Laboratorio 5: Representación tf y tf-idf

Huerta Aguilar, Jesus., Huitzil Juárez, Guadalupe Quetzalli., Pérez Sánchez, Jasmine., Ruiz Ramírez, Gabino
Benemérita Universidad Autónoma de Puebla
Recuperación de la Información
Fecha: 27 Febrero 2024

Resumen: Con ayuda de los resultados obtenidos en los laboratorios 1,2,3 y 4 realizaremos una matriz mostrando la frecuencia de las palabras que hay en cada documento, para ello nos apoyaremos del vocabulario obtenido en el laboratorio 4, una vez obtenido el tf, (frecuencia de termino), calcularemos el idf-tf, para ello primo con ayuda de nuestra formula aprendida en clase calcularemos el idf y por último lo multiplicaremos con el tf.

Palabras clave: tokens, palabras vacías, re, nltk, Python, porter, lower, truncar, Split, stopwords, NPL, vocabulario, sorted, tf, idf-tf.

I. INTRODUCCIÓN

En esta práctica, exploraremos en detalle dos aspectos fundamentales de la recuperación de información: la frecuencia de término (TF) y la frecuencia de término-inversa de documento (TF-IDF). Estos conceptos proporcionan una base sólida para comprender cómo se representa y se mide la importancia de los términos en un conjunto de documentos. Recordemos que el TF mide la frecuencia con la que un término específico aparece en un documento, mientras que el TF-IDF, combina el TF con la Inversa de Frecuencia de Documento (IDF), que mide la rareza de un término en el corpus de documentos. La multiplicación del TF y el IDF permite dar más peso a los términos que son frecuentes en un documento específico pero raros en el corpus en general, lo que ayuda a identificar términos más relevantes.

II. OBJETIVO Y PLANTEAMIENTO DEL PROBLEMA

El objetivo principal de esta práctica es familiarizarse con el modelo vectorial en la recuperación de información y comprender cómo se representan documentos y consultas utilizando este enfoque. Además, se busca aplicar los conceptos de Frecuencia de Término (TF) e Inversa de Frecuencia de Término en Documento (IDF-TF) para mejorar la representación y la relevancia de los términos en el proceso de recuperación de información.

III. DESARROLLO EXPERIMENTAL

Para realizar esta práctica necesitaremos:

- Software Python
- Librería NLTK
- Colección NPL de la página: https://ir.dcs.gla.ac.uk/resources/test_collections/npl/
- Laboratorios 1, 2, 3 y 4.

Para poder realizar nuestro código debemos tomar en cuenta dos cosas:

- La dimensión de los vectores será igual al tamaño del vocabulario reducido que obtuvimos en la práctica 4.
- Nos apoyaremos del documento en que se encuentra el vocabulario reducido.

Contemplando estos dos puntos procederemos a desarrollar nuestro código.

1. Primero representaremos cada documento empleando el modelo vectorial y el esquema de pesado tf y guardamos la matriz generada.

   Creamos nuestra función que permite leer el contenido del documento, en seguida creamos una función que permita generar la matriz en un documento de acuerdo con el tamaño del vocabulario, tomando en cuenta la línea del documento.

   En seguida recuperamos el vocabulario guardado en el archivo de vocabulario, este lo guardaremos en una lista para que sea más fácil al momento de emplearlo.

```python
# leer un archivo y devolver su contenido como una lista de líneas
def leer_archivo(nombre_archivo):
    with open(nombre_archivo, 'r', encoding='utf-8') as archivo:
        contenido = archivo.readlines()
    return contenido

# escribir una matriz en un archivo de texto
def escribir_matriz(nombre_archivo, matriz):
    with open(nombre_archivo, 'w', encoding='utf-8') as archivo:
        contador = 0
        for fila in matriz:
            contador += 1
            progress_bar(contador, 11419) #LLAMADA A FUNCIÓN 'progress_bar()'
            archivo.write('\t'.join(map(str, fila)) + '\n')

# generar el vocabulario a partir de un archivo de texto
def generar_vocabulario(archivo_vocabulario):
    vocabulario = {}
    lineas = leer_archivo(archivo_vocabulario)
    contador = 0
    for linea in lineas:
        contador += 1
        palabras = linea.strip().split()
        for palabra in palabras:
            vocabulario[palabra] = len(vocabulario)
    return vocabulario
```

   En seguida vectorizaremos el documento es decir asignaremos a cada documento un vector numérico que capture alguna característica relevante del texto. Para ello creamos una función que permita realizar este proceso, donde crearemos una matriz para almacenar las representaciones vectoriales, creamos una lista de frecuencias de cada palabra del vocabulario que se encuentre en el documento con ayuda de un contador, una vez obtenida la frecuencia de la palabra en el documento actual este valor se guardara en la matriz.

```python
def vectorizar_documentos(archivo_documentos, vocabulario):
    lineas = leer_archivo(archivo_documentos)
    matriz = []
    print("\nVECTORIZANDO DOCUMENTOS...")
    for linea in lineas:
        frecuencias = [0] * len(vocabulario)
        palabras = linea.strip().split()
        for palabra in palabras:
            if palabra in vocabulario:
                indice = vocabulario[palabra]
                frecuencias[indice] += 1
        matriz.append(frecuencias)
    return matriz
```

Una vez creada nuestras funciones a ocupar procederemos a guardar nuestros documentos a ocupar en variables para hacer más fácil el manejo de estos, por ejemplo, el archivo del vocabulario, el archivo donde se encuentran los documentos, y el archivo donde se guardará la matriz dada. En seguida llamaremos a la función generar vocabulario creada anteriormente, vectorizaremos los documentos y por último, guardaremos la matriz en el archivo de salida llamado matriz-docs.txt. Para corroborar que la matriz cuente con el numero correcto de filas y columnas imprimiremos ese valor para compararlo después con los documentos.

```python
# Archivos de entrada
archivo_vocabulario = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\vocabularioReducidoT.txt'
archivo_documentos  = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\documento-TRUNCADO.txt'
# Archivo de salida
archivo_salida      = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\matriz-docs.txt'

# Generar vocabulario
vocabulario = generar_vocabulario(archivo_vocabulario)

# Vectorizar documentos
matriz = vectorizar_documentos(archivo_documentos, vocabulario)

# Escribir matriz en archivo de salida
print("\n\nCREANDO MATRIZ TF...")
escribir_matriz(archivo_salida, matriz)

print("\n\nMATRIZ TF GENERADA Y GUARDADA EN:", archivo_salida)
# Imprimir la cantidad de filas y columnas de la matriz
print("\n\tFILAS  DE  LA  MATRIZ TF:", len(matriz))
print("\tCOLUMNAS DE LA MATRIZ TF:", len(matriz[0]))

```

2. De la misma manera que el paso anterior, representaremos ahora el archivo de queries empleando el modelo vectorial y esquema de pesado tf.

   Cambiando únicamente el valor del total de los documentos(filas) pasando de 11419 a 93

```python
# escribir una matriz en un archivo de texto
def escribir_matriz(nombre_archivo, matriz):
    with open(nombre_archivo, 'w', encoding='utf-8') as archivo:
        contador = 0
        for fila in matriz:
            contador += 1
            progress_bar(contador, 93) #LLAMADA A FUNCIÓN 'progress_bar()'
            archivo.write('\t'.join(map(str, fila)) + '\n')
```

   También cambiando el valor donde guardaremos el resultado en un nuevo archivo llamado matriz-query.txt que en variable también es archivo de salida.

```
# Archivos de entrada
archivo_vocabulario = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\vocabularioReducidoT.txt'
archivo_documentos  = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\cueri-TRUNCADO.txt'
# Archivo de salida
archivo_salida      = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\matriz-query.txt'

```

3. Repita los pasos 1 y 2 con esquema de pesado tf-idf para documento y para queries.

   Para este caso crearemos una función que permita calcular el IDF, primero calculando el número de documentos (N) que hay en la colección, el número de palabras y el número de documentos con esa palabra (N_t), aplicamos la formula aprendida en clase para calcular el idf:

$$idf = \log\frac{N}{N_t} + 1$$

   Después crearemos una función que permita escribir el vocabulario junto con su idf y esto lo guardaremos en un archivo llamado IDF-docs, en variable llamado archivo_idf

```
# Función para calcular el IDF
def calcular_idf(matriz):
    num_documentos = len(matriz)
    num_palabras = len(matriz[0])
    idf = [0] * num_palabras
    print("\nCALCULANDO IDF...")
    for i in range(num_palabras):
        progress_bar(i + 1, num_palabras) #LLAMADA A FUNCIÓN 'progress_bar()'
        # Contar en cuántos documentos aparece la palabra
        num_documentos_con_palabra = sum([1 for documento in matriz if documento[i] > 0])
        # Calcular el IDF
        idf[i] = math.log(num_documentos / (num_documentos_con_palabra)) + 1
    return idf

# Calcular IDF
idf = calcular_idf(matriz)

# Función para escribir el vocabulario junto con sus valores IDF en un archivo de texto
def escribir_vocabulario_idf(nombre_archivo, vocabulario, idf):
    with open(nombre_archivo, 'w', encoding='utf-8') as archivo:
        for palabra, valor_idf in zip(vocabulario, idf):
            archivo.write(palabra + '\t' + str(valor_idf) + '\n')

# Generar archivo con vocabulario e IDF
archivo_idf = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\IDF-docs.txt'
escribir_vocabulario_idf(archivo_idf, vocabulario, idf)

print("\n\nVALORES IDF GUARDADOS EN:", archivo_idf)
```

   Una vez obtenido el idf procederemos a multiplicarlo por el tf para así obtener el IDF-TF guardando la matriz obtenida en el archivo de salida llamado matriz-TF-IDF-docs.txt.

```
# Función para calcular el TF-IDF
def calcular_tfidf(matriz_tf, idf):
    print("\nCALCULANDO TF - IDF...")
    matriz_tfidf = []
    contador = 0
    for tf_vector in matriz_tf:
        contador += 1
        tfidf_vector = [tf * idf[i] for i, tf in enumerate(tf_vector)]
        matriz_tfidf.append(tfidf_vector)
        progress_bar(contador, 11419) #LLAMADA A FUNCIÓN 'progress_bar()'
    return matriz_tfidf

# Calcular TF-IDF
matriz_tfidf = calcular_tfidf(matriz, idf)

# Archivo de salida para la matriz TF-IDF
archivo_salida_tfidf = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\matriz-TFIDF-docs.txt'

# Escribir matriz TF-IDF en archivo de salida
print("\n\nCREANDO MATRIZ TF-IDF...")
escribir_matriz(archivo_salida_tfidf, matriz_tfidf)

print("\n\nMATRIZ TF-IDF GENERADA Y GUARDADA EN:", archivo_salida_tfidf)
```

De la misma manera calcularemos el TF-IDF para los queries: únicamente cambiando los archivos en los que se guardará:

```
# Generar archivo con vocabulariocls IDF
archivo_idf = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\IDF-query.txt'
escribir_vocabulario_idf(archivo_idf, vocabulario, idf)

print("\n\nVALORES IDF GUARDADOS EN:", archivo_idf)
```

```
# Archivo de salida para la matriz TF-IDF
archivo_salida_tfidf = r'C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\matriz-TFIDF-query.txt'

# Escribir matriz TF-IDF en archivo de salida
print("\n\nCREANDO MATRIZ TF-IDF...")
escribir_matriz(archivo_salida_tfidf, matriz_tfidf)

print("\n\nMATRIZ TF-IDF GENERADA Y GUARDADA EN:", archivo_salida_tfidf)
```

IV. DISCUSIÓN Y RESULTADOS

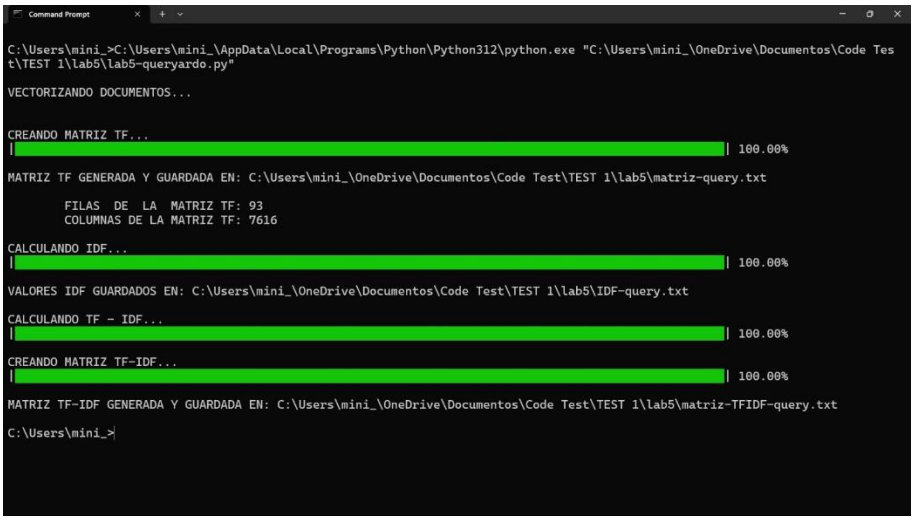
Como resultados de nuestra primera matriz de TF obtuvimos lo siguiente:

Para documentos al momento de ejecutar obtenemos que el número de filas y columnas es correcto, ya que consta de 11429 documentos y 7616 términos:

```
C:\Users\mini_>C:\Users\mini_\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\mini_\OneDrive\Documentos\Code Tes
t\TEST 1\lab5\lab5-docs.py"

VECTORIZANDO DOCUMENTOS...


CREANDO MATRIZ TF...
| 100.09%

MATRIZ TF GENERADA Y GUARDADA EN: C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\matriz-docs.txt

       FILAS  DE  LA  MATRIZ TF: 11429
       COLUMNAS DE LA MATRIZ TF: 7616

CALCULANDO IDF...
| 100.00%

VALORES IDF GUARDADOS EN: C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\IDF-docs.txt

CALCULANDO TF - IDF...
| 100.09%

CREANDO MATRIZ TF-IDF...
| 100.09%

MATRIZ TF-IDF GENERADA Y GUARDADA EN: C:\Users\mini_\OneDrive\Documentos\Code Test\TEST 1\lab5\matriz-TFIDF-docs.txt

C:\Users\mini_>
```

Nuestra matriz es la siguiente observamos que la mayor parte está conformada por 0 ya que la mayor parte del vocabulario no se encuentra en cada documento de esta forma observamos que es más difícil encontrar un 1 u otro valor:

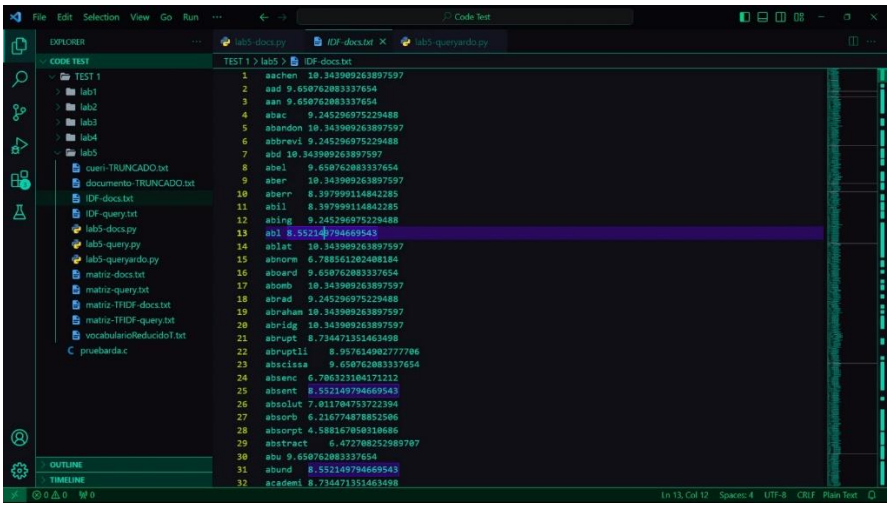
Para queries al momento de ejecutar podemos observar de igual manera que el valor es correcto donde las filas es el número de consulta mientras que las columnas corresponden al número de termino:



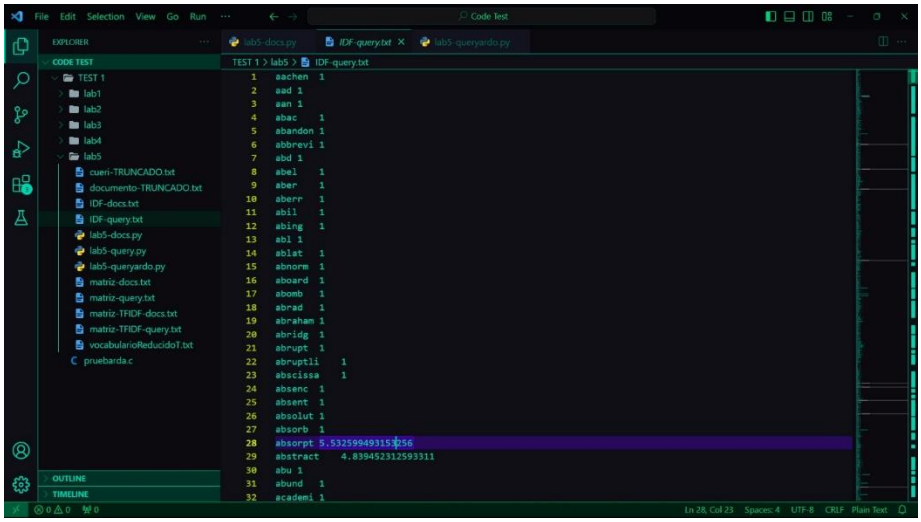
Así mismo observamos de igual manera que la matriz esta conformada por mas 0 que 1 u otro valor:

Como IDF obtuvimos el siguiente:

Documentos aquí podemos observar el término junto con su IDF, para corroborar el resultado tomaremos en cuenta el termino 23 con su IDF:



Queries: de igual forma que en los documentos podemos observar el término con su IDF correspondiente y para corroborar tomaremos en cuenta el IDF del término resaltado en azul y lo buscaremos en el siguiente paso



Por último, nuestro resultado de matriz TF-IDF fue el siguiente:

Para documentos recordando el valor anterior y ajustando la matriz podemos observar que en el documento 2732 (fila) se encuentra el resultado TF-IDF correspondiente al termino(columna):

Para queries: recordando el término con su IDF seleccionado anteriormente podemos observar que en la consulta 21 se encuentra el resultado correcto y correspondiente al término; en este caso era el termino 28 y ajustando la matriz podemos observar que se encuentra correctamente.

V. CONCLUSIONES

En resumen, en este laboratorio hemos explorado la representación de documentos y consultas en el modelo vectorial, utilizando herramientas como Python y NLTK. A partir de los archivos pre-procesados de la colección NPL y los queries proporcionados, hemos aplicado el esquema de pesado tf y tf-idf para crear matrices que representan tanto nuestros documentos como nuestras consultas.

Al emplear el esquema de pesado tf, pudimos asignar pesos a cada término en función de su frecuencia en los documentos, lo que nos brinda una manera efectiva de representar la importancia relativa de cada palabra dentro de un texto. Por otro lado, al aplicar el esquema tf-idf, consideramos tanto la frecuencia del término en el documento como su importancia en toda la colección, lo que nos permite capturar mejor la relevancia de un término en un contexto más amplio.

Estas representaciones vectoriales nos brindan una manera eficaz de comparar documentos y consultas en un espacio matemático, facilitando la implementación de algoritmos de recuperación de información. Al guardar las matrices generadas, hemos creado una base de datos eficiente que podemos utilizar en futuras prácticas y análisis.

Este laboratorio ha sido fundamental para comprender cómo las técnicas de modelado de texto nos permiten estructurar y analizar grandes colecciones de documentos.

VI. BIBLIOGRAFÍA

1. NLTK. (2022). SnowballStemmer. Recuperado de https://www.nltk.org/api/nltk.stem.SnowballStemmer.html?highlight=stopwords
2. NLTK. (2022). WordNetLemmatizer. Recuperado de https://www.nltk.org/api/nltk.stem.WordNetLemmatizer.html?highlight=wordnet