ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

RECUPERACIÓN DE LA INFORMACIÓN

Proyecto del Primer Bimestre

Alumnos :

Alexander Saavedra
Andrés Suárez Suárez
Pablo Arcos Astudillo

FECHA DE ENTREGA : 19-06-2024

Tabla de Contenidos

Introducción

El objetivo de este proyecto es diseñar, construir, programar y desplegar un Sistema de Recuperación de Información (SRI) utilizando el corpus Reuters-21578. A continuación, se describe detalladamente cada fase del proyecto, incluyendo la adquisición de datos, preprocesamiento, representación de datos, indexación, diseño del motor de búsqueda, evaluación del sistema e interfaz web de usuario.

1. Adquisición de Datos

Para comenzar, se descargó el corpus Reuters-21578 desde el github en donde estaba el archivo *reuters.rar*. Este conjunto de datos fue descomprimido y guardado en una carpeta que se llama reuters

2. Preprocesamiento

2.1 Importación de librerías

Se tiene una lista de todas las importaciones de las librerías para Python para poder trabajar con nuestro corpus.

```
import os
import shutil
import re
import os
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
import spacy
import snowballstemmer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import jaccard_score
from collections import defaultdict
import numpy as np
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import itertools
from scipy.sparse import csr_matrix
```

2.2 Transformación de archivos a formato .txt

Al descomprimir los archivos se va a tomar en cuenta la carpeta training, se notó que no tienen ninguna extensión por lo cual se va a proceder a modificar todos los archivos para que tengan la extensión .txt y estos guardarlos en una nueva carpeta llamada training_txt.

```
# Ruta carpeta training
src_folder = 'reuters/training'
```

```
# Ruta nueva carpeta
dest_folder = 'reuters/training_txt'

# crear carpeta
os.makedirs(dest_folder, exist_ok=True)

# Iteramos sobre cada archivo para cambiar la extension
for filename in os.listdir(src_folder):
    src_file_path = os.path.join(src_folder, filename)
    dest_file_path = os.path.join(dest_folder, f"{filename}.txt")
    shutil.copy(src_file_path, dest_file_path)
```

2.3 Eliminación de stop words

En la carpeta *Reuters* se tiene un archivo llamado *stopwords.txt* en donde se tiene una lista de palabras que son consideradas como stop words, además de hacer todas las palabras minúsculas, las cuales deben ser eliminadas de todos nuestros archivos de nuestra carpeta de *training* y luego de procesar todos los archivos los vamos a guardar en una nueva carpeta llamada *training_stop_words_txt.*

```
# Cargamos el archivo con las stop words
stop_words_file = 'reuters/stopwords.txt'
with open(stop_words_file, 'r', encoding='utf-8') as file:
    stop_words = set(file.read().split())

# Nueva carpeta con los archivos eliminados las stop words
src_folder = 'reuters/training_txt'
dest_folder = 'reuters/training_stop_words_txt'

# Crea la carpeta si no existe
os.makedirs(dest_folder, exist_ok=True)

# Funcion para leer cada archivo
def read_file_with_encodings(file_path):
    encodings = ['utf-8', 'latin-1', 'iso-8859-1']
    for encoding in encodings:
            with open(file_path, 'r', encoding=encoding) as file:
                return file.read()

for filename in os.listdir(src_folder):
    src_file_path = os.path.join(src_folder, filename)
    content = read_file_with_encodings(src_file_path).lower()
    # Removemos las stop words
    cleaned_content = ' '.join([word for word in content.split() if word not in stop_words])
    dest_file_path = os.path.join(dest_folder, filename)
    with open(dest_file_path, 'w', encoding='utf-8') as file:
```

```
        file.write(cleaned_content)
```

2.4 Eliminación de caracteres especiales

Como penúltimo paso tenemos la eliminación de caracteres especiales de nuestros documentos, como por ejemplo comas, puntos, tildes, etc. Y lo guardamos en una nueva carpeta *final_txt.*

```
# El directorio final
input_directory = 'reuters/training_stop_words_txt'
output_directory = 'reuters/final_txt'

# Lo creamos si no existe
if not os.path.exists(output_directory):
    os.makedirs(output_directory)

# Funcion para limpiar caracteres especiales
def clean_text(text):
    cleaned_text = re.sub(r'[^A-Za-z0-9\s]', '', text)
    return cleaned_text

for filename in os.listdir(input_directory):
    with open(os.path.join(input_directory, filename), 'r', encoding='utf-8') as file:
        content = file.read()
        cleaned_content = clean_text(content)

    with open(os.path.join(output_directory, filename), 'w', encoding='utf-8') as output_file:
        output_file.write(cleaned_content)
```

2.5 Stematizacion y lematización

Vamos a stematizar y lematizar cada documento, esto quiere decir reducir a la raíz de cada palabra para asi poder tener normalizado cada documento, la diferencia entre la lemaizacion y la stematizacion es que una es más agresiva que otra. Luego de realizar este procedimiento se crearon dos carpetas llamadas *lemmatized* y *stemmed.*

```
# Cargar el modelo de lenguaje de spaCy
nlp = spacy.load('en_core_web_sm')

# Inicializar el stemmer
stemmer = snowballstemmer.stemmer('english')
```

```
# Función para lematizar y stematizar texto
def preprocess_text(text):
    doc = nlp(text)
    stemmed_tokens = [stemmer.stemWord(token.text) for token in doc]
    lemmatized_tokens = [token.lemma_ for token in doc]
    return ' '.join(stemmed_tokens), ' '.join(lemmatized_tokens)
```

```
# Directorios de entrada y salida
```

```
input_dir = 'reuters/final_txt'
output_dir_stemmed = 'final/Stemmed'
output_dir_lemmatized = 'final/lemmatized'

# Crear directorios de salida si no existen
os.makedirs(output_dir_stemmed, exist_ok=True)
os.makedirs(output_dir_lemmatized, exist_ok=True)

for filename in os.listdir(input_dir):
    with open(os.path.join(input_dir, filename), 'r', encoding='utf-8') as file:
        text = file.read()
        stemmed_text, lemmatized_text = preprocess_text(text)

    # Guardar texto stematizado
    with open(os.path.join(output_dir_stemmed, filename), 'w', encoding='utf-8') as file:
        file.write(stemmed_text)

    # Guardar texto lematizado
    with open(os.path.join(output_dir_lemmatized, filename), 'w', encoding='utf-8') as file:
        file.write(lemmatized_text)

print("Procesamiento completado.")
```

3. Representación de datos en espacios vectoriales

Se utilizaron las técnicas de Bag of Words (BoW) y TF-IDF para vectorizar el texto de las dos carpetas la de *lemmatized* y la de *stemmed*, vamos a realizar esto para poder comparar las dos técnicas para reducir las palabras a su raíz y ver si influyen o no en nuestro resultado final de nuestras búsquedas mediante querys.

3.1 Vectorizacion utilizando Bag of Words

```
# Función para vectorizar textos utilizando Bag of Words
def Bag_of_Words(texts):
    # Vectorización usando Bag of Words
    vectorizer = CountVectorizer()
    X_bow = vectorizer.fit_transform(texts)

    return X_bow, vectorizer
```

3.1.1 Utilizando la carpeta lemmatized

Vamos a vectorizar con Bag of Words que es la función que ya está definida anteriormente, pero con la carpeta que tiene los archivos lematizados.

```
# Vectorizar los documentos
X_bow_lemmatized, bow_vectorizer_lemmatized = Bag_of_Words(documents_lemmatized)

# Ver los resultados de Bag of Words
print("Bag of Words (BoW) Lemmatized:")
print(X_bow_lemmatized.toarray())
print("Caracteristicas de BoW Lemmatized:", bow_vectorizer_lemmatized.get_feature_names_out())
```

3.1.2 Utilizando la carpeta Stemmed

Vamos a vectorizar con Bag of Words que es la función que ya está definida anteriormente, pero con la carpeta que tiene los archivos stematizados.

```
# Vectorizar los documentos
X_bow_stemmed, bow_vectorizer_stemmed = Bag_of_Words(documents_stemmed)

# Ver los resultados de Bag of Words
print("Bag of Words (BoW) Stemmed:")
print(X_bow_stemmed.toarray())
print("Caracteristicas de BoW Stemmed:", bow_vectorizer_stemmed.get_feature_names_out())
```

3.2 Vectorizacion utilizando TF-IDF

```
# Función para vectorizar textos utilizando TF-IDF
def TF_IDF(texts):
    # Vectorización usando TF-IDF
    tfidf_vectorizer = TfidfVectorizer()
    X_tfidf = tfidf_vectorizer.fit_transform(texts)

    return X_tfidf, tfidf_vectorizer
```

3.2.1 Utilizando la carpeta lemmatized

Vamos a vectorizar con TF-IDF que es la función que ya está definida anteriormente, pero con la carpeta que tiene los archivos lematizados.

```
# Vectorizar los documentos
X_tfidf_lemmatized, tfidf_vectorizer_lemmatized = TF_IDF(documents_lemmatized)

# Ver los resultados de TF-IDF
print("TF-IDF Lemmatized:")
print(X_tfidf_lemmatized.toarray())
print("Caracteristicas de TF-IDF Lemmatized:", tfidf_vectorizer_lemmatized.get_feature_names_out())
```

3.2.2 Utilizando la carpeta Stemmed

Vamos a vectorizar con TF-IDF que es la función que ya está definida anteriormente, pero con la carpeta que tiene los archivos stematizados.

```
# Vectorizar los documentos
```

```
X_tfidf_Stemmed, tfidf_vectorizer_Stemmed = TF_IDF(documents_stemmed)

# Ver los resultados de TF-IDF
print("TF-IDF Stemmed:")
print(X_tfidf_Stemmed.toarray())
print("Caracteristicas de TF-IDF Stemmed:", tfidf_vectorizer_Stemmed.get_feature_names_out())
```

4. Diseño de motor de búsqueda

En esta sección se desarrolló la lógica para procesar consultas, implementando algoritmos de similitud como similitud coseno y Jaccard. Se lo realizo de dos formas una se encuentra en el archivo *preprocessing.ipynb* y el código que esta implementado en el back-end en donde se ocupa los archivo .pkl para poder serializar y desrealizar los datos.

4.1 Búsqueda similitud coseno con TF-IDF lematizado

```
def lemmatized_tfidf_cosine_similarity_search(query):
    # cargar la data transformada y los vectores
    with open('final/TF-IDF/lemmatized_vectorizer_tfidf.pkl', 'rb') as vec_file:
        vectorizer_tfidf = pickle.load(vec_file)

    with open('final/TF-IDF/lemmatized_X_tfidf.pkl', 'rb') as xt_file:
        X_tfidf = pickle.load(xt_file)

    with open('final/TF-IDF/lemmatized_filenames.pkl', 'rb') as f_file:
        filenames = pickle.load(f_file)

    query_vector = vectorizer_tfidf.transform([query])

    # calcular la similitud coseno
    cosine_sim_scores = cosine_similarity(X_tfidf, query_vector)

    # se crea un dataframe
    similarity_df = pd.DataFrame({'Filename': filenames, 'Cosine_Similarity': cosine_sim_scores.flatten()})

    # se organiza el documento
    similarity_df = similarity_df.sort_values(by='Cosine_Similarity', ascending=False)

    return similarity_df
```

4.2 Búsqueda similitud coseno con TF-IDF stematizado

```
def stemmed_tfidf_cosine_similarity_search(query):
    # cargar la data transformada y los vectores
    with open('final/TF-IDF/Stemmed_vectorizer_tfidf.pkl', 'rb') as vec_file:
        vectorizer_tfidf = pickle.load(vec_file)

    with open('final/TF-IDF/Stemmed_X_tfidf.pkl', 'rb') as xt_file:
        X_tfidf = pickle.load(xt_file)
```

```
    with open('final/TF-IDF/Stemmed_filenames.pkl', 'rb') as f_file:
        filenames = pickle.load(f_file)

    query_vector = vectorizer_tfidf.transform([query])

    # calcular la similitud coseno
    cosine_sim_scores = cosine_similarity(X_tfidf, query_vector)

     # se crea un dataframe
    similarity_df = pd.DataFrame({'Filename': filenames, 'Cosine_Similarity': cosine_sim_scores.flatten()})

     # se organiza el documento
    similarity_df = similarity_df.sort_values(by='Cosine_Similarity', ascending=False)

    return similarity_df
```

4.3 Búsqueda similitud jaccard con BoW lematizado

```
def lemmatized_bow_jaccard_similarity_search(query):
    # cargar la data transformada y los vectores
    with open('final/BOW/lemmatized_vectorizer_bow.pkl', 'rb') as vec_file:
        vectorizer_bow = pickle.load(vec_file)
    print("Loaded vectorizer")

    with open('final/BOW/lemmatized_X_bow.pkl', 'rb') as xb_file:
        X_bow = pickle.load(xb_file)
    print("Loaded X_bow")

    with open('final/BOW/lemmatized_filenames.pkl', 'rb') as f_file:
        filenames = pickle.load(f_file)
    print("Loaded filenames")

    # convertir la matriz de BoW a binaria
    X_bow_binary = (X_bow > 0).astype(int)
    print("Converted X_bow to binary")

    # Transformar la consulta a su representación binaria BoW
    query_vector = vectorizer_bow.transform([query])
    query_vector_binary = (query_vector > 0).astype(int)
    print("Converted query vector to binary")

   # Se asegura de que ambas matrices sean matrices de CRS
    query_vector_binary = csr_matrix(query_vector_binary)
    X_bow_binary = csr_matrix(X_bow_binary)

    # Depurar formas y tipos
    print("query_vector_binary type:", type(query_vector_binary), "shape:", query_vector_binary.shape)
```

```
    print("query_vector_binary values:", query_vector_binary.toarray())
    print("X_bow_binary type:", type(X_bow_binary), "shape:", X_bow_binary.shape)
    print("X_bow_binary values:", X_bow_binary.toarray())

    # Calcular puntuaciones de similitud de Jaccard
    jaccard_sim_scores = jaccard_similarity(query_vector_binary, X_bow_binary)
    print("Calculated Jaccard similarity scores")

    if jaccard_sim_scores is None:
        print("Error in calculating Jaccard similarity scores")
        return None

    # Verifique que las longitudes coincidan
    if len(filenames) != len(jaccard_sim_scores):
        print(f"Length mismatch: {len(filenames)} filenames vs {len(jaccard_sim_scores)} scores")
        return None

    # Se crea un dataframe
    similarity_df = pd.DataFrame({'Filename': filenames, 'Jaccard_Similarity': jaccard_sim_scores})
    print("Created similarity DataFrame")

    # Ordenar documentos según puntuaciones de similitud
    similarity_df = similarity_df.sort_values(by='Jaccard_Similarity', ascending=False)
    print("Sorted documents by similarity scores")

    return similarity_df
```

4.4 Búsqueda similitud jaccard con BoW stematizado

```
def stemmed_bow_jaccard_similarity_search(query):
    # cargar la data transformada y los vectores
    with open('final/BOW/Stemmed_vectorizer_bow.pkl', 'rb') as vec_file:
        vectorizer_bow = pickle.load(vec_file)
    print("Loaded vectorizer")

    with open('final/BOW/Stemmed_X_bow.pkl', 'rb') as xb_file:
        X_bow = pickle.load(xb_file)
    print("Loaded X_bow")

    with open('final/BOW/Stemmed_filenames.pkl', 'rb') as f_file:
        filenames = pickle.load(f_file)
    print("Loaded filenames")

    # convertir la matriz de BoW a binaria
    X_bow_binary = (X_bow > 0).astype(int)
    print("Converted X_bow to binary")
```

```
    # Transformar la consulta a su representación binaria BoW
    query_vector = vectorizer_bow.transform([query])
    query_vector_binary = (query_vector > 0).astype(int)
    print("Converted query vector to binary")

    # Se asegura de que ambas matrices sean matrices de CRS
    query_vector_binary = csr_matrix(query_vector_binary)
    X_bow_binary = csr_matrix(X_bow_binary)

    # Depurar formas y tipos
    print("query_vector_binary type:", type(query_vector_binary), "shape:", query_vector_binary.shape)
    print("query_vector_binary values:", query_vector_binary.toarray())
    print("X_bow_binary type:", type(X_bow_binary), "shape:", X_bow_binary.shape)
    print("X_bow_binary values:", X_bow_binary.toarray())

    # Calcular puntuaciones de similitud de Jaccard
    jaccard_sim_scores = jaccard_similarity(query_vector_binary, X_bow_binary)
    print("Calculated Jaccard similarity scores")

    if jaccard_sim_scores is None:
        print("Error in calculating Jaccard similarity scores")
        return None

    # Verifique que las longitudes coincidan
    if len(filenames) != len(jaccard_sim_scores):
        print(f"Length mismatch: {len(filenames)} filenames vs {len(jaccard_sim_scores)} scores")
        return None

    # Se crea un dataframe
    similarity_df = pd.DataFrame({'Filename': filenames, 'Jaccard_Similarity': jaccard_sim_scores})
    print("Created similarity DataFrame")

    # Ordenar documentos según puntuaciones de similitud
    similarity_df = similarity_df.sort_values(by='Jaccard_Similarity', ascending=False)
    print("Sorted documents by similarity scores")

    return similarity_df
```

6. Evaluación del Sistema

Para realizar la evaluación del sistema se usó el archivo cats.txt el cual relaciona a cada archivo de Reuters con una o varias categorías. El primer paso es leer este archivo y crear una estructura de datos que relacione el nombre del archivo con las categorías:

```
# Read the categories file
categories_file = 'reuters/cats.txt'
```

```
doc_categories = {}

with open(categories_file, 'r', encoding='utf-8') as f:
    for line in f:
        parts = line.strip().split()
        filename = parts[0].split('/')[-1]  # Extraer solo el nombre del archivo
        category_type = parts[0].split('/')[0]  # Obtener si es training o test
        categories = parts[1:]

        if category_type == 'training':
            doc_categories[filename] = categories
```

Ahora vamos a crear nuestra ground_truth, para esto usaremos una estructura de datos de tipo "defaultdict", para almacenar las categorías, siendo cada categoría una "key" del diccionario, y los títulos de los archivos como los atributos:

```
ground_truth = defaultdict(list)

for filename, categories in doc_categories.items():
    for category in categories:
        ground_truth[category].append(filename)
```

A continuación, se desarrolló la función evaluation, la cual para calcualr el rendimiento de un sistema de clasificacióny graficar su matriz de confusión . La función toma como parámetros el conjunto de datos ground_truth, un vectorizador de categorías category_vectorizer, la matriz de características vectorizadas del corpus X_corpus_vectorized, el método de similitud similarity_method (que puede ser cosine_similarity o jaccard_similarity), y un umbral threshold para determinar la clasificación.

Primero, se inicializan listas vacías para almacenar las matrices de confusión, precisiones y recalls de cada categoría. Luego, se itera sobre cada categoría en el conjunto de datos de la verdad fundamentada. Para cada categoría, se crean contadores para verdaderos positivos, falsos positivos, falsos negativos y verdaderos negativos. La categoría se transforma en un vector utilizando el vectorizador de categorías.

Dependiendo del método de similitud seleccionado, se calcula la similitud entre el vector de la categoría y la matriz de características vectorizadas del corpus. Si el método es cosine_similarity, se usa la función cosine_similarity de sklearn; si es jaccard_similarity, se utiliza jaccard_score.

A continuación, se itera sobre los puntajes de similitud calculados. Si el puntaje de similitud para un documento supera el umbral especificado, se verifica si el documento pertenece a la categoría actual en el conjunto de la verdad fundamentada. Si es así, se incrementa el contador de verdaderos positivos, de lo contrario, se incrementa el contador de falsos positivos. Si el puntaje de similitud no supera el umbral, se verifica si el documento debería haber sido clasificado en la categoría actual; si es así, se incrementa el contador de falsos negativos, de lo contrario, se incrementa el contador de verdaderos negativos.

Después de procesar todos los documentos para una categoría, se calcula la precisión y el recall para esa categoría. La precisión se calcula como la proporción de verdaderos positivos sobre el total de predicciones positivas, y el recall como la proporción de verdaderos positivos sobre el total de elementos positivos reales. Se crea una matriz de confusión para la categoría y se añade a la lista de matrices de confusión. Las precisiones y recalls calculados también se añaden a sus respectivas listas.

Finalmente, se combinan todas las matrices de confusión individuales en una matriz de confusión general sumando los valores correspondientes. Esta matriz de confusión general se muestra gráficamente utilizando matplotlib. La función también imprime y devuelve la matriz de confusión general, junto con la precisión y el recall promedio del sistema. La visualización de la matriz de confusión incluye etiquetas para las categorías predichas y reales, y los valores de la matriz se muestran en el gráfico con colores diferenciados para facilitar la interpretación. A continuación, se muestra el código de dicha función:

```
def evaluation(ground_truth, category_vectorizer, X_corpus_vectorized, similarity_method='cosine_similarity',
threshold=0.0):
    confusion_matrices = []
    precisiones = []
    recalls = []

    for cat in list(ground_truth.keys()):
        true_positives = 0
        false_positives = 0
        false_negatives = 0
        true_negatives = 0
        n_cat = 0
        cat_vector = category_vectorizer.transform([cat])

        if similarity_method == 'cosine_similarity':
            scores = cosine_similarity(cat_vector, X_corpus_vectorized)[0]
        elif similarity_method == 'jaccard_similarity':
            scores = jaccard_score(cat_vector.toarray()[0], X_corpus_vectorized.toarray())
        else:
            raise ValueError("Método de similitud no válido. Use 'cosine_similarity' o 'jaccard_similarity'.")

        for idx, score in enumerate(scores):
            filenumber = int(filenames_lemmatized[idx].split('.')[0])
            if score > threshold:
                if filenumber in [int(file) for file in ground_truth[cat]]:
                    true_positives += 1
                    n_cat += 1
                else:
                    false_positives += 1
            else:
```

```
            if filenumber in [int(file) for file in ground_truth[cat]]:
                false_negatives += 1
            else:
                true_negatives += 1

    # Calcular precisión y recall
    precision = true_positives / (true_positives + false_positives) if (true_positives + false_positives) > 0 else 0
    recall = true_positives / (true_positives + false_negatives) if (true_positives + false_negatives) > 0 else 0

    # Crear la matriz de confusión y añadirla a la lista
    cm = np.array([
        [true_positives, false_positives],
        [false_negatives, true_negatives]
    ])
    confusion_matrices.append(cm)

    precisiones.append(precision)
    recalls.append(recall)

    print("--------------------------------")
    print("Categoria: ", cat)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("--------------------------------")


# Combinar todas las matrices de confusión en una matriz general
combined_cm = np.sum(confusion_matrices, axis=0)

print("Matriz de confusión general: ")
print(combined_cm)

# Graficar la matriz de confusión general
plt.figure(figsize=(8, 6))
plt.imshow(combined_cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Matriz de Confusión General')
plt.colorbar()

tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Predicho Positivo', 'Predicho Negativo'], rotation=45)
plt.yticks(tick_marks, ['Actual Positivo', 'Actual Negativo'])

thresh = combined_cm.max() / 2.
for i, j in itertools.product(range(combined_cm.shape[0]), range(combined_cm.shape[1])):
    plt.text(j, i, format(combined_cm[i, j], 'd'),
```

```
            horizontalalignment="center",
            color="white" if combined_cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Etiqueta Verdadera')
    plt.xlabel('Etiqueta Predicha')

    plt.show()

    print("Precision del sistema: ", np.mean(precisiones))
    print("Recall del sistema: ", np.mean(recalls))

    return
```

Ahora para realizar la evaluación de cada sistema simplemente se llama al método previamente definido, cambiando los argumentos de entrada dependiendo el sistema de recuperación de información.

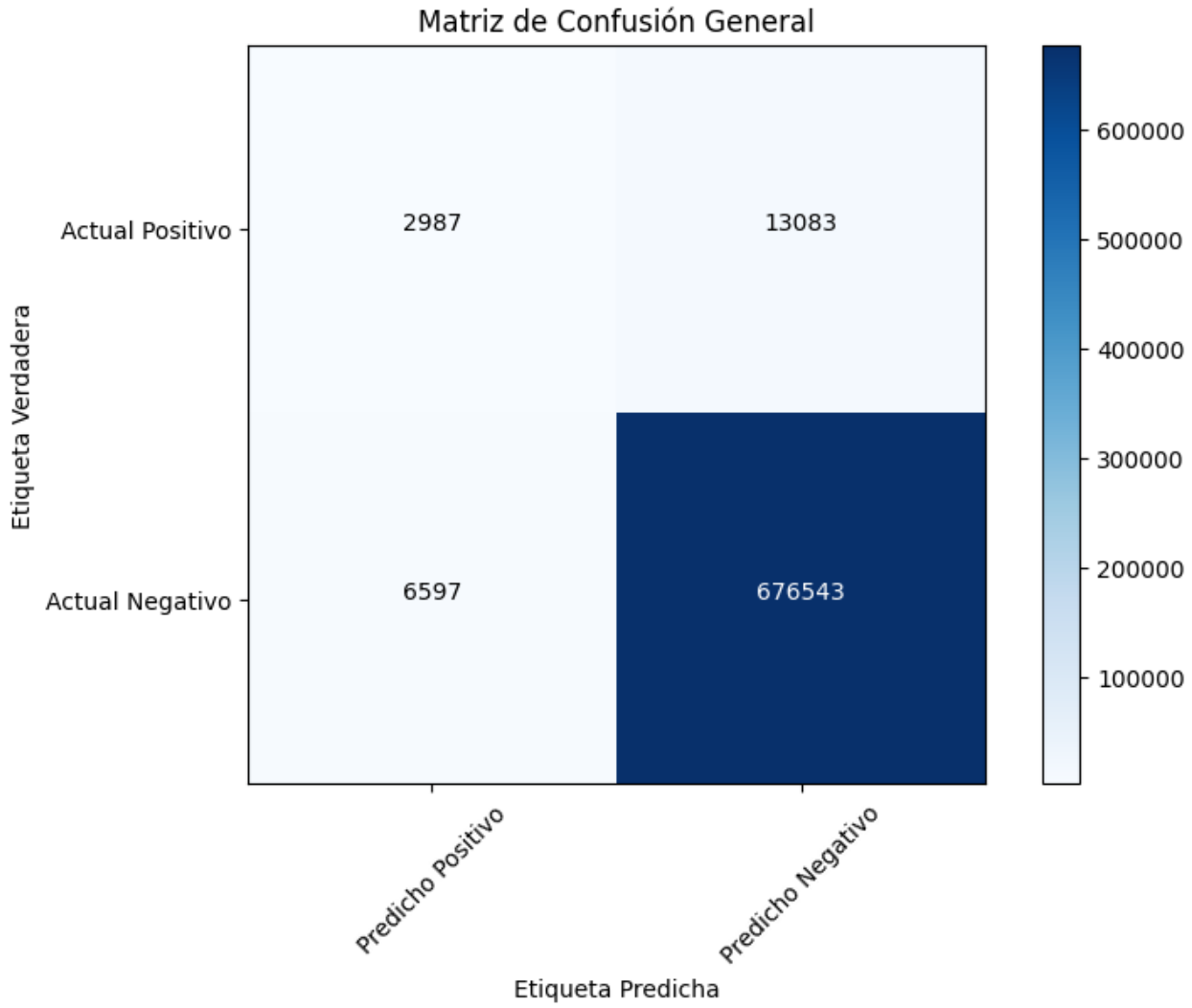
6.1. Jaccard Lemmatized

Para el primer método, se llamó a la función evaluation de la siguiente manera:

```
evaluation(ground_truth, bow_vectorizer_lemmatized, X_bow_lemmatized,
similarity_method='jaccard_similarity', threshold=0)
```

Obteniendo los siguientes resultados:

```
Precision del sistema:  0.3282360013330195
Recall del sistema:  0.6249198600790823
```



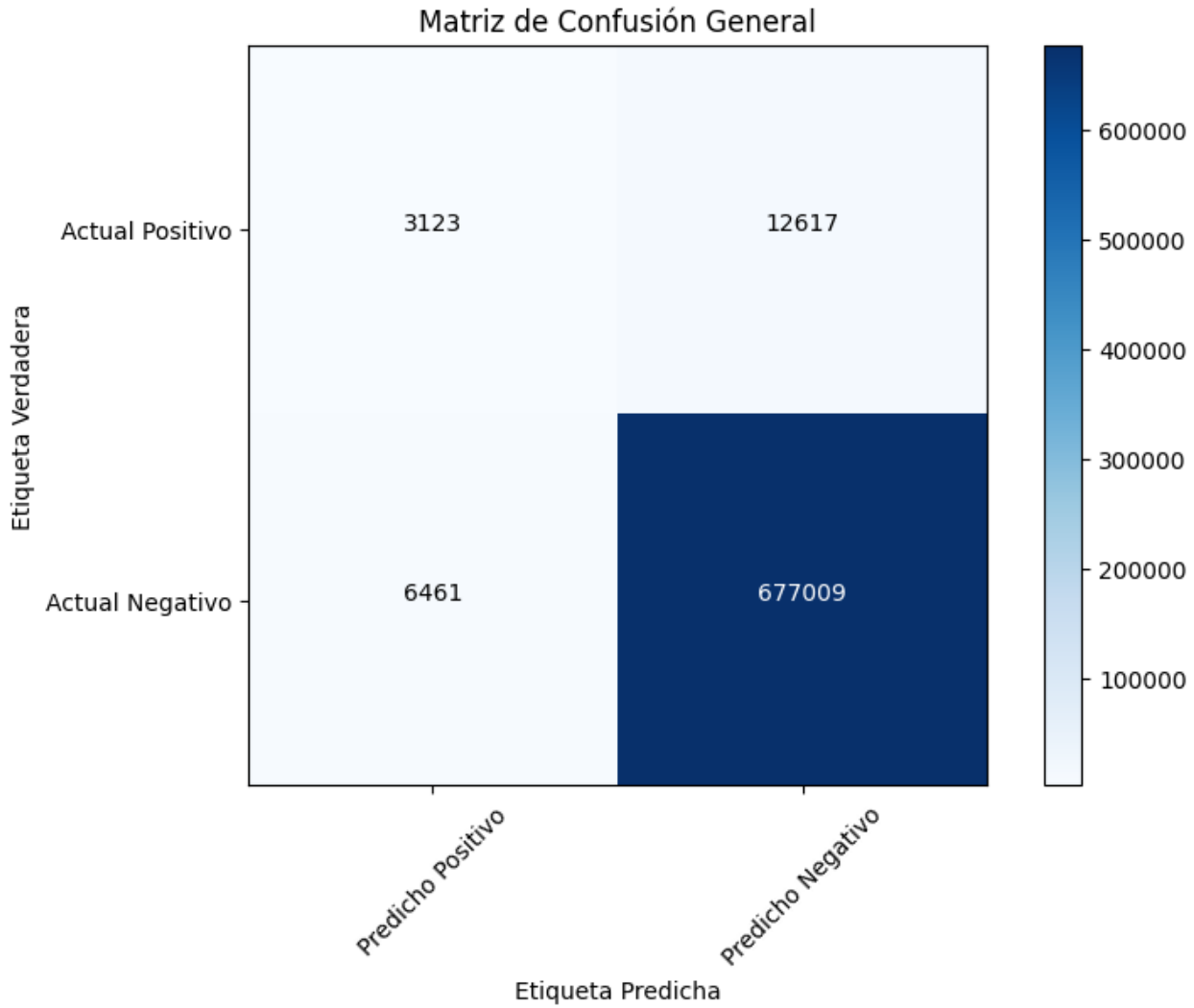
6.2. Jaccard Stemmed

En este caso tenemos la llamada al método:

```
evaluation(ground_truth, bow_vectorizer_stemmed, X_bow_stemmed, similarity_method='jaccard_similarity', threshold=0.001)
```

Aquí hemos modificado el umbral a 0.001 dando como resultado:

```
Precision del sistema:  0.2834349774495125
Recall del sistema:  0.5722688172388687
```

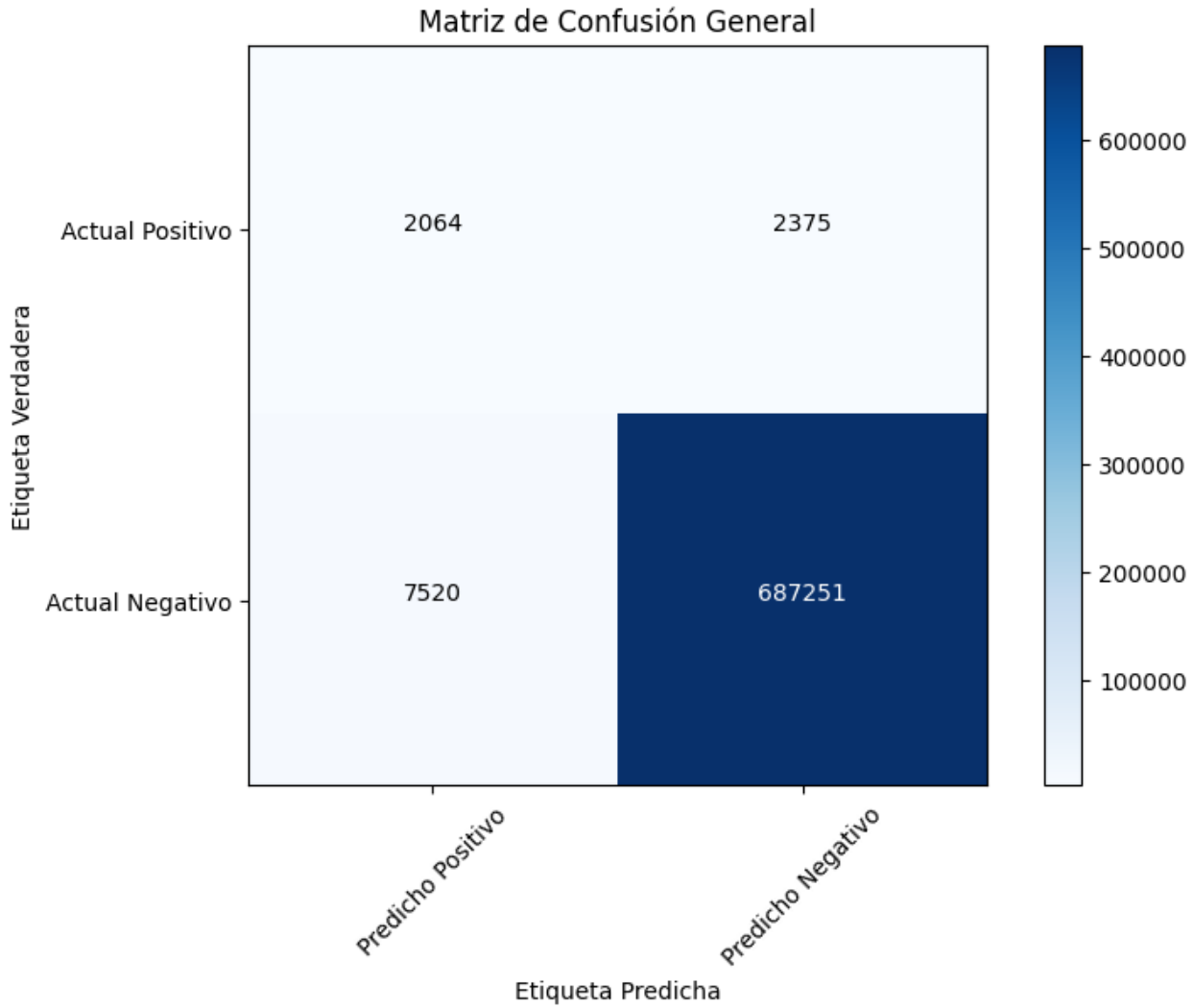


6.3. Cosine Lemmatized

Pasamos al método de similitud de coseno, en el caso lemmatized se hace la siguiente llamada a la función:

```
evaluation(ground_truth, tfidf_vectorizer_lemmatized, X_tfidf_lemmatized,
similarity_method='cosine_similarity', threshold=0.1)
```

```
Precision del sistema:  0.40125870631356714
Recall del sistema:  0.4730903719541281
```



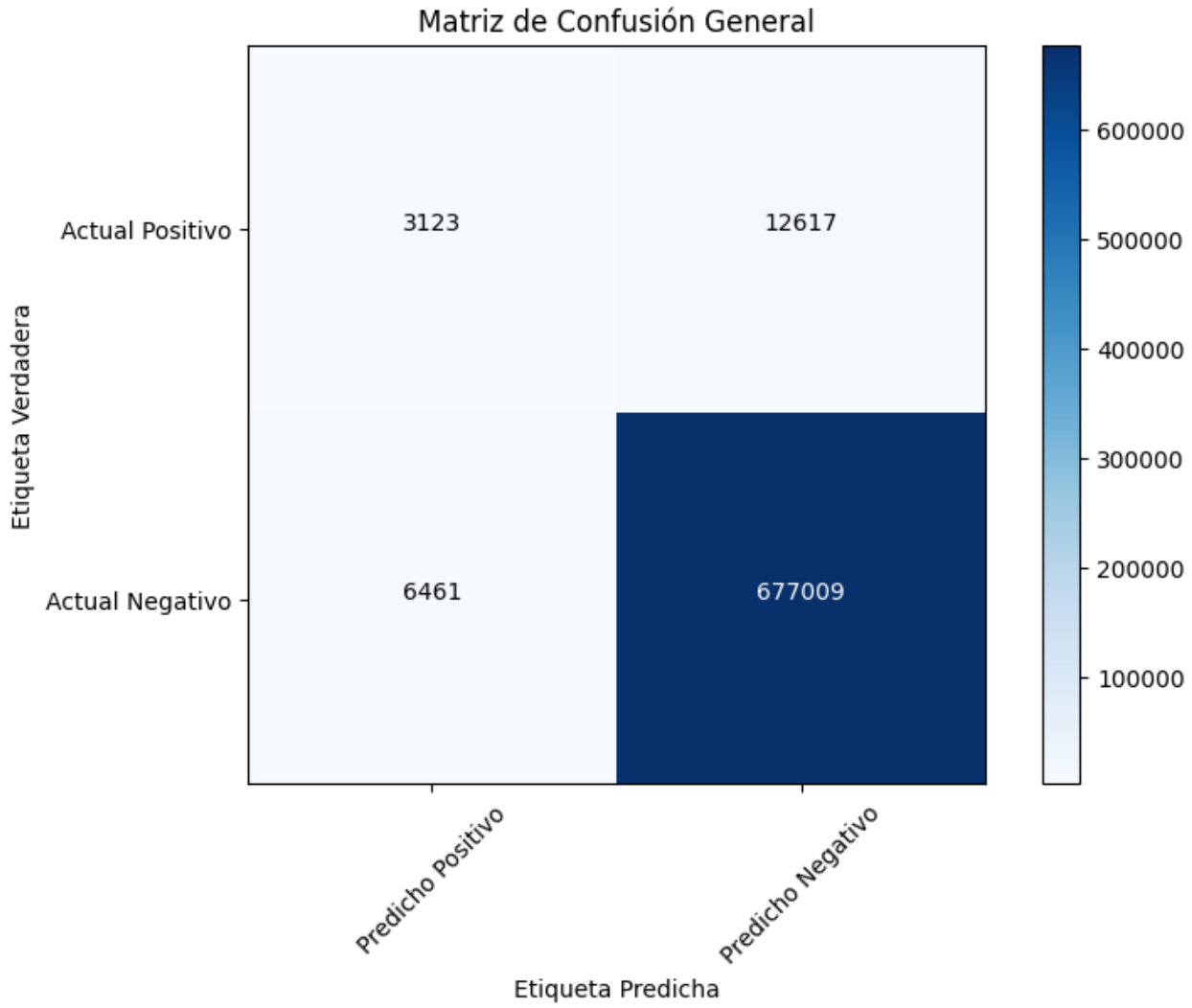
6.4. Cosine Stemmed

En este caso se usó un umbral de 0, es decir todo archivo que tenga algún nivel de similitud será considerado como positivo:

```
evaluation(ground_truth, tfidf_vectorizer_Stemmed, X_tfidf_Stemmed, similarity_method='cosine_similarity', threshold=0)
```

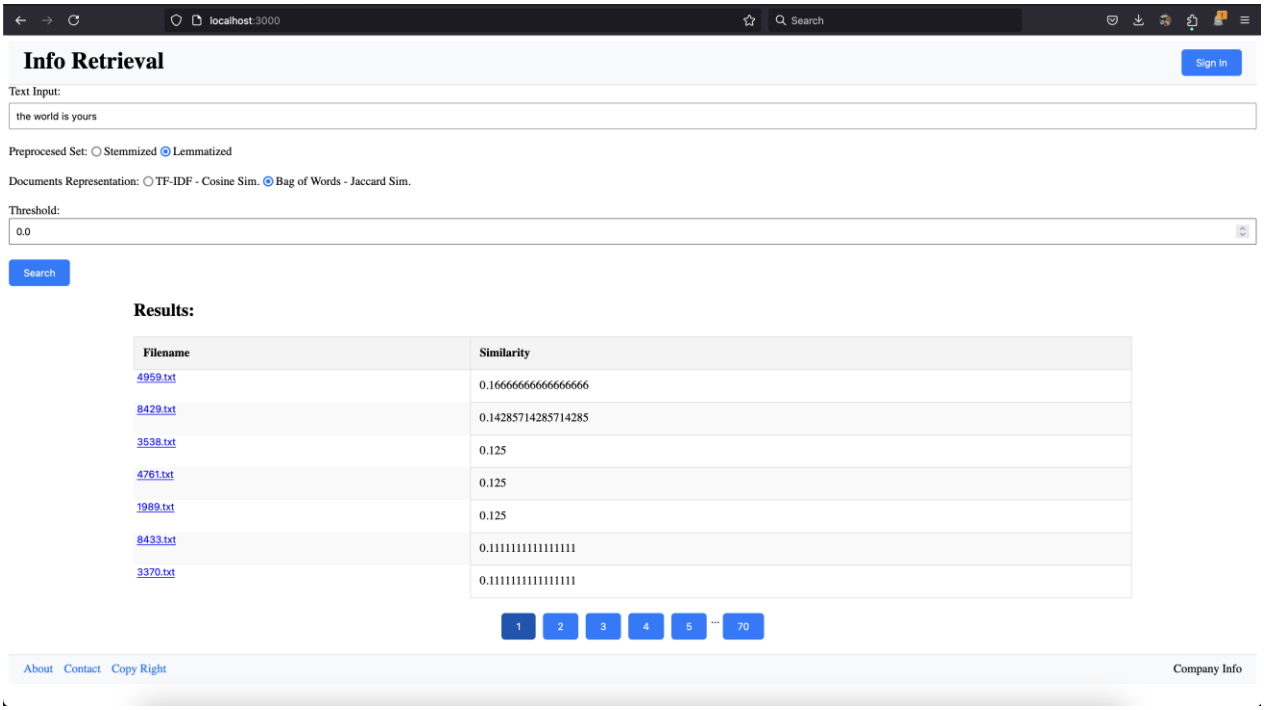
Obteniendo los siguientes resultados:

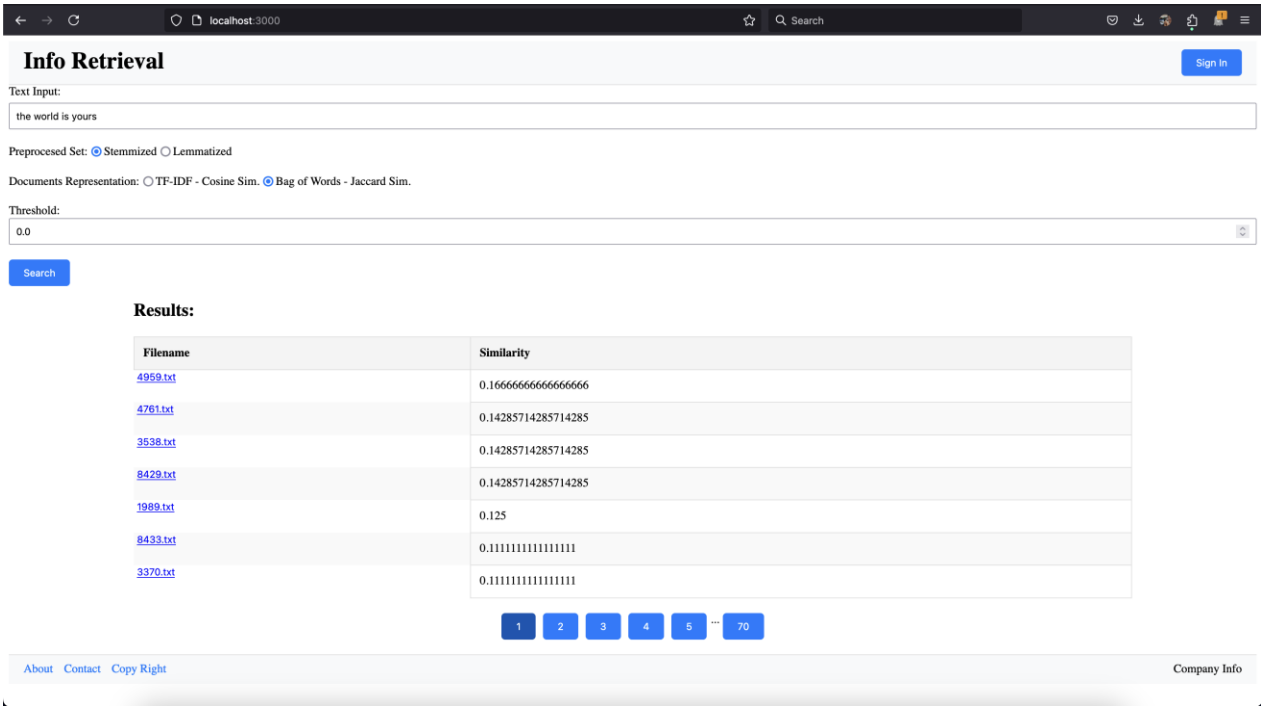
```
Precision del sistema:  0.2834349774495125
Recall del sistema:  0.5722688172388687
```

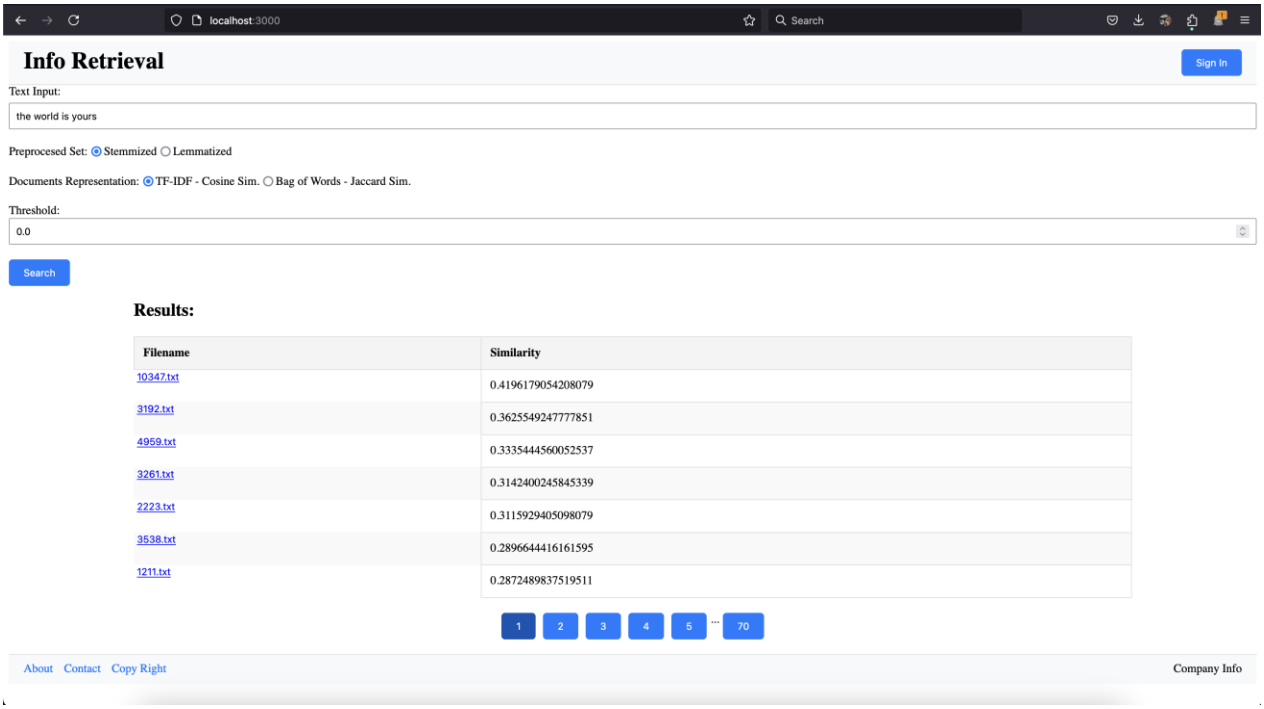


7. Interfaz Web de Usuario

Finalmente, se creó una interfaz web utilizando Flask para el back-end y React para la parte del front-end, permitiendo a los usuarios ingresar consultas y visualizar los resultados de manera clara y ordenada. Se implementaron características adicionales como filtros y opciones de visualización, con las cuales los usuarios pueden elegir si el corpus esté lematizado o stematizado y si quiere realizarlo mediante TF-IDF o mediante Bag of Words, además de poder elegir el umbral de las respuestas e implementar una paginación para poder ver todos los resultados. De la misma forma, se pueden recuperar los textos completos de los documentos enlistado pulsando el nombre de cada documento en la tabla de resultados.


localhost:3000
Search
Info Retrieval
Sign In
Text Input:
the world is yours
Preprocesed Set: Stemmized Lemmatized
Documents Representation: TF-IDF - Cosine Sim. Bag of Words - Jaccard Sim.
Threshold:
0.0
Search
Results:
Filename
Similarity
4959.txt
0.1666666666666666
8429.txt
0.14285714285714285
3538.txt
0.125
4761.txt
0.125
1989.txt
0.125
8433.txt
0.11111111111111
3370.txt
0.11111111111111
1
2
3
4
5
70
About
Contact
Copy Right
Company Info


localhost:3000
Search
Info Retrieval
Sign In
Text Input:
the world is yours
Preprocesed Set: Stemmized Lemmatized
Documents Representation: TF-IDF - Cosine Sim. Bag of Words - Jaccard Sim.
Threshold:
0.0
Search
Results:
Filename
Similarity
4959.txt
0.1666666666666666
4761.txt
0.14285714285714285
3538.txt
0.14285714285714285
8429.txt
0.14285714285714285
1989.txt
0.125
8433.txt
0.11111111111111
3370.txt
0.11111111111111
1
2
3
4
5
70
About
Contact
Copy Right
Company Info



El framework utilizado fue Next.js el cual es un flavor de react. En este caso, el proyecto consume los End-Points del proyecto Back-End. El código se muestra a continuación:

```
inforRetrieval.js   page.js   modal.js   styles.mo
src > repos > inforRetrieval.js > stemmed_tfidf_cosine > bodyString >
1   import axios from "axios";
2
3   const url = axios.create({
4     baseURL: 'http://127.0.0.1:5000'
5   });
6
7   export const getDocumentByName = (name) => {
8     return url.get(`/doc/?doc_id=${name}`)
9       .then((result) => {
10        console.log("from repo text: ", result.data);
11        return result.data;
12      }).catch((error) => {
13        console.log(`repo error: ${error}`);
14        console.log(error.response);
15      });
16  };
```

```
export const lemmatized_tfidf_cosine = (body) => {
  const bodyString = {query: body}
  const headers = {
    'Content-Type': 'application/json',
  }
  return url.post('/lemmatized/tfidf/cosine', bodyString, {headers})
    .then((result) => {
      console.log("from repo");
      console.log(result.data);
      return result.data;
    }).catch((error) => {
      console.log(`repo error: ${error}`);
      console.log(error.response);
    });
};
```

```
export const stemmed_tfidf_cosine = (body) => {
  const bodyString = {query: body}
  const headers = {
    'Content-Type': 'application/json',
  }
  return url.post('/stemmed/tfidf/cosine', bodyString, {headers})
    .then((result) => {
      console.log("from repo");
      console.log(result.data);
      return result.data;
    }).catch((error) => {
      console.log(`repo error: ${error}`);
      console.log(error.response);
    });
};
```

```
export const stemmed_bow_jaccard = (body) => {
  const bodyString = {query: body}
  const headers = {
    'Content-Type': 'application/json',
  }
  return url.post('/stemmed/bow/jaccard', bodyString, {headers})
    .then((result) => {
      console.log("from repo");
      console.log(result.data);
      return result.data;
    }).catch((error) => {
      console.log(`repo error: ${error}`);
      console.log(error.response);
    });
};
```

```
export const lemmatized_bow_jaccard = (body) => {
  const bodyString = {query: body}
  const headers = {
    'Content-Type': 'application/json',
  }
  return url.post('/lemmatized/bow/jaccard', bodyString, {headers})
    .then((result) => {
      console.log("from repo");
      console.log(result.data);
      return result.data;
    }).catch((error) => {
      console.log(`repo error: ${error}`);
      console.log(error.response);
    });
};
```

De esa forma se presenta la capa de consumo de API End-Points. La página principal implementa la lógica de funcionalidad el Front-End:

page.js | modal.js | styles.module.css

src > app > page.js > Home > handleSubmit

```
'use client';
import { useState } from 'react';
import Styles from './styles.module.css';
import Modal from './modal';

import {
  getDocumentByName,
  lemmatized_tfidf_cosine,
  stemmed_tfidf_cosine,
  stemmed_bow_jaccard,
  lemmatized_bow_jaccard,

} from '../repos/inforRetrieval';

function getPaginationRange(currentPage, totalPages, siblingCount = 1) {
  const totalPageNumbers = siblingCount + 5;

  if (totalPages <= totalPageNumbers) {
    return [...Array(totalPages).keys()].map(n => n + 1);
  }

  const leftSiblingIndex = Math.max(currentPage - siblingCount, 1);
  const rightSiblingIndex = Math.min(currentPage + siblingCount, totalPages);

  const shouldShowLeftDots = leftSiblingIndex > 2;
  const shouldShowRightDots = rightSiblingIndex < totalPages - 2;
```

```
const firstPageIndex = 1;
const lastPageIndex = totalPages;

if (!shouldShowLeftDots && shouldShowRightDots) {
  const leftItemCount = 3 + 2 * siblingCount;
  const leftRange = [...Array(leftItemCount).keys()].map(n => n + 1);
  return [...leftRange, '...', totalPages];
}

if (shouldShowLeftDots && !shouldShowRightDots) {
  const rightItemCount = 3 + 2 * siblingCount;
  const rightRange = [...Array(rightItemCount).keys()].map(n => totalPages - rightItemCount + n + 1);
  return [firstPageIndex, '...', ...rightRange];
}

if (shouldShowLeftDots && shouldShowRightDots) {
  const middleRange = [...Array(rightSiblingIndex - leftSiblingIndex + 1).keys()].map(n => leftSiblingIndex + n);
  return [firstPageIndex, '...', ...middleRange, '...', lastPageIndex];
}
}
```

page.js | modal.js | styles.module.css

src > app > page.js > Home > handleSubmit

```
49  export default function Home() {
50    const [threshold, setThreshold] = useState(0.1);
51
52    const [loading, setLoading] = useState(false);
53
54    const [formData, setFormData] = useState({
55      textInput: '',
56      preprocess: '',
57      representation: '',
58      comparison: '',
59    });
60
61    const [result, setResult] = useState([]);
62
63    const [currentPage, setCurrentPage] = useState(1);
64    const resultsPerPage = 7;
65
66    const [modalOpen, setModalOpen] = useState(false);
67    const [modalContent, setModalContent] = useState("");
68    const openModal = async (filename) => {
69      const data = await getDocumentByName(filename)
70      setModalContent(data.content);
71      setModalOpen(true);
72    };
```

```
75  const indexOfLastResult = currentPage * resultsPerPage;
76  const indexOfFirstResult = indexOfLastResult - resultsPerPage;
77  const currentResults = result.slice(indexOfFirstResult, indexOfLastResult);
78
79  const paginate = (pageNumber) => setCurrentPage(pageNumber);
80
81  const handleChange = (e) => {
82    const { name, value } = e.target;
83
84    setFormData(prevFormData => {
85      let updatedFormData = { ...prevFormData, [name]: value };
86
87      if (name === 'representation') {
88        if (value === 'bow') {
89          updatedFormData.comparison = 'jaccard';
90        } else if (value === 'tf_idf') {
91          updatedFormData.comparison = 'cosine';
92        }
93      }
94
95      console.log("options: ", updatedFormData);
96      return updatedFormData;
97    });
98  };
```

```
const handleSetResult = (myResults) => {
  const filteredResults = myResults.filter(item =>
    (item.Jaccard_Similarity && item.Jaccard_Similarity > threshold) ||
    (item.Cosine_Similarity && item.Cosine_Similarity > threshold)
  );
  setResult(filteredResults);
};

const handleSetThreshold = (e) => {
  setThreshold(e.target.value);
}

const handleSubmit = async (e) => {
  e.preventDefault();

  if (formData.preprocess === '' && formData.representation === '' && formData.comparison === '') {
    alert("Please select all options");
    return;
  };

  setLoading(true);
```

```
  try {
    if (formData.preprocess === 'lemmatized' && formData.representation === 'tf_idf' && formData.comparison === 'cosine') {
      handleSetResult(await lemmatized_tfidf_cosine(formData.textInput));
    };

    if (formData.preprocess === 'stemmized' && formData.representation === 'tf_idf' && formData.comparison === 'cosine') {
      handleSetResult(await stemmed_tfidf_cosine(formData.textInput));
    };

    if (formData.preprocess === 'stemmized' && formData.representation === 'bow' && formData.comparison === 'jaccard') {
      handleSetResult(await stemmed_bow_jaccard(formData.textInput));
    };

    if (formData.preprocess === 'lemmatized' && formData.representation === 'bow' && formData.comparison === 'jaccard') {
      handleSetResult(await lemmatized_bow_jaccard(formData.textInput));
    };
  } catch (e) {
    console.log("Error fetching results: ", e);
  } finally {
    setLoading(false);
  }
};
```

```
const totalPages = Math.ceil(result.length / resultsPerPage);
const paginationRange = getPaginationRange(currentPage, totalPages);

return (
  <div>
    <div>
      <form className={Styles.form} onSubmit={handleSubmit}>
        <div className={Styles.form_group}>
          <label htmlFor="textInput" className={Styles.label}>Text Input:</label>
          <input type="text" id="textInput" name="textInput" className={Styles.input_text} onChange={handleChange} value={formData.textInput}><
        </div>
        <div className={Styles.form_group}>
          <label>Preprocesed Set:</label>
          <input type="radio" id="stemmized" name="preprocess" value="stemmized" checked={formData.preprocess === 'stemmized'} onChange={handle
          <label htmlFor="stemmized">Stemmized</label>
          <input type="radio" id="lemmatized" name="preprocess" value="lemmatized" checked={formData.preprocess === 'lemmatized'} onChange={han
          <label htmlFor="lemmatized">Lemmatized</label>
        </div>
        <div className={Styles.form_group}>
          <label>Documents Representation:</label>
          <input type="radio" id="tf_idf" name="representation" value="tf_idf" checked={formData.representation === 'tf_idf'} onChange={handleC
          <label htmlFor="tf_idf">TF-IDF - Cosine Sim.</label>
          <input type="radio" id="bow" name="representation" value="bow" checked={formData.representation === 'bow'} onChange={handleChange}></
          <label htmlFor="bow">Bag of Words - Jaccard Sim.</label>
        </div>
```

```
        <div className={Styles.form_group}>
          <label>Threshold:</label>
          <input type="number" id="threshold" name="threshold" value={threshold} onChange={handleSetThreshold} className={Styles.input_text}
        </div>
        <div className={Styles.form_group}>
          <button type="submit" className={Styles.submit_button} disabled={loading}>Search</button>
        </div>
      </form>
    </div>
    <div className={Styles.tableContainer}>
      <h2>Results:</h2>
      {loading ? (
        <div className={Styles.loading}>Loading...</div>
      ) : (
        <div>
          <table className={Styles.table}>
            <thead>
              <tr>
                <th>Filename</th>
                <th>Similarity</th>
              </tr>
            </thead>
            <tbody>
              {currentResults.map((item, index) => (
                <tr key={index}>
```

```
                <button onClick={() => openModal(item.Filename)} className={Styles.linkButton}>
                  {item.Filename}
                </button>
                <td>{item.Cosine_Similarity || item.Jaccard_Similarity}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    )}
    <div className={Styles.pagination}>
      {paginationRange.map((page, index) => {
        if (page === '...') {
          return <span key={`dots-${index}`} className={Styles.dots}>...</span>;
        }
        return (
          <button
            key={`page-${page}`}
            onClick={() => paginate(page)}
            className={page === currentPage ? Styles.active : ''}
          >
            {page}
          </button>
        );
      })}
    </div>
```

```
      </div>
      <Modal isOpen={modalOpen} onClose={() => setModalOpen(false)} content={modalContent} />
    </div>
  );
}
```

En el código de Front-End podemos ver la implementación de pop-up, los estados de carga, las peticiones web, la asignación de valore cuando se cambian las opciones de búsqueda, el ordenamiento y despliegue de los resultados.

Con esto, se completó el desarrollo del Sistema de Recuperación de Información basado en el corpus Reuters-21578, cubriendo todas las fases descritas en el proyecto. Este sistema no solo es capaz de procesar y vectorizar grandes volúmenes de texto, sino también de proporcionar resultados de búsqueda eficientes y precisos a través de una interfaz web amigable.

Referencias:

[1] Scikit-learn, "sklearn.metrics.pairwise.cosine_similarity," Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html. [Accessed: 17-Jun-2024].
[2] Stack Overflow, "Install Spacy in a Jupyter Notebook," Available: https://stackoverflow.com/questions/64268889/install-spacy-in-a-jupyter-notebook. [Accessed: 17-Jun-2024].
[3] GeeksforGeeks, "How to Calculate Jaccard Similarity in Python," Available: https://www.geeksforgeeks.org/how-to-calculate-jaccard-similarity-in-python/. [Accessed: 17-Jun-2024].
[4] Towards Data Science, "Text Vectorization: Bag of Words (BoW)," Available: https://towardsdatascience.com/text-vectorization-bag-of-words-bow-441d1bfce897. [Accessed: 17-Jun-2024].
[5] Scikit-learn, "sklearn.feature_extraction.text.TfidfVectorizer," Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. [Accessed: 17-Jun-2024].