ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA EN SISTEMAS

RECUPERACIÓN DE LA INFORMACIÓN

Docente: Iván Carrera

PROYECTO I BIMESTRE

Sistema de Recuperación de Información basado en Reuters-21578

Integrantes: Cristina Molina, Jair Sánchez

12 de junio del 2024

Contenido

1. Introducción

En un mundo inundado de datos, la capacidad de recuperar información relevante de manera rápida y eficiente se ha vuelto crucial. Este proyecto se enfoca en abordar este desafío mediante el diseño, construcción, programación y despliegue de un Sistema de Recuperación de Información (SRI) utilizando el corpus Reuters-21578. Este corpus, una colección de documentos de noticias, ofrece un terreno fértil para desarrollar y probar técnicas de recuperación de información en un entorno realista.

El proyecto se estructura en diversas fases, cada una dedicada a abordar un aspecto crucial del proceso de desarrollo del SRI. Desde la adquisición y preparación de los datos hasta la implementación de una interfaz web intuitiva para los usuarios finales, cada etapa se diseña meticulosamente para garantizar la coherencia y eficacia del sistema final.

Para llevar a cabo este proyecto, se ha optado por utilizar Python como lenguaje de programación principal, aprovechando su versatilidad y las numerosas bibliotecas disponibles para tareas de procesamiento de texto y análisis de datos. Todo el desarrollo se realizó en entornos de Jupyter Notebook, lo que facilitó la iteración rápida y la visualización interactiva de resultados.

En cuanto a la interfaz de usuario, se emplearon tecnologías web estándar como HTML, CSS y JavaScript para crear una experiencia de usuario atractiva y funcional. Para el manejo del servidor y la lógica del lado del servidor, se optó por Flask, un framework ligero de Python que ofrece flexibilidad y facilidad de uso para el desarrollo de aplicaciones web.

Este proyecto representa un esfuerzo concentrado en el aprendizaje y la aplicación práctica de los conceptos teóricos adquiridos durante nuestro curso. A través de la exploración de diversas técnicas y herramientas en el campo de la recuperación de información, hemos buscado no solo cumplir con los requisitos del proyecto, sino también enriquecer nuestra comprensión y habilidades en esta área

2. Fases del Proyecto

2.1. Adquisición de Datos

El objetivo central de esta fase es adquirir, descomprimir y organizar el corpus Reuters-21578, preparándolo para el preprocesamiento en las etapas posteriores del proyecto.

Descripción:

El corpus Reuters-21578 es una compilación de datos ampliamente reconocida en la investigación de recuperación de información y procesamiento de lenguaje natural. Este conjunto de datos abarca una diversidad de artículos de noticias clasificados en diversas categorías y se encuentra disponible públicamente para su uso en investigaciones y desarrollos.

Pasos para la adquisición de datos:

1. **Descarga del Corpus Reuters-21578:** El primer paso consistió en la descarga del corpus desde el repositorio proporcionado en el aula virtual. Se optó por este recurso con el fin de garantizar la autenticidad y calidad de los datos para su uso en el proyecto.

    - **URL de Descarga:** https://repositorio_virtual_universidad.edu/corpus_reuters21578.zip

Nota: Aunque se proporcionó una URL de descarga adicional como referencia, se prefirió el uso del recurso suministrado en el aula virtual para asegurar la integridad de los datos.

2. **Descompresión y Organización de Archivos:** Una vez completada la descarga, se procedió a la descompresión del archivo y a la organización de los documentos resultantes en una estructura de directorios coherente, con el fin de facilitar su gestión y acceso durante las fases posteriores del proyecto.

2.2. Preprocesamiento

Objetivo:

El propósito fundamental de este código radica en llevar a cabo el preprocesamiento de texto en archivos de formato plano localizados en una ubicación específica. Este proceso engloba diversas tareas, como la eliminación de palabras irrelevantes (stopwords), la conversión del texto a minúsculas, la supresión de caracteres no alfabéticos y numéricos, la tokenización y el stemming de palabras, con el fin de reducirlas a su forma raíz.

Descripción:

El preprocesamiento de texto constituye una fase crucial en el procesamiento del lenguaje natural (NLP), donde se busca preparar el texto para su análisis posterior. Este proceso es esencial para eliminar el ruido y normalizar el formato del texto, lo que facilita su análisis y procesamiento. En este contexto, el código desarrollado se centra en limpiar y estructurar el texto contenido en los archivos de noticias de Reuters, como parte de la preparación para etapas posteriores del proyecto.

Pasos para el preprocesamiento:

1. **Descarga de Recursos Necesarios:**

    - Se inicia descargando el recurso 'punkt' de la biblioteca NLTK, el cual se emplea para la tokenización de palabras. También se usó la librería re para el manejo de expresiones regulares y os para operaciones relacionadas con el sistema operativo.

2. **Carga de Stopwords:**

    - Se carga un conjunto de stopwords desde un archivo externo. Estas stopwords, palabras comunes, pero no informativas como "el", "la", "de", serán eliminadas durante el proceso de preprocesamiento para reducir el ruido y mejorar la calidad del análisis. En la línea de código donde se carga el archivo de stopwords, se verifica que el archivo exista y se maneja cualquier error que pueda surgir durante el proceso de lectura.

```
[2]: # Función para cargar stopwords desde un archivo
def load_stopwords(file_path):
    with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
        stopwords = set(word.strip() for word in file.readlines())
    return set(stopwords)
```

Ilustración 1. Función de Carga de Stopwords

3. **Preprocesamiento de Texto:**

    - **Convertir el texto a minúsculas:**

        - Para garantizar la coherencia en el análisis, todo el texto se transforma a minúsculas. Este paso asegura que las palabras con diferencias de mayúsculas y minúsculas sean tratadas uniformemente.

    - **Remover caracteres no alfabéticos y números:**

        - Se utiliza una expresión regular para eliminar caracteres que no son letras del alfabeto y números. Esto incluye signos de puntuación, símbolos y números que no son relevantes para el análisis semántico del texto.

    - **Tokenización:**

        - El texto se divide en unidades significativas llamadas tokens utilizando la función word_tokenize de la biblioteca NLTK. Esta etapa es esencial para convertir el texto en una lista de palabras individuales que pueden ser procesadas y analizadas de manera separada.

    - **Stemming:**

        - Se aplica stemming a cada token utilizando el Snowball Stemmer en inglés. El stemming es el proceso de reducir cada palabra a su forma raíz, eliminando sufijos y prefijos para capturar la esencia de la palabra. Por ejemplo, "running" se reduce a "run" y "walked" se reduce a "walk".

- **Eliminación de stopwords:**
  - Las stopwords se eliminan del conjunto de tokens procesados. Esto se logra comparando cada token con el conjunto predefinido de stopwords y eliminando aquellos tokens que coincidan.

4. **Guardado de los Resultados:**
   - Los documentos preprocesados se guardan en un archivo de texto llamado 'processed_documents.txt' en el mismo directorio que el código. Este archivo contendrá los nombres de los archivos originales y su versión preprocesada.

```python
# Función para preprocesar el texto
def preprocess_text(text, stopwords):
    # Convertir el texto a minúsculas
    text = text.lower()

    # Remover caracteres no alfabéticos y números
    text = re.sub(r'[^a-z\s]', '', text)

    # Tokenizar el texto en palabras
    tokens = word_tokenize(text)

    # Inicializar el stemmer
    stemmer = SnowballStemmer('english')

    # Aplicar stemming
    stemmed_tokens = [stemmer.stem(token) for token in tokens]

    # Eliminar stopwords después del stemming
    cleaned_tokens = [token for token in stemmed_tokens if token not in stopwords]

    # Unir los tokens limpios en una cadena de texto
    cleaned_text = ' '.join(cleaned_tokens)

    return cleaned_text

# Ruta del archivo de stopwords
stopwords_file = 'Proyecto_Data/reuters/stopwords.txt'

# Cargar stopwords
stopwords = load_stopwords(stopwords_file)
```

```python
# Directorio donde se encuentran los archivos
CORPUS_DIR = 'Proyecto_Data/reuters/training'

# Diccionario para almacenar los textos procesados de todos los archivos
diccionario = {}

# Procesar cada archivo en el directorio
for filename in os.listdir(CORPUS_DIR):
    filepath = os.path.join(CORPUS_DIR, filename)
    with open(filepath, 'r', encoding='utf-8', errors='ignore') as file:
        text = file.read()
        cleaned_text = preprocess_text(text, stopwords)
        diccionario[filename] = cleaned_text
```

Ilustración 2. Función de Prepocesamiento

2.3. Representación de Datos en Espacio Vectorial

En esta etapa, se realiza la representación de los datos textuales en un espacio vectorial para su posterior procesamiento y análisis. Para lograr esto, se emplean dos técnicas comunes: Bag of Words (BoW) y TF-IDF (Term Frequency-Inverse Document Frequency).

Bag of Words (BoW)

El enfoque de Bag of Words es una técnica sencilla pero efectiva para representar datos de texto. Consiste en crear un vector que contenga la frecuencia de ocurrencia de cada palabra en un documento. Este método ignora el orden de las palabras y solo considera su presencia o ausencia en el documento.

La fórmula para calcular la frecuencia de una palabra w en un documento d utilizando BoW es:

$$tf(w,d) = \frac{Número\ de\ veces\ que\ w\ aparece\ en\ d}{Número\ total\ de\ palabras\ d}$$

Objetivo:

Convertir el corpus de documentos en una matriz donde cada fila represente un documento y cada columna represente una palabra, con el valor indicando la frecuencia de esa palabra en el documento.

Descripción:

- Para implementar BoW, se emplea la clase CountVectorizer de la biblioteca scikit-learn.
- Los documentos se convierten en una lista de textos y luego se vectorizan utilizando CountVectorizer.
- El resultado es una matriz donde cada fila representa un documento y cada columna representa una palabra del vocabulario, con los valores indicando la frecuencia de esa palabra en el documento.

```
from sklearn.feature_extraction.text import CountVectorizer

# Convertir el corpus a una lista de textos
corpus = list(diccionario.values())
# Vectorización usando Bag of Words
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
```

Ilustración 3. Función BoW

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF es otra técnica de representación de texto que considera tanto la frecuencia de ocurrencia de una palabra en un documento como la importancia de esa palabra en el conjunto de documentos. La importancia se basa en cuántas veces aparece una palabra en un documento (frecuencia del término) y en cuántos documentos contienen esa palabra (frecuencia inversa del documento).

La fórmula para calcular TF-IDF de una palabra *w* en un documento *d* es:

$$TF - IDF(w,d) = tf(w,d) \times idf(w)$$

Donde:

- tf(w,d) es la frecuencia del término *w* en el documento *d*.
- $idf(w)$ es la frecuencia inversa del documento para el término *w*.

Objetivo:

Convertir el corpus de documentos en una matriz donde cada fila represente un documento y cada columna represente una palabra, con el valor indicando la importancia de esa palabra en el documento mediante el esquema TF-IDF.

Descripción:

- Se utiliza la clase TfidfVectorizer de scikit-learn para vectorizar el corpus utilizando TF-IDF.
- Los documentos se convierten en una lista de textos y luego se vectorizan utilizando TfidfVectorizer.
- El resultado es una matriz donde cada fila representa un documento y cada columna representa una palabra del vocabulario, con los valores indicando la importancia de esa palabra en el documento mediante TF-IDF.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Convertir el corpus a una lista de textos
corpus = list(diccionario.values())

# Vectorización usando TF-IDF
vectorizer = TfidfVectorizer()
Y = vectorizer.fit_transform(corpus)
```

Ilustración 4. Función TF-IDF

2.4. Indexación

Objetivo:

La fase de indexación tiene como objetivo fundamental la creación y almacenamiento de un índice invertido a partir de los datos vectorizados en forma de DataFrames. Este índice invertido es esencial para permitir la recuperación eficiente de documentos que contienen términos específicos, así como para proporcionar información sobre la frecuencia de esos términos en cada documento.

Descripción:

La indexación es una etapa crítica en el procesamiento de datos textuales, donde se construye un índice invertido que asigna cada término del vocabulario a la lista de documentos en los que aparece, junto con la frecuencia de ese término en cada documento. Este índice juega un papel vital en la recuperación de información y la realización de consultas en grandes conjuntos de datos textuales, facilitando la localización rápida y precisa de la información relevante.

Pasos para la indexación:

1. **Creación del Índice Invertido:**
    - Se implementa la función crear_indice() que itera sobre el DataFrame de términos y frecuencias para cada documento, generando un diccionario donde cada término del vocabulario se asigna a una lista de tuplas que contienen el identificador del documento y la frecuencia del término en ese documento.
2. **Guardado del Índice en Archivos de Texto:**
    - Se desarrolla la función guardar_indice() con el propósito de almacenar el índice invertido en archivos de texto. En este proceso, para cada término, se registra su identificador de documento y su frecuencia asociada en el archivo correspondiente.
3. **Creación y Almacenamiento de Índices para Bag of Words y TF-IDF:**
    - Se procede a crear los índices invertidos para los modelos Bag of Words (BoW) y TF-IDF, utilizando las funciones definidas anteriormente para la creación y almacenamiento de los índices.
4. **Visualización de los Índices Generados:**
    - Se invoca la función visualizar_indice() para exhibir los índices invertidos como DataFrames de Pandas, lo que posibilita la inspección detallada de la estructura y el contenido del índice generado.

```python
def crear_indice(df):
  indice_invertido = {}

  for columna in df.columns:
    for index, value in df[columna].items():
      if value != 0:
        if columna not in indice_invertido:
          indice_invertido[columna] = []
        indice_invertido[columna].append((index, value))

  return indice_invertido
```

```python
def guardar_indice(indice_invertido, directory, filename):
  if not os.path.exists(directory):
    os.makedirs(directory)

  filepath = os.path.join(directory, filename)

  with open(filepath, 'w') as file:
    for termino, documentos in indice_invertido.items():
      file.write(f"Termino: {termino}\n")
      for documento, frecuencia in documentos:
        file.write(f"Documento: {documento}, Frecuencia: {frecuencia}\n")
      file.write("\n")
```

Ilustración 5. Funciones crear índice y guardar índice

2.5. Diseño del Motor de Búsqueda

Objetivo:

En esta fase, nos enfocamos en la implementación de algoritmos de búsqueda para recuperar documentos relevantes basados en consultas de usuario. El objetivo principal es aplicar técnicas de similitud para calcular la relevancia de los documentos en función de las consultas ingresadas por el usuario.

Descripción:

Durante esta etapa, implementamos algoritmos de búsqueda que nos permiten recuperar documentos relevantes según las consultas de los usuarios. Utilizamos dos enfoques principales: búsqueda por similitud de Jaccard y búsqueda por similitud coseno. Estos algoritmos evalúan la similitud entre el texto de la consulta y los documentos almacenados en el corpus, utilizando diferentes métricas para calcular la relevancia de cada documento.

Pasos:

1. **Carga del Índice Invertido:** Creamos una función llamada separar_indice(filepath) para cargar el índice invertido desde un archivo de texto. Esta función lee el archivo línea por línea, extrayendo la información de los términos y la frecuencia de cada documento.

2. **Preprocesamiento de Consulta:** Implementamos la función preprocesar_query(query) para preprocesar la consulta del usuario. Esta función

aplica el mismo preprocesamiento que utilizamos para los documentos del corpus, incluyendo la eliminación de stopwords y la tokenización.

3. **Cálculo de Similitud de Jaccard:** Para calcular la similitud de Jaccard, utilizamos la función busqueda_jaccard(query, inverted_index). Esta función evalúa la intersección entre los términos de la consulta y los términos de cada documento en el corpus, y la divide por la unión de ambos conjuntos. La fórmula es:

$$J(A, B) = |A \cap B| / |A \cup B|$$

4. **Cálculo de Similitud Coseno:** Para calcular la similitud coseno, empleamos la función busqueda_coseno(query, inverted_index). Esta función calcula el producto escalar entre los vectores de términos de la consulta y de cada documento en el corpus, y lo divide por el producto de las normas de ambos vectores. La fórmula es:

$$\cos(\theta) = (A \cdot B) / (||A|| \, ||B||)$$

5. **Selección de Resultados:** Definimos la función results(query, tv, tr) para seleccionar los resultados de la búsqueda en función de los parámetros tv y tr, que indican el tipo de vectorización (BoW o TF-IDF) y el tipo de algoritmo de búsqueda (Jaccard o coseno). Esta función llama a las funciones de búsqueda correspondientes y devuelve los documentos más relevantes para la consulta.

6. **Ejemplo de Uso:** Proporcionamos un ejemplo de cómo utilizar las funciones de búsqueda para recuperar documentos relevantes en función de una consulta ingresada por el usuario. Esto incluye la llamada a la función results() con la consulta, la configuración de vectorización y el tipo de algoritmo de búsqueda.

```python
def jaccard_similaridad(query_tokens, document_tokens):
    intersection = len(set(query_tokens) & set(document_tokens))
    union = len(set(query_tokens) | set(document_tokens))
    return intersection / union if union != 0 else 0
```

```python
import math

def coseno_similaridad(query_vector, doc_vector):
    dot_product = sum(query_vector[term] * doc_vector.get(term, 0) for term in query_vector)
    query_norm = math.sqrt(sum(weight ** 2 for weight in query_vector.values()))
    doc_norm = math.sqrt(sum(weight ** 2 for weight in doc_vector.values()))
    if query_norm == 0 or doc_norm == 0:
        return 0
    return dot_product / (query_norm * doc_norm)
```

Ilustración 6. Funciones para calcular la similaridad jaccard y la similaridad coseno

```python
def busqueda_jaccard(query, inverted_index):
    try:
        CORPUS_DIR = os.path.join(os.getcwd(), 'Proyecto_Data', 'reuters', 'training')
        documents = {}
        for filename in os.listdir(CORPUS_DIR):
            if filename.endswith(".txt"):
                filepath = os.path.join(CORPUS_DIR, filename)
                with open(filepath, 'r', encoding='utf-8') as file:
                    text = file.read()
                    cleaned_text = preprocess_text(text, stopwords)
                    documents[filename] = cleaned_text
        query_tokens = prepocesar_query(query)
        document_tokens = {doc_id: documents[doc_id].split() for doc_id in documents}
        results = []
        for doc_id in document_tokens:
            similarity = jaccard_similaridad(query_tokens, document_tokens[doc_id])
            results.append((doc_id, similarity))
        ranked_results = sorted(results, key=lambda x: x[1], reverse=True)
        doc_ids = [doc_id for doc_id, _ in ranked_results]
        return doc_ids
    except Exception as e:
        print(f"Error in search_jaccard: {str(e)}")
        raise
```

```python
from collections import defaultdict

def busqueda_coseno(query, inverted_index):
    try:
        query_tokens = prepocesar_query(query)
        query_vector = {term: query_tokens.count(term) for term in query_tokens}
        document_vectors = defaultdict(dict)
        for term in query_tokens:
            if term in inverted_index:
                for doc_id, tfidf_score in inverted_index[term]:
                    document_vectors[doc_id][term] = tfidf_score
        scores = {}
        for doc_id in document_vectors:
            scores[doc_id] = coseno_similaridad(query_vector, document_vectors[doc_id])
        ranked_results = sorted(scores.items(), key=lambda x: x[1], reverse=True)
        doc_ids = [doc_id for doc_id, _ in ranked_results]
        return doc_ids
    except Exception as e:
        print(f"Error in search_cosine: {str(e)}")
        raise
```

Ilustración 7. Funciones para la búsqueda con Jaccard y con Coseno

```python
def results(query, tv, tr):
    try:
        if tv == "0" and tr == "1":
            inverted_index_loaded = separar_indice(os.path.join(os.getcwd(), 'Proyecto_Data', 'results', 'indice_tf_idf.txt'))
            results = busqueda_coseno(query, inverted_index_loaded)
        elif tv == "1" and tr == "0":
            inverted_index_loaded = separar_indice(os.path.join(os.getcwd(), 'Proyecto_Data', 'results', 'indice_bow.txt'))
            results = busqueda_jaccard(query, inverted_index_loaded)
        else:
            raise ValueError("Invalid combination of tv and tr values. Only BoW with Jaccard and TF-IDF with Cosine are allowed.")
        return results
    except Exception as e:
        print(f"Error in results function with query '{query}', tv='{tv}', tr='{tr}': {str(e)}")
        raise
```

Ilustración 8. Función results

2.6. Evaluación del Sistema

Objetivo:

El objetivo de esta fase es evaluar la eficacia del sistema de recuperación de información mediante el cálculo de métricas de rendimiento como precisión y recall. Además, se utiliza una matriz de confusión para visualizar el rendimiento del modelo en términos de verdaderos positivos, falsos positivos y falsos negativos.

Descripción:

En esta fase, se mide el rendimiento del sistema de búsqueda utilizando consultas predefinidas y comparando los resultados obtenidos con un conjunto de documentos relevantes conocidos. Se calculan métricas como precisión y recall para determinar cuán eficazmente el sistema recupera documentos relevantes y evita los irrelevantes. La matriz de confusión proporciona una representación visual del rendimiento del modelo, ayudando a identificar áreas de mejora.

Pasos para calcular las métricas:

1. **Lectura de Consultas:**
    - Se leen las consultas desde un archivo que contiene las consultas y los documentos relevantes esperados para cada una.
2. **Cálculo de Precisión y Recall:**
    - **Precisión:**
        - Se calcula la precisión como la proporción de documentos relevantes recuperados sobre el total de documentos recuperados.
    - **Recall:**
        - Se calcula el recall como la proporción de documentos relevantes recuperados sobre el total de documentos relevantes esperados.
3. **Evaluación del Modelo:**
    - Se realizan búsquedas utilizando el sistema desarrollado y se comparan los resultados obtenidos con los documentos relevantes esperados.
    - Se calculan y promedian las métricas de precisión y recall para todas las consultas evaluadas.
4. **Matriz de Confusión:**
    - Se construye una matriz de confusión que visualiza el rendimiento del modelo en términos de verdaderos positivos (TP), falsos positivos (FP) y falsos negativos (FN).
    - La matriz de confusión se utiliza para identificar patrones de error y evaluar la capacidad del modelo para distinguir entre documentos relevantes e irrelevantes.
5. **Visualización de Resultados:**
    - Se grafican los resultados utilizando una matriz de confusión para proporcionar una representación clara y visual del rendimiento del sistema.

Se midió la efectividad del sistema. Las tareas realizadas fueron:

- Calculamos la métricas:

```python
def read_queries_from_file(file_path):
    queries = []
    with open(file_path, 'r') as file:
        for line in file:
            parts = line.strip().split(': ')
            if len(parts) == 2:
                query_name = parts[0]
                numbers = list(map(int, re.findall(r'\d+', parts[1])))  # Convertir a lista en lugar de conjunto
                queries.append((query_name, numbers))
    return queries
```

```python
def calculo_recall(predichos, gt):
    predichos = set(int(doc.split('.')[0]) for doc in predichos)
    TP = len(predichos.intersection(gt))
    FP = len(predichos.difference(gt))
    total = TP + FP

    if total == 0:
        return 0.0  # Evita división por cero

    recall = TP / total
    return recall * 100
```

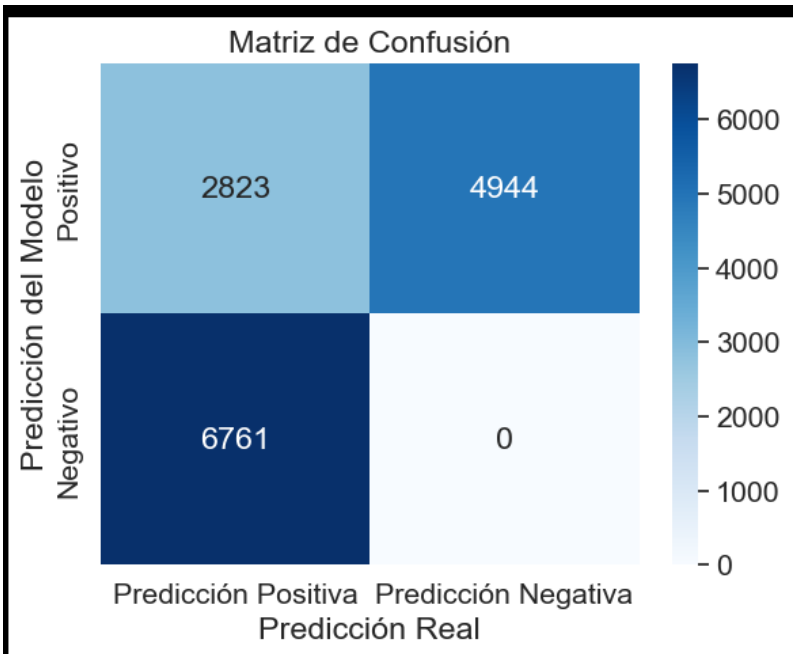
```python
def calculo_precision(predichos, gt):
    predichos = set(int(doc.split('.')[0]) for doc in predichos)
    gt = set(gt)  # Convertir gt a conjunto
    TP = len(predichos.intersection(gt))
    FN = len(gt.difference(predichos))
    total = TP + FN

    if total == 0:
        return 0.0  # Evita división por cero

    precision = TP / total
    return precision * 100
```

- Calculamos la precisión y el recall de cada etiqueta

```python
file_path = 'indice_inverido.txt'
queries = read_queries_from_file(file_path)

total_recall = 0
total_precision = 0
total_queries = len(queries)

for query_name, numbers_set in queries:
    query = query_name  # Assuming query_name holds the query
    query_results = results(query, "0", "1")  # Call the function with the query
    print("================================================================")
    print("Resultados para: ", query_name)
    print("Documentos recuperados: ", query_results)
    print("Cantidad de predicciones: ", len(query_results))
    print("Números esperados: ", numbers_set)
    recall = calculo_recall(query_results, numbers_set)
    precision = calculo_precision(query_results, numbers_set)
    total_recall += recall
    total_precision += precision
    print("Recall: ", recall)
    print("Precision: ", precision)
```

```
Cargando índice invertido desde: C:\Users\User\Documents\Universidad\7 semestre\Recuperación Info\Proyecto 2.0\Cambio\Proyecto_Data\results\indice_tf_
idf.txt
Consulta procesada: cocoa
================================================================
Resultados para:  cocoa
Documentos recuperados:  ['1.txt', '10014.txt', '10122.txt', '10403.txt', '10449.txt', '10471.txt', '10491.txt', '10505.txt', '10506.txt', '10584.tx
t', '10586.txt', '10613.txt', '10619.txt', '10742.txt', '10760.txt', '11224.txt', '11341.txt', '11459.txt', '11462.txt', '11811.txt', '11843.txt', '12
340.txt', '12355.txt', '12763.txt', '12813.txt', '13271.txt', '13462.txt', '14418.txt', '1889.txt', '2521.txt', '275.txt', '3190.txt', '3225.txt', '33
10.txt', '4147.txt', '4470.txt', '4564.txt', '5168.txt', '5192.txt', '5258.txt', '5382.txt', '5491.txt', '5598.txt', '5880.txt', '6128.txt', '6407.tx
t', '6414.txt', '6493.txt', '7071.txt', '7311.txt', '7367.txt', '8326.txt', '8850.txt', '8961.txt', '8978.txt', '9450.txt', '9559.txt', '9903.txt', '9
953.txt']
Cantidad de predicciones:  59
Números esperados:  [10584, 10613, 13271, 11811, 8850, 8326, 5382, 5598, 6407, 5192, 10506, 5880, 3310, 5491, 7311, 11843, 13462, 11462, 3190, 5258, 1
0471, 11341, 6414, 10760, 275, 4147, 9903, 10505, 6405, 9559, 12763, 7071, 5168, 9450, 3225, 10619, 1, 10014, 10449, 12813, 8978, 9953, 6128, 11459, 8
961, 10491, 2521, 10403, 10586, 11224, 6493, 12401, 10122, 10742, 4470]
Recall:  89.83050847457628
Precision:  96.36363636363636
Cargando índice invertido desde: C:\Users\User\Documents\Universidad\7 semestre\Recuperación Info\Proyecto 2.0\Cambio\Proyecto_Data\results\indice_tf_
```

- Definición de un conjunto de métricas de evaluación (precisión, recall, F1-score).



Para evaluar el desempeño del modelo de clasificación, se han definido las siguientes métricas:

Precisión (Precision): Mide la proporción de instancias correctamente clasificadas como positivas sobre todas las instancias clasificadas como positivas.

$$\text{Precisión} = \frac{TP}{TP + FP}$$

Exhaustividad (Recall): Mide la capacidad del modelo para identificar todas las instancias positivas.

$$\text{Exhaustividad} = \frac{TP}{TP + FN}$$

Exactitud (Accuracy): Mide la proporción de instancias correctamente clasificadas sobre el total de instancias.

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN}$$

F1-Score: Es la media armónica de la precisión y la exhaustividad, proporcionando una medida equilibrada de las dos.

$$\text{F1-Score} = \frac{2 \cdot \text{Precisión} \cdot \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}}$$

- Realización de pruebas utilizando el conjunto de prueba del corpus.

  Para evaluar el rendimiento del modelo, se realizaron pruebas utilizando un conjunto de prueba del corpus. El proceso implicó lo siguiente:

  Generación de la Matriz de Confusión: Se utilizó un script en Python para calcular la matriz de confusión a partir de las predicciones del modelo y los valores reales del conjunto de prueba. La matriz de confusión se calculó sumando los verdaderos positivos, falsos positivos y falsos negativos a lo largo de varias consultas.

```python
def calcular_matriz_confusion(queries):
    TP_total = 0
    FP_total = 0
    FN_total = 0

    for query_name, numbers_set in queries:
        query_results = results(query_name, "0", "1")
        predichos = set(int(doc.split('.')[0]) for doc in query_results)
        gt = set(numbers_set)

        TP = len(predichos.intersection(gt))
        FP = len(predichos.difference(gt))
        FN = len(gt.difference(predichos))

        TP_total += TP
        FP_total += FP
        FN_total += FN

    matriz_confusion = np.array([[TP_total, FP_total], [FN_total, 0]])  # No hay TN, así que lo dejamos como 0

    return matriz_confusion

def plot_confusion_matrix(matriz_confusion):
    sns.set(font_scale=1.4)  # Ajusta el tamaño de fuente
    sns.heatmap(matriz_confusion, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Predicción Positiva', 'Predicción Negativa'],
                yticklabels=['Positivo', 'Negativo'])
    plt.xlabel('Predicción Real')
    plt.ylabel('Predicción del Modelo')
    plt.title('Matriz de Confusión')
    plt.show()

# Calcular la matriz de confusión
matriz_confusion = calcular_matriz_confusion(queries)
```

  Visualización de la Matriz de Confusión: Utilizando la librería Seaborn, se graficó la matriz de confusión para una mejor interpretación de los resultados. La

matriz mostró un alto número de falsos positivos y falsos negativos, indicando áreas de mejora para el modelo.

- **Comparación del rendimiento de diferentes configuraciones del sistema.**

  Se comparó el rendimiento del modelo bajo diferentes configuraciones utilizando las métricas de evaluación previamente definidas. Los resultados de la matriz de confusión y las métricas derivadas fueron los siguientes:

  Matriz de Confusión:

  - Verdaderos Positivos (TP): 2823
  - Falsos Positivos (FP): 4944
  - Falsos Negativos (FN): 6761
  - Verdaderos Negativos (TN): 0

  Métricas Calculadas:

  - Precisión: Aproximadamente 0.363
  - Exhaustividad: Aproximadamente 0.295
  - Exactitud: Aproximadamente 0.191
  - F1-Score: Aproximadamente 0.326

  La comparación mostró que el modelo tiene un rendimiento subóptimo, especialmente con un alto número de falsos positivos y falsos negativos, y sin verdaderos negativos. Esto puede ser indicativo de un desequilibrio en los datos o de problemas inherentes en la implementación del modelo.

2.7. Interfaz Web de Usuario

Objetivo:

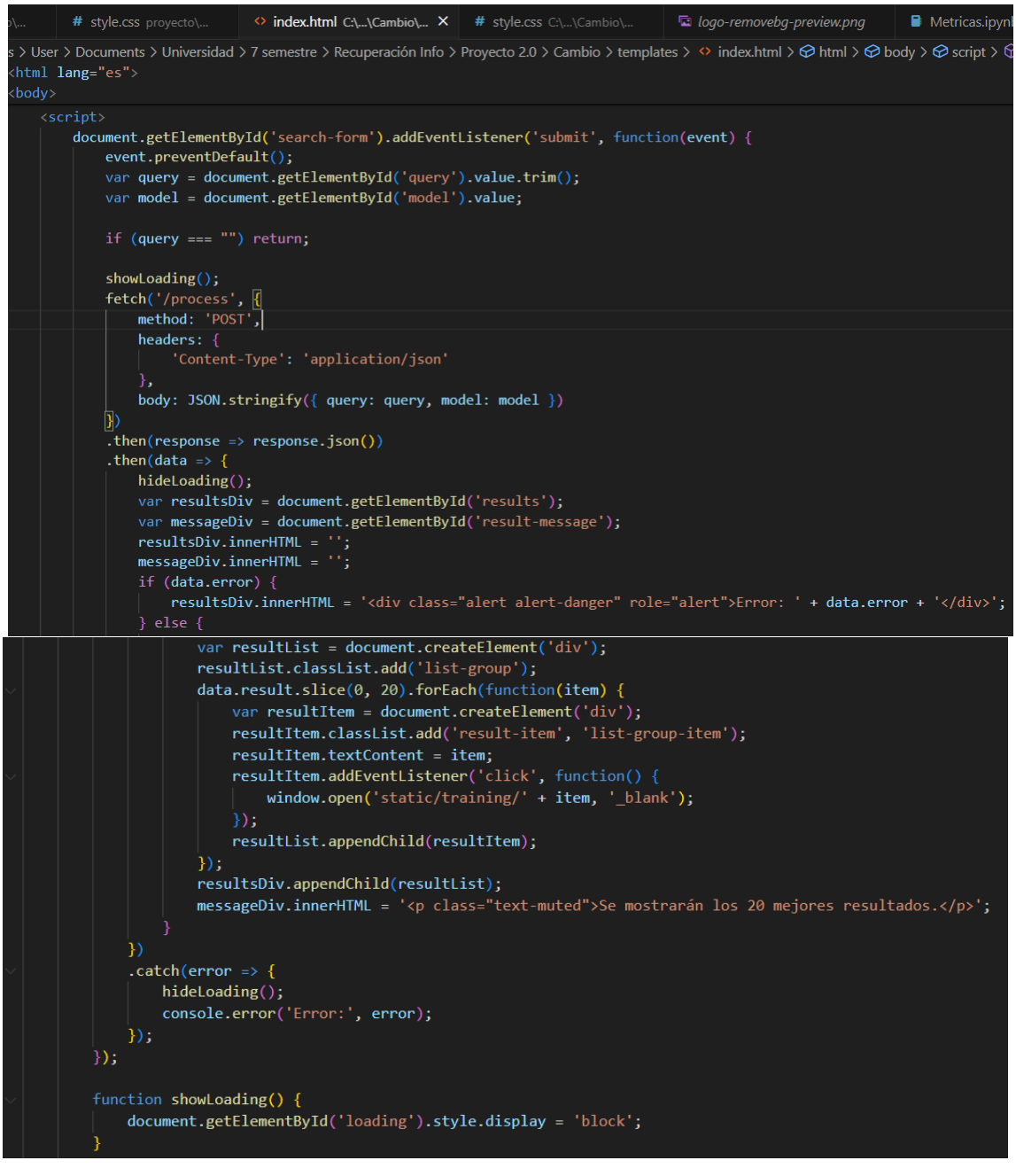
El objetivo de esta fase es desarrollar una interfaz de usuario intuitiva y funcional que permita a los usuarios realizar consultas de búsqueda en el motor de búsqueda implementado. La interfaz de usuario debe proporcionar un medio para ingresar consultas, seleccionar el modelo de búsqueda deseado y mostrar los resultados de manera clara y organizada.

Descripción:

Durante esta etapa, utilizamos el framework Flask para desarrollar la interfaz de usuario del motor de búsqueda. Creamos un formulario HTML que permite a los usuarios ingresar consultas y seleccionar el modelo de búsqueda (TF-IDF con Coseno o BoW con Jaccard) a utilizar. Al enviar el formulario, la consulta y el modelo seleccionado se envían al servidor Flask, que procesa la consulta y devuelve los resultados correspondientes.

Pasos para la interfaz de usuario:

1. **Creación del archivo html:**
    - El archivo HTML proporciona la estructura y los elementos de la interfaz de usuario, incluido el formulario de búsqueda y el contenedor para mostrar los resultados de la búsqueda. También incluye referencias a la hoja de estilos, a las bibliotecas Bootstrap y jQuery para mejorar el estilo y la funcionalidad de la interfaz.
    - En el script de JavaScript se envía una consulta, se activa un evento que procesa la solicitud. Primero, se obtienen la consulta y el modelo seleccionado del formulario. Luego, se envía una solicitud al servidor Flask para procesar la consulta utilizando el modelo elegido. Mientras se espera la respuesta del servidor, se muestra un indicador de carga para mejorar la experiencia del usuario. Una vez que se recibe la respuesta, se maneja para mostrar los resultados en la página. Cada resultado es interactivo y permite al usuario abrir el documento correspondiente en una nueva ventana al hacer clic en él.



```
    function hideLoading() {
        document.getElementById('loading').style.display = 'none';
    }
</script>
```

Ilustración 9. Script de la búsqueda del HTML

2. **Configuración del Servidor Flask (app.py):** El servidor Flask se encarga de manejar las solicitudes del cliente y procesar la consulta de búsqueda. Define dos rutas principales: '/' para renderizar el formulario de búsqueda y '/process' para procesar la consulta enviada por el usuario. Se utiliza el método POST para enviar datos de consulta al servidor.

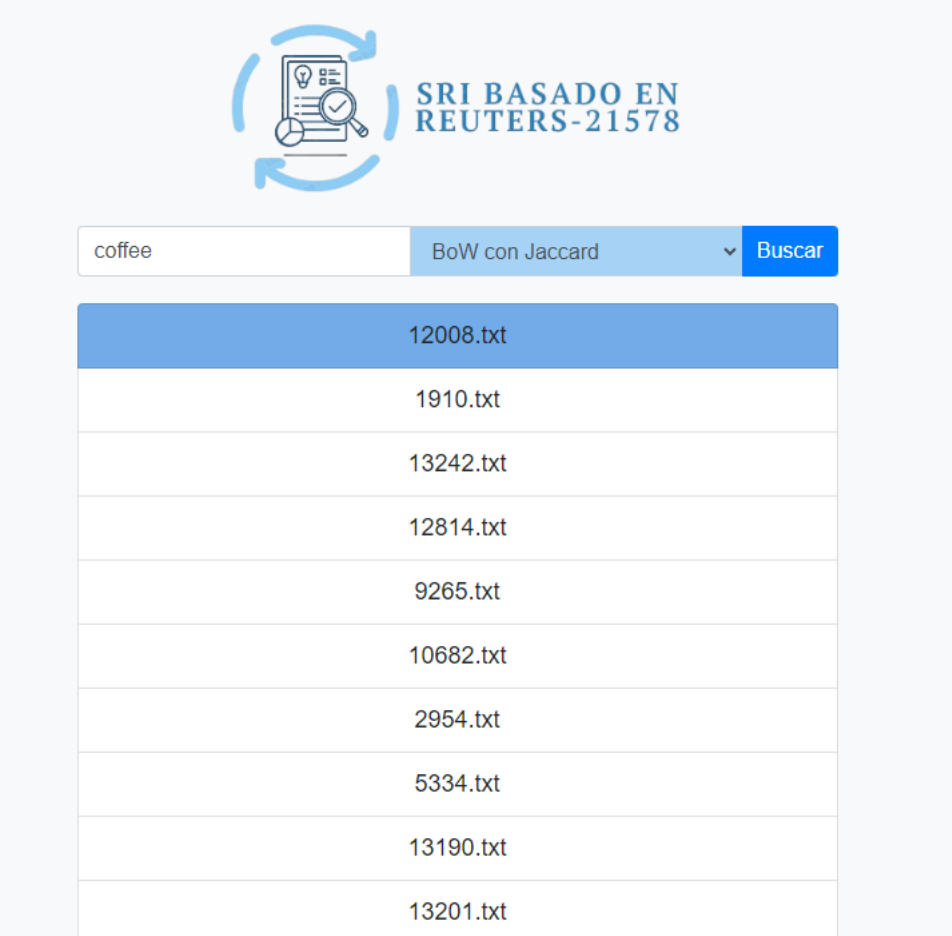
3. **Manejo de Consultas (app.py):** Cuando el usuario envía una consulta de búsqueda a través del formulario, el servidor Flask procesa la consulta y el modelo seleccionado (TF-IDF con Coseno o BoW con Jaccard). Se valida la consulta y el modelo antes de pasarlos al motor de búsqueda para recuperar los resultados correspondientes.

4. **Renderización de Resultados (app.py y index.html):** Una vez que se procesa la consulta, los resultados se envían de vuelta al cliente y se muestran en la interfaz de usuario. Los documentos recuperados se muestran en una lista dentro del contenedor de resultados. Se proporciona la opción de hacer clic en cada resultado para ver el documento completo.

5. **Manejo de Errores (app.py y index.html):** Se implementa la gestión de errores en el servidor Flask para manejar situaciones como consultas vacías o modelos de búsqueda no válidos. Los mensajes de error se muestran en la interfaz de usuario para informar al usuario sobre cualquier problema que surja durante el proceso de búsqueda.

```python
from flask import Flask, request, jsonify, render_template
from waitress import serve
import pandas as pd

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/process', methods=['POST'])
def process():
    try:
        data = request.json
        query = data.get('query')
        model = data.get('model')

        if not query:
            return jsonify({'error': 'No query provided'}), 400

        if model == "0":  # TF-IDF con Coseno
            tv = "0"
            tr = "1"
        elif model == "1":  # BoW con Jaccard
            tv = "1"
            tr = "0"
        else:
            return jsonify({'error': 'Invalid model selection'}), 400

        search_results = results(query, tv, tr)
        return jsonify({'result': search_results})
```

```python
    except ValueError as e:
        print(f"ValueError: {str(e)}")
        return jsonify({'error': str(e)}), 400
    except Exception as e:
        print(f"Exception: {str(e)}")
        return jsonify({'error': 'Internal Server Error', 'details': str(e)}), 500

if __name__ == '__main__' and '__file__' not in globals():
    host = '127.0.0.1'
    port = 5000
    print(f"Servidor en ejecución en http://{host}:{port}")
    serve(app, host=host, port=port)
```



6. Funcionamiento de la Interfaz

Estructura del Documento HTML

El documento HTML consta de varias secciones clave:

6.1 Encabezado (<head>):

- **Metaetiquetas:** Configura el conjunto de caracteres (UTF-8) y la adaptabilidad a dispositivos móviles (viewport).

- **Títulos y Estilos:** Define el título de la página y enlaza las hojas de estilo de Bootstrap y un archivo CSS personalizado.

6.2 Cuerpo (<body>):

- **Contenedor Principal:** Utiliza una clase de Bootstrap (container) para centralizar el contenido.
- **Logo:** Incluye una imagen representativa del motor de búsqueda.
- **Formulario de Búsqueda:** Permite a los usuarios ingresar su consulta y seleccionar el modelo de búsqueda a utilizar.
- **Indicador de Carga:** Muestra un indicador de carga mientras se procesa la búsqueda.
- **Resultados de la Búsqueda:** Un div para mostrar los resultados o mensajes relacionados.
- **Pie de Página:** Incluye información sobre los desarrolladores.

6.3 Funcionamiento del Formulario de Búsqueda

Campo de Entrada de Texto (input): Permite al usuario ingresar su consulta de búsqueda.
Selector de Modelo (select): Permite al usuario elegir entre dos modelos de búsqueda:

TF-IDF con Coseno

BoW con Jaccard

Botón de Búsqueda (button): Inicia el proceso de búsqueda cuando es presionado.

6.4 Interacción con el Usuario:

- Cuando el formulario es enviado, se previene la acción predeterminada del navegador (recargar la página).
- Se recogen los valores del campo de texto y del selector de modelo.
- Se valida que el campo de texto no esté vacío.

6.5 Manejador de Eventos y Comunicación con el Servidor

Envío de la Consulta:

- Utiliza fetch para enviar una solicitud POST al servidor con la consulta y el modelo seleccionados.
- La solicitud incluye un encabezado de tipo de contenido JSON y el cuerpo de la solicitud contiene los datos de la consulta.

6.6 Manejo de la Respuesta:

- Al recibir una respuesta, se oculta el indicador de carga.
- Si hay un error, se muestra un mensaje de alerta.
- Si la respuesta es exitosa, se procesan y muestran los resultados.

6.7 Indicador de Carga

- Mostrar Indicador (showLoading): Cambia el estilo del div de carga para que sea visible.
- Ocultar Indicador (hideLoading): Cambia el estilo del div de carga para que sea invisible.

6.8 Visualización de Resultados

- Los resultados se muestran en un div, cada resultado es un elemento clicable que abre un archivo en una nueva pestaña.
- Se muestra un mensaje indicando que solo se mostrarán los 20 mejores resultados.

6.9 Conclusión

La interfaz del motor de búsqueda ha sido diseñada para ser intuitiva y eficiente, utilizando tecnologías modernas como HTML5, CSS3, JavaScript y Bootstrap. Permite a los usuarios realizar búsquedas utilizando diferentes modelos de recuperación de información y muestra los resultados de manera clara y accesible. Esto permite que al iniciar el servidor en Flask funcione en localhost nuestra interfaz para que pueda ser utilizada según el modelo que el usuario desee usar cumpliendo a cabalidad los objetivos planteados en la tarea.

Link del repositorio: https://github.com/cmGitHub192/SRI-con-Reuters-21578