PROYECTO DE PROGRAMACION

KEVIN MARQUEZ VEGA

C - 121

ALGORITMO DEL MOTOR DE BUSQUEDA

- Cargar los documentos.
- Almacenar el contenido de los documentos.
- Crear vectores documento con los pesos de las palabras.
- Ejecutar consulta(Estandarizar y crear vector query).
- Hallar similitud entre cada vector documento y el vector query (score).
- Ordenar la base de datos.
- Devolver los documentos mas relevantes a la búsqueda..

1. CARGAR LOS DOCUMENTOS

- El programa contiene una clase llamada BBDD la cual representa la base de datos, posee un campo llamado docs que consiste en un array de elementos tipo Document
- El metodo SearchFiles utiliza la clase Directory y obtiene las rutas de todos los archivos con extension .txt que esten en la carpeta Content. Luego, por cada ruta se crea un objeto tipo Document.

1. CARGAR LOS DOCUMENTOS

- La clase Document contiene 3 campos : route, Vd (un objeto tipo vector) y score
- El constructor de Document recibe un string que representa la ruta y lo guarda en route, lo que se utilizara mas adelante para leer el contenido del documento

2. ALMACENAR CONTENIDO

- La clase BBDD posee un campo Content tipo diccionario , que relaciona cada palabra con un diccionario cuyas parejas de valores representan Indice del Documento : veces que aparece la palabra(almacenando solo los documentos donde aparece la palabra) , esto permite optimizar el uso de memoria ya que no se almacenan los valores de los documentos donde no aparezca la palabra

2. ALMACENAR CONTENIDO

- El metodo SetContent de la clase BBDD se encarga de leer el contenido de los documentos de la base de datos (a traves del metodo File.ReadAllText) , estandarizarlo(ver mas adelante) e iterar sobre cada palabra, añadiendola al diccionario si no se encuentra, y en caso contrario aumenta en 1 las ocurrencias de la palabra en el documento de indice i.

2. ALMACENAR CONTENIDO

Ejemplo :

Content

{

…

“arbol” : [<1: 23> ; <3: 5> ; <2198 : 44> ; …] ;

“python” : [<23: 116> ; <104 : 22> ; …]

…

}

{ “palabra” : < Doc : n > }

Doc : indice del documento
n : cantidad de repeticiones de la palabra en el documento

3. CREAR VECTORES DOCUMENTO

- La representacion vectorial de un documento se realiza a traves del calculo del peso de cada palabra en él.
- Los objetos tipo vector poseen un único campo, que es un diccionario que relaciona la palabra con su peso (las palabras que no aparezcan el documento no estaran ya que su peso seria 0), para su calculo utilizamos la fórmula de tf-idf

3. CREAR VECTORES DOCUMENTO

Para el calculo del tf-idf de la palabra i utilizamos la siguientes formulas

$$tf = (fq\ i\ /\ maxfq\ d)$$

$$idf = Log10(\ D + 1\ /\ nj + 1)$$

fq i : veces que se repite la palabra en el documento
maxfq : frecuencia de la palabra que mas se repite en el documento
D : total de documentos en la base de datos
nj : cantidad de documentos donde aparece la palabra

*en la formula del idf se adiciona 1 para evitar división por cero

```
}
public void CreateVectors()
{
    for(int i = 0 ; i < docs.Length ; i++)
    {
        docs[i].Vd = new Vector();

        int maxfq = Tools.MaxFq(Content , i) ; // la freq de la palabra que mas se repite en el doc(para no calcularla varias veces)
        foreach(string word in Content.Keys)
        {
            if(Content[word].ContainsKey(i)) // si el documento contiene la palabra
            {
                double tf = Content[word][i] / (maxfq + 1.0) ;
                double idf = Math.Log10( (docs.Length + 1.0) / (Content[word].Count + 1.0) );
                docs[i].Vd[word] = tf * idf ;
            }

        }
    }
```

3. CREAR VECTORES

Ejemplo :

Vector documento

{

< “arbol” : 0.214 > ; < “python” : 1.32 > ;

< “de” : 0.00004> ; < “levenshtein” : 2.78 > …

}

4. EJECUTAR CONSULTA

- Una vez el usuario realiza una consulta en la interfaz de Moogle, esta se guarda en la variable query, la cual se le pasa como parámetro a la función Stdrize(), la cual la normaliza(elimina caracteres extraños y la lleva a minúsculas).
- Utilizando las mismas formulas de tf-idf se crea el vector query, asignándole el peso que tiene cada palabra en la consulta(si la palabra no aparece en la BBDD su peso se le asigna el valor 0).

```
{
    public static string Stdrize(string input)
    {
        // eliminar caracteres especiales(excepto los modificadores)
        Regex rgx = new Regex("[^a-zA-Z0-9*~!áéíóú]");
        string filtered = rgx.Replace(input , " ") ;

        // eliminar espacios
        filtered = Regex.Replace(filtered , @"\s+" , " ");

        // Convertir a minusculas
        filtered = filtered.ToLower();

        // Devolver Resultado
        return filtered;
    }
```

5. HALLAR EL SCORE

- Una vez creado los vectores documento y el vector query basta con hallar cuan parecidos son, esto se realiza con la formula de similitud del coseno descrita como :

$$sim(d_j, q) = \frac{\sum_{i=1}^{n} w_{i,j} \times w_{iq}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{n} w_{i,q}^2}}$$

5. HALLAR EL SCORE (PRODUCTO ESCALAR)

- El producto escalar de dos vectores se define como (foto de PEsc)

$$\sum_{i=1}^{n} w_{i,j} \times w_{iq}$$

- En este caso se itera sobre las claves del vector query, si esa clave también se encuentra en el vector documento, se adiciona a una variable suma la multiplicación de su peso en cada vector(si la palabra no se encuentra su peso seria 0 por lo cual no modifica el resultado)

5.HALLAR EL SCORE(NORMA)

- La norma de un vector se define como :

$$\sqrt{\sum_{i=1}^{n} w_{i,j}^2}$$

- La función SimCos(Vq , Vd) devuelve un valor entre 0 y 1(mientras mas se acerque a 1 mas relevante es el documento para esa consulta), este resultado se guarda en el campo score de cada objeto tipo documento

6.ORDENAR LA BBDD Y DEVOLVER DOCUMENTOS

- Los documentos se organizan según su score implementando un SelectionSort
- Una vez ordenada la BBDD se devuelven la cantidad máxima(menor que 5) de documentos cuyo score sea mayor que 0
- La clase Documento tiene dos métodos GetTittle y GetSnippet para retornar el titulo y el snippet del documento
- El método GetTittle utiliza la ruta del documento para devolver su titulo utilizando Substring y IndexOf('/') ;

6.ORDENAR LA BBDD Y DEVOLVER DOCUMENTOS

- El método GetSnippet busca la palabra en el vector Documento con mayor peso e itera sobre el documento buscandola (resaltar que hay que comparar la palabra con cada palabra del documento original estandarizada para que haya coincidencias)
- Una vez encontrada la palabra devuelve un string con las 10 palabras anteriores(o desde el inicio del documento si la palabra aparece en una posición anterior a la 10) y las siguientes 30 palabras(o el final del documento si lo sobrepasa)

7. IDEAS PARA MEJORAR

- Agregar los operadores (* ~ ^ y !)
- Al realizar la búsqueda utilizar el método de Distancia de Levenshtein para buscar palabras similares en caso que no aparezca en la base de datos
- Durante el almacenamiento del contenido utilizar bucles Parallel.For y Parallel.Foreach para mejorar la velocidad de carga del programa
- Agregar en la pagina html un botón 'historial' que permita al usuario ver sus ultimas busquedas

7.IDEAS PARA MEJORAR

Por una cuestión de tiempo no he sido capaz de implementar estas funcionalidades, pero iré implementándolas a medida que se vaya desarrollando el proyecto