Informe de Moogle!

Kendry Javier del Pino Barbosa

1 Clase Documents

La clase representa un conjunto de documentos en formato de texto plano y proporciona varios métodos para acceder y manipular la información de los documentos. A continuación se describen los métodos de esta clase:

- GetContent(): El método es utilizado en la clase **Documents** y se encarga de leer el contenido de cada archivo de texto dentro de una carpeta especificada en la construcción de la instancia de la clase.

En primer lugar, se crea una lista vacía **content** que será utilizada para almacenar el contenido de cada archivo. Luego, se utiliza un **foreach** para iterar sobre cada archivo de texto en la carpeta especificada.

Dentro del **try** se lee el contenido del archivo con **File.ReadAllText**, se eliminan los caracteres de control con **new string(text.Where(c => !char.IsControl(c)).ToArray()),** se obtiene el nombre del archivo con la lista **FileName** y finalmente se añade el contenido y el nombre del archivo a la lista **content**.

En caso de haber algún error al leer el archivo, se captura la excepción en el **catch** y se imprime un mensaje de error. Al final, la lista **content** que contiene el nombre y el contenido de cada archivo de texto es devuelta.

- GetName(): El método es una función que devuelve una lista de strings con los nombres de los archivos presentes en el directorio especificado en **documentsPath**.

Esta función utiliza la clase **Directory** de C# para obtener la lista de archivos presentes en el directorio, y luego utiliza la función **Path.GetFileNameWithoutExtension** para obtener el nombre de archivo sin la extensión ".txt" y almacenarlos en la lista **Names**. Finalmente, la función devuelve la lista **Names** que contiene los nombres de los archivos presentes en el directorio.

- GetFileName(int a): Devuelve el nombre de archivo de texto con el índice **a** en la lista de nombres de archivo.

- GetWords(): Este método recibe el contenido de cada documento y devuelve una lista de listas de palabras de cada documento.

Primero se crea una lista vacía de listas de palabras llamada **Words**. Luego, para cada documento en la lista de contenido, se crea una lista de palabras llamada **documentWords**.

Después, se define un conjunto de separadores que incluyen varios signos de puntuación y se usa el método **Split**() para separar las palabras en cada documento utilizando los separadores. Luego se recorre cada palabra, se verifica si es nula o espacio en blanco, se convierte a minúscula y se agrega a la lista **documentWords**.

Finalmente, la lista **documentWords** se agrega a la lista **Words** y se repite el proceso para el siguiente documento. La función devuelve la lista de listas de palabras **Words** con todas las palabras de los documentos convertidas a minúsculas y sin signos de puntuación.

- GetWords(int a): Devuelve una lista de cadenas de caracteres que representan las palabras del documento con el índice `a` en la lista de documentos.

- GetTF(): Este método recibe como entrada la lista de palabras **Words** de cada documento y devuelve un diccionario con la frecuencia de términos de cada documento.

Primero se crea un diccionario **termFrequencies** vacío. Luego se itera a través de la lista **Words**, que contiene las palabras de cada documento. Dentro del bucle **for** anidado, se crea otro diccionario **docTermFrequencies** vacío para almacenar la frecuencia de cada término en el documento actual. Luego, se itera a través de cada palabra en el documento actual y se actualiza su frecuencia en el diccionario **docTermFrequencies**. Si la palabra ya está en el diccionario, se incrementa su frecuencia en 1. Si no, se agrega la palabra al diccionario y se establece su frecuencia en 1.

Una vez que se han procesado todas las palabras en un documento, se agrega el diccionario **docTermFrequencies** al diccionario **termFrequencies**, utilizando un número entero como clave para identificar cada documento. Finalmente, se devuelve el diccionario **termFrequencies** completo, con la frecuencia de términos de cada documento.

- GetTermFrequencies(int a,string b): Devuelve la frecuencia de términos del término **b** en el documento con el índice **a** en la lista de documentos.

- GetIDF(): Este método calcula el valor **IDF** (Inverse Document Frequency) para cada término o palabra en los documentos. La fórmula empleada para calcular **IDF** es **log(N/df)**, donde **N** es el número total de documentos y **df** es el número de documentos que contienen el término..

El método comienza creando un diccionario vacío llamado **IDF**. Luego, calcula el número total de documentos y recorre cada documento para contar el número de documentos que contienen cada término. Si un término ya está en el diccionario **IDF**, se incrementa su valor. De lo contrario, se agrega al diccionario con un valor inicial de 1.

Finalmente, se recorre el diccionario **IDF** y se calcula el valor **IDF** para cada término utilizando la fórmula mencionada anteriormente. El valor **IDF** se almacena en el diccionario y, finalmente, se devuelve.

- GetSnippet(string fileName, string query): Devuelve una cadena de caracteres que representa un fragmento del documento con el nombre de archivo **fileName** que contiene la consulta **query**. Este método utiliza el modelo de recuperación de información BM25 para calcular la puntuación de cada término en la consulta y seleccionar las oraciones más relevantes del documento.

2 Clase Query

La clase **Query** representa una consulta en un modelo de recuperación de información.La clase tiene los siguientes métodos:

- GetText(): Este método devuelve la consulta como un string.

- GetWords(): Este método obtiene una lista de palabras a partir de la consulta. El método separa la consulta por espacios y signos de puntuación y agrega cada palabra a la lista.

- GetTermsFrecuency(): Este método calcula la frecuencia de términos en la consulta. El método recorre la lista de palabras y utiliza un diccionario para almacenar la frecuencia de cada término. Si el término ya está en el diccionario, se incrementa su frecuencia. De lo contrario, se agrega al diccionario con una frecuencia de 1.

3 Clase ModeloVectorial

La clase **ModeloVectorial** representa un modelo de recuperación de información vectorial que permite obtener la similitud entre una consulta y un conjunto de documentos. La clase tiene los siguientes métodos:

- GetMatrix(): El método es utilizado para calcular la matriz de términos de los documentos en relación a la consulta. La matriz resultante es una matriz **numDocumentos x numWords**, donde **numDocumentos** es la cantidad de documentos en la colección y **numWords** es la cantidad de palabras en la consulta. Cada elemento de la matriz representa la frecuencia de una palabra en un documento, multiplicada por su **IDF**.

Primero, se inicializa la matriz con ceros. Luego, se recorre cada documento y se obtiene su diccionario de **TF**. Después, se recorren las palabras de la consulta y se busca si la palabra está en el diccionario de frecuencias del documento. Si la palabra está en el diccionario, se calcula su frecuencia **TF** y su **IDF**. Si el **IDF** es menor a **0.9**, entonces la frecuencia es considerada como 0, lo que significa que la palabra no es relevante para el documento. De lo contrario, se multiplica el **TF** por el **IDF** y se guarda el resultado en la matriz.

- ObtenerSimilitudDocumentos(): Este método calcula la similitud de cosenos entre cada documento y la consulta. El método recorre cada documento y llama al método `CalcularSimilitudCosenos` para calcular la similitud de cosenos entre el documento y la consulta. Luego, agrega la similitud a la lista **Score** junto con el nombre del documento correspondiente. Finalmente, devuelve la lista **Score**.

- CalcularSimilitudCosenos(double[,] matriz, int indiceDocumento): El método utiliza el concepto de similitud de cosenos para calcular la similitud entre un documento y una consulta.

Primero, el método verifica si la consulta es una sola palabra, en cuyo caso se calcula la similitud de cosenos para esa palabra. De lo contrario, se calcula la similitud de cosenos para la consulta normalmente.

Luego, se crea un vector de consulta y un vector de documento, donde el valor de cada elemento del vector documento representa su respectivo valor en la matriz y a cada elemento del vector query se la da 1 asumiendo que cada elemento de la misma tiene el mismo peso o importancia

A continuación, se calcula la norma del documento dividiendo la suma de los cuadrados de los valores del vector documento por su raíz cuadrada. La similitud de cosenos se calcula multiplicando cada valor correspondiente en los vectores de consulta y de documento, y sumando los resultados.

- GetScore(): Este método devuelve la lista **Score**.

- GetScoreOrganizado(): Este método ordena la lista **Score** en orden descendente según la similitud de cosenos y la devuelve.

4 Clase Moogle

Esta clase es utilizada para realizar búsquedas en documentos. A continuación se explica su funcionamiento:

Primero, se crea una instancia de la clase **Documents** para obtener acceso a los documentos que serán utilizados en la búsqueda.

A continuación, se crea una instancia de la clase **Query**, que representa la consulta que se desea realizar.

Se crea una instancia de la clase **ModeloVectorial**, que utiliza la matriz de términos-frecuencias de los documentos y la consulta para calcular la similitud entre cada documento y la consulta.

Se llama al método **GetScoreOrganizado**() de la clase **ModeloVectorial**, que devuelve una lista de tuplas que contienen el nombre del documento y su puntuación de similitud con la consulta, ordenadas de mayor a menor.

Se seleccionan los tres primeros documentos con las puntuaciones de similitud más altas y se crean instancias de la clase **SearchItem** para cada uno de ellos, que contienen información como el nombre del documento, el fragmento de texto relevante y la puntuación de similitud.

Finalmente, se devuelve una instancia de la clase **SearchResult**, que contiene los tres **SearchItem** y la consulta original.

5 Operadores

Se usan los siguientes operadores:

1. Operador de negación "!" - Se utiliza para indicar que una palabra no debe aparecer en los resultados. Por ejemplo, si la query es "perros !gatos", los resultados no incluirán documentos que contengan la palabra "gatos".

En el código, cuando se encuentra una palabra que comienza con "!", simplemente se salta esa palabra y se continúa con la siguiente en la consulta.

2. Operador de multiplicación "*" - Se utiliza para darle más valor a las palabras que lo tengan al principio, este es acumulativo. Por ejemplo, si la query es "*perros **casa", se le dará más importancia a la palabra casa. Si una palabra tiene uno o más asteriscos al principio, se quitan los asteriscos y se incrementa la variable "numAsteriscos" en 1. Luego, se multiplica la frecuencia de esa palabra por el valor resultante de "1 +

numAsteriscos", lo que significa que la frecuencia de la palabra aumenta en función de la cantidad de asteriscos que tenga.

3. Operador de obligatoriedad "^" - Se utiliza para indicar que una palabra debe aparecer en los resultados. Por ejemplo, si la query es "perros ^casa", los resultados solo incluirán documentos que contengan la palabra "casa". Si una palabra tiene el símbolo "^" al principio, se quita ese símbolo y se establece la variable "obligatorio" en "true", lo que significa que la presencia de esa palabra es obligatoria en la búsqueda.