Informe Moogle

Alumno: José Ernesto Morales Lazo

C-122-

El namespace Moogle Engine cuenta con 6 clases. La clase Documentos, la clase Puntuacion , la clase Matriz y la clase Query – estas cuatro implementadas por mi – y las clases SearchItem y SearchResult.

Clase Documentos :

Esta clase es la que se encarga de procesar cada Documento de la colección, un objeto tipo Documento es en esencia una lista de listas donde cada lista corresponde a un documento de la colección . Cada lista es el resultado del procesamiento de cada texto, dígase eliminar caracteres especiales, espacios y demás cuestiones que no favorecerían al desarrollo del programa, es así que dicha lista contiene las palabras de cada texto .

Métodos :

En el método constructor se inicializa un objeto tipo texto, lo hace primero inicializando una lista de listas de string a la cual

se le aplica el método RecorrerDocumento, el mismo se explica detalladamente a continuación.

Este método devuelve una lista de listas de string y es el encargado de procesar los textos y dividirlos en palabras , de manera que se eliminen los caracteres especiales, espacios, etc.

Se utiliza el método GetFiles de la clase Directory para extraer las rutas de todos los archivos de la carpeta Content y archivarlas en un array de string. En un ciclo for se recorre este array , en este ciclo primero se inicializa una lista que a la postre se va a añadir a la lista de listas que se va a devolver . En segundo lugar se emplea el método ReadAllText de la clase File para almacenar en una cadena cada texto y en tercero se inicializa un array en el que usando la clase Regex y una expresión regular de la misma que define los delimitadores que se toman como referencia para hacer Split en la cadena en la que anteriormente se almaceno el texto. Ya por último se recorre este array , en primer lugar se aplica el método ToLower a cada elemento y y el método IsNullOrEmpty , con el objetivo de que no pasen cadenas vacias (que serian los espacios que contenia el texto) y además que las palabras pasen en minúsculas a la lista a las que se añaden en este ciclo . Dicha lista se añade a la lista de listas y ya tendríamos los textos procesados palabra por palabra y listos para utilizarlos .

El método Repeticiones de la clase Documentos es un método que devuelve un diccionario con clave string y clave double . En un método de instancia que se le aplica a un objeto tipo

Documentos con la finalidad de almacenar cuantas veces se repite una palabra en un texto determinado. Para esto se recorre cada lista de la lista de listas de string (Objeto Tipo Documentos) y con empleando condicionales se hace lo siguiente: Si la palabra ya esta en el diccionario se aumenta en uno su valor en el mismo , si no se añade al mismo con valor uno . De esta manera cada vez que aparezca la palabra en el texto su valor en el diccionario va a aumentar. Cuando este ciclo se acaba se añade el diccionario a la lista de diccionarios. Al finalizar tendremos una lista con un diccionario por cada texto, en el que aparece la palabra del texto y cuantas veces se repite en él. Esto no tiene ningún interés a la hora de procesar los textos, pero si lo tendrá a la hora de hacer los cálculos del Term Frequency en la clase Matriz. Lo mismo pasa con el método Vocabulario que se describe a continuación.

Este método recorre un objeto tipo texto y almacena en un Hash Set de string todas las palabras de cada lista sin que se repiten. Se emplea el HashSet por su utilidad para almacenar elementos que no se desea que se repitan. Esto será de utilidad en los cálculos posteriores de la clase Matriz.

El método Index de la clase Documentos devuelve el Count de una lista de un objeto tipo Documentos -Lista de listas - .Se le pasa por parámetro la posición de la lista de la cual se desea conocer la posición .

Cabe mencionar también el método CopiarDocumentos que realiza una copia de un objeto tipo Documentos, esto para evitar problemas que se pueden dar a la hora de recorrer estos objetos en ciclo.

Clase Matriz

La clase Matriz cuenta con un método constructor en el cual se inicializa un objeto tipo Documentos que se le pasará como parámetro al método CalcularTDIDF el cual da valor a la variable tfidf – una lista de diccionarios con clave string y valor double -, la cual será nuestro objeto tipo Matriz propiamente dicho. Este objeto contendrá los valores del TFIDF de cada palabra en cada texto.

Método TFDoc: Lo primero que se hace en este método es llamar al método Repeticiones de la clase Documentos explicado anteriormente, así tendremos la frecuencia de

cada palabra de cada texto. Luego recorremos la lista de diccionarios procedente del método Repeticiones. En otro bucle recorremos cada diccionario y calculamos el tf de cada clave: el valor de la misma (su frecuencia en el texto dividido) dividido por el texto. Index – devuelve el Count de la lista en cuestión, es decir, la cantidad de palabras del texto y se le pasa por parámetro un

contador el cual representa la posición de la lista de cuyas palabras se le calcula el TF -. Cada resultado se le añade a un diccionario que se inicializa en el primer bucle. Dicho diccionario al final el proceso se añade a la lista de diccionarios. Es así como se obtiene la lista de diccionarios, con un diccionario por texto y con valor del TF de sus palabras.

El método Copiar Hash de esta clase recibe un Hashset como parámetro, el cual será el que contenga todas las palabras del vocabulario. Devuelve un diccionario con clave string (todas las palabras del vocabulario) y valor 0, es cero porque es un diccionario cuya utilidad se basa en aumentar su frecuencia en el método siguiente.

El método NumeroenDocumentos es el método empleado para saber en cuantos de los textos de la colección se repite cada palabra del vocabulario. Para esto se utiliza un HashSet con todas las palabras del vocabulario (Método Vocabulario de la clase Documentos) y un diccionario con todas estas palabras y valor 0 (Método Copiar Hash). Usando el método CopiarDocumentos se realiza una copia del objeto tipo Documentos (Lista de listas con todas las palabra de los textos) , esto para poder recorrer dicho objeto .Luego se itera sobre esta copia y en cada ciclo se crea un HashSet (llamémoslo palabrasTexto) que será el que contendrá todas las palabras de cada texto (las palabras de cada lista). Luego en un bucle dentro de este se itera sobre el HashSet procedente del método Vocabulario (todas las palabras de la colección) y empleando el método Contains al HashSet palabrasTexto y si devuelve true aumentamos en 1 el valor de la palabra en el diccionario frecuencias (procedente de la llamada al método CopiarHash) . De esta manera se obtiene en cuantos documentos aparece cada palabra de la colección de textos .

En el método IDF lo que se hace es después de llamar al anterior método usar estos datos para calcular el IDF con formula logaritmo natural del cociente de la cantidad total de documentos en la colección entre la cantidad de documentos en los que aparece la palabra en cuestión. En mi caso yo lo calcule como resta de logaritmos de igual base ya que la cantidad de documentos va a ser constante ya le calculo el valor afuera del ciclo, a dicho valor se le resta el logaritmo natural de la cantidad de documentos en los que

aparece la palabra. Luego este valor se almacena en un diccionario en el cual estarán todos los valores del IDF del vocabulario.

Y ya por último el método TDIDFDoc que nos devuelve el objeto tipo Matriz propiamente dicho. En un ciclo se recorre la lista de diccionarios que almacena los valores del TF de cada palabra de cada documento, en otro ciclo más interno se recorre cada diccionario y se multiplica el valor de cada clave (el TF de cada palabra) por el valor del IDF de la palabra en el diccionario que almacena los mismo. Estos valores se almacenan en un diccionario que posteriormente se añaden a la lista de diccionarios. De esta manera se obtiene una lista de diccionarios, un diccionario para cada texto en el cual se almacena el TF-IDF de cada palabra.

Clase Query

La utilidad de la implementación de esta clase el proyecto se basa en el procesamiento de la query. Aunque su manejo puede ser muy similar al de los de los documentos considere oportuno procesarla en una clase aparte.

El objeto tipo Query consta de un constructor el cual recibe como parámetro la query insertada por el usuario . Cuando se llama al método constructor se le da valor a un Diccionario con el método CalcularTFIDF (recibe como parámetro la

query del usuario que se la pasa como parámetro al constructor.

Esta clase consta de un método CopiarQuery que realiza una copia de un objeto tipo Query para evitar errores a la hora de recorrerlos mediante un ciclo en otros métodos.

Para procesar la query se utiliza el método CogerPalabras . Este recibe como parámetro la query del usuario y usando una expresión regular de la clase Regex -que determina los separadores que se deben tomar como referencias para hacer Split (dígase comas , comillas , paréntesis y demás) – se almacena en un array de string cada palabra por separado y ya sin los caracteres especiales . Luego se recorre este array y aplicando el método IsNullOrEmpty (para evitar que los espacios pasen a la lista con las palabras de la query que va a devolver este método) se agrega el elemento a la lista .Se agrega en minúsculas después de aplicar el método ToLower a la palabra . Es así que a partir de este método se obtiene una lista con las palabras de la query .

Luego está el método Repeticiones, este se usa para ver cuántas veces se repite cada palabra de la query en dicha query. Se recibe como parámetro la lista de las palabras de la query y se itera sobre ella, con el método ContainsKey se

comprueba si la palabra ya esta en el diccionario que se va a devolver como resultado del método, si lo esta se aumenta en uno su valor, si no, se añade la palabra al diccionario con valor 1. De esta manera ya se obtiene el diccionario con la frecuencia de cada palabra de la query .

En el método que calcula el TF de las palabras en la query se llama al método CogerPalabras y al método Repeticiones . El primero para obtener la lista de las palabras de la query y saber el total de palabras de la misma a través del método Count() aplicado sobre la lista y el segundo para tener las frecuencias de las mismas en la query . Acto seguido se itera sobre el diccionario que contiene las frecuencias y se divide el valor de la palabra en el diccionario entre la cantidad total de palabras del texto , el resultado de esta operación se añade al diccionario tfs que es el que va a ser devuelto eventualmente por el método TF .

Esta clase también contiene métodos como CogerUna y CopiarHash . El primero devuelve un HashSet que es el resultado de iterar sobre la lista que contiene todas las palabras de la query , obteniendo así el vocabulario de la misma . El segundo devuelve un diccionario cuya clave es cada palabra del vocabulario y su valor 0 . La utilidad de este ultimo radica en que se va a usar para aumentar su frecuencias y guardar los datos del siguiente método.

El método NumeroenDocumentos es el método que calcula en cuantos documentos de la colección aparece cada palabra de la query . Para esto se llama en primer lugar a tres métodos de la clase . CogerPalabras , CogerUna y CopiarHash. Del primero se obtiene la lista con todas las palabras de la query , el segundo recibe como parámetro esta lista y construye el vocabulario de la query y lo guarda en un HashSet y el tercero recibe como parámetro a este HashSet y construye el diccionario(la clave son las palabras y el valor es 0) que será devuelto al final del método . Se hace uso del método GetFiles de la clase Directory para almacenar en un array todas las rutas de los documentos de la carpeta Content . A continuación, se itera sobre el Vocabulario de la query (HashSet del método CogerUna) y en un segundo ciclo se itera sobre el array con las rutas de los documentos y haciendo uso del método ReadAllText de la clase File se almacena en un string cada texto . Luego se aplica el método Contains a este string y se devuelve true se aumenta en una el valor de la palabra en el diccionario resultante del método CopiarHash . Asi ya se obtiene en un diccionario estos valores que se usaran para el calculo del IDF .

En el método para calcular el IDF se utiliza el diccionario que obtenemos del método anterior (llamando al método) .Luego al igual que antes hacemos uso del método GetFiles de la clase Directory para guardar en un array todas las rutas

de los archivos (la utilidad de esto acá es aplicarle el método Length al array obtener la cantidad de documentos en la colección de documentos) . Ya luego se itera sobre el diccionario frecuencias que es el que almacena los datos del método NumeroenDocumentos (cantidad de documentos en que aparece cada palabra de la query) . Si el valor de alguna de las palabras es cero se añade con valor 0 al diccionario que devolverá este método . Si el valor de algunas es igual a la cantidad total de documentos de la colección se agrega al diccionario con valor 0.00001 , muy cercano a cero . Esto lo hago para evitar que el método SimilitudentreCosenos devuelva NaN en el caso particular de que la query aparezca en todos los documentos . Y ya por último si nada de esto ocurre entonces se calcula el IDF de la palabra . Previamente fuera del bucle se inicializo una variable con el valor del logaritmo natural de la cantidad de documentos de la colección para evitar hacerlo dentro del bucle ya que es un valor constante . Aunque la formula del IDF es Logaritmo Natural de la cantidad total de documentos de la colección entre la cantidad de documentos en los que aparece la palabra acá hago uso de la propiedad de los logaritmos que afirma que el logaritmo de una división es igual a la resta de los logaritmos de ambos componentes de la división .

Ya por último el método CalcularTDIDF realiza la operación de multiplicar cada valor de los diccionarios que almacenan los valores del TF y del IDF . Se inicializan dos diccionarios llamando a los respectivos métodos ya explicados previamente y en un ciclo se multiplican ambos valores .

Clase Puntuación

Esta clase es donde se realizan todos los procesos para llegar al score .

En el método Nombres de esta clase se hace uso del método GetFiles de la clase Directory para almacenar en un array las rutas de los archivos de la carpeta Content . Luego se itera sobre este array y haciendo uso del método GetFileName(como parámetro se la pasa cada ruta del archivo) se obtiene el nombre de cada archivo , cada nombre se guarda en una lista . Así se han obtenido todos los nombres de los documentos de la colección .

El método NewCollection es el método encargado de crear un vocabulario en común entre la query y el documento , esto en pos de que los dos tengan la misma longitud y poder aplicar sobre ellos la formula de la similitud entre cosenos . Este método recibe dos parámetros que vendrían siendo los correspondientes a los valores del TF-IDF de la query y los documentos y devuelve un diccionario con clave string y una

tupla de valores , el primer valor de a tupla corresponde a un diccionario y el segundo a otro . Primero se recorre uno de los diccionarios y se añade cada palabra al diccionario que se devolverá eventualmente , además se añade una tupla , el primer valor es el correspondiente a la clave del diccionario que se recorre y el segundo valor es 0.Luego de terminado este proceso en otro ciclo se recorre el segundo diccionario , si el diccionario que va a ser devuelto ya contiene alguna de las claves entonces simplemente se añade el valor a la tupla . Si no contiene a la clave entonces el primer valor de la tupla se agrega en 0 y en el segundo se agrega el valor correspondiente a la clave del diccionario que se esta recorriendo . De esta manera al finalizar se obtiene un diccionario cuyas claves son todas las palabras del vocabulario entre la query y el documento y cuya tupla de valores representan los valores de las claves en los dos diccionarios que se recibieron como parámetro .

El método ProductoEscalar recibe como parámetro este diccionario anterior y recorriéndolo realiza el producto de ambos valores de la tupla , cada resultado del producto se va agregando dentro del ciclo a una variable que va conteniendo la suma de estos productos , así al final se obtendrá el valor del Producto escalar entre los vectores de la query y el documento .

El método ProductoMagnitudVectores recibe el mismo diccionario que el anterior método y calcula la magnitud de cada vector . Se recorre el diccionario en un ciclo y en dos variables suma1 y suma2 se van almacenando las sumas de los resultados de la operación de elevar al cuadrado cada componente del vector . Aquí cabe aclarar que cada vector acá es un valor de la Tupla . Por ejemplo, se tiene una clave y una tupla , el primer valor corresponde a un vector y el segundo a otro . Luego cuando termina el ciclo se le halla el valor de la raíz cuadrada de cada suma y se multiplican teniendo así el producto de la magnitud de ambos vectores necesario para calcular la similitud entre cosenos .

En el método SimilitudEntreCosenos (como parámetro se reciben dos diccionarios , uno correspondiente a los valores del TF-IDF de un documento y el otro correspondiente a la query) se van a emplear los tres métodos explicados anteriormente :

1.NewCollection – para construir el vocabulario común entre los dos vectores , igualar su longitud y además almacenarlos en un mismo diccionario .

2.ProductoEscalar- Se le pasa como parámetro el diccionario anterior para obtener el producto escalar de ambos vectores

3.ProductoMagnitudVectores:Tambien se le pasa como parámetro el diccionario de NewCollection y realiza el producto de la magnitud de ambos vectores.

Los valores del producto escalar y el producto entre la magnitud de los vectores se almacenan en una variable respectivamente y luego la variable similitud almacena el cociente entre las dos variables anteriores , resultando así la Similitud entre cosenos .

El método Scores devuelve una lista de doubles con los valores de la similitud entre cosenos entre la query y todos los documentos de la colección . Primero se realiza una copia del objeto tipo matriz, así como de la query para evitar errores a la hora de recorrerlos en un ciclo . En un bucle foreach se recorre cada diccionario en la copia de la matriz y se calcula la similitud entre cosenos entre el diccionario (el vector de cada documento) y el vector de la query , se va añadiendo cada valor a la lista con los scores.

Con el método Union unimos las listas resultantes de los métodos Nombres y Scores , de manera que se obtenga un diccionario en el que a cada archivo (su nombre) se le asocie un score.

El método OrdenarScores se le aplica a un objeto tipo Puntuación con el objetivo de ordenar los scores de mayor a menor . Primero se crea una lista y se recorre el objeto tipo Puntuación de manera de que a la lista se le vaya añadiendo los valores de los scores que se encuentran en el diccionario . Cuando termina el ciclo se le aplica el método Sort a la lista quedando así ordenados de menor a mayor . Luego en un ciclo se va a recorrer la lista empezando por el ultimo valor (el mayor) y terminando por el primero (el menor) . Dentro de este mismo ciclo se va a recorrer el objeto tipo Puntuación . Si un valor de la lista coincide con un valor del diccionario (objeto tipo Puntuación) entonces si el diccionario que se va a devolver (llamémoslo orden) , si orden ya contiene esa clave no se va a hacer nada , si no la contiene entonces se añade la clave con su valor de score correspondiente . De esta manera se obtiene un diccionario donde a cada titulo se le asocio su valor del score , y además en orden .

El método del Snippet devuelve un diccionario en el que a dos claves (nombre y snippet) se le asocia un valor (score).

Se le pasa un diccionario como parámetro que será el que contenga los scores ordenados . Luego se itera sobre este diccionario . Aquí se inicializa una variable tipo string (llamémosla texto) , se utiliza el método string.Format para añadirle a la ruta de la carpeta Content el nombre de cada archivo de manera tal que a la variable que se inicializa luego

usando el método ReadAllText de la clase File (se le pase como parámetro la ruta del documento ej. : rutacontent\nombrearchivo) .Luego hacemos texto.IndexOf(se le pasa como parámetro un punto indicando que devuelva la posición del primer punto y además se vuelve a llamar a texto.Index (se le pasa como parámetro un punto) – esta vez indicando que devuelva la posición del primer punto después del primer punto encontrado), de esta manera se obtiene la posición del segundo punto en el texto , delimitando así las dos primeras oraciones. Luego se inicializa otra variable string y se llama al método substring (texto.Substring (0 , posición del segundo punto +1) , indicando que archive en la variable las dos primeras oraciones del texto . Ya por ultimo se añade esta subcadena al diccionario como la segunda clave . Quedando estructurado de la siguiente manera : Nombre , snippet : score.

Al método constructor de esta clase se le pasa como parámetro un objeto tipo matriz y un objeto tipo Query . En el interior se inicializan en primer lugar dos variables : una lista de string resultado de llamar al método Nombres (con todos los nombre de los archivos) y otra lista de dobles (con todos los scores resultado de llamar al método Scores).Luego se inicializa un Diccionario de string y double con el método

Union de manera que cada nombre de archivo quede asociado a su score .

Método Query

Justo afuera del método Query en la clase Moogle se inicializa la variable estática matriz que es un objeto tipo Matriz (Lista de Diccionarios con los valores de los TF-IDF de cada palabra de cada documento de la colección)

Cuando se llama al Método Query se inicializa un objeto tipo Query , en donde se procesa la query como se ha explicado previamente . Luego se inicializa un objeto tipo Puntuación (llamémosle scores), este sería el Diccionario con los valores de los scores asignados a cada texto . Luego se inicializa un diccionario con el método scores.OrdenarScores(); aquí se obtienen los valores de los scores ordenados de mayor a menor . Después de esto se llama el método estático de la clase Puntuación al que se le pasa como parámetro el diccionario anterior , de acá se obtiene un diccionario con el snippet integrado , dos claves (nombre,snippet) y el valor del score .

Ya tenemos lo que seria el resultado de la búsqueda , nos falta integrarlo , por decirlo de alguna manera al array items de objetos tipo SearchItem . Para esto inicializamos uno (con el Length igual al Count del diccionario con los snippet integrados) . Luego en un ciclo recorremos el diccionario y con un contador vamos agregando los tres elementos al array (los parámetros que se le pasa al constructor de SearchItem).

Finalmente se devuelve un objeto tipo SearchResult cuyo constructor recibe dos parámetros , el array items y la query .