Universidad de la Habana

Facultad de Matemática y Computación

Proyecto Intrasemestral de Programación
Moogle!

Autor:
Christopher Guerra Herrero
Septiembre, 2022

1 Información básica

1.1 Google, pero con M

Moogle es un *buscador* creado con C# para el back-end y Razor para la interfaz. Su objetivo es, dado una *consulta* y un conjunto de ficheros .txt, indicarle al usuario que documentos se asemejan más a su búsqueda.
Con en el fin de hacer más grata la experiencia del usuario, nuestro programa cuenta con una serie de herramientas para hacer más eficiente la búsqueda (operadores), y sugiere consultas semejantes a la hecha por el usuario, cuando esta no tiene resultados. Para alcanzar estas metas, el programa hace uso de diferentes algoritmos que se describirán en los apartados siguientes

1.2 Cómo funciona la búsqueda?

Para brindar resultados, Moogle, hace uso de un *modelo vectorial*, el cual consiste en considerar como vectores n-dimensionales a los archivos .txt y a la consulta; y luego hallar el coseno del ángulo formado entre estos con el fin de hallar los vectores con menor ángulo, es decir, los vectores que más se aproximan.
Para esta abstracción de considerar un texto como un vector n-dimensional se usa el principio TF-IDF. Si el lector desea saber más sobre este principio consulte [1].

1.3 Se pueden usar chirimbolos!

Con el fin de hacer más certera las consulta del usuario, se pueden utilizar los siguientes operadores:

- **Operador ^ :** Se usa antepuesto inmediatamente a una palabra e indica que todos los documentos brindados deben contener dicha palabra (Ejemplo: ^ Garfield).

- **Operador ! :** Se usa antepuesto inmediatamente a una palabra e indica que ninguno de los documentos brindados deben contener dicha palabra (Ejemplo: !Garfield).

- **Operador * :** Se usa antepuesto inmediatamente a una palabra e indica que esa palabra es importante en la búsqueda. Es decir, si un documento contiene esta palabra su Score quedará multiplicado x2. Su efecto es aditivo, es decir, pueden ponerse varios para aumentar aún más la importancia de la palabra (Ejemplo: ***Garfield).

- **Operador ~ :** Se usa entre dos palabras separado por espacios. Indica que entre más cercanas sean estas palabras en un documento más importante será este, puesto que el Score quedará multiplicado por un coeficiente entre 1 y 10. (Ejemplo: Garfield ~ Odi).
- **Operador " :** Se usan para seleccionar una frase, uno al principio y otro al final. Indica que dicha frase debe aparecer textual en todos los resultados de la búsqueda (Ejemplo: "Garfield ama a la lazaña").
- **Operador ' :** Se usan para seleccionar una frase, uno al principio y otro al final. Indica que dicha frase no puede aparecer textual en ninguno de los resultados de la búsqueda (Ejemplo: 'Garfield odia a Odi').

1.4 Puede que hayas querido decir...

Tener faltas de ortografía no nos va a impedir trabajar con Moogle ya que este está diseñado para ofrecer *búsquedas alternativas* cuando la hecha por el usuario no tiene resultados(ya sea por errores en la escrituta o por la no aparición de coincidencias en nuestros documentos). Esta sugerencia aparecerá en la parte inferior del cajón de búsqueda con color de fuente azul(palabra caliente).

2 Sobre la implementación

Nuestro código trabaja con tres clases: Moogle, la clase principal; Metodos, clase para agrupar los diferentes métodos que se necesitan en Moogle; y Ficha, clase que caracteriza a los diferentes documentos y es usada en todo el código indistintamente. La clase Moogle cuenta con dos métodos , el constructor y Query(). Al cargar nuestra página principal en el navegador una instancia de Moogle es creada, por lo cual se ejecuta el constructor de la clase y se cargan los documentos(entiéndase aplicar el principio del TF-IDF a los documentos). Para ello se hace uso de diferentes métodos de la clase Metodos y almacenando las informaciones en diferenres instancias de ficha.

2.1 Flujo

2.1.1 Constructor Moogle()

El constructor de la clase Moogle se encarga de cargar los archivos .txt y calcularles su TF-IDF. En este proceso son utilizados dos métodos estáticos de la clase Metodos. Además de guardar los resultados obtenidos en la

propiedad TFIDF de las instancias de ficha, también son almacenadas las posiciones de cada palabra del documento en un diccionario que asocia cada string con una lista de enteros(esto también constituye una propiedad de ficha), para luego facilitar la implementación del operador ~ .

2.2 Método Query

El método Query comienza inicializando en 1 la propiedad Score de todos los documentos(fichas) que tenemos. La propiedad Score de la clase ficha, contiene un número que representa la importancia de ese documento para una determinada búsqueda. Por defecto esta propiedad es inicializada en 1 y sus incrementos se realizan con el producto usual, es decir: al calcular el coseno entre el vector que representa a la query y el que representa a un documento, este valor se le multiplica a la propiedad Score; al usar determinados operadores que deben incrementar el Score por un coeficiente determinado, simplemente se multiplica el Score por ese valor. La importancia principal de este procedimiento reside en que en determinadas circunstancias es necesario anular algunos Score e independientemente de lo que ocurra luego, estos no incrementan más; al multiplicar un Score por cero se logra lo deseado.

A continuación se procede a calcular el TFIDF de la consulta haciendo uso de otro método de la clase Metodos. En este punto son implementados los operadores de búsqueda también, los cuales afectaran directamente a la propiedad Score de los documentos. Nuestro programa le mostrará al usuario con los resultados un pequeño fragmento de su texto en el que aparezca la palabra con mayor TFIDF de la consulta que contenga el documento. Por esta razón, cuando se calcula el TFIDF de las palabras en la consulta, también se guardan un orden de estas de acuerdo a su importancia.
Ahora estamos listos para hallar el coseno entre el vector que representa la consulta y los que representan a los diferentes documentos. Para esto se llama al método HallaCoseno() de la clase Metodos y guardamos estos valores multiplicándoselos al Score de cada documento. Bajo este criterio ordenaremos los documentos de tal forma que será *mejor* aquel con mayor Score.

Para seleccionar el fragmento de texto que acompañará a los resultados, escogemos la palabra más importante de la consulta(con mayor TF-IDF) que este documento contenga y seleccionamos 300 caracteres después de la primera aparición de esta palabra.

Con los documentos más relevantes para una determinada consulta, Snippet para ellos y sus Score se realizan diferentes instancias de la clase SearchItem.

Para lograr las sugerencias mencionadas anteriormente cuando no hay resultados para la consulta se aplica el algoritmo de Levenshtein. Para mayor comprensión consulte [2].

Con las instancias de SearchItem y las sugerencias, instanciamos un objeto de la clase SearchResult y esta es la que devuelve el método Query, dando así los resultados buscados.

References

[1] https://es.m.wikipiedia.org/wiki/Tf-idf

[2] https://es.m.wikipiedia.org/wiki/Distancia_de_Levenshtein