Motor de Busqueda Moogle

Proyecto Primer Semestres

Josue Rolando Naranjo Sieiro, C112
Universidad de la Habana
November 8, 2022

Que es moogle?

Moogle! es un motor de búsqueda que opera con un conjunto de documentos y una consulta. Para su realización escogimos el modelo vectorial de recuperación de información por las ventajas que ofrece: el esquema de ponderación y la estrategia de coincidencia parcial. Con coincidencia parcial nos referimos a que la consulta no tiene que coincidir exactamente con un documento para ser considerado relevante sino con un subconjunto de ellos. Son precisamente estas diferencias las que permiten realizar una ordenación de los documentos recuperados.

Precálculo

Explicacion de la realizacion del precalculo:

Para el funcionamiento mas optimo de la aplicacion primero realizamos un precalculo de las operaciones que se puede ir tomando de antemano sin saber la consulta asi cuando el usuario decida realizar su consulta consumira menos tiempo. Para se ha creado el metodo Awake() que basta llamarlo antes de iniciar la aplicacion para que lea la carpeta content procesando los articulos y guardandolos en una estructura global y guardando tambien una estructura con todas las palabras del que aparecieron en el content y llamando al resto de metodos utiles para posteriormente procesar el modelo vectorial.

Precalculo

La estructura principal se basa en un Dictionary<string, Dictionary<string, double», donde el primer string seria el path al documento en el content como key y su valor seria un Dictionary con las palabras que tiene el documento como key acompanadas de la cantidad de veces que se repite esa palabra como value.

El mismo llamado del la estructura principal aprovechando el poder de calculo del indexado de las palabras a los diccionarios de los documento se guarda un Diccionario global con todas las palabras que estan presetnes en el content con un value de la cantidad de documentos en los aparecio la palabra

Precalculo

Posteeriormente se precalcula el Inverse Docuemnt Frequency (IDF), el Time Frequency (TF) y el Word Value Weight de la estructura principal para luego de realizada la consulta sea mas optima esta. Para el ultimo paso del precalculo se inicializa un Dataset de sinonimos para luego realizar una consulta mas optima buscando tambien los sinonimos de esta.

Query

Luego del precalculo realizado la aplicacion levanta y el usuario esta listo para realizar su consulta. Esta es procesada y se toman los modificadores en caso de haberlos (exolicado mas adelante) para guardarlos en un Diccionario global.

Score

Una vez procesada la query se puede realizar el primer calculo del modelo vectorial, se calcula el peso de la consulta con el mismo metodo del Word Value Weight y teniendo esto se procede a realizar el calculo del score utilizando la formula del cosine similarity

La utilizacion de diccionarios como estructuras fundamentales optimizo los procesamientos y calculos de las operaciones. (Una de las principales optimizacion fue el metodo ContainsKey() el cual tiene una complejidad Log N)

Sugestion

De no aparecer alguna de las palabras de la consulta en documentos del content se procedera a realizar la sugestion. El metodo Levenshtein distance es una medida de similaridad entre dos strings. La distancia es el numero de caracteres borrados, anadidos, o sustituidos para transformar la primera palabra en la segunda. Se procede a calcular el Levenshtein distance entre la palabra no encotrada y todas la palabras del content y se devuelve la palabra con menor distancia sustituyendola asi en la consulta, de tal manera que si el usuario escribio "reculsibidá" se le sugiere la palabra "recursividad".

Snipet

Una vez listos para devolver los resultados de la busqueda se retorna un pequeno texto (50 palabras) con una importancia significativa. Para calcular semejante importancia el metodo snipet recibe el texto como cadena de palabras y se toman las primeras 50 palabras para calcular el valor inicial de su importancia, y se va comparando la importancia con los textos de secuencia < i = 0 , j = 50, i++ , j++ > hasta el final del texto. la importancia esta dada por el Words Value Weight del texto dado en la query.

Operadores: Introduccion

-> Un símbolo ! delante de una palabra (e.j., "algoritmos de búsqueda !ordenación") indica que esa palabra no debe aparecer en ningún documento que sea devuelto.

-> Un símbolo entre dos o más términos indica que esos términos deben aparecer cerca, o sea, que mientras más cercanos estén en el documento mayor será la relevancia. Por ejemplo, para la búsqueda "algoritmos ordenación", mientras más cerca están las palabras "algoritmo" y "ordenación", más alto debe ser el score de ese documento.

Operadores: Introduccion

-> Un símbolo ĝdelante de una palabra (e.j., "algoritmos de ôrdenación") indica que esa palabra tiene que aparecer en cualquier documento que sea devuelto.

Cualquier cantidad de símbolos * delante de un término indican que ese término es más importante, por lo que su influencia en el score debe ser mayor que la tendría normalmente (este efecto será acumulativo por cada *, por ejemplo "algoritmos de **ordenación" indica que la palabra "ordenación" tiene dos veces más prioridad que "algoritmos").

Operadores: ! y ^>0

Para el operador !: Se hace un analisis de la palabra !string sobre todos los docuementos (indexados en diccionarios) pregutando ContainsKey(string) de recibir un true como respues se pasa a dejar el score de dicho documento en 0.

Para el operador ^ Se realiza la misma operacion que para el operador anterior solo que cambiando la pregunta por !ContainsKey(string), de contenar el escore del documento pasa a ser 0.

Operadores: *

Para el operador de *, se recolectan la cantidad de veces que aparece dicho operador delante de la palabra y luego se realiza la operacion

[document score] = [document score] * [0,75 * cantidad de veces que el modificador aparecio]

Operadores: ~>0

Para el operador de carcania, se guarda la cadena de palabras que deben estar juntas. Luego se llama al metodo NearWordsValueNotOrder (porque el metodo calcula cuan cerca estuvieron las palabras sin importar el orden). El metodo recorre todo el documento y guarda un int [] del tamano de la consulta donde sobreescribira los indices donde aparecieron las palabras de la consulta y calculara que tan cerca aparecieron anadiendole 1/[que tan cerca aparecieron] al valor del documento, para asi si la distancia fue 1 entonces se le anade 1 al valor del documento.

Sinonimos

Si aparecen palabras relacionadas (por ejemplo si la búsqueda es "computadora" y el documento tiene "ordenador"), también queremos devolver esos documentos pero con menor score

Para ello se crea un segundo score con una consulta nueva. Para la consulta nueva se itera por cada palabra de esta y se le anade hasta un maximo de 3 sinonimos y se realiza la busque (ej. "Estudiante y profesor de la universidad de la habana", se se realiza una segunda consulta que seria "estudiante alumno pupilo profesor maestro de la universidad de la habana")

Sinonimos

Una vez calculado el score de los documentos para esta nueva consulta se crea el score final que seria la suma del score calculado previamente mas la mitad del nuevo score de los sinonimos obteniendose resultados mas optimos en cuanto a respuestas.

De la nueva consulta realizada es posible quitar las palabras a las cuales se le buscaran los sinonimos pero en ese caso se deberia bajar el valor de modificacion de la nueva consulta porque quedarian beneficiados los documentos donde aparecen sinonimos de palabras en cuanto a documentos donde aparecen las palabras de la consulta.