Proyecto Moogle!

Javier Mustelier Garrido C-111

MATCOM

Universidad de la Habana
MATCOM - Facultad de Matemática y Computación

Modelo vectorial

Este proyecto se basa principalmente en la construcción de un motor de búsqueda de documentos de texto. Utilizando el lenguaje de programación C# y conceptos propios de las matemáticas como espacios vectoriales o vectores, se busca obtener, a partir de una consulta, una respuesta más o menos precisa que seria una lista de los documentos más relevantes respecto a la consulta.

La primera problemática que se hizo presente era quizás la más obvia de todas: ¿Cómo calcular que tan similar o relevante es un documento con respecto a una palabra, frase u otro documento?. Como respuesta al problema se usó el modelo vectorial, que no es más que una forma de ver lo que representa un documento dentro de un conjunto de estos; cada documento representa un vector dentro de un espacio vectorial, que es el grupo de documentos a los que se quiere hacer la consulta, espacio donde cada palabra diferente existente entre todos los documentos es una dimensión distinta, permitiendo representar entonces a cada documento en un espacio de tantas dimensiones como palabras existan en él. Luego así se tiene que:

Si tenemos los documentos:

doc1 = “el perro corre tras el gato”
doc2 = “el gato persigue al ratón”

Entonces el universo de palabras seria:

{“el”, “perro”, ”corre”, ”tras”, ”gato”, ”persigue”, ”al”, ”ratón”}

Siguiendo lo que plantea el modelo vectorial los documentos se representan en vectores de la siguiente manera:

	el	perro	corre	tras	gato	persigue	al	ratón
doc1	2	1	1	1	1	0	0	0
doc2	1	0	0	0	1	1	1	1

Así es posible realizar las operaciones propias de los espacios vectoriales sobre los documentos (especialmente las referentes a la geometría analítica), operaciones que incluyen, convenientemente, cálculos para hallar la similitud entre dos vectores.

TF-IDF(term frequency - inverse document frequency)

Una vez con la idea del modelo vectorial en mente, se presenta el siguiente problema: “No es suficiente representar los documentos como vectores donde el

valor de la dimensión sea la cantidad de veces que se repita un término dentro del texto; tengo que saber que tan importante es dentro de cada documento, así si esta palabra forma parte de la consulta, se tendrá en cuenta los documentos donde esta sea más importante, pero, ¿Cómo?".

Para resolver dicho problema se utilizo un algoritmo, el TF-IDF(term frequency - inverse document frequency), para darle peso (importancia) a cada palabra. El primer paso de este algoritmo consiste en calcular la frecuencia de cada palabra en el documento por la formula:

$$\mathrm{TF} = \frac{\boldsymbol{t}}{\boldsymbol{T}}$$

Donde ***t*** es la cantidad de veces que aparece el término y ***T*** es la cantidad total de términos. Luego el TF de cada término se multiplica por el IDF de cada palabra respectivamente; el calculo del IDF se realiza mediante la formula:

$$\mathrm{IDF} = \log\frac{\boldsymbol{D}}{\boldsymbol{d}}$$

Donde ***D*** es la cantidad total de documentos y ***d*** la cantidad de documentos en donde aparece el término correspondiente. Con el IDF mientras en más documentos aparezcan la palabra va perdiendo importancia hasta que si aparece en todas las archivos esta pierda importancia por completo, permitiendo eliminar así las palabras poco relevantes como las conjunciones o las preposiciones.

Luego en el vector del documento cada dimension toma el valor del TF-IDF correspondiente a la palabra, de la siguiente manera:

Hacer el calculo del TF en cada término:

	el	perro	corre	tras	gato	persigue	al	ratón
doc1	0.33	0.16	0.16	0.16	0.16	0	0	0
doc2	0.2	0	0	0	0.2	0.2	0.2	0.2

Luego cada TF se multiplica por el IDF de cada término

	el	perro	corre	tras	gato	persigue	al	ratón
doc1	0	0.049	0.049	0.049	0	0	0	0
doc2	0	0	0	0	0	0.06	0.06	0.06

De esta forma se obtiene una matriz Documento-Término donde cada casilla es la relevancia del término en el documento correspondiente.

Matriz Documento-Término

Ya que todos los documentos están contenidos en una carpeta y estos no cambian en cada consulta, el proceso de computar todos los archivos para obtener la matriz se puede realizar una sola vez.

Justo cuando se inicia la aplicación, se realiza este proceso: al método que va devolver los documentos más relevantes, se le va a pasar esta matriz documento-término; así cada consulta la utiliza sin tener que volver a realizar todo nuevamente solo para obtener misma matriz cada vez.

Para hallar la matriz se utiliza el siguiente algoritmo:

Declarar el diccionario que va a actuar de Matriz Documento término

Declarar el diccionario del universo palabras y la cantidad de documentos en que aparece cada una.

Obtener una lista del cuerpo de documentos

Por cada documento{

Leer y procesar para obtener las palabras

Declarar un nuevo diccionario de palabras del documento y la cantidad de repeticiones

Por cada palabra del documento{

Si la palabra no existe en el universo de palabras**{**

Agregarla al respectivo diccionario con 0 apariciones

}

Si la palabra no existe en el diccionario de palabras del documento**{**

Agregarla al mismo con 1 repetición

Aumentar en 1 la cantidad de documentos en que aparece

} si es lo contrario y la palabra existe **{**

Aumentar en 1 las repeticiones de la palabra en el diccionario

}

}

Por cada palabra en el diccionario de palabras del documento{

Calcular y guardar en el diccionario el ***TF*** de la palabra en el documento

}

Agregar a la Matriz Documento-término el documento y el diccionario de palabras que contiene con el ***TF*** de cada palabra.

}

Por cada palabra en el Universo de palabras**{**
Calcular y guardar en el diccionario el IDF de la palabra.
}

Por cada palabra **de cada** documento en la Matriz Documento Término{
Calcular el TF-IDF y actualizar el valor de la palabra en el respectivo documento

Con este algoritmo se asegura la obtención de la Matriz Documento-término que se va a utilizar en el método para calcular la similitud de cada documento con la consulta para entonces devolver los más relevantes.

La Consulta

La consulta se va a representar como un vector del espacio vectorial de los documentos, pero para esto hay que tener en cuanta, tres operadores que pueden estar presente y que le añade un significado a las palabras que afectan.

El operador ^ tiene que aparecer delante de un palabra y significa que la palabra debe existir en todos los documentos sugeridos.

El operador ! parecido al anterior pero esta vez la palabra no puede aparecer en ningún documento devuelto.

El operador * que añade importancia por cada signo * que aparece delante de la palabra

Y el operador ~ que añade relevancia a los documentos que tengan la pareja de términos entre los que aparece, más cerca.

Para obtener obtener y guardar esta información de la consulta se separó y procesó la consulta en términos, donde si alguno de estos términos contenía alguno de los operadores se guardaba en una lista dedicada a cada operador; lista que luego hay que tener en cuanta a la hora de devolver los archivos. En el proceso se verifica que la consulta sea valida, o sea, que tenga escrito algo con lo que se pueda realizar una búsqueda, de forma que si esta vacía o no contiene ninguna palabra relevante el motor de búsqueda no te devuelve nada que no sea una advertencia de que la consulta no es valida. En este proceso se obtiene el vector con las palabras relevantes de la consulta, donde a cada palabra se le asocia un peso calculado con el ***TF*** dentro de la propia consulta y el ***IDF*** universo de palabras calculado anteriormente. Teniendo en cuenta cuantos operadores (*) estén relacionados a la palabra y si la palabra esta repetida en la consulta, a su peso (***TF-IDF***) se le sera sumado el peso de la palabra con mayor relevancia de la consulta tantas veces tenga relacionado operadores (*) y en el caso de que la palabra se repita solo sera multiplicado su propio peso por la cantidad de veces que se repita

Con respecto a la obtención de las palabras implicadas con los vectores; en el caso de los operadores de necesidad (^), no-aparición(!) y de importancia (*) es sencillo, pues estos operadores solo son operadores cuando están al inicio de la

palabra o seguido de un (o varios) operador de importancia (*). Mientras que la obtención de las palabras relacionadas con el operador de cercanía (~) es un proceso un poco más complejo, ya que en este ultimo proceso se ha de obtener las posiciones de dos palabras de forma dinámica, ya que estas pueden variar según la aparición del operador y también se he de asegurar que las parejas de palabras que se obtengan no se encuentren ya en la lista: dado que “perro, gato” es lo mismo que “gato, perro”. Para completar este proceso elimine los operadores de cercanía al inicio y al final de la consulta (ya que no aportan nada) y obtuve las palabras posterior y anterior a la posición del operador, luego ya verifico que la pareja de palabras obtenidas no estén ya registradas, de ser así las guardo.

Una vez terminado este proceso ya he obtenido lo que me interesa de la consulta que es el vector y las lista de las palabras con los operadores.

Documentos Relevantes(Cálculo del Score)

Ya con los vectores de cada documento y el vector de consulta se puede pasar entonces a aplicar las operaciones matemáticas, para calcular la relevancia de cada documento. Existen muchas formas de calcular la similitud o distancia entre dos vectores de un espacio: La formula de la distancia Euclidiana y la similitud del coseno, son las formas más utilizadas en estos casos, pero elegí la similitud del coseno dado su facilidad de implementación (dado la forma en que he procesado los datos) y rapidez con la que se computa el algoritmo.

La formula para calcular la similitud del Coseno entre dos vectores, se basa en que tan semejantes son los ángulos de los vectores utilizando sus cosenos (dado que si un vector se superpone a otro significa que es linealmente dependiente o sea es el mismo vector escalado, por lo que a vistas practicas es el mismo vector). La formula es la siguiente:

$$D(x, y) = \frac{\sum x_i * y_i}{\sum x_j^2 * \sum y_k^2}$$

Mientras más cercano a 1 es el resultado más similar son los vectores, mientras que 0 significa que los vectores no comparten palabras. Pues este resultado es el que dicta que tan alto es la relevancia de un documento, pero este no es aún el **score** del documento; primero se ha de tener en cuenta el operador de cercanía.

Puesto que mientras menos palabras existan entre una pareja de palabras más cercana son, entonces tomando la cercanía como un parámetro en porciento (mientras más alta mayor es la cercanía), se ha de recorrer el documento buscando por una de las palabras de la pareja, cuando se halla (si no se halla, por supuesto en porcentaje de cercanía es 0) se comienza contar las palabras que existen hasta que se encuentre la otra palabra de la pareja (si se encuentra la misma palabra antes de llegar a su pareja entonces la cuenta se reinicia) y así se va llevando una cuenta de la distancia mínima. Una vez recorrido el documento y obtenida la distancia mínima de cada pareja, se hace un promedio entre ellas y eso se divide entre la cantidad de palabras del documento, dando así una proporción que restada a 1 y multiplicada

por 100 representa que el porciento de cercanía. Luego, dado que el máximo **score** de un documento es **1** y tengo dos datos a tener en cuenta (la cercanía de las palabras requeridas y la similitud del documento con la consulta), la solución fue mantener la similitud del documento y a lo que le falta al documento para tener un máximo **score: (1 - scoreSimilitud = x)**, se multiplica por el porcentaje de cercanía y se le suma entonces a el score de similitud **(scoreSimilitud + x*porcentCercanía)**. De esta forma si un documento cumple con la condición de cercanía entonces es más (o tan relevante como otro que también la cumpla) relevante que cualquier otro documento.

Por supuesto el **score** se calcula después de tener en cuenta los operadores de la consulta (“^”,”!”).

Snippet

Una vez obtenidos los documentos más relevantes, se han de mostrar al usuario, pero en aras de no agobiar al programa (ni al usuario) se muestra solo los 10 documentos más relevantes como mucho , pero junto con estos también se deben mostrar un pedazo del propio documento que represente la relevancia del documento. Para cumplir con este requisito me decanté por mostrar el pedazo de texto que contenía mayor densidad de palabras de la consulta. Con un tamaño de 100 palabras en cada snippet, estos deben mostrar el pedazo de texto de 100 palabras que dentro de ellas contengan la mayor cantidad de palabras relevantes de la consulta.

Primero se ha de obtener la lista de las posiciones de las palabras relevantes en el documento original (sin procesar) y a partir de cada posición de cada palabra ir contando las palabras que su posición están dentro de del rango de 80 posiciones a partir de la palabra que se esta buscando…. o sea:

Posiciones de la palabras relevantes: X
Densidad desde de la posición 3: Y

X	3	5	57	59	60	72	81	85	110
Y	0	1	2	3	4	5	6	6	6

De esta forma el **snippet** es el pedazo de texto tomado desde 10 posiciones antes y 10 posiciones después de las 80 palabras con mayor densidad.

Sugerencia

En caso de que la consulta no arroje ningún o relativamente pocos documento se debe mostrar una sugerencia que muestre una consulta lo más semejante posible y con mejores resultados. En caso de que el usuario se haya equivocado a la hora de escribir una palabra, la sugerencia debería ser capaz de

corregir este hecho. El algoritmo que hace posible este “feature” es el de la Distancia de Levenshtein, que se usa principalmente para calcular cantidad mínima de cambios en una palabra quese necesitan para convertirla en otra. Así la palabra más cercana a ella es el ella misma con 0 cambios.

En este caso se ha de realizar el proceso con todas las palabras del universo de palabras que tengan un IDF mayor a uno determinado, dado que queremos una consulta fructuosa.

Dadas las dos palabras a buscar la distancia, lo primero es crear un vector que representa la **primera palabra**, un vector de tamaño **n+1**, donde **n** es la longitud de la palabra y se inicializa de 0 hasta **n-1,** este sera el vector Previo o vector de Referencia ya que es el vector que se ha de tomar de referencia para hacer las operaciones. Una vez inicializado el **vector Referencia** (**vR**), se toma un vector de igual longitud y en la primera casilla se va a colocar el valor correspondiente al **vR + 1**, este va a ser el vector en el cual se van a realizar las operaciones, por tanto **vector de Cambio (vC)**. Toca llenar el **vC** por lo que por cada casilla sin llenar del **vC** (empezando por la primera) se va a tomar el mínimo resultado de la comparación entre **sumarle 1** a la casilla actual del **vC, sumarle 1** a la casilla siguiente correspondiente al **vR** y sumarle el **costo** (que mientras los caracteres de las palabras correspondientes en la **Matriz (**en la figura de más adelante lo aclaro**)** no sean iguales siempre es **1**, en caso contrario es **0**) a la casilla correspondiente al vector **vR**. Este proceso se repite tantas veces como caracteres existan en la **palabra “objetivo”**, siempre intercambiando el **vR** con el **vC** y el **vC** por un nuevo vector en cada ciclo.

Primera palabra: “del” **Palabra objetivo:** “al”

Matriz de Levenshtein:

		d	e	l
	0	1	2	3
a	1	1	2	3
l	2	2	2	2

Una vez realizado este proceso el valor que te queda en la ultima casilla del ultimo vector es la distancia entre las dos palabras, solo quedaría sustituir la primera palabra por la que menor distancia a ella tenga de entre las palabras del universo. Ya solo queda sustituir en la consulta las palabras que se cambiaron y agregarle los operadores que anteriormente tenían y… ya, tenes una consulta nueva y funcional. Junto con un motor de búsqueda a mi parecer bastante eficiente.