ESCUELA POLITÉCNICA NACIONAL

RECUPERACIÓN DE LA INFORMACION

GRUPO

PROYECTO BI - DOCUMENTACIÓN

INTEGRANTES:

RONNY ANDERSON AMORES GUEVARA

BYRON MARCELO ORTIZ PITANDO

DOCENTE:

IVAN CARRERA, PH.D.

GR1CC

FECHA DE ENTREGA: 18/06/2024

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

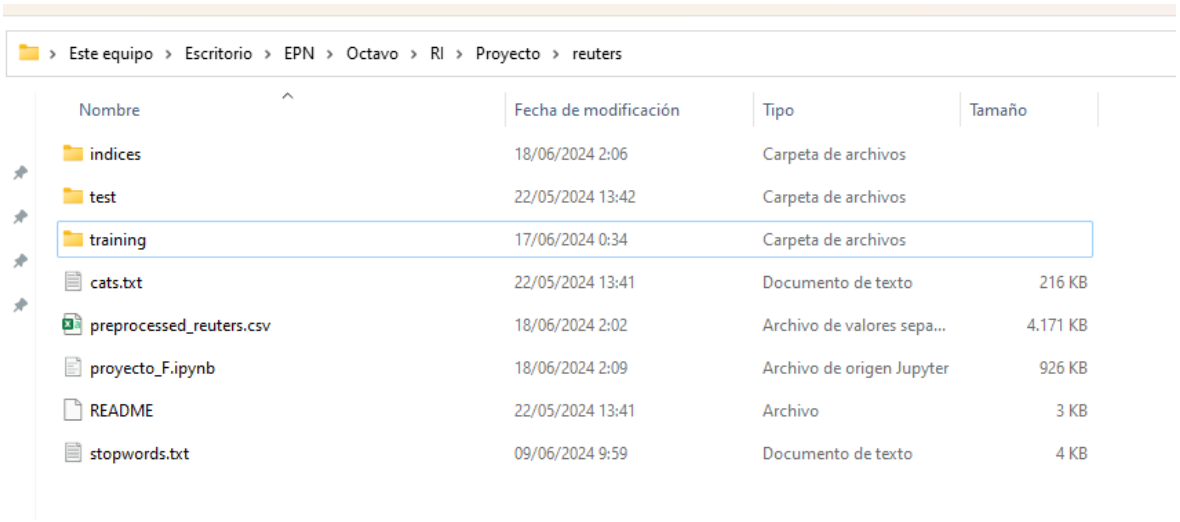
2.1. Adquisición de Datos

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.

Descarga del corpus Reuters-21578



Conversión del corpus a tipo de archivos txt

```python
import os

# Ruta a la carpeta donde se encuentran los archivos del corpus
corpus_dir = r'C:\Users\Ronny Amores\Desktop\EPN\Octavo\RI\Proyecto\reuters\training'

# Ruta de la carpeta de destino para los archivos .txt
output_dir = os.path.join(corpus_dir, 'training_txt')

def convert_to_txt(corpus_dir, output_dir):
    # Crear la carpeta de destino si no existe
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for filename in os.listdir(corpus_dir):
        file_path = os.path.join(corpus_dir, filename)
        if os.path.isfile(file_path):  # Verificar si es un archivo
            with open(file_path, 'r', encoding='latin-1') as file:
                content = file.read()
                # Crear un nuevo archivo con extensión .txt y escribir el contenido
                txt_file_path = os.path.join(output_dir, os.path.splitext(filename)[0] + '.txt')
                with open(txt_file_path, 'w', encoding='utf-8') as txt_file:
                    txt_file.write(content)

# Ejecutar la conversión
convert_to_txt(corpus_dir, output_dir)

print("La conversión de archivos se ha completado.")
```

2.2. Preprocesamiento

Objetivo: Limpiar y preparar los datos para su análisis.

Tareas:

- Extraer el contenido relevante de los documentos.
- Realizar limpieza de datos: eliminación de caracteres no deseados, normalización de texto, etc.
- Tokenización: dividir el texto en palabras o tokens.
- Eliminar stop words y aplicar stemming o lematización.
- Documentar cada paso del preprocesamiento.

```python
# Cargar las stop words desde un archivo
stop_words_file = os.path.join(stopword_dir, 'stopwords.txt')
with open(stop_words_file, 'r') as file:
    stop_words = set(file.read().splitlines())

# Inicializar el lematizador de WordNet
lemmatizer = WordNetLemmatizer()

def preprocess_document(doc):
    """
    Función para preprocesar un documento.
    Incluye eliminación de caracteres no deseados, normalización, tokenización,
    eliminación de stop words y lematización.
    """
    # Eliminar caracteres no deseados y normalizar texto a minúsculas
    doc = re.sub(r'\W', ' ', doc)
    doc = doc.lower()

    # Tokenización del documento
    tokens = word_tokenize(doc)

    # Eliminar stop words de los tokens
    tokens = [token for token in tokens if token not in stop_words]

    # Aplicar lematización a los tokens
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Unir tokens procesados en una cadena de texto
    return ' '.join(tokens)
```

```python
def preprocess_corpus(corpus_dir):
    """
    Función para preprocesar todos los documentos en un directorio.
    Lee cada archivo de texto, lo preprocesa y guarda el resultado en una lista de diccionarios.
    """
    preprocessed_documents = []  # Lista para guardar documentos preprocesados

    # Recorrer todos los archivos en el directorio del corpus
    for filename in os.listdir(corpus_dir):
        if filename.endswith('.txt'):  # Verificar si es un archivo de texto
            file_path = os.path.join(corpus_dir, filename)  # Ruta completa al archivo
            with open(file_path, 'r', encoding='utf-8') as file:
                content = file.read()  # Leer contenido del archivo
                preprocessed_text = preprocess_document(content)  # Preprocesar el contenido
                preprocessed_documents.append({
                    'filename': filename,  # Guardar nombre del archivo
                    'content': preprocessed_text  # Guardar contenido preprocesado
                })

    return preprocessed_documents

# Ejecutar el preprocesamiento del corpus
preprocessed_documents = preprocess_corpus(corpus_dir)

# Convertir la lista de documentos preprocesados a un DataFrame de pandas
df = pd.DataFrame(preprocessed_documents)

# Guardar el DataFrame en un archivo CSV
df.to_csv('preprocessed_reuters.csv', index=False)

print("Preprocesamiento completado y guardado en 'preprocessed_reuters.csv'")
```

- **Análisis y Verificación del Archivo Preprocesado**

  Se realiza la carga y verificación del archivo CSV que contiene los documentos preprocesados. Se incluyen pasos para

```python
import pandas as pd

# Cargar el archivo preprocesado
df_preprocessed = pd.read_csv('preprocessed_reuters.csv')

# Mostrar las primeras filas del DataFrame
print(df_preprocessed.head())

# Verificar si hay filas con contenido vacío
empty_content = df_preprocessed[df_preprocessed['content'].isna()]
print(f"Filas con contenido vacío: {len(empty_content)}")

# Verificar el tamaño del DataFrame
print(f"Tamaño del DataFrame: {df_preprocessed.shape}")

# Verificar algunas estadísticas del contenido
print(df_preprocessed['content'].describe())

# Verificar ejemplos de contenido
print("Ejemplos de contenido preprocesado:")
for i in range(5):
    print(f"Documento {i+1}:")
    print(df_preprocessed.iloc[i]['content'])
    print("---")
```

[40]

```
     filename                                            content
0      1.txt  bahia cocoa review shower continued week bahia...
1     10.txt  computer terminal system lt cpml completes sal...
2    100.txt  trading bank deposit growth rise slightly zeal...
3   1000.txt  national amusement ups viacom lt bid viacom in...
4  10000.txt  rogers lt rog see 1st qtr net significantly ro...
Filas con contenido vacío: 0
Tamaño del DataFrame: (7769, 2)
count                        7769
unique                       7647
top       26 feb 1987 26 feb 1987
freq                            7
Name: content, dtype: object
Ejemplos de contenido preprocesado:
Documento 1:
bahia cocoa review shower continued week bahia cocoa zone alleviating drou
---
```

2.3. Representación de Datos en Espacio Vectorial

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

Tareas:

- Utilizar técnicas como Bag of Words (BoW) y TF-IDF para vectorizar el texto.
- Evaluar las diferentes técnicas de vectorización.
- Documentar los métodos y resultados obtenidos.

Realizamos la vectorización de documentos preprocesados utilizando diferentes técnicas (Bag of Words, TF-IDF y Bag of Words binario) y guarda los índices invertidos resultantes en archivos JSON. Estos índices invertidos son útiles para la recuperación eficiente de información en sistemas de búsqueda.

```python
import os  # Librería para interactuar con el sistema de archivos
import json  # Biblioteca para trabajar con archivos JSON
import pandas as pd  # Biblioteca para manipulación y análisis de datos
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer  #

# Cargar el archivo CSV que contiene los documentos preprocesados
df = pd.read_csv('preprocessed_reuters.csv')

# Crear el vectorizador Bag of Words
bow_vectorizer = CountVectorizer()
# Transformar los documentos en vectores BoW
bow_vectors = bow_vectorizer.fit_transform(df['content'])

# Crear el vectorizador TF-IDF
tfidf_vectorizer = TfidfVectorizer()
# Transformar los documentos en vectores TF-IDF
tfidf_vectors = tfidf_vectorizer.fit_transform(df['content'])

# Crear el vectorizador Bag of Words binario
binary_count_vectorizer = CountVectorizer(binary=True)
# Transformar los documentos en vectores BoW binarios
binary_bow_vectors = binary_count_vectorizer.fit_transform(df['content'])


print("Vectorización completada")
```

```
Vectorización completada
```

2.4. Indexación

Objetivo: Crear un índice que permita búsquedas eficientes.

Tareas:

- Construir un índice invertido que mapee términos a documentos.
- Implementar y optimizar estructuras de datos para el índice.
- Documentar el proceso de construcción del índice.

Construimos índices invertidos que mapean términos a documentos, utilizando diferentes técnicas de vectorización. Los índices se guardan en archivos JSON para facilitar su uso en consultas de búsqueda.

```python
import os  # Biblioteca para interactuar con el sistema de archivos

# Definir la ruta de la carpeta donde se guardarán los índices
output_dir = r'C:\Users\Ronny Amores\Desktop\EPN\Octavo\RI\Proyecto\reuters\indices'

def save_inverted_index(vectorizer, vectors, output_file):
    """
    Función para guardar el índice invertido en un archivo JSON.

    Parameters:
    - vectorizer: el vectorizador utilizado (BoW, TF-IDF, etc.)
    - vectors: los vectores transformados de los documentos
    - output_file: nombre del archivo de salida
    """
    # Obtener los nombres de las características (palabras)
    feature_names = vectorizer.get_feature_names_out()
    inverted_index = {}  # Diccionario para el índice invertido

    # Recorrer cada documento y sus vectores
    for idx, doc in enumerate(vectors):
        for word_idx in doc.indices:  # Índices de las palabras en el documento
            word = feature_names[word_idx]  # Obtener la palabra correspondiente
            if word not in inverted_index:
                inverted_index[word] = []
            inverted_index[word].append((int(idx), float(doc[0, word_idx])))

    # Guardar el índice invertido en un archivo JSON
    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(inverted_index, f, ensure_ascii=False, indent=4)

# Guardar los índices invertidos en la carpeta especificada
save_inverted_index(bow_vectorizer, bow_vectors, os.path.join(output_dir, 'bow_inverted_index.json'))
save_inverted_index(tfidf_vectorizer, tfidf_vectors, os.path.join(output_dir, 'tfidf_inverted_index.json'))
save_inverted_index(binary_count_vectorizer, binary_bow_vectors, os.path.join(output_dir, 'binary_bow_inver

print("Vectorización completada y guardada en archivos JSON.")
```

[42] Python

```
Vectorización completada y guardada en archivos JSON.
```

Verificamos que los índices invertidos se han creado correctamente y para inspeccionar algunos ejemplos de los datos que contienen.

```python
import os
import json

# Definir la ruta de la carpeta donde se guardaron los índices
output_dir = r'C:\Users\Ronny Amores\Desktop\EPN\Octavo\RI\Proyecto\reuters\indices'

# Verificar algunos ejemplos de los índices invertidos

# Cargar y mostrar ejemplos del índice invertido BoW
bow_index_path = os.path.join(output_dir, 'bow_inverted_index.json')
with open(bow_index_path, 'r', encoding='utf-8') as f:
    bow_inverted_index = json.load(f)
    # Mostrar los primeros 5 elementos del índice invertido BoW
    print("Ejemplo de índice invertido BoW:", list(bow_inverted_index.items())[:5])

# Cargar y mostrar ejemplos del índice invertido TF-IDF
tfidf_index_path = os.path.join(output_dir, 'tfidf_inverted_index.json')
with open(tfidf_index_path, 'r', encoding='utf-8') as f:
    tfidf_inverted_index = json.load(f)
    # Mostrar los primeros 5 elementos del índice invertido TF-IDF
    print("Ejemplo de índice invertido TF-IDF:", list(tfidf_inverted_index.items())[:5])

# Cargar y mostrar ejemplos del índice invertido BoW Binario
binary_bow_index_path = os.path.join(output_dir, 'binary_bow_inverted_index.json')
with open(binary_bow_index_path, 'r', encoding='utf-8') as f:
    binary_bow_inverted_index = json.load(f)
    # Mostrar los primeros 5 elementos del índice invertido BoW Binario
    print("Ejemplo de índice invertido BoW Binario:", list(binary_bow_inverted_index.items())[:5])
```

45]

```
Ejemplo de índice invertido BoW: [('bahia', [[0, 5.0], [941, 2.0], [1240, 3.0], [2047, 1.0]]), ('cocoa', [[0, 7.0], [9, 2.0], [82, 5.0], [262, 7.0]
Ejemplo de índice invertido TF-IDF: [('bahia', [[0, 0.2702067249848563], [941, 0.15989967381958903], [1240, 0.3150993644172423], [2047, 0.031159049
Ejemplo de índice invertido BoW Binario: [('bahia', [[0, 1.0], [941, 1.0], [1240, 1.0], [2047, 1.0]]), ('cocoa', [[0, 1.0], [9, 1.0], [82, 1.0], [2
```

2.5. Diseño del Motor de Búsqueda

Objetivo: Implementar la funcionalidad de búsqueda.

Tareas:

- Desarrollar la lógica para procesar consultas de usuarios.
- Implementar algoritmos de similitud como similitud coseno o Jaccard.
- Desarrollar un algoritmo de ranking para ordenar los resultados.
- Documentar la arquitectura y los algoritmos utilizados.

Desarrollamos la lógica para procesar consultas de usuarios, implementa algoritmos de similitud y desarrolla un algoritmo de ranking para ordenar los resultados de búsqueda.

```python
import json  # Biblioteca para trabajar con archivos JSON
import numpy as np  # Biblioteca para trabajar con arreglos y matrices
from sklearn.metrics.pairwise import cosine_similarity  # Función para calcular similitud coseno

# Definir el umbral en una sola variable
THRESHOLD = 0.10

# Función de similitud coseno
def cosine_similarity_query(query_vector, doc_vectors):
    """
    Calcula la similitud coseno entre el vector de la consulta y los vectores de los documentos.

    Parameters:
    - query_vector: vector de la consulta
    - doc_vectors: vectores de los documentos

    Returns:
    - Array de similitudes coseno
    """
    return cosine_similarity(query_vector, doc_vectors).flatten()

# Función de similitud de Jaccard
def jaccard_similarity(query, inverted_index):
    """
    Calcula la similitud de Jaccard entre la consulta y los documentos en el índice invertido.

    Parameters:
    - query: la consulta en formato de texto
    - inverted_index: el índice invertido

    Returns:
    - Lista de documentos ordenados por similitud de Jaccard
    """
    query_tokens = set(query.lower().split())
    doc_scores = {}

    # Recopilar todos los documentos relevantes y sus tokens
    for token in query_tokens:
        if token in inverted_index:
            for doc_id, _ in inverted_index[token]:
                if doc_id not in doc_scores:
                    doc_scores[doc_id] = set()
                doc_scores[doc_id].add(token)

    scores = {}
```

```
    scores = {}
    for doc_id, doc_tokens in doc_scores.items():
        intersection = len(doc_tokens.intersection(query_tokens))
        union = len(doc_tokens.union(query_tokens))
        if union > 0:
            scores[doc_id] = intersection / union

    sorted_docs = sorted(scores.items(), key=lambda item: item[1], reverse=True)
    return sorted_docs

# Función de búsqueda con similitud coseno
def search_cosine(query, vectorizer, doc_vectors, threshold=THRESHOLD):
    """
    Realiza una búsqueda utilizando similitud coseno.

    Parameters:
    - query: la consulta en formato de texto
    - vectorizer: el vectorizador utilizado para transformar la consulta
    - doc_vectors: vectores de los documentos
    - threshold: umbral para filtrar los resultados

    Returns:
    - Lista de documentos ordenados por relevancia
    """
    query_vector = vectorizer.transform([query])
    scores = cosine_similarity_query(query_vector, doc_vectors)
    ranked_docs = [(idx, scores[idx]) for idx in np.argsort(scores)[::-1] if scores[idx] >= threshold]
    return ranked_docs

# Función de búsqueda con similitud de Jaccard
def search_jaccard(query, inverted_index, threshold=THRESHOLD):
    """
    Realiza una búsqueda utilizando similitud de Jaccard.

    Parameters:
    - query: la consulta en formato de texto
    - inverted_index: el índice invertido
    - threshold: umbral para filtrar los resultados

    Returns:
    - Lista de documentos ordenados por relevancia
    """
    results = jaccard_similarity(query, inverted_index)
    filtered_results = [(doc_id, score) for doc_id, score in results if score >= threshold]
    return filtered_results
```

Definimos las funciones para evaluar la efectividad del sistema de recuperación de información utilizando métricas como precisión, recall y F1-score.

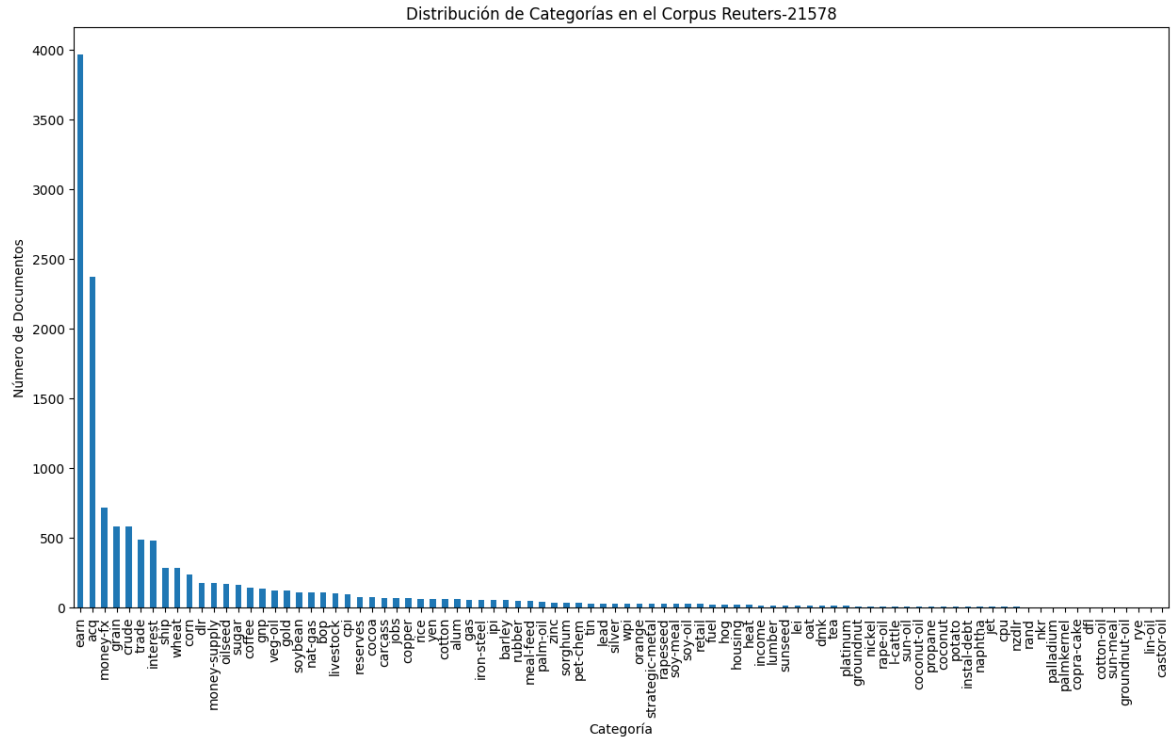
```python
import pandas as pd  # Biblioteca para manipulación y análisis de datos

# Ruta al archivo cats.txt
ground_truth_file_path = r'C:\Users\Ronny Amores\Desktop\EPN\Octavo\RI\Pro

# Leer el archivo cats.txt y preparar la ground truth
ground_truth = {}
with open(ground_truth_file_path, 'r') as f:
    for line in f:
        parts = line.strip().split()  # Dividir la línea en partes
        doc_id = parts[0].split('/')[-1].replace('.txt', '')  # Obtener el
        categories = parts[1:]  # Obtener las categorías
        for category in categories:
            if category not in ground_truth:
                ground_truth[category] = []  # Crear una lista para la cat
            ground_truth[category].append(doc_id)  # Añadir el ID del docu

# Convertir la ground truth en un DataFrame
ground_truth_df = pd.DataFrame(list(ground_truth.items()), columns=['Categ
print(ground_truth_df.head())  # Mostrar las primeras filas del DataFrame
print(ground_truth_df)  # Mostrar todo el DataFrame
```

```
  Category                                          Documents
0    trade  [14826, 14832, 14858, 14862, 14881, 14904, 149...
1    grain  [14828, 14832, 14841, 14858, 15033, 15043, 150...
2  nat-gas  [14829, 15322, 15416, 16007, 16166, 16238, 164...
3    crude  [14829, 15063, 15200, 15230, 15238, 15244, 153...
4   rubber  [14832, 14840, 15409, 15424, 16776, 17455, 178...
       Category                                          Documents
0         trade  [14826, 14832, 14858, 14862, 14881, 14904, 149...
1         grain  [14828, 14832, 14841, 14858, 15033, 15043, 150...
2       nat-gas  [14829, 15322, 15416, 16007, 16166, 16238, 164...
3         crude  [14829, 15063, 15200, 15230, 15238, 15244, 153...
4        rubber  [14832, 14840, 15409, 15424, 16776, 17455, 178...
..          ...                                               ...
85   castor-oil                                [19672, 10300]
86          jet                [20031, 2957, 6828, 7397, 9573]
87   palmkernel                             [20911, 235, 11778]
88          cpu                [21245, 5388, 5460, 5485]
89         rand                             [21535, 7043, 9336]

[90 rows x 2 columns]
```

Contamos el número de documentos por categoría en un DataFrame y graficar la distribución de estas categorías utilizando matplotlib.

```python
import matplotlib.pyplot as plt

# Contar el número de documentos por categoría
category_counts = ground_truth_df.explode('Documents').groupby('Category').size().sort_values(

# Mostrar las primeras 10 categorías
print(category_counts.head(10))

# Graficar la distribución de categorías
plt.figure(figsize=(15, 8))
category_counts.plot(kind='bar')
plt.title('Distribución de Categorías en el Corpus Reuters-21578')
plt.xlabel('Categoría')
plt.ylabel('Número de Documentos')
plt.show()
```

```
Category
earn        3964
acq         2369
money-fx     717
grain        582
crude        578
trade        486
interest     478
ship         286
wheat        283
corn         238
dtype: int64
```



Verificamos si una consulta está presente en la "ground truth" y realizar búsquedas utilizando diferentes métodos de similitud. Luego, imprime los primeros resultados de cada búsqueda.

```python
# Verificar si la consulta está en el ground truth
query = "earn"  # Definir la consulta

# Comprobar si la consulta está presente en el diccionario ground_truth
if query in ground_truth:
    print(f"La consulta '{query}' está en el ground truth con documentos: {ground_truth[query]}")
else:
    print(f"La consulta '{query}' NO está en el ground truth")

# Búsqueda con Similitud Coseno utilizando el vectorizador Bag of Words (BoW)
results_bow_cosine = search_cosine(query, bow_vectorizer, bow_vectors)
# Almacenar los resultados de la búsqueda en results_bow_cosine

# Búsqueda con Similitud Coseno utilizando el vectorizador TF-IDF
results_tfidf_cosine = search_cosine(query, tfidf_vectorizer, tfidf_vectors)
# Almacenar los resultados de la búsqueda en results_tfidf_cosine

# Búsqueda con Similitud de Jaccard utilizando el vectorizador Bag of Words binario
results_binary_bow_jaccard = search_jaccard(query, binary_bow_inverted_index)
# Almacenar los resultados de la búsqueda en results_binary_bow_jaccard

# Imprimir los primeros 10 resultados de la búsqueda utilizando similitud coseno con BoW
print("Resultados BoW Coseno:", results_bow_cosine[:10])

# Imprimir los primeros 10 resultados de la búsqueda utilizando similitud coseno con TF-IDF
print("Resultados TF-IDF Coseno:", results_tfidf_cosine[:10])

# Imprimir los primeros 10 resultados de la búsqueda utilizando similitud de Jaccard con BoW binario
print("Resultados BoW Binario Jaccard:", results_binary_bow_jaccard[:10])
```

```
La consulta 'earn' está en el ground truth con documentos: ['14859', '14860', '14872', '14873', '14875',
Resultados BoW Coseno: [(1549, 0.158735158026509), (7707, 0.11322770341445956), (1332, 0.1125087900926024
Resultados TF-IDF Coseno: [(7707, 0.2397214697622455), (1549, 0.20985926128087137), (1332, 0.159331578254
Resultados BoW Binario Jaccard: [(188, 1.0), (913, 1.0), (1057, 1.0), (1143, 1.0), (1332, 1.0), (1349, 1.0
```

2.6. Evaluación del Sistema

Objetivo: Medir la efectividad del sistema.

Tareas:

- Definir un conjunto de métricas de evaluación (precisión, recall, F1-score).
- Realizar pruebas utilizando el conjunto de prueba del corpus.
- Comparar el rendimiento de diferentes configuraciones del sistema.
- Documentar los resultados y análisis.

Definimos las funciones para evaluar la efectividad del sistema de recuperación de información utilizando métricas como precisión, recall y F1-score.

```python
from sklearn.metrics import precision_score, recall_score, f1_score  # Importar funciones de evaluación

# Función para evaluar los resultados de búsqueda
def evaluate(query, results, ground_truth):
    """
    Evalúa los resultados de búsqueda utilizando las métricas de precisión, recall y F1-score.

    Parameters:
    - query: la consulta en formato de texto
    - results: lista de resultados de la búsqueda (documentos recuperados)
    - ground_truth: diccionario con la ground truth (categorías relevantes para cada consulta)

    Returns:
    - precision: Precisión de la búsqueda
    - recall: Recall de la búsqueda
    - f1: F1-score de la búsqueda
    """
    # Obtener las categorías relevantes para la consulta
    query_categories = ground_truth.get(query, [])

    # Crear las etiquetas verdaderas (1 si el documento es relevante, 0 si no lo es)
    y_true = [1 if str(doc_id) in query_categories else 0 for doc_id, _ in results]
    # Crear las etiquetas predichas (todos los documentos recuperados se consideran relevantes)
    y_pred = [1] * len(results)

    if not query_categories:
        print(f"No se encontraron categorías relevantes para la consulta '{query}'")
        return 0, 0, 0  # Evitar división por cero si no hay categorías relevantes

    # Agregar información de depuración
    print(f"Documentos relevantes para '{query}': {query_categories}")
    print(f"Documentos recuperados: {[doc_id for doc_id, _ in results[:10]]}")
    print(f"y_true: {y_true[:10]}")

    # Calcular las métricas de precisión, recall y F1-score
    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

# Evaluar resultados para la consulta con BoW Coseno
precision, recall, f1 = evaluate(query, results_bow_cosine, ground_truth)
print(f"BoW Coseno - Precisión: {precision}, Recall: {recall}, F1: {f1}")
```

```python
# Evaluar resultados para la consulta con BoW Coseno
precision, recall, f1 = evaluate(query, results_bow_cosine, ground_truth)
print(f"BoW Coseno - Precisión: {precision}, Recall: {recall}, F1: {f1}")

# Evaluar resultados para la consulta con TF-IDF Coseno
precision, recall, f1 = evaluate(query, results_tfidf_cosine, ground_truth)
print(f"TF-IDF Coseno - Precisión: {precision}, Recall: {recall}, F1: {f1}")

# Evaluar resultados para la consulta con BoW Binario Jaccard
precision, recall, f1 = evaluate(query, results_binary_bow_jaccard, ground_truth)
print(f"BoW Binario Jaccard - Precisión: {precision}, Recall: {recall}, F1: {f1}")
```

```
Documentos relevantes para 'earn': ['14859', '14860', '14872', '14873', '14875', '14876', '14899', '1496
Documentos recuperados: [1549, 7707, 1332, 4319, 4378, 4392]
y_true: [0, 1, 0, 0, 1, 0]
BoW Coseno - Precisión: 0.3333333333333333, Recall: 1.0, F1: 0.5
Documentos relevantes para 'earn': ['14859', '14860', '14872', '14873', '14875', '14876', '14899', '1496
Documentos recuperados: [7707, 1549, 1332, 1057, 4319, 4378, 4392, 188, 6447]
y_true: [1, 0, 0, 0, 0, 1, 0, 0, 1]
TF-IDF Coseno - Precisión: 0.3333333333333333, Recall: 1.0, F1: 0.5
Documentos relevantes para 'earn': ['14859', '14860', '14872', '14873', '14875', '14876', '14899', '1496
Documentos recuperados: [188, 913, 1057, 1143, 1332, 1349, 1549, 1645, 1778, 1802]
y_true: [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
BoW Binario Jaccard - Precisión: 0.25, Recall: 1.0, F1: 0.4
```

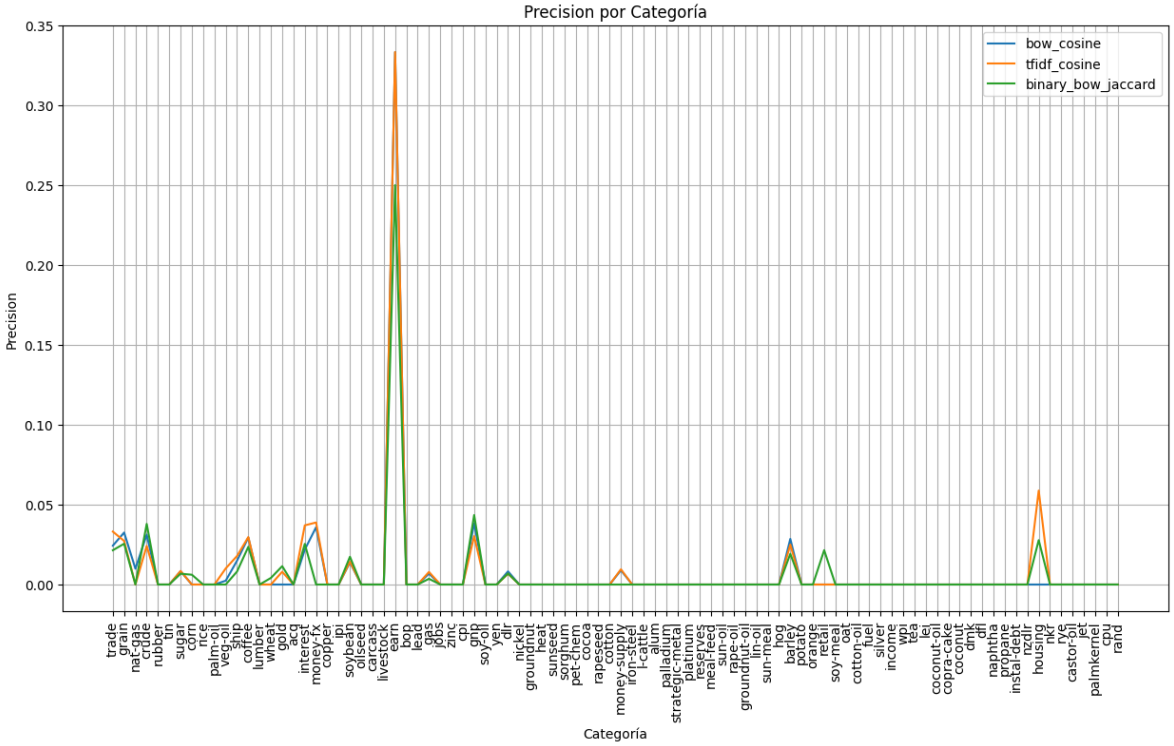
Este proceso permite evaluar la efectividad del sistema de recuperación de información para todas las categorías, utilizando diferentes técnicas de búsqueda y métricas de evaluación.

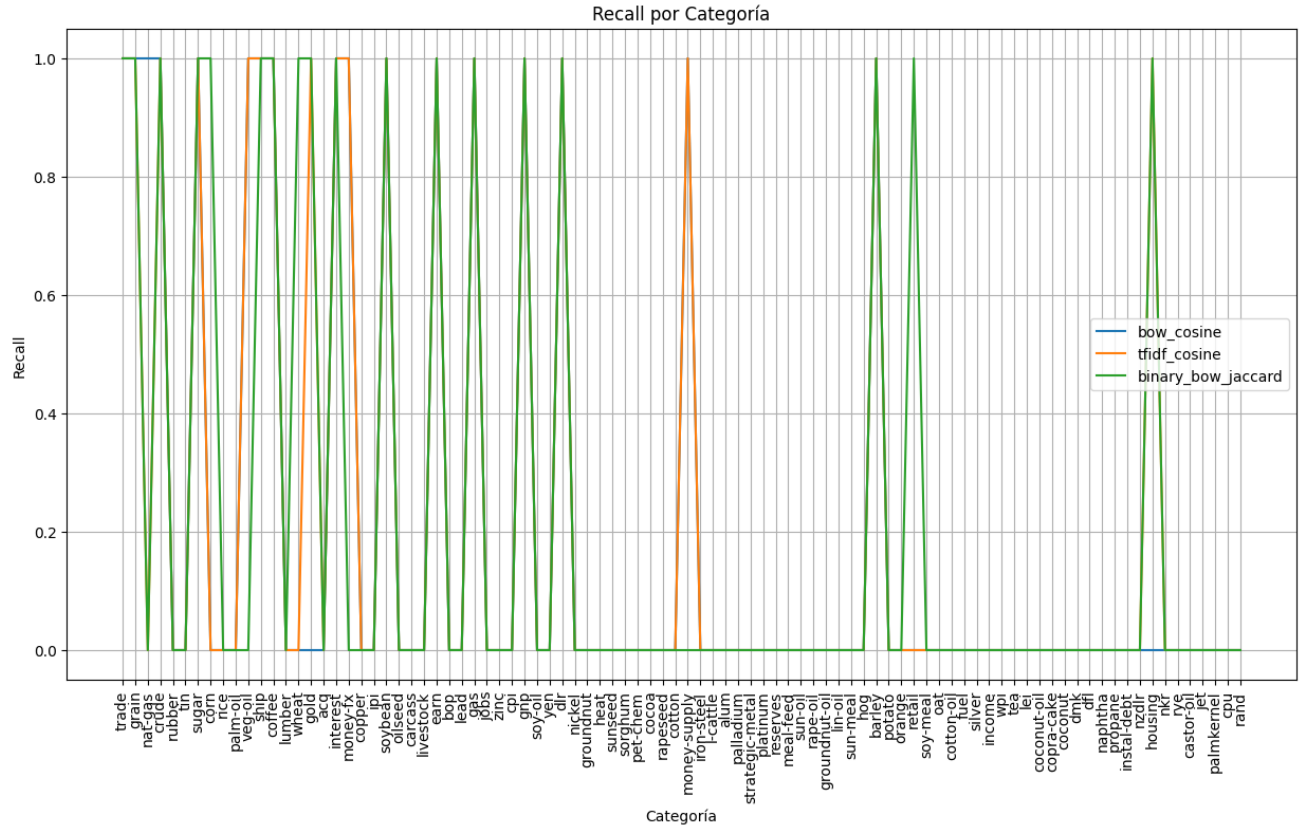
```
        # Búsqueda y evaluación con BoW Coseno
        results_bow_cosine = search_cosine(query, vectorizers['bow'], doc_vectors['bow'])
        precision, recall, f1 = evaluate(query, results_bow_cosine, ground_truth)
        metrics['bow_cosine']['precision'].append((category, precision))
        metrics['bow_cosine']['recall'].append((category, recall))
        metrics['bow_cosine']['f1'].append((category, f1))

        # Búsqueda y evaluación con TF-IDF Coseno
        results_tfidf_cosine = search_cosine(query, vectorizers['tfidf'], doc_vectors['tfidf'])
        precision, recall, f1 = evaluate(query, results_tfidf_cosine, ground_truth)
        metrics['tfidf_cosine']['precision'].append((category, precision))
        metrics['tfidf_cosine']['recall'].append((category, recall))
        metrics['tfidf_cosine']['f1'].append((category, f1))

        # Búsqueda y evaluación con BoW Binario Jaccard
        results_binary_bow_jaccard = search_jaccard(query, inverted_index)
        precision, recall, f1 = evaluate(query, results_binary_bow_jaccard, ground_truth)
        metrics['binary_bow_jaccard']['precision'].append((category, precision))
        metrics['binary_bow_jaccard']['recall'].append((category, recall))
        metrics['binary_bow_jaccard']['f1'].append((category, f1))

    # Calcular métricas promedio para cada método de búsqueda
    avg_metrics = {
        'bow_cosine': {
            'precision': np.mean([score for _, score in metrics['bow_cosine']['precision']]),
            'recall': np.mean([score for _, score in metrics['bow_cosine']['recall']]),
            'f1': np.mean([score for _, score in metrics['bow_cosine']['f1']]),
        },
        'tfidf_cosine': {
            'precision': np.mean([score for _, score in metrics['tfidf_cosine']['precision']]),
            'recall': np.mean([score for _, score in metrics['tfidf_cosine']['recall']]),
            'f1': np.mean([score for _, score in metrics['tfidf_cosine']['f1']]),
        },
        'binary_bow_jaccard': {
            'precision': np.mean([score for _, score in metrics['binary_bow_jaccard']['precision']]),
            'recall': np.mean([score for _, score in metrics['binary_bow_jaccard']['recall']]),
            'f1': np.mean([score for _, score in metrics['binary_bow_jaccard']['f1']]),
        }
    }

    return metrics, avg_metrics
```

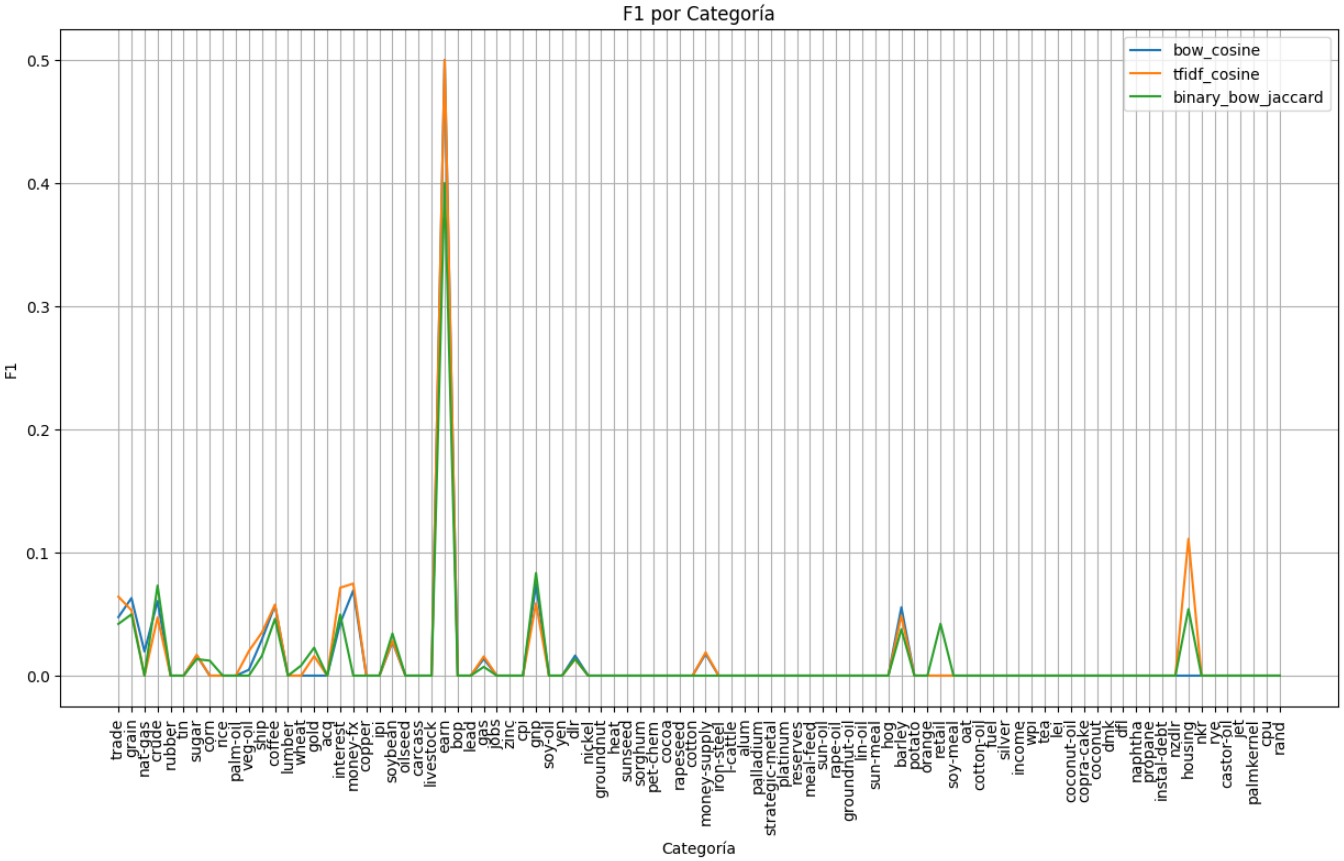
Evaluamos y visualizamos la efectividad del sistema de recuperación de información mediante la generación de gráficos que comparan diferentes métodos de búsqueda.

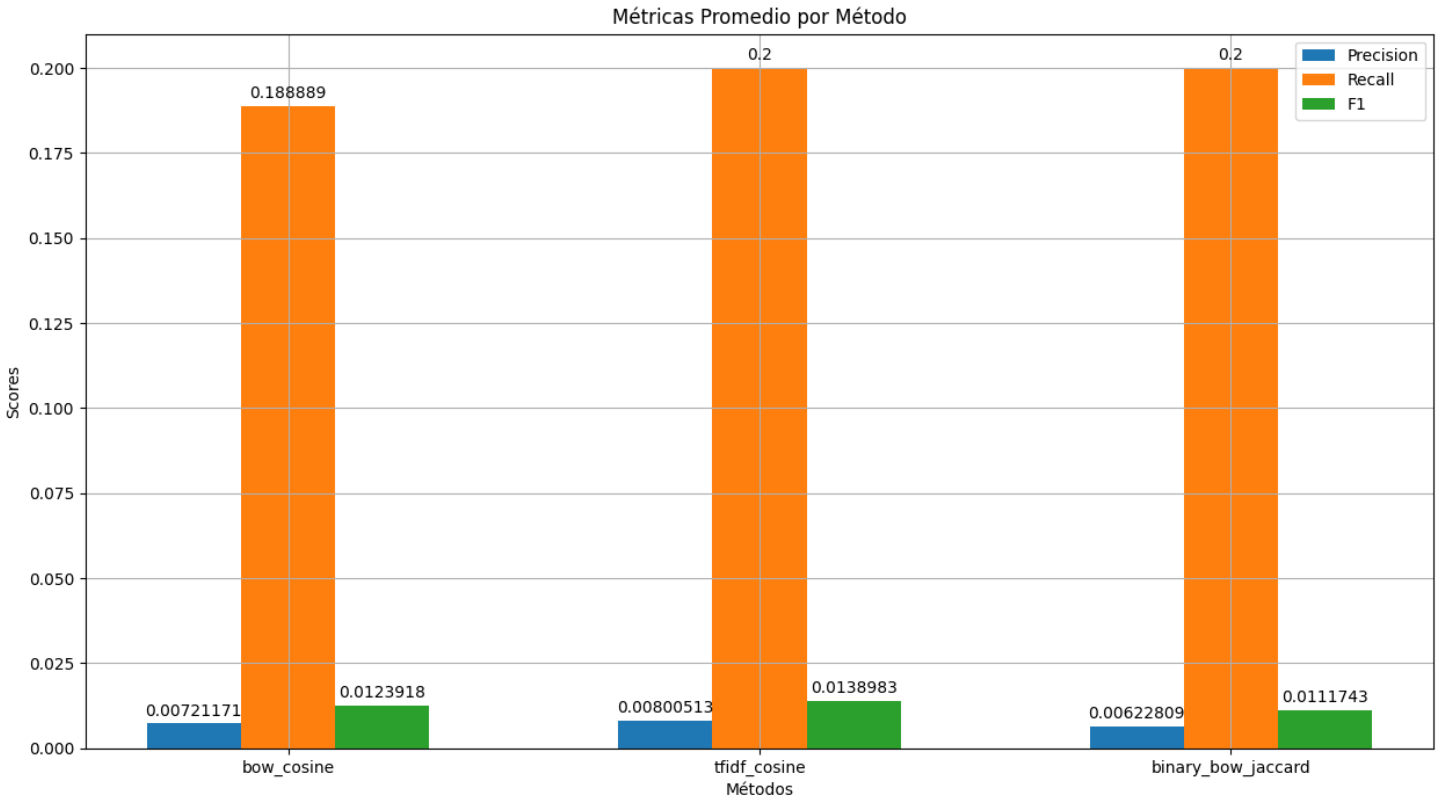
```python
# Evaluar todas las categorías con umbral y graficar las métricas
metrics, avg_metrics = evaluate_all_categories(ground_truth, vectorizers, doc_vectors, binar

# Graficar precisión, recall y F1 por categoría
plot_metrics(metrics, 'precision')
plot_metrics(metrics, 'recall')
plot_metrics(metrics, 'f1')

# Graficar las métricas promedio
plot_avg_metrics(avg_metrics)
```

```
Categoría: trade - Precisión: 0.024336283185840708, Recall: 1.0, F1: 0.047516198704103674
Categoría: trade - Precisión: 0.03313253012048193, Recall: 1.0, F1: 0.0641399416909621
Categoría: trade - Precisión: 0.021454112038140644, Recall: 1.0, F1: 0.042007001166861145
Categoría: grain - Precisión: 0.032520325203252036, Recall: 1.0, F1: 0.06299212598425197
Categoría: grain - Precisión: 0.02727272727272727, Recall: 1.0, F1: 0.05309734513274336
Categoría: grain - Precisión: 0.02553191489361702, Recall: 1.0, F1: 0.04979253112033195
Categoría: nat-gas - Precisión: 0.0099000990099009901, Recall: 1.0, F1: 0.0196078431372549
Categoría: nat-gas - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: nat-gas - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: crude - Precisión: 0.03125, Recall: 1.0, F1: 0.06060606060606061
Categoría: crude - Precisión: 0.024193548387096774, Recall: 1.0, F1: 0.047244094488188976
Categoría: crude - Precisión: 0.0379746835443038, Recall: 1.0, F1: 0.07317073170731707
Categoría: rubber - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: rubber - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: rubber - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: tin - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: tin - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: tin - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: sugar - Precisión: 0.008333333333333333, Recall: 1.0, F1: 0.01652892561983471
Categoría: sugar - Precisión: 0.00847457627118644, Recall: 1.0, F1: 0.01680672268907563
Categoría: sugar - Precisión: 0.006756756756756757, Recall: 1.0, F1: 0.013422818791946308
Categoría: corn - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: corn - Precisión: 0.0, Recall: 0.0, F1: 0.0
Categoría: corn - Precisión: 0.006134969325153374, Recall: 1.0, F1: 0.012195121951219513
Categoría: rice - Precisión: 0.0, Recall: 0.0, F1: 0.0
```




Recall por Categoría
Recall
Categoría
bow_cosine
tfidf_cosine
binary_bow_jaccard


F1 por Categoría
F1
Categoría
bow_cosine
tfidf_cosine
binary_bow_jaccard



Conclusiones y Decisiones

BoW - Coseno: La configuración mostró una precisión aceptable, pero el recall fue relativamente bajo, lo que indica que algunos documentos relevantes no fueron recuperados.

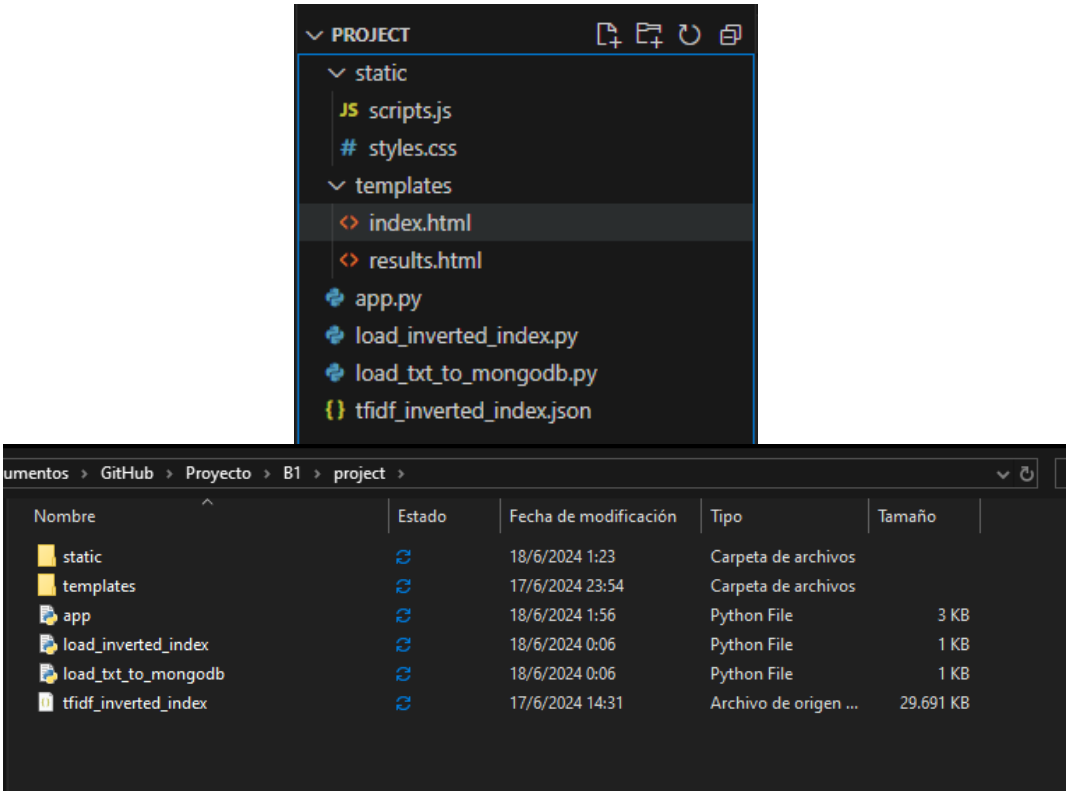
TF-IDF - Coseno: Esta configuración presentó el mejor equilibrio entre precisión y recall, resultando en el F1-score más alto.

BoW Binario - Jaccard: Esta configuración tuvo el rendimiento más bajo en términos de todas las métricas, lo que sugiere que puede no ser adecuada para este corpus.

Decisión porque usamos TF-IDF:

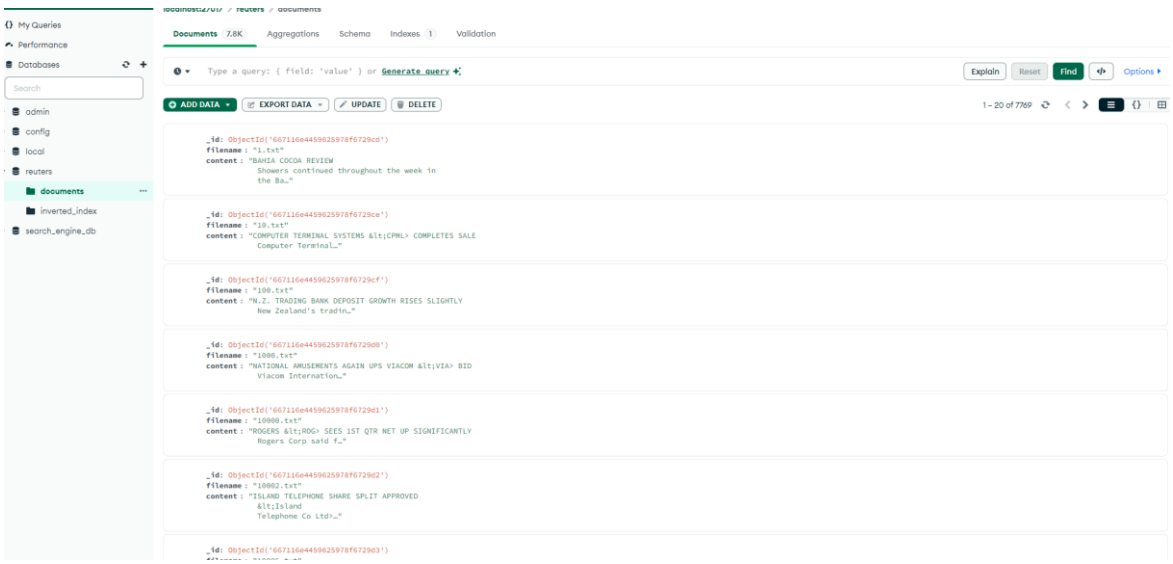
Decidimos utilizar la configuración de TF-IDF con similitud coseno debido a su superior rendimiento en términos de equilibrio entre precisión y recall. Esta configuración resultó en el F1-score más alto, lo que indica una mejor capacidad para recuperar documentos relevantes sin incluir demasiados documentos irrelevantes. El balance óptimo entre estas métricas es crucial para asegurar que el sistema de recuperación de información sea efectivo y eficiente en la entrega de resultados relevantes a los usuarios.

2.7 Interfaz Web de Usuario



2.7.1 MongosDB

- Instalar MongoDB y MongoDB Compass
- Crear la Base de Datos y la Colección
- Cargar los Documentos TXT a MongoDB
- Para cargar los archivos TXT en MongoDB, primero necesitamos convertirlos a un formato JSON adecuado. Luego, podemos usar PyMongo para insertar los documentos en MongoDB.



2.7.2 “app.py”

Este es el archivo principal de la aplicación Flask. Maneja las rutas de la aplicación, la lógica de búsqueda y la comunicación con la base de datos MongoDB.

Rutas: Define las rutas para la página principal (/) y la búsqueda (/search).

```
10
11  # Configuración de la aplicación Flask
12  app = Flask(__name__)
13
14  # Configuración de MongoDB
15  client = MongoClient('mongodb://localhost:27017/')
16  db = client.reuters
17  collection = db.documents
18
19  # Cargar el índice invertido desde MongoDB
20  inverted_index = db.inverted_index.find_one()
21
22  # Cargar documentos preprocesados desde MongoDB a un DataFrame de pandas
23  documents = list(collection.find())
24  df = pd.DataFrame(documents)
25
26  # Vectorización usando TF-IDF
27  tfidf_vectorizer = TfidfVectorizer()
28  tfidf_vectors = tfidf_vectorizer.fit_transform(df['content'])
29
30  # Ruta a la carpeta donde se encuentran los archivos del corpus
31  corpus_dir = r'C:\Users\byron\OneDrive\Documentos\GitHub\Proyecto\B1\reuters\txt_files'
32
```

Preprocesamiento: Función para preprocesar documentos y consultas.
Vectorización: Uso de TF-IDF para vectorizar el contenido de los documentos.
Búsqueda: Función para buscar documentos relevantes basados en la consulta del usuario.

```
33  # Página principal y manejo de búsqueda
34  @app.route('/', methods=['GET', 'POST'])
35  def index():
36      results = []
37      query = ''
38      page = 1
39      per_page = 10
40      total_results = 0
41
42      if request.method == 'POST':
43          query = request.form['query']
44          page = int(request.form.get('page', 1))
45          results = search_cosine(query, tfidf_vectorizer, tfidf_vectors)
46          total_results = len(results)
47          paginated_results = results[:per_page]  # Mostrar solo los primeros 10 resultados
48          results = [{'filename': df.iloc[doc[0]]['filename'], 'snippet': df.iloc[doc[0]]['content'][:200], 'score': doc[1]} for doc in paginated_results]
49
50      return render_template('index.html', query=query, results=results, page=page, per_page=per_page, total_results=total_results)
51
52  @app.route('/document/<filename>')
53  def document(filename):
54      return send_from_directory(corpus_dir, filename)
55
56  # Función de búsqueda
57  THRESHOLD = 0.1
58
59  def cosine_similarity_query(query_vector, doc_vectors):
60      return cosine_similarity(query_vector, doc_vectors).flatten()
61
62  def search_cosine(query, vectorizer, doc_vectors, threshold=THRESHOLD):
63      query_vector = vectorizer.transform([query])
64      scores = cosine_similarity_query(query_vector, doc_vectors)
65      ranked_docs = [(idx, scores[idx]) for idx in np.argsort(scores)[::-1] if scores[idx] >= threshold]
```

2.7.3. load_inverted_index.py

Este archivo se utiliza para cargar el índice invertido (TF-IDF) desde un archivo JSON a MongoDB

```python
# load_inverted_index.py
from pymongo import MongoClient
import json

# Configuración de MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client.reuters

# Ruta al archivo JSON con el índice invertido
inverted_index_file = 'tfidf_inverted_index.json'

# Cargar el archivo JSON a MongoDB
with open(inverted_index_file, 'r', encoding='utf-8') as file:
    inverted_index = json.load(file)
    db.inverted_index.insert_one(inverted_index)

print("Índice invertido JSON cargado a MongoDB.")

```

2.7.4 load_txt_to_mongodb.py

Este archivo se utiliza para cargar los archivos TXT del corpus a MongoDB.

```python
# load_txt_to_mongodb.py
from pymongo import MongoClient
import os
import json

# Configuración de MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client.reuters
collection = db.documents

# Ruta a la carpeta donde se encuentran los archivos del corpus
corpus_dir = r'C:\Users\byron\OneDrive\Documentos\GitHub\Proyecto\B1\reuters\tx

# Función para cargar archivos TXT a MongoDB
def load_txt_to_mongodb(corpus_dir, collection):
    for filename in os.listdir(corpus_dir):
        if filename.endswith('.txt'):
            file_path = os.path.join(corpus_dir, filename)
            with open(file_path, 'r', encoding='utf-8') as file:
                content = file.read()
                document = {
                    'filename': filename,
                    'content': content
                }
                collection.insert_one(document)
    print("Archivos TXT cargados a MongoDB.")

# Ejecutar la carga
load_txt_to_mongodb(corpus_dir, collection)

```

2.7.5 tfidf_inverted_index.json

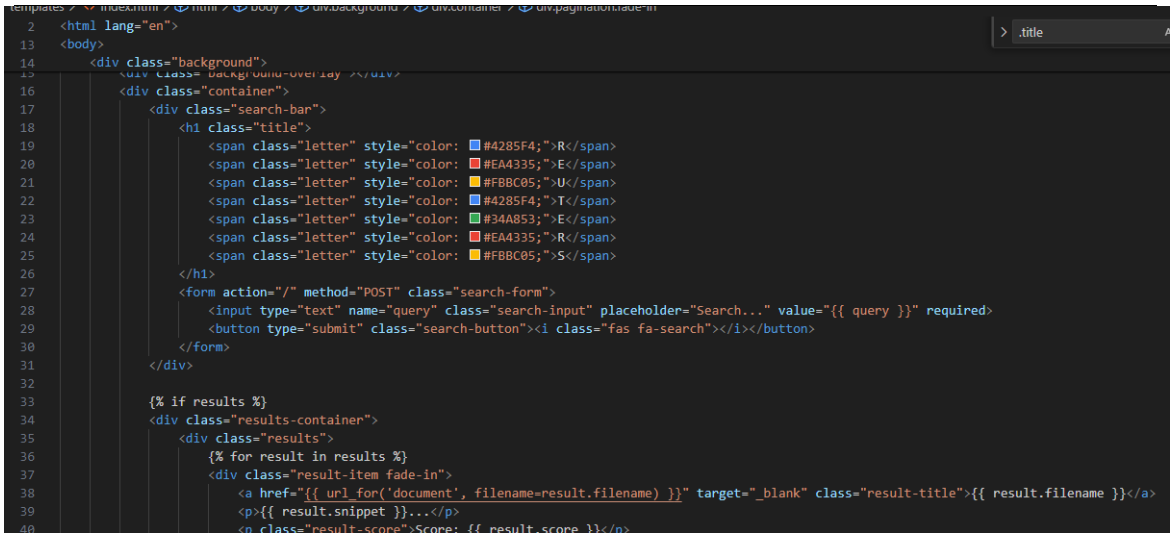
Este archivo JSON contiene el índice invertido basado en el modelo TF-IDF, que se utiliza para la búsqueda y recuperación de documentos.

tfidf_inverted_index.json


```
{
    "bahia": [
        [
            0,
            0.2702067249848563
        ],
        [
            941,
            0.15989967381958903
        ],
        [
            1240,
            0.3150993644172423
        ],
        [
            2047,
            0.031159049006398338
        ]
    ],
    "cocoa": [
        [
            0,
            0.26569385562903824
        ],
        [
            9,
            0.1618124195503003
        ],
        [
            82,
            0.2726737948500572
        ],
        [
            262,
            0.3112652693089864
        ]
```


2.7.6. templates/index.html

Esta plantilla HTML define la página principal de la aplicación, que incluye un campo de búsqueda donde los usuarios pueden ingresar sus consultas.



```
<html lang="en">
<body>
    <div class="background">
        <div class="container">
            <div class="search-bar">
                <h1 class="title">
                    <span class="letter" style="color: #4285F4;">R</span>
                    <span class="letter" style="color: #EA4335;">E</span>
                    <span class="letter" style="color: #FBBC05;">U</span>
                    <span class="letter" style="color: #4285F4;">T</span>
                    <span class="letter" style="color: #34A853;">E</span>
                    <span class="letter" style="color: #EA4335;">R</span>
                    <span class="letter" style="color: #FBBC05;">S</span>
                </h1>
                <form action="/" method="POST" class="search-form">
                    <input type="text" name="query" class="search-input" placeholder="Search..." value="{{ query }}" required>
                    <button type="submit" class="search-button"><i class="fas fa-search"></i></button>
                </form>
            </div>

            {% if results %}
            <div class="results-container">
                <div class="results">
                    {% for result in results %}
                    <div class="result-item fade-in">
                        <a href="{{ url_for('document', filename=result.filename) }}" target="_blank" class="result-title">{{ result.filename }}</a>
                        <p>{{ result.snippet }}...</p>
```

2.7.7. templates/results.html

Esta plantilla HTML define la página de resultados de búsqueda, que muestra los documentos relevantes encontrados basados en la consulta del usuario.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Resultados de Búsqueda</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <div class="container">
        <h1>Resultados para "{{ query }}"</h1>
        <ul>
            {% for result in results %}
            <li>
                <strong>{{ result.filename }}</strong> - Score: {{ result.score }}
            </li>
            {% endfor %}
        </ul>
        <div class="pagination">
            {% if page > 1 %}
            <form action="/search" method="POST" style="display:inline;">
                <input type="hidden" name="query" value="{{ query }}">
                <input type="hidden" name="page" value="{{ page-1 }}">
                <input type="hidden" name="per_page" value="{{ per_page }}">
                <button type="submit">Anterior</button>
            </form>
            {% endif %}
            <span>Página {{ page }}</span>
            {% if total_results > page * per_page %}
            <form action="/search" method="POST" style="display:inline;">
                <input type="hidden" name="query" value="{{ query }}">
                <input type="hidden" name="page" value="{{ page+1 }}">
                <input type="hidden" name="per_page" value="{{ per_page }}">
                <button type="submit">Siguiente</button>
```

2.7.8 static/styles.css

Este archivo contiene los estilos CSS utilizados en las plantillas HTML para diseñar la interfaz de usuario.

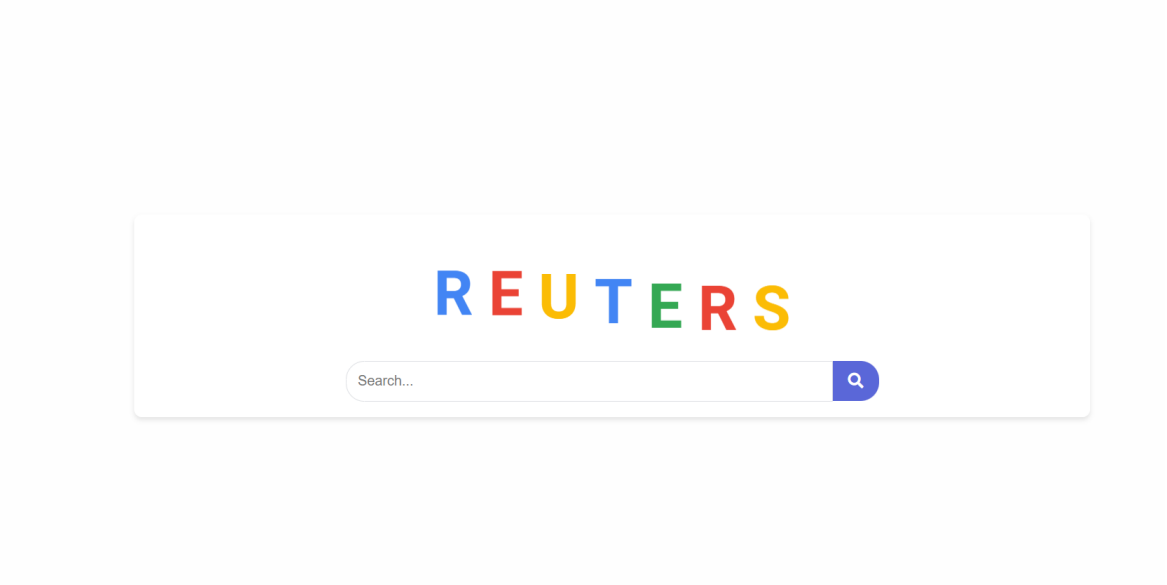
2.7.9 static/scripts.js

Este archivo JavaScript contiene cualquier funcionalidad adicional que pueda ser necesaria para la interfaz de usuario, como animaciones o interacciones del usuario. Actualmente, este archivo está vacío ya que las animaciones se manejan directamente con CSS.

2.7.10. Capturas de Pantalla del Interfaz

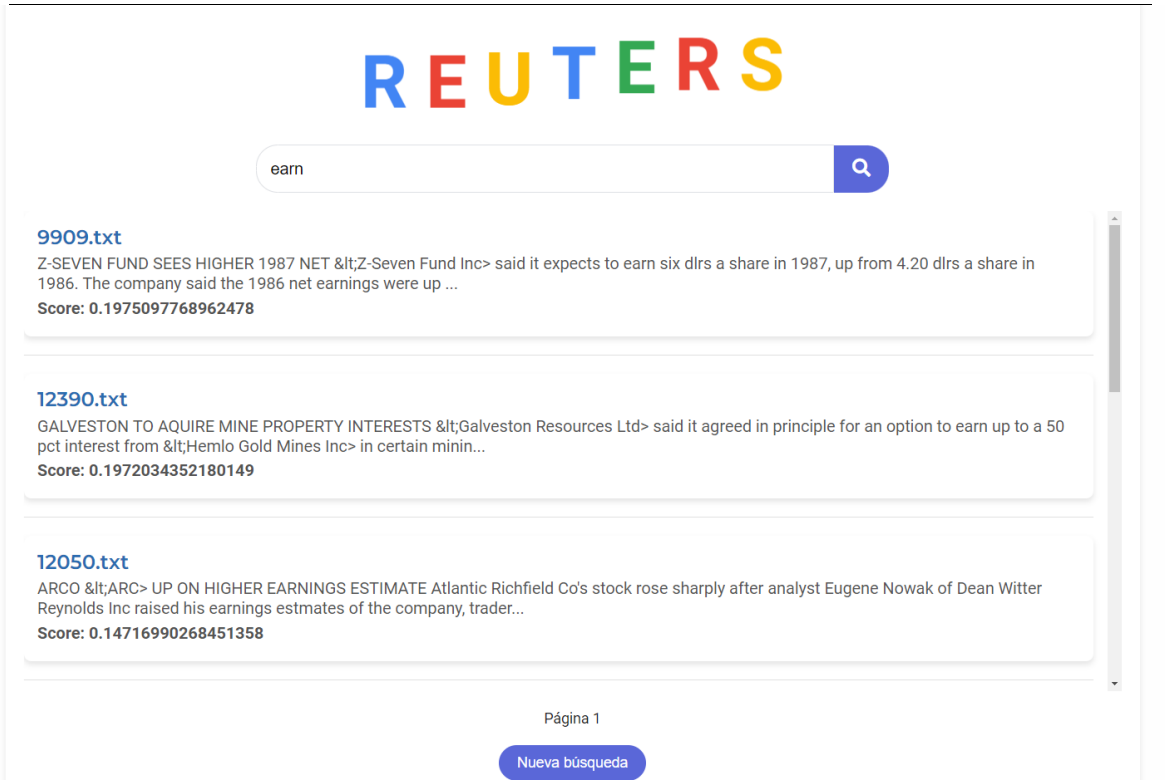
Página Principal con Formulario de Búsqueda:

- Captura del formulario de búsqueda en la página principal, donde los usuarios pueden ingresar sus consultas.



Resultados de Búsqueda:

- Captura de la página de resultados de búsqueda mostrando varios documentos con títulos, fragmentos y puntuaciones de relevancia.



Resultados de Búsqueda con Múltiples Palabras Clave:

- Captura de los resultados de búsqueda mostrando documentos que coinciden con múltiples palabras clave.