Moogle! : Proyecto de Programación I

Cristhian Delgado García

1 Introducción

Moogle! es un pequeño motor de búsqueda hecho en **C#** basado en el **Modelo de Espacio Vectorial** que permite hacer consultas sobre un conjunto de archivos con extensión *.txt*; dichas consultas aspiran a arrojar los resultados más adecuados sin comprometer la simplicidad del buscador. A lo largo de este informe se explicarán brevemente los fundamentos matemáticos que dan paso a su implementación y su implementación en sí.

2 Desarrollo

2.1 Los fundamentos matemáticos

- **Modelo de Espacio Vectorial** : Se conoce como modelo de espacio vectorial a un modelo algebraico utilizado para filtrado, recuperación, indexado y cálculo de relevancia de información. Representa documentos en lenguaje natural de una manera formal mediante el uso de vectores (de identificadores, por ejemplo términos[1] de búsqueda) en un espacio lineal multidimensional. En el área de recuperación de información normalmente se usa una expresión vectorial, donde las dimensiones del vector representan términos, frases o conceptos que aparecen en el documento. En este aspecto la representación más adoptada es la conocida como bolsa de palabras: una colección de documentos compuesta por **n** documentos indexados y **m** términos representados por una matriz documentos-términos de **n** x **m**. Donde los m-vectores fila[2] representan los **n** documentos; y el valor asignado a cada componente refleja la importancia o frecuencia ponderada que produce el término, frase o concepto t_i en la representación semántica del documento j. Cada documento[3] puede ser escrito como:

$$d_j = \left[w_1 j \ w_2 j \ \ldots \ w_m j\right] \tag{1}$$

[1] Se considera en esta implementación como término a una palabra tomada sin repitición del cuerpo de documentos.

[2] Aunque aquí se explican la representación de los documentos en vectores fila y los términos en vectores columna en esta implementación serán tratados de manera análoga pero al revés (los documentos son representados como vectores columna y los términos como vectores fila)

[3] La query será tratada como un documento más cuando se considere pertinente para simplificar la implementación y la explicación.

Donde **m** es la cardinalidad del conjunto de términos y cada w_{ij} representa el peso del término t_i en el documento d_j. En la práctica hay varias maneras de calcular estos pesos y hacer la matriz términos-documentos, esta implementación usará **TF-IDF**.

- **TF-IDF** (del ingles *Term frequency-Inverse document frequency*) es una medida numérica que expresa que tan relevante es una palabra para un documento en un cuerpo de documentos. El valor del tfidf aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensado por la frecuencia de la palabra en la colección de documentos lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras. Además puede utilizarse exitosamente para el filtrado de las *stop-words*[4] (palabras que suelen usarse en casi todos los documentos). Tf-idf es el producto de dos medidas, frecuencia de término y frecuencia inversa del documento. Existen varias maneras de determinar el valor de ambas.

  Para hallar la **frecuencia del término** tf(t,d) la opción que se usará es la frecuencia absoluta del término t en el documento d, o sea el número de veces que el término t ocurre en el documento d. Si se denota la frecuencia absoluta de t en el documento d por f(t,d), entonces:

$$\mathrm{tf}(t,d) = \mathrm{f}(t,d) \tag{2}$$

  .

  La **frecuencia inversa del documento** es una medida de si el término es común o no, aunque hay varias maneras de calcularlo, esta implementación usará su expresión más sencilla:

$$\mathrm{idf}(t,D) = \log \frac{N}{n_t} \tag{3}$$

  Donde N es la cantidad de documentos en el cuerpo de documentos y n_t es la cantidad de documentos que contienen al término t. Matemáticamente la base del logaritmo no es importante y constituye un factor constante en el resultado final.

  Finalmente *tfidf* se calcula como:

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \times \mathrm{idf}(t,D) \tag{4}$$

  Un peso alto en *tfidf* se alcanza con una elevada frecuencia del término (en el documento dado) y una pequeña frecuencia del término en la colección completa de documentos. Además el valor de esta medida siempre es no negativo.

[4]Esto será de utilidad en esta implementación para poder hacer una búsqueda basada en el peso de las palabras y no solo en su frecuencia absoluta.

- La **similitud coseno** es una medida de similitud entre dos vectores no nulos, la usaré para calcular la similitud entre un documento y una consulta (representados como vectores por supuesto). Se puede hallar mediante la siguiente expresión:

$$\cos(v, w) = \frac{v \cdot w}{|v||w|} \tag{5}$$

  El numerador representa el producto punto entre los vectores v y w que se calcula mediante la siguiente fórmula:

$$v \cdot w = \sum_{i=1}^{n} v_i w_i \tag{6}$$

  El denominador representa el producto entre los módulos de los vectores v y w. El módulo de un vector puede calcularse así:

$$\|v\| = \sqrt{\sum_{i=1}^{n} v_i^2} \tag{7}$$

  Donde los v_i y w_i representan las componentes del vector v y w respectivamente.

  Como los vectores documento tienen sus componentes no negativas la similitud coseno tendrá un valor no negativo.

A modo de resumen, los documentos pueden ser representados como vectores, cuyas entradas son los pesos de los términos del cuerpo de documentos. Para hallar esos pesos se puede usar la medida **TF-IDF**. Para calcular la similitud o *puntaje de relevancia* entre un documento y la consulta se usa la **similitud coseno** entre los vectores que los representan[5].

2.2 La implementación

El código principal de la "lógica" del buscador se encuentra en el archivo *Moogle.cs*. Este archivo se divide en cuatro clases : `Moogle`, `Documents`, `Query` y `Vector`. La estructura de cada clase es igual y sigue este orden:

1. Variables de instancia (o campos en el caso de la clase `Moogle`).
2. Propiedades.
3. Métodos.

Los métodos que tienen un prefijo `Get` forman parte de la "lógica" de una propiedad, por lo que son ejecutados al instanciar cada clase (excepto en la clase

[5] Resta ordenar en forma descendiente los *puntajes de relevancia* calculados y se obtendrá el *orden de relevancia*.

`Moogle` por supuesto). Dicho esto se explicará el funcionamiento del código, explicando cada clase del archivo, así como los métodos y principales propiedades de cada clase.

Clases:

- `Documents` : se encarga de procesar un cuerpo de documentos con extensión *.txt* dentro de un directorio.

  Métodos:

  - `Documents(string DirRoute)`: es el constructor de la clase, toma como parámetro un `string` con la ruta del directorio que contiene el cuerpo de documentos.
  - `TokenizeDocument(string DocRoute)` : toma la ruta de un documento y devuelve un *array* de `string` con las palabras que conforman el documento sin símbolos extraños ni cadenas vacías.
  - `ComputeDocTF()`: devuelve un diccionario donde cada clave es la ruta de cada uno de los archivos *.txt* y cada valor es un diccionario que tiene como claves las palabras (sin repetir) de su respectivo archivo y como valores sus respectivas frecuencias absolutas en él. Utiliza a `TokenizeDocument()` como un método auxiliar y a la propiedad `FilesRoutes` (Véase en la subsección de ***Propiedades***).
  - `GetWordSet()` : devuelve un *array* de `string` con todos las palabras (sin repetición)o términos del cuerpo de documentos. Utiliza la estructura `HashSet` para una mayor eficiencia y la propiedad `DocTF` (derivada del método `ComputeDocTF()`).
  - `GetTF()` : devuelve un *array* bidimensional de `int` que representa la matriz **TF** del cuerpo de documentos, donde el elemento en la i-ésima fila y la j-ésima columna corresponde al tf del i-ésimo término en el j-ésimo documento. Utiliza la propiedad `DocTF` y la propiedad `WordSet` (derivada del método `GetWordSet()`).
  - `GetIDF()` : devuelve un *array* con los idf de cada término en el conjunto de documentos, donde el i-ésimo elemento corresponde al idf del i-ésimo término en el cuerpo de documentos. Utiliza la propiedad `TF` (derivada del método `GetTF()`).
  - `GetTF_IDF()` : devuelve un *array* bidimensional de `double` que representa la matriz **TF-IDF** del cuerpo de documentos, donde el elemento en la i-ésima fila y en la j-ésima columna representa el tfidf del i-ésimo termino en el j-ésimo documento. Utiliza la propiedad `TF` y la propiedad `IDF` (derivada del método `GetIDF()`).

  Propiedades:

  - `FilesRoutes`: *array* de `string` con las rutas de los archivos *.txt* contenidos en la carpeta cuya ruta fue pasada como parámetro en el constructor de la clase.

- `DocTF` : derivada del método `ComputeDocTF()`.
- `WordSet` : derivada del método `GetWordSet()`.
- `IDF` : derivada del método `GetIDF()`.
- `TF_IDF` : derivada del método `GetTF_IDF()`.

- `Query` : se encarga de procesar la consulta hecha por el usuario.

  Métodos:

  - `Query(string query, string[] WordSet, double[] IDF)` : es el constructor de la clase, recibe como primer parámetro un `string` con la consulta hecha por el usuario, como segundo un *array* de `string` con el conjunto de palabras del cuerpo de documentos y como tercero un *array* de `double` con el idf de cada término en el conjunto de documentos.
  - `GetTokenizedQuery()`: devuelve un *array* de `string` con las palabras que componen la consulta hecha por el usuario (pasada en el primer parámetro del constructor de la clase) sin símbolos extraños ni cadenas vacías.
  - `GetTF()` : retorna un *array* de `int` que representa la matriz **TF** de la consulta, donde el i-ésimo elemento corresponde a la frecuencia absoluta del i-ésimo término del cuerpo de documentos en la consulta (para esto utiliza el conjunto de palabras o términos pasado en el segundo parámetro del constructor de la clase).
  - `GetTF_IDF()` : retorna un *array* de `double` que representa la matriz **TF-IDF**, donde el i-ésimo elemento representa el tfidf del i-ésimo término del conjunto de documentos en la consulta. Utiliza la propiedad `TF` (derivada del método `GetTF()`) y el *array* pasado en el tercer parámetro del constructor de la clase.

  Propiedades:

  - `TF_IDF`: derivada del método `GetTF_IDF()`.

- `Vector` : crea un vector a partir de un *array* numérico con sus componentes y define operaciones entre vectores.

  Métodos:

  - `Vector(double[] components)` : constructor de la clase, toma como único parámetro un *array* de `double` con las componentes del vector.
  - `GetModule()` : devuelve el módulo del vector, calculado a partir de sus componentes (pasadas como parámetro).
  - `DotProduct(Vector v1, Vector v2)` : método estático que toma como parámetros dos vectores y devuelve un `double` que representa su producto punto a partir de sus componentes (pasadas como parámetro).

- `Cos(Vector v1, Vector v2)` : Devuelve un `double` que representa el coseno entre dos vectores si ambos tienen módulos distintos de cero y **0** si alguno tiene el módulo igual a cero. Utiliza la propiedad `Module` (derivada del método `GetModule()`) y el método `DotProduct(...)`.

- `Moogle` : clase estática que se encarga de realizar consultas sobre el cuerpo de documentos(los archivos con extensión *.txt* que se encuentran en la carpeta *Content*).

  Propiedades:

  - `initDocs()` : inicializa el campo `documents`, que es una instancia de la clase `Documents`, con la ruta de la carpeta *Content*. Es ejecutado mientras se renderiza la interfaz gráfica.
  - `ExtractText(string filePath, string query)`:extrae un fragmento de texto de un documento que contenga alguna de las palabras de la consulta, y lo devuelve como un `string`. Si no hay ninguna coincidencia, devuelve un mensaje indicándolo. Utiliza expresiones regulares.
  - `GetScores(Query userInp)`: devuelve un *array* de `double` donde el i-ésimo elemento corresponde al puntaje de relevancia o score de la query con el i-ésimo documento. Para ello utiliza la propiedad `TF_IDF` de los objetos `documents` y `userInp` y el método estático `Cos(...)` de la clase `Vector`.
  - `GetResults(double[] scores)` devuelve un *array* de `string` donde al i-ésimo elemento le corresponde la ruta del i-ésimo documento. Para esto utiliza la propiedad `DocTF` de `documents`.
  - `Query(string query)`: recibe la consulta hecha por el usuario y devuelve un objeto del tipo `SearchResult` que contiene a la consulta y a un *array* con objetos del tipo `SearchItem`, cada uno con el nombre del documento, un fragmento de ese documento o snnipet y un score o puntaje de coincidencia, ordenados de mayor a menor por el score. Para esto:
    1. Se instancia el objeto userInp de la clase `Query`.
    2. Se calculan los scores usando el método `GetScores(...)` y se almacenan en la variable `scores`.
    3. Se crea una variable `positiveScores` que almacena los scores positivos.
    4. Se almacenan en la variable `results` usando el método `GetResults(...)` las rutas de los documentos.
    5. Se llama al método estático `Sort(...)`de la clase `Array` para ordenar el *array* `results` respecto al *array* `scores` de menor a mayor y luego al método estático `Reverse(...)` de la misma clase con cada *array* para que se ordenen de mayor a menor.
    6. Se crea un *array* de `SearchItems`, `items` que guarda los objetos cada del tipo `SearchItem` con un score positivo, usando un ciclo que se detiene cuando ya no hay más scores no nulos.

7. Se retorna un objeto instanciado de `SearchResult` con el *array* `items` y el `string` `query`.

Así se tienen los documentos que coinciden con la consulta ordenados de más relevante a menos relevante con sus respectivos snippets.

3 Conclusiones

En este informe se ha presentado el diseño e implementación de **Moogle!**, un motor de búsqueda sencillo pero eficaz que utiliza el **Modelo de Espacio Vectorial** para ordenar los documentos más relevantes para una consulta dada. Se han explicado sus fundamentos matemáticos, así como el código empleado para su realización en **C#**. Se ha demostrado que el motor de búsqueda es capaz de procesar consultas sobre un conjunto de archivos de texto y devolver los resultados ordenados por su similitud con la consulta, utilizando medidas como el producto escalar, la distancia euclidiana y el coseno del ángulo entre dos vectores. Se concluye que **Moogle!** es un proyecto didáctico que ilustra los principios básicos de la recuperación de información y que ofrece una experiencia de búsqueda satisfactoria al usuario.