Reporte MoogleEngine

Luis Alejandro Arteaga Morales

Universidad de La Habana (Facultad de Matemática y Computación)

Resumen

La implementación del motor de búsqueda está basada en el modelo vectorial, modelando cada consulta(query) y cada texto como un vector de palabras en R^N, representados con el valor de TF-IDF de cada palabra. Luego se emplea la similitud de cosenos para ver qué tan cerca de ser iguales están dos vectores. De esta manera se tiene una noción de qué documentos son los más similares a lo que buscamos.

Palabras clave: modelo vectorial, vector, TF-IDF, similitud de cosenos

Estructura del Proyecto

El proyecto está dividido en dos partes principales: MoogleServer, que contiene el servidor e implementación del Frontend del motor y MoogleEngine, que contiene la lógica que hay detrás del motor. Nos centraremos en describir esta última en el reporte.

Durante el análisis algunos detalles de implementación serán omitidos. De lo contrario el documento sería demasiado extenso. Para ver estos se pueden revisar los comentarios en el código fuente del programa.

MoogleEngine está compuesto de las siguientes clases, que iremos desglosando en los textos que siguen.

1. Document
2. DocumentCatcher
3. SearchItem
4. SearchResult
5. TFIDFAnalyzer
6. StringMethods
7. Moogle
8. Trie (clase adicional para futuras modificaciones)

1.Document

Nos permite representar objetos de tipo documento, con las propiedades que utilizaremos de ellos para cada una de las consultas.

1. Title (título del documento)
2. Text (texto contenido en el documento)
3. LowerizedText (texto en minúsculas, para solo preocuparnos por la semántica de los datos y no de la capitalización, esta propiedad es usada en la construcción de snippets para nuestra búsqueda en la clase Moogle)
4. Words (contiene un arreglo con todas las palabras del texto separadas, esta separación está dada por los caracteres que no constituyen letras o números, ej: signos de puntuación)
5. WordFrequency (contiene cuantas veces aparece cada palabra en el texto)
6. TF (representa el vector con los valores de TF para cada palabra, en el desglose de la clase TFIDFAnalyzer veremos que significa el TF y el IDF en cada documento)
7. TFIDF (vector de TF * IDF de cada palabra, estos son los vectores que utilizaremos en nuestra búsqueda)
8. WordPos (posiciones de cada palabra en el documento para la implementación del operador ~ en la clase Moogle)

2.DocumentCatcher

Esta clase estática contiene los métodos y constantes necesarios para procesar los archivos y convertirlos a nuestra clase Document.

Métodos:

1. IgnoreASCII() genera todos los caracteres que ignoraremos al procesar los textos y actuarán como separadores de las palabras. Internamente recorre todos los caracteres ASCII y considera todos los que no son letras o números.
2. IgnoreASCIIQuery() este método funciona como el anterior pero no elimina los caracteres especiales para los operadores de búsqueda, véase: ! * ^ ~
3. IsAlphanumeric() comprueba si un carácter es letra o número (esta función realiza la misma función que char.IsLetterOrDigit y sólo permanece en el código por motivos de seguridad, en versiones posteriores puede ser refactorizada y eliminada).
4. IsOperator() comprueba si un caracter es uno de los cuatro operadores especiales (es una función de utilidad empleada en varios lugares del código).
5. InvalidWord() comprueba si una palabra es inválida o no basándose en si contiene letras o números y no símbolos extraños.
6. ReadDocumentsFromFolder() acepta un directorio en forma de string como parámetro y devuelve un arreglo de Document, construido a partir de los .txt contenidos en este

directorio. Esta es la función que utilizamos para construir los documentos sobre los que realizaremos la query.

Propiedades:

1. Delims[] contiene todos los caracteres que actúan como separadores para las palabras de los documentos (es construido a partir de IgnoreASCII)
2. DelimsQuery[] contiene todos los caracteres que actúan como separadores para la query (es construido a partir de IgnoreASCIIQuery)

3.SearchItem

Representa los elementos devueltos en la query, e implementa de la clase IComparable para poder ordenar usando Array.Sort()

1. Title (título del documento devuelto)
2. Snippet (fragmento de texto del documento que contiene al menos una palabra de la query)
3. Score (puntuación del documento empleada para poderlos ordenar de mayor a menor relevancia)

Nota: El método CompareTo permite comparar dos SearchItem en base a sus puntuaciones.

4.SearchResult

Contiene los resultados de la búsqueda y una sugerencia de cómo podríamos obtener mejores resultados de la búsqueda (esta será la misma que la query si encontramos resultados relevantes)

1. items[] (arreglo de SearchItem que contiene los resultados de la búsqueda)
2. suggestion (string que contiene una nueva query con la que podríamos mejorar los resultados)

5.TFIDFAnalyzer

Esta clase estática contiene los métodos necesarios para poder construir los vectores de TF-IDF necesarios para procesar las consultas(queries).

Un vector de TF-IDF está formado digamos por N componentes(donde cada una representa a una palabra), que son números reales. Estos números reales que representaremos dan una cierta noción de que tan raras e importantes o no son cada una de las palabras que estamos procesando en nuestras queries y documentos, y están dadas por las fórmulas siguientes:

$$TF(palabra,\ Documento) = \frac{cantidadDeOcurrenciasDePalabraEnDocumento}{cantidadDePalabrasEnDocumento}$$

$$IDF(palabra) = ln\ \frac{cantidadDeDocumentos}{cantidadDeDocumentosEnLosQueAparecePalabra}$$

Luego el TF-IDF es el producto de estas dos medidas. Notemos que si una palabra aparece en todos los documentos su IDF será $ln(1) = 0$, y el producto del TF-IDF será 0 tambíen, por lo que la palabra no será relevante.

Este tipo de palabras no relevantes son llamadas StopWords y no nos interesan en nuestra búsqueda.

Volviendo a lo anterior nuestros métodos:

1. CalculateDocumentsWithTerm() recorre todas las palabras del vocabulario y determina cuantos documentos contienen a cada una, esta información es almacenada en Dictionary<string,int> DocumentsWithTerm en Moogle.cs
2. CalculateIDFVector() calcula el vector de IDF para todos los documentos y lo almacena en IDFVector en Moogle.cs
3. CalculateTFVector() calcula el vector de TF para cada documento
4. CalculateTFIDFVector() una vez calculado el vector de IDF y los vectores de TF construye el vector de TFIDF para cada documento.
5. Norm() recibe un Dictionary<string,float> vec, que no es más que un vector de TF-IDF y calcula su norma, que es el valor por el cual tenemos que dividir cada valor en el vector para tener un vector unitario, de cierta manera esto es parecido a la Fórmula de Pitágoras pero generalizada a N-dimensiones, ya que nos da algo así como el tamaño del vector. Este método será útil a continuación cuando veamos ComputeRelevance(). Su valor es la raíz cuadrada de la sumatoria de los cuadrados de cada componente.
6. ComputeRelevance() permite determinar que tan relevante es un documento. Para esto como ya tenemos representados los vectores de TF-IDF por decir así como vectores del espacio R^N entonces queremos ver que tan cerca de ser colineales están esos vectores, porque esto nos dice que tan cerca de ser iguales (de contener lo mismo) están, para ello usaremos la siguiente fórmula:

$$Relevance = \frac{\upsilon \times \omega}{||\upsilon|| \times ||\omega||}$$

para el cálculo del numerador recorreremos cada componente del vector de la query verificando que también esté en el documento y haremos producto de vectores con estos. Luego para el denominador usaremos la función Norm().

En la implementación se tiene cuidado con los denominadores nulos, para no causar indefiniciones y errores en tiempo de ejecución.

6.StringMethods

Esta clase estática contiene los métodos necesarios para poder construir la sugerencia en Moogle.cs cuando no encontramos resultados

1. EditDistance() que devuelve la distancia entre dos strings que recibe como parámetro, es decir la mínima cantidad de caracteres que hay que insertar, remover o cambiar para que los dos strings sean iguales
2. LongestCommonSubsequence() devuelve la subsecuencia común más larga entre dos strings que recibe como parámetro.

Ambos métodos son calculados usando Programación Dinámica para lograr mayor eficiencia.

7.Moogle

Este es el núcleo de nuestro proyecto y donde procesamos las consultas a mostrar. Tiene un constructor estático el cual es accedido por Index.razor en MoogleServer para precalcular la información necesaria para las queries.

Métodos:

1. Preproccess() es llamado por static Moogle() para precalcular la información de los documentos, construye los documentos y el cuerpo de vectores de TFIDF(corpus) utilizando los métodos de TFIDFAnalyzer, Document y DocumentCatcher.
2. ResetGlobalVariables() es llamado cada vez que hacemos una nueva query para limpiar la información de la query anterior y poder calcular la nueva sin errores.
3. CalculateNewRelevane() incrementa la relevancia de las palabras en el vector de TF-IDF de la query multiplicando por 2.0f por cada asterisco encontrado en el prefijo de la palabra.
4. UpdateNewRelevance() actualiza la nueva relevancia en el vector de la query
5. FindSearchResults() inserta a la lista de resultados los documentos más relevantes para la búsqueda. Considera los 4 operadores especiales. Recorre todas las palabras de la query y analiza todos los documentos, solo considerando aquellos que tienen palabras obligatorias (de haberlas) y que no tienen palabras prohibidas, sugiriendo los que mayor score tienen, y de todos estos en caso de existir el operador ~ en la query se incrementa la

relevancia según las palabras cercanas. Luego se construye el snippet del documento y se añade a la lista. Se usan expresiones regulares (RegEx) para construir el snippet, osea para encontrar el fragmento de texto que contiene palabras de la query. En caso de no poder construir el snippet por algún símbolo extraño haberlo impedido, entonces no se sugiere.

6. BuildSuggestion() devuelve un string que es la sugerencia de lo que debería ser escrito en el input para obtener un resultado en la búsqueda. Recorre todas las palabras de la query y las sustituye por la palabra con menor EditDistance, de haber varias, escoge la de menor LongestCommonSubsequence.
7. BuildSnippet() construye el snippet a partir de un documento dado (recibe como parámetros las palabras del documento, el texto original, y el texto en minúscula) y recorre las palabras de la query, cuando encuentra una de ellas intenta mostrar 20 caracteres a la izquierda y 200 a la derecha y devuelve este intervalo en el texto original.
8. FindWordsToBeNear() construye una lista que relaciona que palabras tienen que estar cerca de otras para el operador ~. Devuelve un objeto de tipo List<Tuple<int,int>>.
9. Query() recibe un string query que es obtenido desde Index.razor y encuentra todos los documentos relevantes ordenados utilizando los métodos anteriores. Los devuelve como un SearchResult.

8.Trie

Esta clase se encuentra en el programa para poder ser usada en funcionalidades futuras. Ej: Sugerencia de palabras en tiempo real, búsqueda de palabras con el mismo prefijo (esta funcionalidad ya se encontraba en el programa pero tuvo que ser removida porque es demasiado costosa en memoria)

9.Matrix

Esta clase contiene todas las operaciones a evaluar con matrices, y algunas adicionales, como son:

1. Add() que recibe una matriz genérica y devuelve la matriz suma entre la matriz de la que se llamó el método y la otra matriz.
2. Multiply() que recibe una matriz genérica y devuelve la matriz producto entre ambas, en caso de las matrices no poderse multiplicar por no tener la misma cantidad de columnas la primera que de filas la segunda, entonces arroja una excepción.
3. ProductByScalar() recibe un escalar y devuelve la matriz multiplicada por ese escalar.
4. Print() imprime la matriz
5. DeterminantOrder1() calcula el determinante de orden 1.
6. DeterminantOrder2() calcula el determinante de orden 2.
7. DeterminantOrder3() calcula el determinante de orden 3.
8. Determinant() detecta el orden de la matriz y si es cuadrada, en caso de serlo y ser de orden a lo sumo 3 calcula el determinante, sino arroja una excepción. Este método puede ser extendido a órdenes mayores usando desarrollo por menores y recursividad.

Nota: La matriz ha sido implementada genérica para poder trabajar con cualquier tipo de dato numérico