Cargar Documentos Clase BaseDeDatos

La clase **BaseDeDatos** contiene varios campos y métodos para cargar, procesar y almacenar información de documentos. A continuación, se describen los campos y métodos de la clase:

- ***separadores:*** un arreglo de cadenas que contiene los caracteres que se utilizarán para separar las palabras en los documentos.
- ***TFIDF***: un diccionario de diccionarios que contiene los valores de **TF-IDF** para cada palabra en cada documento.
- ***tf:*** un diccionario de diccionarios que contiene los valores de **TF** para cada palabra en cada documento.
- ***idf***: un diccionario que contiene los valores de **IDF** para cada palabra en todos los documentos.
- ***words***: una lista de todas las palabras únicas en todos los documentos.
- ***VectorsDocument***: un diccionario que contiene los vectores de documentos para cada documento.

La

```
1 reference
public BaseDeDatos()
{
    this.TFIDF = Load();
    this.tf = TF();
    this.idf = IDF();
    this.words = idf.Keys.ToList();
    this.VectorsDocument = VectorDocument();
}
```

clase tiene un constructor que inicializa los campos ***TFIDF, tf, idf***, ***words*** y ***VectorsDocument***. El método **Load()** carga los documentos de un directorio y calcula los valores de **TF** e **IDF** para cada palabra en cada documento. El método **TF** devuelve el diccionario de **TF** para todos los documentos. El método **IDF** devuelve el diccionario de **IDF** para todas las palabras en todos los documentos. El método **VectorDocument()** calcula los vectores de documentos para cada documento.

TF_IDF

Para calcular el **TF_IDF** primero se carga en una lista el nombre de todos los documentos con el método **GetFile**(tiene predeterminada mi ruta). Luego iteramos por dicha lista convirtiendo cada texto en **string** a través de **File.ReadAllText** y hacemos un segundo **foreach** para iterar por las palabras cada texto usando el método .**Split**(separadores). De aquí se quiere sacar la mayor cantidad de información posible por lo que creamos un diccionario **wordfordocument** con el objetivo de saber en cuantos documentos esta una palabra lo cual nos servirá más adelante para calcular el **IDF.**

Así mismo se crea un diccionario de frecuencias para saber cuantas veces esta una palabra en un documento y luego se añade al diccionario **TF** una llave documento y la frecuencia de cada palabra en ese documento.

```
foreach (string document in Namedocuments)
{
    string documents = File.ReadAllText(document);

    if(!Tf.ContainsKey(document)) Tf[document] = new Dictionary<string, double>();

    foreach(string word in documents.ToLower().Split(separadores, StringSplitOptions.RemoveEmptyEntries).ToList())
    {
        if(!wordfordocument.ContainsKey(word)) wordfordocument[word] = new List<string>();

        if(!wordfordocument[word].Contains(document)) wordfordocument[word].Add(document);


        if(Tf[document].ContainsKey(word)) Tf[document][word]++; else Tf[document][word] = 1;

    }
    foreach(string key in Tf[document].Keys)
    {
        Tf[document][key] = Tf[document][key] / Tf[document].Values.Max();
    }
}
foreach (string key in wordfordocument.Keys)
{
    int a = wordfordocument[key].Count;
    int b = Namedocuments.Length;
    idf[key] = Math.Log((double)(Namedocuments.Length)/(wordfordocument[key].Count), 10);
}
```

Ahora tenemos un diccionario **wordfordocument** que contiene en cuantos documentos está una palabra y un diccionario **TF** con la frecuencia de cada palabra en cada documento. Entonces a partir de esto calculamos el **IDF** con el logaritmo en base 10 de la división de la cantidad total de documentos sobre la cantidad de documentos en la que se encuentra una palabra y el **TF** dividiendo la frecuencia de la palabra en el documento sobre la frecuencia de la palabra con mayor frecuencia .

Por problemas de optimización y tratar de hacer la mayor cantidad de cosas en las menos iteraciones posibles tenemos dos diccionarios en el mismo método y solo puedo devolver un tipo así que decidí devolver un diccionario que almacene los dos diccionarios para luego separarlos en el constructor.

Vectores

Para crear los vectores de cada documento iteramos por cada documento y por la lista de palabras totales de todos los documentos, si el documento contiene la palabra colocamos el valor **TFIDF** en esa posición si no la contiene colocamos 0.

Clase Query

La clase **Query** tiene dos campos públicos: ***words*** y ***TFquery.*** El constructor de la clase toma una cadena de consulta como entrada y la convierte en una lista de palabras en minúsculas utilizando el método **Split** de la clase **string**. El constructor también llama al método **TF()** para calcular la frecuencia de término (TF) de cada palabra en la consulta y almacenarla en el campo ***TFquery.***

```
4 references
public class Query
{
    2 references
    public List<string> words;
    3 references
    public Dictionary<string, double> TFquery;

    1 reference
    public Query(string query)
    {
        this.words = query.ToLower().Split(new string[]{" ", "\t", "\n", "\r", ".", ",", ";", ":", "?", "!", "(", ")", "[", "]", "
        this.TFquery = TF();
    }
```

El método **TF()** es un método privado que toma la lista de palabras de la consulta y calcula la frecuencia de cada palabra en la lista. El método utiliza un diccionario frecuencias para almacenar la frecuencia de cada palabra y luego calcula la frecuencia de término (TF) de cada palabra dividiendo su frecuencia por la frecuencia máxima de cualquier palabra en la lista. El método devuelve un diccionario **tf** que contiene la frecuencia de término de cada palabra en la lista.

Clase ModeloVectorial

La clase **ModeloVectorial** representa un modelo vectorial utilizado en la recuperación de información. Esta clase tiene varios campos y métodos que se utilizan para calcular la similitud entre un conjunto de documentos y una consulta.

El campo ***documents*** es un objeto de la clase **BaseDeDatos** que representa una colección de documentos. El campo ***query*** es un objeto de la clase ***Query*** que representa una consulta. El campo ***VectorQuery*** es un arreglo de tipo **double** que representa el vector de consulta. El campo ***Score*** es un diccionario que almacena los puntajes de similitud entre los documentos y la consulta. El campo ***MoreFrequency*** es una lista que almacena las palabras más frecuentes en los documentos.

El constructor de la clase **ModeloVectorial** toma dos parámetros: un objeto de la clase **BaseDeDatos** y un objeto de la clase **Query**. Este constructor inicializa los campos ***documents, query, VectorQuery, Score y MoreFrequency*** utilizando los métodos **VectorsQuery(), Scores() y MoreFrequencyDocuments().**

```
public class ModeloVectorial
{
    8 references
    BaseDeDatos documents;
    3 references
    Query query;
    2 references
    double[] VectorQuery;
    2 references
    public Dictionary<string, double> Score;
    6 references
    public List<string> MoreFrequency;
    1 reference
    public ModeloVectorial(BaseDeDatos documents, Query query)
    {
        this.documents = documents;
        this.query = query;
        this.VectorQuery = VectorsQuery();
        this.Score = Scores();
        this.MoreFrequency = MoreFrequencyDocuments(Score);
    }
```

El método **VectorsQuery()** calcula el vector de consulta utilizando la frecuencia de términos de la consulta previamente calculada en la clase **Query** el valor **IDF** (Inverse Document Frequency) de cada término en la colección de documentos.

El método **Scores()** calcula los puntajes de similitud entre los documentos y la consulta utilizando el método **Similitud().**

Los métodos **Normalizar(), Multiplicar() y Similitud()** son métodos auxiliares que se utilizan para calcular la similitud entre dos vectores.

```
    2 references
    public double Normalizar(List<double> vector)
    {
        double result = 0;

        for(int i = 0;i < vector.Count;i++)
        {
            result += vector[i]*vector[i];
        }
        return Math.Sqrt(result);
    }
    1 reference
    public double Multiplicar(List<double> vector1, List<double> vector2)
    {
        double producto = 0;
        for(int i = 0;i < vector1.Count;i++)
        {
            producto += vector1[i]*vector2[i];
        }
        return producto;
    }
    1 reference
    public double Similitud(List<double> vector1, List<double> vector2)
    {
        double score = Multiplicar(vector1, vector2)/Normalizar(vector1)*Normalizar(vector2);
        return score;
    }
```

El método **MoreFrequencyDocuments()** devuelve una lista de las palabras más frecuentes en los documentos. Este método ordena el diccionario frecuencias por valor de forma descendente, toma las cinco primeras palabras del diccionario ordenado y crea una lista con las palabras más frecuentes.

Vector Query

El objetivo de crear el ***VectorQuery*** es calcular la similitud de este con los vectores documentos y así devolver los los cinco documentos que más se asemejen a la query. Para crear este vector iteramos por la lista de palabras words y así si la query contiene la palabra le añadimos su valor **TFIDF** en esa posición de otra manera se inserta 0;

```
1 reference
double[] VectorsQuery()
{
    double[] VectorQuery = new double[documents.words.Count];

    for(int j = 0;j < documents.words.Count;j++)
    {
        if (query.TFquery.ContainsKey(documents.words[j]))
        {
            VectorQuery[j] = query.TFquery[documents.words[j]]*documents.idf[documents.words[j]];
        }
        else
        {
            VectorQuery[j] = 0;
        }
    }
    return VectorQuery;
}
```

Score y MoreFrequencyDocuments

Para crear el diccionario ***Score*** que almacena el nombre del documento y su nivel de importancia con respecto a la query se itera por el diccionario ***VectorsDocument*** y se halla la similitud de cosenos de cada vector documento con el vector de la query y después se extrae una lista ***MoreFrequencyDocuments*** con los 5 primeros documentos del diccionario.

Clase Moogle

En la clase **Moogle** se crea una instancia de la clase **ModeloVectorial** y se le pasa como parámetro las dos instancia de **BaseDeDatos** y de la **Query** y se devuelve los 5 primeros documentos de la lista.

```
1 reference
public static class Moogle
{
    1 reference
    public static SearchResult Querys(string query) {
        // Modifique este método para responder a la búsqueda

        Query QUERY = new Query(query);

        ModeloVectorial Vectors = new ModeloVectorial(Object.document, QUERY);

        SearchItem[] items = new SearchItem[Vectors.MoreFrequency.Count];

        for(int y = 0; y < Vectors.MoreFrequency.Count;y++)
        {
            items[y] = new SearchItem(Vectors.MoreFrequency[y].Substring(44,Vectors.MoreFrequency[y].Length - 48),Snippet(Vecto
        }

        return new SearchResult(items, query);
    }
```