Moogle!

Mauricio Sunde Jimenez C111

Julio, 2023

Abstract

Moogle! es una aplicacion cuyo proposito es buscar inteligentemente un texto en un conjunto de documentos. Es una aplicacion web, desarrollada con tecnologia .NET Core 6.0, especificamente usando Blazor como *framework* web para la interfaz grafica, y en el lenguaje CSharp.

1 Introducción

Moogle! es una aplicación cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos. Esta aplicación dada una query (texto a buscar) del usuario muestra en pantalla los documentos más importantes donde esa query aparece. La idea está en que la búsqueda sea lo más inteligente posible por eso no nos limitamos a solo mostrar exactamente el documento donde aparece exactamente esa query. Moogle implementa un algoritmo de búsqueda, muestra además sugerencias en caso de que la palabra introducida lo necesite, etc. Todo esto será explicado a detalle más adelante.

2 Explicacion General

Para poder obtener la relevancia de cada documento respecto a la búsqueda se ha utilizado la medida de recuperación de información de modelo vectorial y el valor TFIDF. Con este

modelo vectorial vemos a los documentos y la Query como vectores que almacenaran sus valores TF-IDF respectivos. El TF-IDF (Term frequency – Inverse document frequency)no es más que la frecuencia de ocurrencia del término en la colección de documentos, la cual es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección. Este valor aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras. Con estos valores podemos verificar que tan parecidos son estos dos vectores mediante la Similitud del coseno. Esta es una medida de la similitud existente entre dos vectores en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos. De esta manera mientras más cercano a 1 son los valores más cercanos a 0 es el ángulo existente entre los dos vectores y más parecidos son estos.

3 Funcionalidad del Proyecto

El proyecto comienza con la clase principal, la clase Moogle. Esta clase tiene un método Query que recibe a la query y devuelve los resultados de la búsqueda. Primero debemos abstraernos y comenzar a ver cada documento como un vector. Necesitamos leer dado una dirección, cada documento y comenzar el proceso de leer el documento, tokenizarlo, obtener los valores TFIDF todo lo necesario para comenzar a resolver el problema inicial, devolver los mejores resultados de búsqueda posible. Lógicamente necesitamos información sobre el documento donde estamos para esto se implementó la Clase Document. Esta clase contiene toda la información necesaria sobre un documento en específico, su título, un método para hallar el vector TFIDF del documento, el texto del documento y todos los términos del documento guardados en un diccionario y donde se asocia ese término a la cantidad de veces que aparecen en el documento y además la primera posición donde fue encontrada dicha palabra en el documento (de gran utilidad para el snipet).

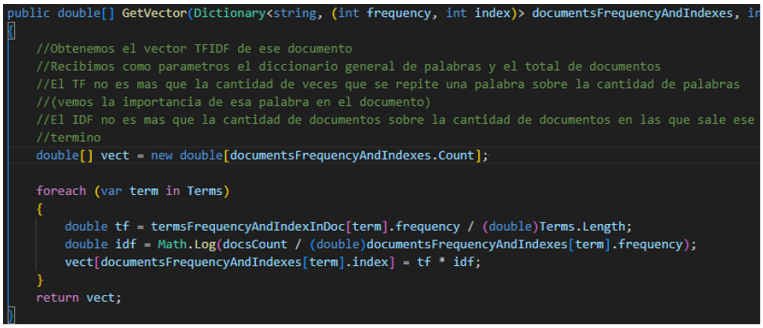
```
using System.Text.RegularExpressions;

namespace MoogleEngine
{
    5 references
    public class Document// Clase encargada de todo el procesamiento de los documentos
    {
        6 references
        private Dictionary<string, (int frequency, int docIndex)> termsFrequencyAndIndexInDoc = new Dict
        //Con este diccionario guardamos todas las palabras unicas del documento ademas de que guardamos
        //cuantas veces se repite ese termino (valor necesario para el calculo de tfidf)
        //y almacenamos el indice de la primera vez que fue encontrada la palabra en el texto
        2 references
        public string Title { get; }
        4 references
        public string DocumentText { get; }
        1 reference
        public Document(string filePath)
        {
            Title = Path.GetFileNameWithoutExtension(filePath);
            DocumentText = File.ReadAllText(filePath);
            termsFrequencyAndIndexInDoc = GetTermsFrequency();
        }
```

Figure 1: Muestra del codigo

Importante destacar que a la hora de la tokenización del texto las palabras con tilde no se les elimina la tilde.

Junto con esta clase tenemos la clase Query podemos decir que es hija de la clase documento. Para poder llevar la query a un vector debemos tratar a la query exactamente

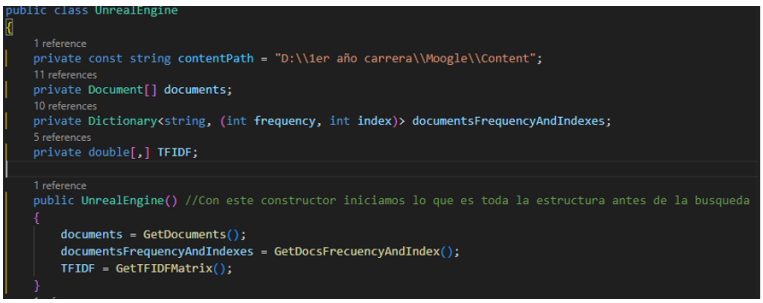


```
public double[] GetVector(Dictionary<string, (int frequency, int index)> documentsFrequencyAndIndexes, in
{
    //Obtenemos el vector TFIDF de ese documento
    //Recibimos como parametros el diccionario general de palabras y el total de documentos
    //El TF no es mas que la cantidad de veces que se repite una palabra sobre la cantidad de palabras
    //(vemos la importancia de esa palabra en el documento)
    //El IDF no es mas que la cantidad de documentos sobre la cantidad de documentos en las que sale ese
    //termino
    double[] vect = new double[documentsFrequencyAndIndexes.Count];

    foreach (var term in Terms)
    {
        double tf = termsFrequencyAndIndexInDoc[term].frequency / (double)Terms.Length;
        double idf = Math.Log(docsCount / (double)documentsFrequencyAndIndexes[term].frequency);
        vect[documentsFrequencyAndIndexes[term].index] = tf * idf;
    }
    return vect;
}
```

Figure 2: Vector TFIDF del documento

como trataríamos a un documento. Esta clase permite saber los términos de la query y su frecuencia en la misma además de cómo obtener su vector TFIDF. Igual que con la tokenizacion de los textos las palabras con tilde de la query no se les elimina la tilde. La clase que ejecutara y trabajara todo es la clase UnrealEngine. Esta clase será la encargada de ejecutar todo lo necesario para la búsqueda. Esta clase posee un arreglo de Objetos documentos de la clase documento, es decir todos los documentos, además de todas las palabras generales del corpus dentro de un diccionario así asociamos a cada palabra (Key o llave) con la cantidad de documentos donde aparece la palabra y un índice asociado para así mantener un control sobre las palabras y proceder a ubicarlas en la matriz TFIDF más cómodamente.



```
public class UnrealEngine
{
    1 reference
    private const string contentPath = "D:\\1er año carrera\\Moogle\\Content";
    11 references
    private Document[] documents;
    10 references
    private Dictionary<string, (int frequency, int index)> documentsFrequencyAndIndexes;
    5 references
    private double[,] TFIDF;

    1 reference
    public UnrealEngine() //Con este constructor iniciamos lo que es toda la estructura antes de la busqueda
    {
        documents = GetDocuments();
        documentsFrequencyAndIndexes = GetDocsFrecuencyAndIndex();
        TFIDF = GetTFIDFMatrix();
    }
```

Figure 3: Class UnrealEngine

Este es el proceso de obtención de palabras generales del corpus es sencillo GetDocsFrecuencyAndIndex () en términos generales recorre cada documento y por cada documento recorre sus términos estos los va almacenando en el diccionario en caso de que ya tenga esa palabra le suma uno a su frecuencia y mantiene su índice en caso contrario agrega 1 a su frecuencia y el índice asociado es el tamaño que posea el diccionario hasta ese momento. Una vez obtenido todas las palabras del corpus y todos los vectores TFIDF necesarios para construir la matriz podemos tener el espacio creado para próximamente realizar las búsquedas.

Proceso de Obtención de la matriz.

Ya una vez con la query tratada como un vector de TFIDF y todos los documentos almacenados en una matriz que guarda el TFIDF de cada vector documento solo queda pasar a la acción y comenzar la búsqueda. Una vez se introduce la búsqueda esta pasa a un vector tamaño n donde pasamos a realizar la antes comentada similitud del coseno entre ese vector de la query y cada uno de los vectores documentos de la matriz. Una vez

```
private double[,] GetTFIDFMatrix()//Aqui vamos a crear la Matriz de documentos
{
    //Va a tener tamaño cantidad de documentos por cantidad de palabras del corpus
    double[,] tfIdf = new double[documents.Length, documentsFrequencyAndIndexes.Count];
    for (int i = 0; i < documents.Length; i++)//vamos recorriendo los documentos
    {
        //Por cada documento vamos a sacar su vector TFIDF e ir rellenando la matriz
        //Este proceso de obtener el vector del documento lo realiza mi clase documento
        //Clase encargada de todo lo necesario para los documentos
        double[] docVector = documents[i].GetVector(documentsFrequencyAndIndexes, documents.Length);
        for (int j = 0; j < docVector.Length; j++)
        {
            tfIdf[i, j] = docVector[j];
        }
    }
    return tfIdf;
```

Figure 4: Obtención de la matriz

terminado el proceso organizamos los score y devolvemos los mejores documentos para el usuario. Es importante recalcar que una vez construido un objeto de tipo UnrealEngine comenzara a cargarse toda la base de datos una y solo una vez, luego de esto estamos listos para comenzar la búsqueda.

4 Implementaciones Extras:

El programa es capaz de devolver además porciones de los textos más relevantes donde aparezca la query deseada. Relativamente sencillo al ya tener las posiciones donde aparece cada termino en el texto. Para esto revisamos cual termino de nuestra query es el más relevante y buscamos su índice en el texto y solo queda obtener tantos caracteres a la izquierda y derecha se desee y se devuelve dicho fragmento. También tiene implementada una funcionalidad para dar sugerencias en caso de que algún término de la query no aparezca en el corpus se le recomendara al usuario la palabra más similar entre todas las palabras del corpus este proceso se hace mediante la distancia de levenshtein. Esta distancia es el mínimo número de operaciones que se deben hacer para convertir una palabra en otra se entiende por operaciones las siguientes tres, Insertar una palabra, Eliminar una palabra y la Sustitución de una palabra. Como un extra en la parte visual podemos dar clic en la sugerencia dada y comenzara una nueva búsqueda en función de la sugerencia.

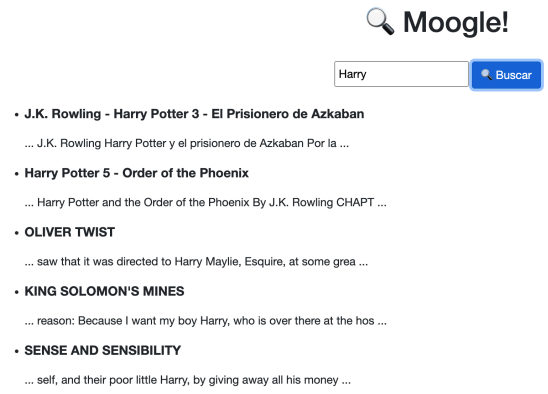


Figure 5: Busqueda realizada con exito y con un fragmento del texto donde la palabra buscada aparece

Figure 6: Ejemplo de como el se ofrece la sugerencia (notese el enlace que puede se tocado en la sugerencia para iniciar una nueva busqueda)

Figure 7: Nueva busqueda con la sugerencia

Contents