Moogle!

Eduardo Brito Labrada

May 8, 2023

Una breve introducción

Moogle! es una aplicación *totalmente original* cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos. Es una aplicación web, desarrollada con tecnología `.NET Core 6.0`, específicamente usando Blazor como *framework* web para la interfaz gráfica, y en el lenguaje `C#`. La aplicación está dividida en dos componentes fundamentales:

- `MoogleServer` es un servidor web que renderiza la interfaz gráfica y sirve los resultados.
- `MoogleEngine` es una biblioteca de clases donde está... ehem... casi implementada la lógica del algoritmo de búsqueda.

¿Para qué sirve?

La idea original del proyecto es buscar en un conjunto de archivos de texto (con extensión `.txt`) que estén en la carpeta `Content`.

¿Cómo usarlo?

Primeramente, se aconseja a quien use esta aplicación tener instalado Linux, ya que no se garantiza la misma eficiencia si esta en un dispositivo que use Windows. En caso de tener instalado Windows, puede optar por instalar Windows Subsystem for Linux (WSL) que añade funcionalidades de Linux en Windows.

Instrucciones

Lo primero que tendrás que hacer para poder trabajar en este proyecto es instalar `.NET Core 6.0`. Luego, te debes parar en la carpeta del proyecto y dependiendo de tu Sistema Operativo hacer lo siguiente:

- **Linux:** Debes tener instalado `make`. Si no lo tienes instalado puedes instalarlo usando `sudo apt update && sudo apt install make`. Luego podrás hacer `make dev`
- **Windows:** Deberías poder ejecutar este proyecto usando `dotnet watch run --project MoogleServer`

Después de hacer lo anterior abre en tu navegador http://localhost:5000 y podrás usar Moogle! introduciendo tu búsqueda en la "entrada" y luego presionando el botón "Buscar".

Motor de búsqueda

El motor de búsqueda usa un modelo vectorial que nos computa para una *query* dada qué tan relevante es un documento determinado. Este modelo vectorial usa *Term Frequency and Inverse Document Frequency (TF-IDF)* con *Cosine Similarity* para computar la relevancia.

Para computar el vector *TF-IDF* uso la fórmula:

$$TFIDF = (\frac{tf}{tw}) \times \ln(\frac{td}{dt})$$

En esta fórmula tenemos que:

- tf es la frecuencia del término en el documento actual.
- tw es la cantidad de palabras totales en el documento actual.
- td es la cantidad total de documentos a analizar.

- dt es la cantidad de documentos que contienen el término.

Nota importante: dado que $\frac{td}{dt}$ puede causar problemas por la división entre 0 decidí que si $dt = 0$ luego $TFIDF = 0$, tiene sentido hacer esto ya que si $dt = 0$ el término no aparece en ningún documento.

Después de hacer lo anterior necesitamos calcular la "similitud" entre el vector de *document* y el vector de la *query*, que para eso utilizo *Cosine Similarity*. Para hacer esto intentamos estimar el "ángulo" comprendido entre el vector de la *query* y el vector de *document*: mientras menor sea este ángulo, mayor "similitud" tendrán estos vectores. Esto lo hacemos usando la fórmula:

$$\cos \alpha = \frac{v_d \cdot v_q}{||v_d||\ ||v_q||}$$

Siendo:

- v_d el vector de *document*
- v_q el vector de *query*
- $||v||$ es la magnitud del vector v

Sobre la implementación

`Utils.cs`

Esta clase está principalmente para métodos que son necesarios en varias clases del proyecto.

- `int EditDistance(string a, string b)`:
- `int LongestCommonPrefix(string a, string b)`: esta clase recibe dos cadenas y devuelve el prefijo común más largo de dichas cadenas.
- `double Distance(string a, string b)`: esta función retorna utilizando el `EditDistance` (en lo adelante ED) y el `LongestCommonPrefix` (en lo adelante LCP) de dos cadenas, la distancia que existe entre ellos. En este método utilizo el cociente de $\frac{ED}{LCP}$