Esta es mi versión del Moogle!, un buscador de texto al que se le agregan documentos .txt antes de ejecutarlo y que tiene las particularidades siguientes:

El proyecto retorna los títulos con la especificación de que son .txt de los documentos dentro de los que aparece el query que se busque.

El score no aparece al devolver los textos, sin embargo es el encargado de el orden de relevancia en el que se presentan al estar los documentos mostrados organizados por descendentes de acuerdo a su score.

Está hecho para que analice textos en español, pues en la búsqueda del snippet evita que las palabras de poca relevancia como artículos o preposiciones sean buscadas, si esto no es de importancia para el usuario puede emplear textos en ingles, solo que un solo idioma a la vez.

El snippet que se retorna es la primera linea en la que aparece la palabra (o frase, pues esta si aparece completa es prioritaria), sin embargo dentro de la linea devuelta puede ocurrir que si buscas una palabra como “casa” que es la raíz de muchas palabras, la coincidencia del snippet (no para determinar el score), pueda ser una palabra como “casada”. Además cuando la query cuenta con palabras comunes (como artículos, conjunciones o preposiciones) y estas no fueron completamente eliminadas por el idf al no aparecer en todos los documentos, te devuelve el título de aquellos documentos en los que aparece pero el snippet es la primera linea del documento; esto cambia si la búsqueda es unicamente esta palabra (como por ejemplo solo buscar “el” y que no se cancele por su idf) en este caso si devuelve la primera línea en la que aparezca dicho artículo.

A continuación se hace un desglose de las clases creadas y los métodos implementados dentro de las misma con una pequeña explicación de su función y proceso:

Clase Obtein

Constructores:

- string[] documentos
- List<Tuple<string, int[]>> vocabulary
- Dictionary<string, string[]> words
- int[] wordsCount

Métodos:

- Vocabulary --»List<Tuple<string, int[]>> vocabulary

Devuelve una lista donde cada tupla lleva una string que representa una palabra única, asociada a un array de int[] con tamaño igual a la cantidad de documentos rellenado con la frecuencia que le corresponde a la palabra en cada documento.

- NormalizeText --» string[] words

A partir de un string (ya sea una sola palabra o un texto completo) pone todas las palabras en minúscula, elimina las tildes y los signos usando el comando Regex(“[^a-zA-Z0-9]”) ; después crea un array de string en el que las palabras están separadas y sin espacios.

- CosineSimilarity --» double a

Es una fórmula exacta sacada de internet.

Recibe dos array double[] que funcionan como vectores para hallar la similitud de cosenos entre ellos y que devuelva un número que sería el score entre un documento y la query introducida.

- Words --» Dictionary<string, string[]> words

Devuelve un relación entre la dirección de un .txt y el texto que contiene ya normalizado y dividido en palabras. Tiene tantas key como documentos se analizan y las palabras de cada string[] no coinciden.

- WordsCount --» int[] wordsCount

Devuelve un array con las cantidad de palabras que conforman los textos de cada documento.

- ComunWords --» string[] palabrasComunes

Contiene los artículos, preposiciones y conjunciones y adverbios para eliminarlas del array introducido, a la vez lo normaliza y divide en palabras sueltas.

Solo se utiliza en el snippet para que en la búsqueda de la sección del texto en la que aparece el query no se traten de buscar palabras sin relevancia.

- FindLineNumber --» int a

Lee el texto de la dirección que se le introduce, lo lleva a minúscula y devuelve el numero de la linea en la que se encuentra el string que recibe, en caso de no hallarse devuelve -1.

- GetLine --» string line

Lee el texto de la dirección introducida y devuelve todo el texto que aparece correspondiente a la linea del numero dado.

Clase TextAnalize

Constructores:

- double[] idf
- List<double> tf_idf
- Obtein a

Métodos:

- TF --» List<double> tf

Devuelve una lista con el tamaño del vocabulario, formada por arrays de tamaño igual a la cantidad de documentos a analizar.

Como la lista del vocabulario ya contiene al frecuencia en la que aparecen las palabras únicas, lo que hace este método es pasar por cada valor de las int[] de las tuplas del vocabulario para aplicar la formula de tf (frecuencia de la palabra en el documento entre cantidad de palabras del documento), este último valor obtenido del int[] wordsCount.

- IDF --» double[] idf

Devuelve un array de tamaño igual a la cantidad de palabras únicas.

Se utiliza la lista del vocabulario para definir, pasando por cada tupla, en cuantos de los valores la frecuencia es distinta de cero y por tanto en cuantos documentos aparece la palabra, el total de elementos del int[] indica cuantos documentos están siendo analizados.

Con estos valores se utiliza la formula matemática estándar para calcular el idf, y una vez obtenido el resultado este se coloca en el lugar correspondiente del array double[] según la posición dentro de la tupla con la que se trabajó en la lista.

- TF_IDF --» List<double> tf_idf

Devuelve una lista del mismo tamaño del tf.

El calculo que se realiza es que multiplica cada valor de la lista de tf por el valor del indice correspondiente del double[] idf.

- QueryVec --» double[] queryVector

Devuelve un array del tamaño del vocabulario en el que las únicas casillas con valor distinto de cero son las correspondientes a palabras coincidentes con el query, si ninguna coincide el valor del array será todo cero y al calcular el score no se devolverá ningún documento.

Se utiliza el vocabulario para hallar las coincidencias lo que indica en que parte del array se deberá colocar el valor de tf*idf. Se revisa cuantas veces esta la palabra en el query y con esa información se calcula el tf, luego se multiplica con el valor del array de double[] idf previamente establecido, y el resultado se coloca en el lugar ya definido dentro del array double[] queryVector.

- Score --» Dictionary<string, double> score

Devuelve un diccionario con el tamaño de la cantidad de documentos analizados, en el que la key es la ruta del documento y el value el score que le corresponde dependiendo de la query introducida.

Para calcular el score se crea un nuevo array que se vuelve a inicializar en cero por cada documento con el tamaño del vocabulario con los valores de tf_idf correspondientes, este se utiliza para con le double[] queryVector realizar la similitud de cosenos que el el numero que se busca.

- Snippet --» string line

Devuelve la primera linea del texto en la que se encuentra una coincidencia con el query, en caso de obtener ninguna devuelve la primera linea del texto.

Para buscar el query primero lo busca como frase completa y si no arroja ningún resultado, busca palabra por palabra, sin contar los artículos, preposiciones y conjunciones que ya fueron eliminados al convertir el query de string a string[].

Utiliza los métodos FindLineNumber para saber en que linea se encuentra y GetLine para que devuelva el texto solicitado en un string.

- Query --» SerchItem[] (title, snippet, score)

Utiliza el método SerchItem[] preestablecido, ahora con tamaño igual a la cantidad de documentos analizados. El title es el nombre del documento, ya separado de las indicaciones de la dirección para encontrarlo, mantiene el tipo de documento que es: en este caso .txt . El snippet es obtenido a través del método Snippet. Se asocian estos valores con una iteración del Dictionary<string, double> score para asociar en cada caso el titulo, la sección en la que aparece el query en el documento y el score que le corresponde.

Clase Moogle

Constructores:

- TextAnalize textAnalize

Modificación del método:

- SearchResult Query

Devuelve el titulo y el snippet asociado.

Dentro de este método se llama al método Query de la clase TextAnalize y se organiza por descendentes dependiendo del score, ademas se desechan aquellos cuyo score es cero pues carecen de relevancia en la búsqueda.

Clase Matriz

Constructor:

- int[,] matrix

Métodos:

- MatrixAdd --» int[,] a

Para dos matrices del mismo tamaño suma los elementos de posiciones iguales [i,j] y los coloca en esa misma posición del nuevo array doble.

A , B pertenece a $M_{mxn}(K)$, $C = A + B$ «--» $c_{ij} = a_{ij} + b_{ij}$ para todo i , j.

- MatrixMult --» int[,] b

Para dos matrices MxN y NxP, se crea un nueva matriz de dimención MxP. El cálculo se realiza de la siguiente manera:

A pertenece a $M_{mxp}(K)$, B pertenece a $M_{pxn}(K)$, $C = AB$ «--» $c_{ij} = \Sigma$ desde k=0 hasta p $a_{ik}b_{kj}$ para todo i , j

- MatrixVec --» int[] c

Se multiplica algebraicamente, moviendo los índices de posición según corresponde, una matriz con un vector, siempre que la matriz tenga tantas columnas como valores posee el vector. Devuelve un array o vector de tamaño igual a la cantidad de filas que tiene la matriz.

El procedimiento de cálculo es una sumatoria desde j=0 hasta el tamaño del vector tal que $c_{ij} = a_{ij}b_{jk}$

- MatrixEsc --» int[,] d

Se obtiene una nueva matriz del mismo tamaño de la matriz introducida, lo que cambia en que cada valor de la matriz original es multiplicado por un número x. La operación algebraica seria la siguiente.

Sean A pertenece a $M_{mxn}(K)$, α pertenece a K entonces $aA = (\alpha a_{ij})$ para todo i , j

Clase Vec

Constructores:

- int[] vector

Métodos:

- VecMult --» int e

Se multiplica algebraicamente dos vectores (dos arrays) y se obtiene un numero. Para calcular se realiza una sumatoria desde k=1 hasta el tamaño del vector tal que $e = a_{ij}b_{ij}$

- VecEsc --» int[] f

Se multiplica un vector con un escalar y se obtiene un nuevo vector en el cual los valores del vector original han sido multiplicados por un numero x.

EJEMPLOS:

Aquí se presentan algunas búsquedas de prueba realizada para comprobar la funcionalidad y eficiencia del proyecto:

Query: gato gordo

- Edgar Allan Poe - El diablo en el campanario.txt

... de cerdo. A su lado se encuentra un gato gordo y manchado, que exhibe en la ...

- Edgar Allan Poe - El entierro prematuro.txt

... -¿Por qué aúlla de esa manera, como un gato montés?- dijo un cuarto. …

Query: Julio Verne (las palabras pueden ser puestas con mayúscula o minúscula y se obtiene el mismo resultado)

- Julio Verne - Aventura de tres Rusos y tres Ingleses.txt

... Julio Verne ...

- 10. EL REMEDIO ANEXIONISTA.txt

... EL REMEDIO ANEXIONISTA. PATRIA, 2 de julio de 1892 ...

- Julio Verne - Ante la bandera.txt

... Del 5 al 25 de Julio.- Han transcurrido dos semanas …

--» En el último caso la palabra que coincide es únicamente “julio”, pues “Verne” no aparece referenciado en el documento. Como se puede observar a la hora de devolver los documentos se le da mayor importancia (por lo que sale primero) a aquel en el que aparece la frase completa para después buscar la coincidencia de al menos una parte del query

Query: caminos

- 07. CON TODOS Y PARA EL BIEN DE TODOS.txt

... Cuba, desolada, vuelve a nosotros los ojos! ¡Que los niños ensayan en los troncos de los caminos la ...

- 01. LECTURA EN STECK HALL.txt

... ni fuerza para contenerla; ni en la política española había caminos, cualesquiera que fueran sus accidentes, ...

- Julio Verne - Aventura de tres Rusos y tres Ingleses.txt

... no son tan fuertes como las tuyas y resisten mal los caminos intrincados como ...

Query: 10

- 02. DISCURSOS CONMEMORATIVOS 10 DE OCTUBRE.txt

... DISCURSOS CONMEMORATIVOS AL 10 DE OCTUBRE (FRAGMENTOS). JOSÉ MARTÍ. ...

- 01. LECTURA EN STECK HALL.txt

... 10 ...

- Julio Verne - Aventura de tres Rusos y tres Ingleses.txt

... 10 ...

- Julio Verne - Ante la bandera.txt

... 10 …