Nombre del Projecto: Moogle!

Nombre del alumno: Raciel Alejandro Simón Domenech

Grupo: C-122

Curso: 2023

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 implementada la lógica del algoritmo de búsqueda.

La búsqueda se realiza utilizando el modelo vectorial del TF-IDF, para lo cual se expresa cada documento y el query como vectores, entonces se calcula el coseno de similitud de cada vector documento con el vector query y se establece este valor como score(valor que determina la importancia del documento).

La biblioteca de clases consta de las siguientes clases

Doc:

Esta clase representa un documento, y contiene toda la información de este, hereda la interfaz IComparable e implementa el método CompareTo para organizar los documentos según la importancia.

Esta clase contiene las siguientes propiedades:

string Title : Representa el título del documento.

double score : Representa el valor numérico de la importancia del documento según la query introducida.

double[] TFIDF : Array de valores double que representa el vector de los pesos TF-IDF de todas las palabras de la colección en el documento.

double[] TF_root : Similar al anterior, pero este representa los pesos TF-IDF de todas las raíces de las palabras de la colección en el documento.

string[] content_unnormalized : Array de strings que contiene todas las palabras del documento en el mismo orden en que aparecen.

string[] content : Similar al anterior pero ya no contiene pronombres, preposiciones, artículos y conjunciones.

string doc_path : String que contiene la dirección en la cual se encuentra el documento.

double module : Representa el módulo del vector de los pesos TF-IDF.

Dictionary<string, double> wordcoll : Diccionario que contiene todas las palabras del documento(sin repetición) y a cada una de ellas se le asocia un valor que representa la cantidad de apariciones en el documento.

Dictionary<string, double> root_words : Similar al anterior, pero contiene las raíces de las palabras y el número de apariciones de esta en el documento (nótese que dos palabras distintas pueden tener la misma raíz, por lo que la cantidad de componentes del vector de los pesos TF-IDF de las raíces probablemente disminuya con respecto al vector de las palabras originales).

Esta clase contiene los siguientes métodos:

Normalize : Método que calcula el módulo del vector de las palabras.

NormalizeR : Similar al anterior pero con el vector de las palabras raíces

InitialiceWordsColl : Inicializa el diccionario de la colección de palabras del documento, si una palabra ya está incluida, le aumenta el número de apariciones en 1, en caso contrario, agrega al diccionario un nuevo elemento cuyo key es la palabra y como value 1.

GetDistance: Método que recibe dos palabras, y en caso de estas estar presentes en el documento, determina la distancia mínima entre las apariciones de esta, para ello calcula todas las distancias y retorna la menor.

SimilarityCosine: Recibe como parámetro una instancia de la clase Query, y calcula el coseno de similitud entre los vectores de los pesos TF-IDF de la clase desde donde se invoca y la clase Query que se le pasa como parámetro, para ello, se realiza la multiplicación escalar de dichos vectores y se divide entre la multiplicación de sus magnitudes, y establece el score en base a este resultado.

SimilarityCosineR: Similar al expuesto anteriormente, pero realiza las operaciones con los vectores de los pesos TF-IDF de las raíces.

CalculateTF: Método que recibe como parámetros un Wordscollection y, un double que representa la cantidad de documentos de la colección, inicializa el vector de los pesos TF-IDF y calcula el valor de cada componente de dicho vector con la fórmula del TF-IDF.

CalculateTFR: Similar al anterior, pero se trabaja con el vector de los pesos TF-IDF de las palabras raíces.

GetFragment : Método que recibe una palabra y devuelve un fragmento del documento que contenga a la palabra, básicamente recorre el array de palabras sin normalizar, y crea un string uniendo palabras a la derecha y a la izquierda de la aparición de este.

GetSnippet : Recibe un array de palabras y devuelve un fragmento del texto para la primera palabra que aparezca en el documento invocando el método GetFragment.

Stemmer() : Método que inicializa el diccionario de las raíces de las palabras utilizando la librería SharpNL.

Query:

Esta clase representa el query vectorizado introducido por el usuario. Hereda de la clase Doc e implementa otras funcionalidades.

Posee las siguientes propiedades:

bool[,] SpChar : Matriz que contiene información sobre los caracteres especiales introducidos por el usuario, específicamente la primera fila representa "^" y la segunda fila "!"

double[] AcumSpChar : Matriz que contiene un valor que se determina según el número de * que se escribieron antes de una palabra.

Dictionary<string, string> ProxSpChar : Diccionario donde cada par de palabras representa las palabras cuya distancia tienen relevancia, esto queda determinado por el carácter ~

Nótese que para las dos matrices mencionadas anteriormente, a cada componente se le asocia una palabra del query, y por tanto, se le asocia un carácter especial.

Esta clase tiene los siguientes métodos:

InicialiceSpecialContent: : Este método inicializa las propiedades mencionadas anteriormente asignándole los caracteres especiales según corresponda.

Initialice: Método que invoca al método anterior.

GetMultiply: Método que recibe una palabra y determina cuántos * son válidos y por tanto su multiplicidad.

ScoreMod : Método que recibe un Doc como parámetro y modifica su score según los caracteres especiales que se apliquen a las palabras del query que aparezcan en el documento.

Wordscollection

Esta clase representa la colección de palabras del documento. Su constructor recibe una lista con todos los Doc e inicializa la propiedad Dictionary<string, double> word_count donde guarda cada palabra del documento y el valor de la cantidad de documentos donde aparece dicha palabra.

Presenta el método Stemmer() que funciona igual que el de la clase Doc e inicializa la propiedad Dictionary<string, double> root_words la cual contiene todas las palabras raíces y el valor de la cantidad de documentos donde aparecen.

DocumentUtils

Clase estática que contiene Métodos importantes a la hora de realizar la búsqueda.

Esta clase contiene la propiedad string[] removable_words array que contiene los pronombres, preposiciones, artículos y conjunciones que son removidas al filtrar los documentos.

Esta clase tiene los métodos:

DocLoader, el cual inicializa, por cada documento de texto de la carpeta content, un instancia del tipo Doc y los devuelve como una Lista de Doc

QueryLoader: Similar al anterior, pero solo crea una instancia de la clase Query con el input del usuario.

ScalarProduct: Método que recibe dos vectores y realiza el producto escalar de estos.

DamerauLevenshteinDistance: Método que calcula la diferencia entre dos palabras.

Where: Método que recibe un Enumerable y un delegado Func<T,bool> y devuelve un Enumerable con los elementos que cumplen con la condición.

FindSimilarWords: Método que se invoca desde un vector de tipo Doc y recibe como parámetro un Wordscollection, sustituye las palabras del query que no tengan aparición por otras similares que sí aparezcan.