Moogle!

-Informe Escrito.

Autor: Amalia Beatriz Valiente Hinojosa.

Grupo: C-112.

El objetivo de este proyecto fue implementar un motor de búsqueda para encontrar documentos en forma de .txt de una forma eficiente.
Para esto fue usado el modelo vectorial de texto, que no es más que un modelo algebraico utilizado para filtrado, recuperación, indexado y cálculo de relevancia de información. Representa documentos en lenguaje natural de una manera formal mediante el uso de vectores en un espacio lineal multidimensional.

Este modelo es un proceso que se divide en una serie de pasos. El primero es calcular el peso de las palabras de los documentos, cuya fórmula ha sido implementada en tres métodos, el primero es para calcular la frecuencia de cada término, para ello e busca cuantas veces aparece la palabra en el texto y luego se divide entre el valor de la palabra que más se repite.

CÁLCULO DE PESOS EN LOS DOCUMENTOS

Sea $freq_{i,j}$ la frecuencia del término t_i en el documento d_j. Entonces, la frecuencia normalizada $f_{i,j}$ del término t_i en el documento d_j está dada por:

$$tf_{i,j} = \frac{freq_{i,j}}{max_l\, freq_{l,j}}$$

donde el máximo se calcula sobre todos los términos del documento d_j. Si el término t_i no aparece en el documento d_j entonces $f_{i,j} = 0$.

El siguiente paso es calcular la frecuencia inversa del documento, para ello se busca en cuantos documentos se encuentra la palabra, luego se aplica la fórmula, Logaritmo en base 10 del resultado de dividir el valor de la cantidad de documentos que hay en la base de datos entre el valor de la cantidad de documentos en el que aparece la palabra.

Por último, multiplicando los resultados de ambos métodos, se obtiene el peso de cada palabra de los documentos de la base de datos.

CÁLCULO DE PESOS EN LOS DOCUMENTOS

Sea N la cantidad total de documentos en el sistema y n_i la cantidad de documentos en los que aparece el término t_i. La frecuencia de ocurrencia de un término t_i dentro de todos los documentos de la colección idf_i (del inglés *inverse document frequency*) está dada por:

$$idf_i = log\frac{N}{n_i}$$

El peso del término t_i en el documento d_j está dado por:

$$w_{i,j} = tf_{i,j} \times idf_i$$

Estos valores se encuentran guardados en un diccionario, cuya key es el título de cada documento y el value es otro diccionario que contiene como keys las palabras y como values los valores de TF*IDF.

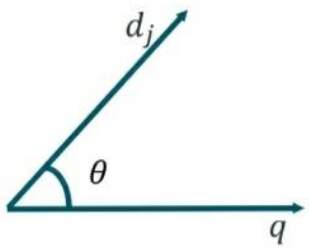
El mismo proceso se realiza con la query, y se almacenan en otro diccionario, cuya key es la palabra y el value es el peso de la palabra.

Para obtener la similitud que hay entre un documento y la query se calcula así:

La correlación se calcula utilizando el coseno del ángulo comprendido entre los vectores documentos d_j y la consulta q.

$$sim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \cdot |\vec{q}|}$$

$$sim(d_j, q) = \frac{\sum_{i=1}^{n} w_{i,j} \times w_{iq}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{n} w_{i,q}^2}}$$

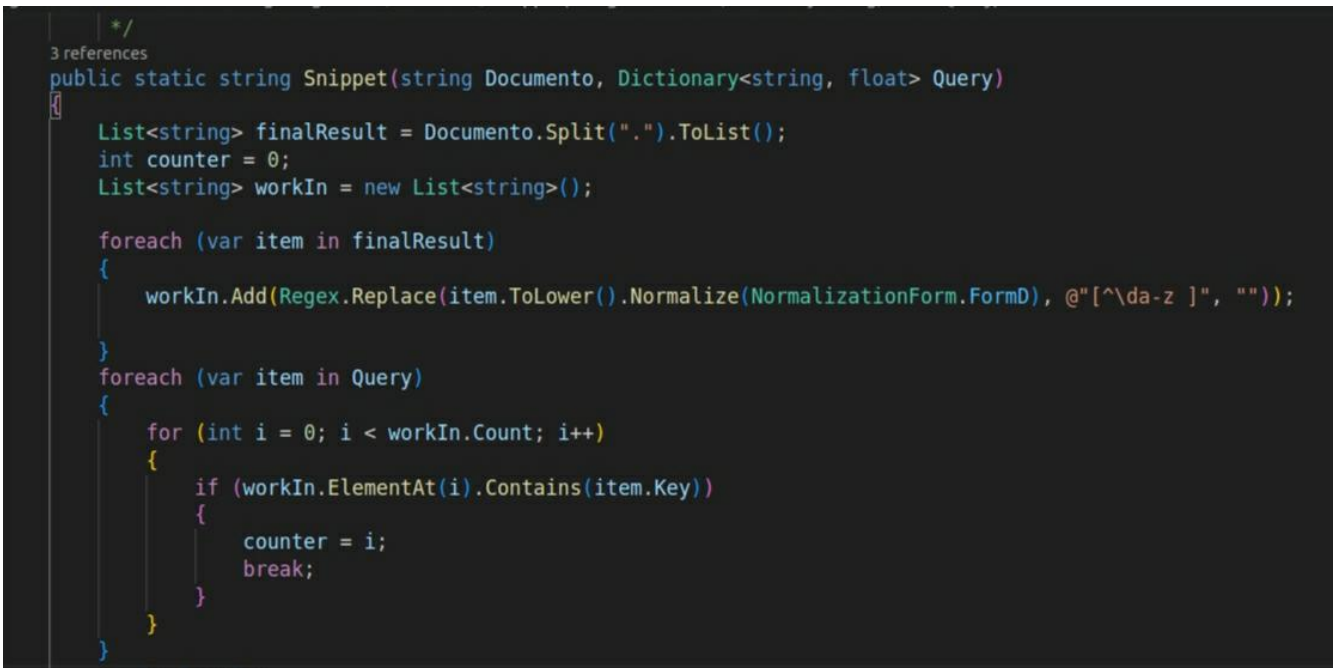


$|\vec{d_j}|, |\vec{q}|$ normas de los vectores documento y consulta respectivamente

En esta fórmula el numerador es suma de la multiplicación del valor del peso de las palabras que están en la query y en el documento.

El denominador se encuentra dividido en dos partes.
La primera parte del denominador es la longitud del vector documento, que no es más que la raíz cuadrada de la suma del cuadrado del peso de cada palabra.
La segunda es la longitud del vector documento.
Esta operación está implementada en el método Score de la clase TFIDF y devuelve un diccionario donde se encuentran ordenados de menor a mayor los titulos de los documentos y su valor de similitud.

Por último y por ahora, se encuentra el Snippet, que es una porción del documento donde aparece al menos una palabra de la query.



```
     */
3 references
public static string Snippet(string Documento, Dictionary<string, float> Query)
{
    List<string> finalResult = Documento.Split(".").ToList();
    int counter = 0;
    List<string> workIn = new List<string>();

    foreach (var item in finalResult)
    {
        workIn.Add(Regex.Replace(item.ToLower().Normalize(NormalizationForm.FormD), @"[^\da-z ]", ""));

    }
    foreach (var item in Query)
    {
        for (int i = 0; i < workIn.Count; i++)
        {
            if (workIn.ElementAt(i).Contains(item.Key))
            {
                counter = i;
                break;
            }
        }
    }
```

Hasta aquí, por cada palabra de la query, se mira si está en alguna oración del documento, si aparece al menos una palabra en una oración, el ciclo finaliza y se conserva en una variable la posición de la oración deseada.

```
string Result;
if ((counter >= 0) && (counter < finalResult.Count - 1))
{
    Result = finalResult.ElementAt(counter) + finalResult.ElementAt(counter + 1);
}
else if ((counter == finalResult.Count - 1) && (finalResult.Count > 1))
{
    Result = finalResult.ElementAt(counter - 1) + finalResult.ElementAt(counter);
}
else
{
    Result = finalResult.ElementAt(counter);
}

return Result;
}
```

Entonces, dependiendo de la posición y la cantidad de oraciones, el método devuelve la oración en la posición guardada y la oración que está después de esa, la que está antes de esa, o sólo esa.