Proyecto de Programación 1 de Ciencias de la Computación de primer año.

Autor: Tiffany Valdés Laferté C-121

Moogle! es un buscador de archivos, es una explicación que le daríamos a alguien para que pudiera entenderlo al momento, pero realmente contiene muchas líneas de código detrás para que lo hagan funcionar. Moogle! permite buscar en una carpeta que contenga un número X de documentos los documentos que tengan mayor coincidencia con la búsqueda que le introduzcamos en el buscador. Si ponemos "manzanas y naranjas", por ejemplo, nos debe devolver los documentos que tengan las palabras clave "manzana" y "naranja" que tengan el score más alto.

Entonces, ¿Cómo es que el Moogle! funciona?. Primero, explicaremos las distintas clases del proyecto y los pasos que sigue para mostrarnos los resultados de la búsqueda.

Antes de poder buscar los archivos debemos meterlos en la carpeta (como se había especificado en los pasos a seguir en el "Readme"), luego empieza a suceder toda la "magia". En la carpeta MoogleServer se encuentra un .cs llamado "Program", con un método llamado "Loading" que se encarga de cargar toda la base de datos de la carpeta, y junto con ello calcular el score y varias funciones más que se mencionaran algo más adelante en el reporte. El método "Loading" se encuentra en la clase Data, en el namespace "MoogleEngine" y se encarga de crear una DataBase, la cual tendrá ya integrada muchos de los requisitos. La DataBase esta conformada por un array de documentos, un diccionario que mapea un string a un diccionario y ese segundo diccionario de enteros a enteros y unos vectores (se explicará más adelante en el informe que hacen los vectores). El método Loading se encuentra realmente también en la clase Data (que es donde está implementado), y en su interior contiene otros 3 métodos, el SearchFiles, SetContenido y CreateVectors, que se encuentran en la misma clase que Loading. SearchFiles usa el método EnumerateFiles, que usa la librería System.IO, su función es la siguiente: dado un directorio que le introduzcamos, este buscara en el los documentos de tipo txt que se encuentren en la ruta del directorio, los documentos se guardarán en una ruta del directorio y estos se agregarán al array de documentos mencionado anteriormente. SetContenido se encarga de recorrer las palabras que se encuentran en los documentos una vez se ha puesto en marcha el SearchFiles, ¿recorrer las palabras en un txt es importante? Si, gracias a la clase Aux, pudimos crear distintas herramientas para que la clase Data en vez de tener unas 100 líneas de código más, estas estén en otra clase para tener una mejor visualización del código. En Aux tenemos un método llamado "tokenizer" que se encarga de recibir las palabras que se obtuvieron de los documentos y que se permita su lectura ya este escrita en mayusculas o minúsculas, elimine los caracteres especiales o separadores que se encuentren delante (o detrás) de estas, devolviendo "tokens", que serían las palabras ya "limpias" para poder ser utilizadas. Una vez hecho esto, se recorre usando un bucle foreach en el que se encuentra este mismo método donde cuenta el número de veces que las palabras se repiten, y así agregarlas al 2do diccionario que enumera el número de veces que cada palabra aparece en los archivos.

Ya tenemos casi completa la DataBase, solo nos falta la vectorización de documentos. El último método dentro de Loading, el método CreateVectors, se encarga de calcular el TF-IDF y de utilizar los métodos de la clase "Vectorizing". En esta clase se crea un diccionario "v" que almacena un conjunto de variables double asociadas a strings y un constructor para poder acceder a este. En la clase Vectorizing se llevan a cabo varias operaciones, las cuales son para hallar el producto escalar entre 2 vectores y la norma Euclidiana y luego dar paso a calcular el modelo vectorial y la similitud de cosenos. Para utilizar todas esas operaciones mencionadas se debe calcular el TF-IDF mencionado previamente, que para resumir su funcionalidad en el código, este se encarga de procesar los documentos y calcular su importancia en el conjunto de documentos, creando un vector que sera utilizado para las operaciones mencionadas. Para más especificaciones de cómo funcionan cada una de las implementaciones mencionadas en este párrafo a manera resumida:

El TF-IDF se calcula en 3 partes: el TF se encarga de calcular la frecuencia de las palabras en un documento específico para medir la relevancia en el documento de manera individual, se calcula dividiendo el número de veces que la palabra aparece en el documento entre el número total de las palabras en el documento; el IDF calcula que tan importante es la palabra entre todos los documentos, y se calcula hallando el logaritmo en base 10 del resultado de la división del número total de documentos sobre el número de documentos que contienen la palabra; luego el TF-IDF que es el producto del TF con el IDF para otorgarle un valor TF-IDF al documento que se le calculó en todo el cuerpo de documentos.

La norma Euclidiana servirá para calcular la magnitud de un vector en un espacio y luego ser usada para calcular la similitud de cosenos.

El modelo vectorial se encargara de un mejor procesamiento para representar mejor las palabras de la query y de los documentos.

La similitud de cosenos se empleará para determinar la similitud de dos vectores que serán los documentos ya asociados a un valor TF-IDF, y esta operación sera la que nos dará el score final.

Los documentos no pasan solamente por este proceso, la query (la búsqueda que introduce el usuario) también pasa por el mismo proceso para que cuando se ejecute la búsqueda esta muestre los resultados con mayor relevancia, si el valor del documento no es 0 entonces se mostrara en los resultados de la búsqueda, enseñando el título del documento y el snippet, que es la zona de mayor relevancia donde aparecen las palabras de la query. Los resultados de la búsqueda se ordenarán desde el que tenga el mayor score al menor.

El proyecto aun no es perfecto y se puede seguir perfeccionando, en el futuro se irá actualizando e incluyendo nuevas funcionalidades para que la búsqueda sea más eficaz o mejore la comodidad del usuario que esta usando.