¡Proyecto de Programación I. Moogle!
Facultad de Matemática y Computación
Universidad de La Habana Curso 2022

Nombre: Dennis Daniel González Duran Grupo:121

En este proyecto realizaremos una aplicación cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos utilizando la tecnología .NET Core 6.0 y el lenguaje de programación C#.

La búsqueda se basa en la utilización de la Frecuencia de Término – Frecuencia Inversa del Documento (TF-IDF) que es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección y nos permite además manejar el hecho de que algunas palabras son generalmente más comunes que otras. Como método de busca principal se utiliza el método conocido como similitud del coseno, que permite conocer que tan parecidos son dos documentos a través del ángulo que existe entre los dos documentos luego de ser expresados en forma vectorial.
Este método usa la formula Cos(&) = Sumapunto / (magnitud_documento * magnitud_query)
Donde Sumapunto = Sumatoria "de 0 a n" de vector_query[n] * vector_documento[n]
Magnitud ^2 = (vector [0]) ^2 + (vector [1]) ^2.....+(vector[n]) ^2.

Además de las clases definidas en la orientación del trabajo, el proyecto usa la clase InfoDocs, dedicada al manejo de la información que se puede obtener a partir de los documentos (Frecuencia de Término (TF), Frecuencia Inversa del Documento (IDF) y palabras cercanas entre sí). También utiliza la clase MakeSearch donde se realiza todo lo referente a la obtención de los vectores (tanto de la query como de los documentos) y al cálculo del coseno entre ambos vectores.

Se procede de la siguiente forma:

CLASE INFODOCS:

A paritr de la clase InfoDocs y los métodos GetPath y GetFiles obtendremos la ruta donde se encuentran los documentos y la dirección de cada documento de forma individual.
Luego con los métodos ManejarTexto y Readfiles leeremos cada documento, reconoceremos cada palabra con los char's delimitadores y luego almacenaremos cada palabra de cada documento en una lista, para luego almacenar cada una de estas listas en una lista llamada ListadeListas.

A partir de esta ListadeListas se utiliza el método CreateD, que nos permitirá obtener un diccionario de la forma <string, double []> este diccionario tendrá entre sus claves todas las palabras de todos los documentos y cada clave tendrá como valor un array que almacenara en la posición 0 el IDF de cada palabra y en el resto de posiciones el TF de la palabra en el documento asociado a esa posición del array.

A partir de esta ListadeListas además, con la utilización del método NearWords podremos obtener una Lista de diccionarios donde cada diccionario representara a un documento. Estos diccionarios contendrán como clave todas las palabras del documento al que representan y como valor una colección de las palabras cercanas a la palabra clave. Para luego ser utilizado en el operador de cercanía.

CLASES MOOGLE y MAKESEARCH:

Con los métodos NormalizeQuery y AnalizeQuery podremos sacar diferente información de la query, como reconocer sus palabras y crear una lista con las palabras de la query, así como otra lista con las palabras de la query que se encuentran modificadas por algún operador.

Luego de esto se creará mediante el método DataVector un array con el TF-IDF de las palabras de la query, que significará el query en forma de vector. Vector que será modificado en caso de aparecer en el query el operador de importancia por el método ImportantOperator, para adaptar la búsqueda a la palabra modificada por este operador.

Luego calcularemos la magnitud del vector de la query usando el método MagnitudeQuery basado en la fórmula: Magnitud ^2 = (vector [0]) ^2 + (vector [1]) ^2.....+(vector[n]) ^2.

Luego con la ayuda de los métodos Sumapunto, MagnitudeDoc y Similarity podremos saber el ángulo entre el vector de cada documento y el query, así como guardar esa información en un diccionario llamado match.

Modificamos luego con el método Operators este diccionario para ajustar los resultados al resto de los operadores y al ordenar ese diccionario tendremos en orden los resultados deseados.

SNIPPETS Y SUGERENCIAS

Para el snippet se identificará el orden de importancia de las palabras de la query y en dependencia de si aparecen o no en los documentos se creará el snippet con la ayuda del método Snippets.
Para la sugerencia tomaremos la query e identificaremos que palabras de la query no aparecen en ningún documento, si la palabra no aparece tal vez el usuario cometió un error al escribirla, por lo que usando el Método H_Distance basado en el algoritmo de distancia de Hamming, encontraremos la palabra más cercana a la palabra que estamos analizando y la agregaremos a la sugerencia.

Bibliografía:
https://es.wikipedia.org/wiki/Tf-idf
https://stuka2013.wordpress.com/2013/01/30/algoritmo-distancia-hamming-levenshtein-en-c/
https://es.m.wikipedia.org/wiki/Similitud_coseno
Miguel Katrib. “Empezar a programar. Un enfoque multiparadigma con C#”.