalTF()

- Recibe como parámetro un array de string que debería de contener los documentos normalizados. Básicamente se realiza un recorrido por cada documento y se le aplica a cada uno el método TF() anterior visto y devuelve un array de diccionarios de la forma Dictionary<string,double>[] el cual contiene en cada posición un diccionario de la forma Dictionary<string,double> los cuales contienen en sus claves las palabras de cada documento y en sus valores sus respectivos TF.

frecPalCorpusPorDoc()

- Recibe como parámetro un array de string que debería de contener los documentos normalizados, a pesar de tener un nombre raro, simplemente devuelve un diccionario de la forma Dictionary<string,int> el cual contiene como clave cada palabra del corpus de documentos y como valor la cantidad de veces que aparece en el corpus, es decir la cantidad de documentos que contiene la palabra en cuestión. (Lo único que me parece curioso de este método es que se implementó la colección genérica HashSet la cual es una buena opción cuando se necesita almacenar elementos únicos y no se requiere un orden específico, aunque también es útil para verificar rápidamente si un elemento ya pertenece en el conjunto)

IDF()

- Recibe como parámetros un diccionario de la forma Dictionary<string,int> con la frecuencia de cada palabra en el corpus (es decir, básicamente el diccionario devuelto en el método "frePalCorpusPorDoc()") y un array de string que contenga los documentos del corpus, en este caso no es obligatorio que estén normalizados los documentos debido a que solo se utiliza para saber la cantidad de documentos que hay en el corpus. Este método devuelve un diccionario de la forma Dictionary<string,double> el cual contiene el IDF de cada palabra del corpus de documentos. (Para hallar el IDF de una palabra se calcula el logaritmo en base de 10 de la división del total de documentos entre la cantidad de documentos que contienen la palabra en cuestión, en este caso se le ha sumado 1 al numerador para evitar una división por 0)

TF-IDF()

- Recibe como parámetros un array de diccionarios el cual contiene en cada posición un diccionario de la forma Dictionary<string,double> que contiene los TF de cada documento (ósea el mismo array de diccionarios que contiene en cada posición los TF de cada palabra que devuelve el método "TotalTF()"), también recibe el diccionario que contiene los IDF de cada palabra del corpus de documentos que se consigue a partir del método anterior ("IDF()") y por último recibe un array de string que contiene los documentos los cuales no necesariamente hace falta que estén normalizados debido a que solo se usa para saber cuantos documentos hay en el corpus. (Para calcular el TF-IDF de una palabra se realiza la multiplicación de su propio TF en el documento por su IDF en el corpus de documentos)

Metodos

En esta clase se encuentra mayormente algunos métodos útiles que sirven para ayudar a métodos de otras clases, trabajar con la query y leer los documentos etc… A continuación se explicará cada método de esta clase:

Direcciones()

- No recibe ningún parámetro, este método solamente devuelve un array de string con las direcciones de cada documento. (Debido a que no trabajo con Linux, no tengo claro si el método detecte bien la carpeta Content en ese sistema, en caso que no lo detecte se debería de solucionar insertando manualmente la dirección en la variable string "dirección")

Titulos()

- Recibe un array de string el cual debería de contener las direcciones de los documentos del corpus (normalmente debería de ser el que devuelve el método "Direcciones()") y devuelve un nuevo array de string con los títulos de cada documento en cada posición respectivamente.

Contenido()

- Recibe como parámetro un array de string con las direcciones de los documentos del corpus ("Direcciones()") y devuelve un array de string con el contenido de cada documento en cada posición respectivamente.

ContenidoMinusculas()

- Recibe como parámetro un array de string con el contenido de cada documento (el visto en "Contenido()") y devuelve básicamente un array de string con los mismos contenidos de antes pero esta vez con todas las palabras en minúsculas. (Este método tiene su utilidad en la clase Snippet)

ContNorm()

- Recibe como parámetro un array de string con el contenido de cada documento (el visto en "Contenido()") y devuelve un array de string con los mismos contenidos pero esta vez normalizados. (Por cada string del array se le aplica el método "NormString()" el cual se explica justo debajo de este)

NormString()

- Recibe como parámetro un string que debe de ser un texto y devuelve un string con el texto ya normalizado. (Este método lo hice básicamente para volver usarlo con la query insertada por el usuario)

PalabrasSeparadas()

- Recibe como parámetros un string el cual debe de ser un texto (aunque también puede ser una simple palabra) y devuelve un array de string con cada palabra del texto insertado (funciona un tanto parecido a ".Split()")

QueryTF()

- Recibe como parámetro un string el cual debería de ser la query ya normalizada (aquí es cuando entra en acción para la query el método "NormString()" mencionado arriba) y devuelve un diccionario de la forma Dictionary<string,double> el cual contiene en sus claves las palabras de la query y en sus valores sus respectivos TF. (Este método requiere del método TF() de la clase "TrabajoSinQuery", aunque este método se use aquí, lo mantuve en dicha clase por un tema de organización)

QueryTFIDF()

- Recibe como parámetros un string el cual debería de ser la query ya normalizada y un diccionario de la forma Dictionary<string,double> el cual debe de contener los IDF de cada palabra del documento (el adquirido en "IDF()") y devuelve un diccionario de la forma Dictionary<string,double> en el cual cada clave contiene una palabra de la query y en cada valor contiene su respectivo TF-IDF.

SumCuadTermDoc()

- Recibe como parámetro un diccionario de la forma Dictionary<string,double> el cual debe de contener el TF-IDF de un documento (o query) y devuelve un double con la suma de los cuadrados de los elementos del diccionario. (Este método será bastante útil para el siguiente método)

Similitud()

- Recibe como parámetros un array de diccionarios de la forma Dictionary<string,double> los cuales deben contener los TF-IDF de cada documento y un diccionario también de la forma Dictionary<string,double> el cual debe contener el IDF de cada palabra del corpus. Este método devuelve un array de double con el valor de score en double de cada documento con respecto a la query (cálculo de similitud del coseno). El cálculo de este score se calcula a partir del cociente entre la sumatoria del producto de los TF-IDF de los documentos y el TF-IDF de la query, entre el producto de la raíz cuadrada de la sumatoria de los cadrados de los elementos del TF-IDF del documento por la raíz cuadrada de la sumatoria de los cuadrados de los elementos del TF-IDF de la query.

OrdenarDocDescend()

- Recibe como parámetros un array de string con los documentos (no es necesario que estén normalizados debido a que solo se usa para saber la cantidad de documentos que hay) y un array de double con los resultados del cálculo del peso de los documentos del método anterior ("Similitud()"). Este método devuelve una lista de tuplas del tipo Tuple<double,string> el cual los item1 son el peso del documento y el item2 la posición del documento que coincide con el peso específico. Esta lista de tuplas se devuelve ordenadas de forma descendente con respecto al peso de los documentos.

Snippet

La clase Snippet básicamente se concentra en el fragmento de texto que mejor coincida con la query que será mostrado al usuario. Solo tiene dos métodos los cuales son:

SepOraciones()

- Recibe como parámetro un string que debería de ser un texto y no hace más que devolver una lista (List<string>) con las oraciones ya separadas.

Fragmento()

- Recibe como parámetros un string que sería el texto a buscar el fragmento y un diccionario de la foma Dictionary<string,double> el cual debe de contener el TF-IDF de la query. Este método utiliza el anterior método ("SepOraciones()") para separar las oraciones del texto insertado y posteriormente toma la palabra con mayor TF-IDF del diccionario y busca la primera oración que contenga dicha palabra, en caso de no contenerla , pasa a la segunda palabra con mayor TF-IDF y así sucesivamente hasta encontrar una palabra del TF-IDF que se encuentre el alguna oración, posteriormente devuelve un string con la oración que se usará como snippet posteriormente.

Últimas Ideas con respecto a la relación del proyecto con Álgebra Lineal

Al usar el TF-IDF mediante el uso de diccionarios, se está utilizando el Modelo Vectorial. En este caso los documentos se representan como vectores en un espacio de términos, donde cada término es una clave en el diccionario y el valor asociado a la clave es la frecuencia del término en el documento. A partir de esta representación vectorial de los documentos, se pueden realizar operaciones de álgebra lineal para buscar y clasificar los documentos según su relevancia. Por otro parte, al usar el cálculo de similitud del coseno mediante diccionarios se está aplicando Álgebra Lineal ya que la similitud del coseno es una medida de parecido entre dos vectores, lo cual involucra la utilización de productos por escalares.

Además, este proceso involucra también la aplicación de conceptos matemáticos como el Teorema de los Valores Propios con el cual es posible encontrar los vectores “más relevantes o destacados”, estos son conceptos que son fundamentales en Algebra Lineal.