



ESCUELA POLITÉCNICA NACIONAL



**FACULTAD DE INGENIERÍA EN SISTEMAS
CARRERA DE INGENIERÍA EN COMPUTACIÓN**

(ICCD753) - RECUPERACION DE INFORMACIÓN

GR1CC

Proyecto IB:

**Sistema de Recuperación de Información
basado en Reuters-21578**

INTEGRANTES:

- Paola Aucapiña
- Kevin Maldonado
- Raquel Zumba

DOCENTE:

Iván Carrera, Ph.D.

FECHA DE ENTREGA:

19/06/2024

PERIODO:

2024-A

Contenido

1. Introducción.....	3
2. Fases del Proyecto.....	3
2.1. Adquisición de Datos Objetivo:	3
2.2. Preprocesamiento.....	3
2.3. Representación de Datos en Espacio Vectorial	5
2.3. Representación de Datos en Espacio Vectorial	5
2.4. Indexación	7
2.5. Diseño del Motor de Búsqueda	8
2.6. Evaluación del Sistema.....	11
2.7. Interfaz Web de Usuario.....	16

1. Introducción

El objetivo de este proyecto es diseñar, construir, programar y desplegar un Sistema de Recuperación de Información (SRI) utilizando el corpus Reuters-21578. El proyecto se dividirá en varias fases, que se describen a continuación.

2. Fases del Proyecto

Para facilitar el desarrollo del proyecto, se creó un ambiente virtual de Python, para obtener un entorno aislado y controlado para la gestión de dependencias. Posteriormente, se instaló Flask, un framework ligero de Python para aplicaciones web, utilizando el gestor de paquetes pip. Flask proporciona las funcionalidades necesarias para construir y desplegar aplicaciones web de manera eficiente, facilitando la creación de rutas, gestión de peticiones HTTP y renderización de plantillas, entre otros aspectos clave para el desarrollo web.

2.1. Adquisición de Datos Objetivo:

Objetivo: Obtener y preparar el corpus Reuters-21578.

Tareas:

- Descargar el corpus Reuters-21578.
- Descomprimir y organizar los archivos.
- Documentar el proceso de adquisición de datos.

Se procedió a la obtención y preparación del corpus Reuters-21578 utilizando el archivo comprimido proporcionado previamente, el cual contenía los directorios de prueba (test) y entrenamiento (training), así como los archivos adicionales stopwords.txt y cats.txt.

2.2. Preprocesamiento

Objetivo: Limpiar y preparar los datos para su análisis

Carga de los stopwords

La función leer_stopwords lee un archivo de stopwords, lo convierte en un conjunto que luego lo utilizaremos para realizar el preprocesamiento de los documentos.

```
[4] def leer_stopwords(stopwords_path):  
    #Abre el archivo donde se encuentra las stopwords  
    with open(stopwords_path, 'r') as file:  
        #Lee todo el contenido del archivo y lo devuelve como una cadena de texto.  
        stopwords_list = file.read().splitlines()  
        #Devuelve el conjunto de stopwords  
    return set(stopwords_list)
```

Procesamiento de los documentos

La función **preprocesar_archivos** invoca a la función **leer_stopwords**, luego prepara los documentos, para ello se hace la normalización, tokenización, se eliminan los stopwords y se realiza el stemming. Luego une las palabras preprocesadas en solo documento y guarda los resultados en un DataFrame.

```
def preprocesar_archivos(training_path, stopwords_path):  
  
    # Leer stopwords  
    stopwords_set = leer_stopwords(stopwords_path)  
  
    # Almacenar los datos de los archivos procesados  
    data = []  
  
    # Procesar cada archivo en la carpeta /training  
    for root, dirs, files in os.walk(training_path):  
        for file in files:  
            file_path = os.path.join(root, file)  
            with open(file_path, 'r', encoding='utf-8') as f:  
                texto = f.read()  
  
            # Eliminación de caracteres no deseados y normalización  
            texto = re.sub(r'\W', ' ', texto)  
            texto = re.sub(r'\s+', ' ', texto)  
            texto = texto.lower()  
  
            # Tokenizar el texto  
            tokens = word_tokenize(texto)  
  
            # Eliminar stopwords  
            tokens = [word for word in tokens if word not in stopwords_set]  
  
            # Aplicar stemming utilizando el stemmer declarado fuera de la función  
            tokens = [stemmer.stem(word) for word in tokens]  
  
            # Unir las palabras procesadas en un solo texto  
            texto_procesado = ' '.join(tokens)  
  
            # Guardar la información en data  
            data.append({  
                'Archivo': file,  
                'PATH': file_path,  
                'Texto': texto_procesado  
            })
```

Resultados después del preprocesamiento, como se observa en la imagen tenemos los archivos y el contenido de cada documento.

```
# Mostrar el DataFrame
print(corpus_df.head())
```

Archivo	Texto
0 1	bahia cocoa review shower continu week bahia c...
1 10	comput termin system lt cpml complet sale comp...
2 100	trade bank deposit growth rise slightli zealan...
3 1000	nation amus up viacom lt bid viacom intern lt ...
4 10000	roger lt rog see 1st qtr net significantli rog...

En esta imagen se muestra que el total de documentos que existe en el corpus son 7769 y el total de tokens una vez que se aplicó la limpieza es de 647706.

```
# Obtener el número total de documentos procesados
total_documentos = len(corpus_df)
# Imprimir el resultado
print("Número total de documentos procesados en el corpus:", total_documentos)

# Calcular el número total de tokens en el corpus
total_tokens = corpus_df['Texto'].apply(lambda x: len(x.split())).sum()
# Imprimir el resultado
print("Número total de tokens en el corpus:", total_tokens)
```

Número total de documentos procesados en el corpus: 7769
Número total de tokens en el corpus: 647706

2.3. Representación de Datos en Espacio Vectorial

Objetivo: Convertir los textos en una forma que los algoritmos puedan procesar.

- Bag of Words

La función `create_bow_representation` lo que hace es convertir la colección de documentos en una matriz, luego se crea un dataframe donde las columnas contienen los términos y las filas representan los documentos.

2.3. Representación de Datos en Espacio Vectorial

Objetivo: Convertir los textos en una forma que los algoritmos puedan procesar.

- Bag of Words

La función `create_bow_representation` lo que hace es convertir la colección de documentos en una matriz, luego se crea un dataframe donde las columnas contienen los términos y las filas representan los documentos.

```
[ ] def create_bow_representation(corpus_df):
    #Inicializa el objeto CountVectorizer que se encargara de convertir la colección de documentos de texto en una matriz de tokens
    vectorizer = CountVectorizer(binary=True)
    #Convierte los documentos de texto en una matriz dispersa
    X = vectorizer.fit_transform(corpus_df['Texto'])
    #Convierte la matriz dispersa X a un array denso
    #Crea un nuevo dataframe donde las columnas son los terminos de los documentos
    bow_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
    #Añade una nueva columna al DataFrame bow_df llamada 'Archivo',
    bow_df['Archivo'] = corpus_df['Archivo']
    # Devuelve el dataframe con los terminos y los documentos
    return bow_df
```

En esta imagen se muestra los resultados de la impresión de la representación BOW, como se observa en la parte de las columnas se tiene los términos y en las filas están los documentos. Lo que hace BOW representar la presencia o ausencia de una palabra en un archivo dado.

```
print("Representación Bow:")
bow_df
```

Representación Bow:

	00	000	0000	00000	0009	001	002	003	0037	004	...	zubeidi	zuccherifici	zuckerman	zulia	zurich	zuyuan	zverev	zy	zzzz	Archivo
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	10
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	100
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1000
4	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	10000
...
7764	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	999
7765	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9992
7766	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9993
7767	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9994
7768	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9995

7769 rows x 19448 columns

- TF-IDF

La función **create_tfidf_representation** calcula los valores TF-IDF para cada término en cada documento utilizando **TfidfVectorizer**, y devuelve un nuevo DataFrame como se observa en la siguiente imagen donde cada fila representa un documento con valores TF-IDF.

```
[14] def create_tfidf_representation(corpus_df):
    #Inicializacion del objeto TfidfVectorizer
    tfidf_vectorizer = TfidfVectorizer()
    # Ajusta y transforma el texto en una matriz dispersa de pesos TF-IDF
    X_tfidf = tfidf_vectorizer.fit_transform(corpus_df['Texto'])
    # Crea un DataFrame con los valores TF-IDF
    tfidf_df = pd.DataFrame(X_tfidf.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
    tfidf_df['Archivo'] = corpus_df['Archivo']
    return tfidf_df
```

En esta imagen se muestra los resultados de la impresión de la representación TF-IDF, como se observa en la parte de las columnas

se tiene los términos y en las filas están los documentos. Lo que hace TF-IDF es mostrar el peso de cada uno de los términos los cuales nos van a ayudar a determinar la importancia.

```
print("Representación Bow:")
bow_df
```

Representación Bow:

	00	000	0000	00000	0009	001	002	003	0037	004	...	zubeidi	zuccherifici	zuckerman	zulia	zurich	zuyuan	zverev	zy	zzzz	Archivo
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	10
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	100
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1000
4	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	10000
...
7764	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	999
7765	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9992
7766	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9993
7767	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9994
7768	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	9995

7769 rows x 19448 columns

2.4. Indexación

Objetivo: Crear un índice que permita búsquedas eficientes

Índice Invertido

La función **calcular_indice_invertido** construye un índice invertido a partir de la representación BoW o TF-IDF. Este índice nos permite buscar rápidamente qué términos están presentes en qué documento.

```
[17] #Funcion que toma como argumento un DataFrame que contiene la representacion
def calcular_indice_invertido(df):
    # Creamos un diccionario vacio que almacenará el índice invertido.
    indice_invertido = {}
    # Iteramos por fila, donde cada fila representa un documento
    for index, row in df.iterrows():
        archivo = row['Archivo']
        for termino, valor in row.drop('Archivo').items():
            if valor > 0:
                if termino not in indice_invertido:
                    indice_invertido[termino] = []
                indice_invertido[termino].append(archivo)

    return indice_invertido
```

Índice invertido para BOW

En esta imagen se puede observar en que documentos se encuentran cada termino.

Termino		Archivos
0	06	[1, 10146, 10233, 10297, 10319, 10376, 10537, ...]
1	13	[1, 100, 10008, 1003, 10032, 10037, 10046, 100...
2	15	[1, 1003, 10057, 10080, 10083, 10089, 1011, 10...
3	155	[1, 10617, 1184, 12098, 12429, 12463, 12787, 1...
4	1986	[1, 100, 10011, 10023, 10025, 10041, 10046, 10...
...
19442	additivi	[9984]
19443	genecor	[9984]
19444	genencor	[9984]
19445	genentech	[9984]
19446	km	[9992]
19447 rows x 2 columns		

Índice invertido para TF-IDF

La tabla muestra el índice invertido para TF-IDF se puede observar que cada termino se encuentra en los respectivos documentos.

Termino		Archivos
0	06	[1.0, 10146.0, 10233.0, 10297.0, 10319.0, 1037...
1	13	[1.0, 100.0, 10008.0, 1003.0, 10032.0, 10037.0...
2	15	[1.0, 1003.0, 10057.0, 10080.0, 10083.0, 10089...
3	155	[1.0, 10617.0, 1184.0, 12098.0, 12429.0, 12463...
4	1986	[1.0, 100.0, 10011.0, 10023.0, 10025.0, 10041....
...
19442	additivi	[9984.0]
19443	genecor	[9984.0]
19444	genencor	[9984.0]
19445	genentech	[9984.0]
19446	km	[9992.0]
19447 rows x 2 columns		

2.5. Diseño del Motor de Búsqueda

Objetivo: Implementar la funcionalidad de búsqueda.


```
def preprocesar_consulta(consulta, stopwords_set):  
    # Eliminación de caracteres no deseados y normalización  
    texto = re.sub(r'\W', ' ', consulta) # Corregido: usar 'consulta' en lugar de 'texto'  
    texto = re.sub(r'\s+', ' ', texto)  
    texto = texto.lower()  
  
    # Tokenizar el texto  
    tokens = word_tokenize(texto) # Corregido: usar 'texto' en lugar de 'consulta'  
  
    # Eliminar stopwords  
    tokens = [word for word in tokens if word not in stopwords_set]  
  
    # Aplicar stemming utilizando el stemmer declarado fuera de la función  
    tokens = [stemmer.stem(word) for word in tokens]  
  
    # Unir las palabras procesadas en un solo texto  
    texto_procesado = ' '.join(tokens)  
  
    return texto_procesado
```

Motor de búsqueda

La función `motor_busqueda_u`` realiza una búsqueda de consulta en un corpus representado por un DataFrame `corpus_df``, utilizando dos enfoques de representación: BoW (Bag of Words) y TF-IDF. Primero, carga y procesa un conjunto de stopwords desde un archivo especificado por `stopwords_path``. Luego, genera representaciones BoW y TF-IDF del corpus utilizando funciones auxiliares como `create_bow_representation`` y `create_tfidf_representation``. Posteriormente, preprocesa la consulta eliminando stopwords y otros pasos de limpieza mediante la función `preprocesar_consulta``. La consulta procesada se vectoriza utilizando los vectorizadores BoW y TF-IDF previamente entrenados. A continuación, se calculan las similitudes coseno entre la consulta vectorizada y los documentos del corpus para ambas representaciones. Se aplican umbrales de similitud (`umbral``) para filtrar los resultados y se obtienen las mejores coincidencias ordenadas por similitud descendente. Finalmente, la función devuelve dos listas de hasta diez tuplas cada una, conteniendo los nombres de archivo y sus similitudes más altas según los enfoques BoW y TF-IDF respectivamente.

```
def motor_busqueda_u(consulta, corpus_df, stopwords_path, umbral=0.2):
    """Realiza la búsqueda de la consulta utilizando las representaciones Bow y TF-IDF."""
    stopwords_set = set(Leer_stopwords(stopwords_path)) # Convertir a conjunto para búsquedas rápidas

    # Crear representaciones Bow y TF-IDF
    bow_df = create_bow_representation(corpus_df)
    tfidf_df = create_tfidf_representation(corpus_df)

    # Crear vectorizadores
    bow_vectorizer = CountVectorizer(binary=True)
    bow_vectorizer.fit(corpus_df['Texto'])
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_vectorizer.fit(corpus_df['Texto'])

    # Preprocesar la consulta
    consulta_procesada = preprocesar_consulta(consulta, stopwords_set)

    # Vectorizar la consulta
    query_vector_bow = vectorizar_consulta(consulta_procesada, bow_vectorizer)
    query_vector_tfidf = vectorizar_consulta(consulta_procesada, tfidf_vectorizer)

    # Matrices de documentos (excluyendo la columna 'Archivo')
    document_matrix_bow = bow_df.drop(columns=['Archivo'])
    document_matrix_tfidf = tfidf_df.drop(columns=['Archivo'])

    # Calcular similitudes
    similitudes_bow = calcular_similitud_coseno(query_vector_bow, document_matrix_bow)
    similitudes_tfidf = calcular_similitud_coseno(query_vector_tfidf, document_matrix_tfidf)

    # Crear resultados ordenados aplicando el umbral
    resultados_bow = [(archivo, similitud) for archivo, similitud in zip(bow_df['Archivo'], similitudes_bow) if similitud >= umbral]
    resultados_tfidf = [(archivo, similitud) for archivo, similitud in zip(tfidf_df['Archivo'], similitudes_tfidf) if similitud >= umbral]

    # Ordenar los resultados por similitud (en orden descendente)
    resultados_ordenados_bow = sorted(resultados_bow, key=lambda x: x[1], reverse=True)[:10]
    resultados_ordenados_tfidf = sorted(resultados_tfidf, key=lambda x: x[1], reverse=True)[:10]

    return resultados_ordenados_bow, resultados_ordenados_tfidf
```

Aquí vamos a realizar una consulta *Showers continued throughout the week in the Bahia cocoa zone*.

Estos resultados muestran los documentos más relevantes para la consulta ***Showers continued throughout the week in the Bahia cocoa zone*** utilizando la representación Bag of Words . Cada entrada incluye el nombre del archivo y su similitud coseno con la consulta, donde una similitud más alta indica mayor relevancia del documento para la consulta. Así también, los resultados con TF-IDF donde pondera los términos según su importancia relativa en cada documento y en el corpus total, ofreciendo una medida más precisa de la relevancia. Pues como conclusión se puede decir que la respectiva consulta se encuentra en el documento 1 por lo tanto hay mayor similitud.

Resultados de la consulta 'Showers continued throughout the week in the Bahia cocoa zone' usando Bow

=====

1. Archivo: 3190 - Similitud: 0.2462

Resultados de la consulta 'Showers continued throughout the week in the Bahia cocoa zone' usando TF-IDF

=====

1. Archivo: 1 - Similitud: 0.3187
2. Archivo: 10505 - Similitud: 0.2930
3. Archivo: 10506 - Similitud: 0.2485
4. Archivo: 5258 - Similitud: 0.2409
5. Archivo: 10586 - Similitud: 0.2307
6. Archivo: 9953 - Similitud: 0.2211
7. Archivo: 10760 - Similitud: 0.2203
8. Archivo: 9450 - Similitud: 0.2139
9. Archivo: 13271 - Similitud: 0.2107

2.6. Evaluación del Sistema

Objetivo: Medir la efectividad del sistema.

La función obtener_indice_invertido_categoria:

- * Inicializa un diccionario vacío
- * Divide las líneas en partes y extrae el nombre del documento y las categorías

Luego se muestra las categorías y en que documentos se encuentran asociadas dicha categoría.

```
[28] def obtener_indice_invertido_categoria(lines):  
    # Inicialización de un diccionario vacío para almacenar el índice invertido  
    indice = {}  
    # Itera sobre cada línea  
    for line in lines:  
        # Verifica si la línea comienza con 'training/'  
        if line.startswith('training/'):  
            parts = line.strip().split()  
            if len(parts) > 1:  
                documento = parts[0].split('/')[1]  
                categorias = parts[1:]  
                # Itera sobre cada categoría  
                for categoria in categorias:  
                    # Si la categoría ya está en el índice, añade el documento a la lista existente  
                    if categoria in indice:  
                        indice[categoria].append(documento)  
                    # Si la categoría no está en el índice, crea una nueva entrada con el documento en una lista  
                    else:  
                        indice[categoria] = [documento]  
    # Devuelve el índice invertido  
    return indice  
  
# Ejemplo :  
with open(cats_path, 'r') as file:  
    indice_invertido = obtener_indice_invertido_categoria(file.readlines())  
  
# Imprimir el índice invertido  
print("Índice Invertido cat :")  
for categoria, documentos in indice_invertido.items():
```

✓ 0 s se ejecutó 4:47 a.m.

En la tabla presentada se muestran todas las categorías junto con los documentos en los que se encuentran asociadas. Este índice invertido de categorías se utilizará como el conjunto de datos de referencia (ground truth) para el sistema.

Índice Invertido Categorías como DataFrame:

	Categoría	Documentos
0	cocoa	[1, 275, 2521, 3190, 3225, 3310, 4147, 4470, 5...
1	sorghum	[5, 6, 97, 1131, 1369, 1582, 1843, 3981, 5467,...
2	oat	[5, 97, 197, 417, 855, 1405, 8759, 13852]
3	barley	[5, 395, 501, 1067, 1652, 1970, 2044, 2171, 21...
4	corn	[5, 6, 57, 97, 193, 197, 235, 327, 395, 501, 5...
...
85	rand	[7043, 9336]
86	coconut	[8112, 10712, 10720, 11836]
87	castor-oil	[10300]
88	nkr	[11972]
89	sun-meal	[12361]

90 rows x 2 columns

Evaluación del sistema

La función **evaluate** nos permite evaluar la efectividad del sistema, para ello toma tres parametros categoría, results y el ground_truth, luego obtiene los documentos relevantes para esa categoría y compara con los resultados de la búsqueda con esos documentos relevantes para calcular tres métricas de evaluación:

```
[38] #FUNCION DE EVALUACION
from sklearn.metrics import precision_score, recall_score, f1_score

# Evalúa la efectividad de un sistema de búsqueda para una categoría dada
def evaluate(category, results, ground_truth):
    # Obtiene la lista de documentos relevantes para la categoría desde el ground_truth.
    relevant_docs = ground_truth.get(category, [])
    # Crea una lista y_true donde 1 indica que el documento es relevante y 0 que no lo es
    y_true = [1 if str(doc_id) in relevant_docs else 0 for doc_id, _ in results]
    # Crea una lista y_pred donde se asume que todos los documentos en results son relevantes 1
    y_pred = [1] * len(results)

    # Si no hay documentos relevantes para la categoría, se imprime un mensaje y se devuelven métricas 0.
    if not relevant_docs:
        print(f"No se encontraron documentos relevantes para la categoría '{category}'")
        return 0, 0, 0 # Evitar división por cero si no hay documentos relevantes

    # Calcular precisión, recall y F1-score utilizando y_true y y_pred
    precision = precision_score(y_true, y_pred, zero_division=0)
    recall = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)

    return precision, recall, f1
```

Precisión y recall por cada query

Para calcular la Precisión, recall y F1 para cada query, creamos dos listas para almacenar los resultados. Después hacemos un for sobre las categorías predefinidas.

Utilizamos la función de motor de búsqueda para buscar los documentos relevantes utilizando BoW y TD-IDF

Luego se evalúa los resultados utilizando la función ****evaluate**** el cual calcula precisión, recall y F1 comparando los resultados con el ground Truth.

Guardamos los resultados en rows_bow y rows_tfidf

Por último, imprimimos los resultados.

```
# Crear listas vacías para almacenar los resultados
rows_bow = []
rows_tfidf = []

# Dentro del bucle
for categoria in categorias:
    # Obtener documentos asociados a la categoría
    documentos = indice_invertido_cat_df.loc[indice_invertido_cat_df['Categoría'] == categoria, 'Documentos'].iloc[0]

    # Obtener resultados de búsqueda utilizando tu motor de búsqueda
    resultados_bow, resultados_tfidf = motor_busqueda(categoria, corpus_df, stopwords_path)

    # Evaluar resultados para BoW
    precision_bow, recall_bow, f1_bow = evaluate(categoria, resultados_bow, ground_truth)

    # Evaluar resultados para TF-IDF
    precision_tfidf, recall_tfidf, f1_tfidf = evaluate(categoria, resultados_tfidf, ground_truth)

    # Agregar los resultados de BoW a la lista de filas
    rows_bow.append({'categoria': categoria, 'precision': precision_bow, 'recall': recall_bow, 'f1': f1_bow})

    # Agregar los resultados de TF-IDF a la lista de filas
    rows_tfidf.append({'categoria': categoria, 'precision': precision_tfidf, 'recall': recall_tfidf, 'f1': f1_tfidf})

# Crear DataFrames a partir de las listas de filas
df_resultados_bow = pd.DataFrame(rows_bow)
df_resultados_tfidf = pd.DataFrame(rows_tfidf)
```

En esta tabla se muestra la precisión, recall y f1 para cada categoría utilizando BoW y TD-IDF.

Como se puede observar en la imagen en general, muchas **categorías** muestran un recall perfecto de 1.0, lo que indica que el sistema logra recuperar todos los documentos relevantes. Sin embargo, la precisión varía significativamente entre categorías. Algunas categorías logran una precisión perfecta de 1.0, mientras que otras tienen una precisión baja, indicando que solo una fracción de los documentos recuperados son relevantes.

Métricas de evaluación del SRI con un umbral de 0.2 por cada query/categoría
Resultados para Bow con similitud coseno:

	categoria	precision	recall	f1
0	groundnut	0.0	0.0	0.000000
1	ship	0.9	1.0	0.947368
2	cocoa	1.0	1.0	1.000000
3	l-cattle	0.0	0.0	0.000000
4	interest	0.3	1.0	0.461538
..
85	wheat	1.0	1.0	1.000000
86	meal-feed	0.5	1.0	0.666667
87	housing	0.1	1.0	0.181818
88	cpu	0.0	0.0	0.000000
89	palladium	0.0	0.0	0.000000

[90 rows x 4 columns]

Resultados para TF-IDF con similitud coseno:

	categoria	precision	recall	f1
0	groundnut	0.0	0.0	0.000000
1	ship	1.0	1.0	1.000000
2	cocoa	1.0	1.0	1.000000
3	l-cattle	0.4	1.0	0.571429
4	interest	0.6	1.0	0.750000
..
85	wheat	1.0	1.0	1.000000
86	meal-feed	0.8	1.0	0.888889
87	housing	0.3	1.0	0.461538
88	cpu	0.0	0.0	0.000000
89	palladium	1.0	1.0	1.000000

[90 rows x 4 columns]

Precisión y un recall de todo el sistema

Para evaluar el promedio del sistema de recall, precisión y f1 utilizando las técnicas BOW Y TF-IDF utilizamos las funciones:

df_resultados_bow: Contiene los valores de recall, precisión y f1 calculados utilizando la técnica BoW para varias categorías de documentos.

df_resultados_tfidf: Contiene los valores de recall, precisión y f1 calculados utilizando la técnica TF-IDF para las mismas categorías de documentos.

```
print ("Promedio de las metricas de evaluación del SRI con un umbral de 0.2")  
print(df_resultados_umbral)
```

```
Promedio de las metricas de evaluación del SRI con un umbral de 0.2  
      Recall Precision      F1  
0  Bow  0.500000  0.318735  0.361363  
1  TF-IDF 0.722222  0.505794  0.566358
```

Se pudo observar que los resultados muestran que el enfoque TF-IDF con un umbral de 0.2 tiene un desempeño superior en términos de recall, precisión y F1-score en comparación con el enfoque BoW bajo las mismas

condiciones de evaluación en el Sistema de Recuperación de Información (SRI).

2.7. Interfaz Web de Usuario

Objetivo: Crear una interfaz para interactuar con el sistema.



El diseño y desarrollo de la Interfaz Web de Usuario para nuestro sistema de recuperación se fundamenta en la premisa de ofrecer una experiencia óptima y eficiente. Orientada a facilitar la interacción intuitiva con la plataforma, esta interfaz ha sido meticulosamente diseñada para permitir a los usuarios navegar y recuperar información de manera fluida y precisa.