ESCUELA POLITÉCNICA NACIONAL

RECUPERACIÓN DE LA INFORMACION

GRUPO

PROYECTO BI - Informe de Evaluación

INTEGRANTES:

RONNY ANDERSON AMORES GUEVARA

BYRON MARCELO ORTIZ PITANDO

DOCENTE:

IVAN CARRERA, PH.D.

GR1CC

FECHA DE ENTREGA: 18/06/2024

Informe de Evaluación del Sistema de Recuperación de Información

1. Introducción

El presente informe detalla la evaluación del Sistema de Recuperación de Información (SRI) desarrollado utilizando el corpus Reuters-21578. El objetivo principal de esta evaluación es medir la efectividad del sistema en términos de su capacidad para recuperar información relevante de un gran conjunto de documentos.

2. Metodología

2.1. Corpus Utilizado

El corpus Reuters-21578 es una colección de documentos de noticias ampliamente utilizada para la investigación en recuperación de información y minería de texto. Contiene miles de documentos categorizados en varias clases temáticas.

2.2. Preprocesamiento

Para preparar los datos para el análisis, se llevaron a cabo los siguientes pasos de preprocesamiento:

- Extracción del contenido relevante de los documentos.
- Limpieza de datos: eliminación de caracteres no deseados y normalización del texto.
- Tokenización: división del texto en palabras o tokens.
- Eliminación de stop words y aplicación de lematización.

Código de Preprocesamiento:

```
python
Copiar código
import os
import re
import nltk
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Descargar recursos necesarios de NLTK
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

# 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\training_txt'
stopword_dir = r'C:\Users\Ronny
Amores\Desktop\EPN\Octavo\RI\Proyecto\reuters'

# 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 lematizer
lemmatizer = WordNetLemmatizer()

def preprocess_document(doc):
    # Eliminar caracteres no deseados y normalizar texto
    doc = re.sub(r'\W', ' ', doc)
    doc = doc.lower()

    # Tokenización
    tokens = word_tokenize(doc)

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

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

    return ' '.join(tokens)

# Leer y preprocesar todos los documentos del corpus
def preprocess_corpus(corpus_dir):
    preprocessed_documents = []

    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)
            with open(file_path, 'r', encoding='utf-8') as file:
                content = file.read()
                preprocessed_text = preprocess_document(content)
                preprocessed_documents.append({
                    'filename': filename,
                    'content': preprocessed_text
                })

    return preprocessed_documents

# Ejecutar el preprocesamiento
preprocessed_documents = preprocess_corpus(corpus_dir)

# Guardar los documentos preprocesados en un archivo CSV
df = pd.DataFrame(preprocessed_documents)
df.to_csv('preprocessed_reuters.csv', index=False)

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

2.3. Vectorización

Se utilizaron tres técnicas principales de vectorización para convertir el texto en una representación numérica que los algoritmos pueden procesar:

- **Bag of Words (BoW):** Cada documento se representa como un vector de frecuencia de palabras.
- **TF-IDF:** Cada documento se representa como un vector ponderado que refleja la importancia de las palabras en el corpus.
- **Bag of Words Binario:** Cada documento se representa como un vector binario indicando la presencia o ausencia de palabras.

Código de Vectorización:

```
python
Copiar código
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer

# Cargar los documentos preprocesados
df = pd.read_csv('preprocessed_reuters.csv')

# Crear el vectorizador BoW
bow_vectorizer = CountVectorizer()
bow_vectors = bow_vectorizer.fit_transform(df['content'])

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

# Crear el vectorizador BoW binario
binary_count_vectorizer = CountVectorizer(binary=True)
binary_bow_vectors =
binary_count_vectorizer.fit_transform(df['content'])

# Función para guardar los índices invertidos en archivos JSON
def save_inverted_index(vectorizer, vectors, output_file):
    feature_names = vectorizer.get_feature_names_out()
    inverted_index = {}

    for idx, doc in enumerate(vectors):
        for word_idx in doc.indices:
            word = feature_names[word_idx]
            if word not in inverted_index:
                inverted_index[word] = []
            inverted_index[word].append((int(idx), float(doc[0,
word_idx])))

    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(inverted_index, f, ensure_ascii=False, indent=4)

# Guardar índices invertidos BoW, TF-IDF y BoW Binario
save_inverted_index(bow_vectorizer, bow_vectors,
'bow_inverted_index.json')
save_inverted_index(tfidf_vectorizer, tfidf_vectors,
'tfidf_inverted_index.json')
save_inverted_index(binary_count_vectorizer, binary_bow_vectors,
'binary_bow_inverted_index.json')

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

2.4. Algoritmos de Similitud

Para medir la similitud entre documentos y consultas, se implementaron los siguientes algoritmos:

- **Similitud Coseno:** Mide la similitud del coseno entre dos vectores.
- **Similitud Jaccard:** Mide la similitud entre dos conjuntos como la intersección dividida por la unión de los conjuntos.

Código de Similitud:

```
python
Copiar código
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Definir el umbral en una sola variable
THRESHOLD = 0.10

# Función de similitud coseno
def cosine_similarity_query(query_vector, doc_vectors):
    return cosine_similarity(query_vector, doc_vectors).flatten()

# Función de similitud de Jaccard
def jaccard_similarity(query, inverted_index):
    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 = {}
    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):
    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):
    results = jaccard_similarity(query, inverted_index)
    filtered_results = [(doc_id, score) for doc_id, score in results
if score >= threshold]
    return filtered_results
```

3. Métricas de Evaluación

Las métricas utilizadas para evaluar el rendimiento del SRI incluyen:

- **Precisión:** Proporción de documentos relevantes recuperados respecto al total de documentos recuperados.
- **Recall:** Proporción de documentos relevantes recuperados respecto al total de documentos relevantes disponibles.

- **F1-Score:** La media armónica de la precisión y el recall, proporcionando un equilibrio entre ambas métricas.

4. Resultados de la Evaluación

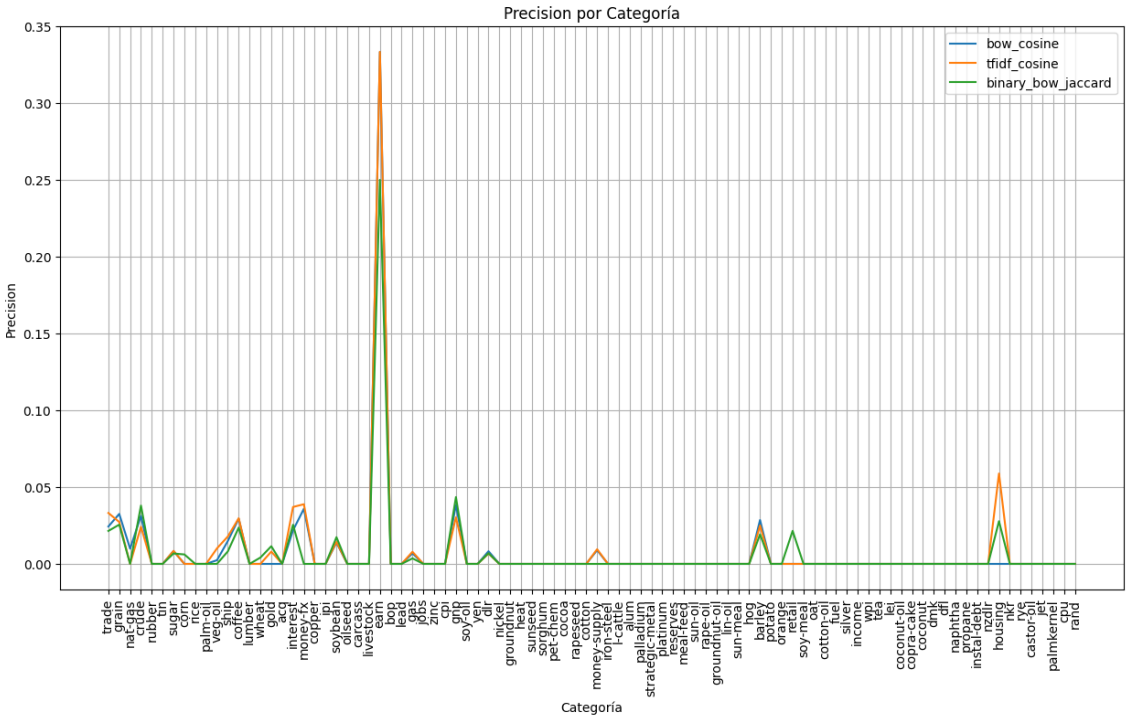
4.1. Configuraciones Evaluadas

Se evaluaron varias configuraciones del SRI utilizando las técnicas de vectorización y algoritmos de similitud descritos anteriormente.

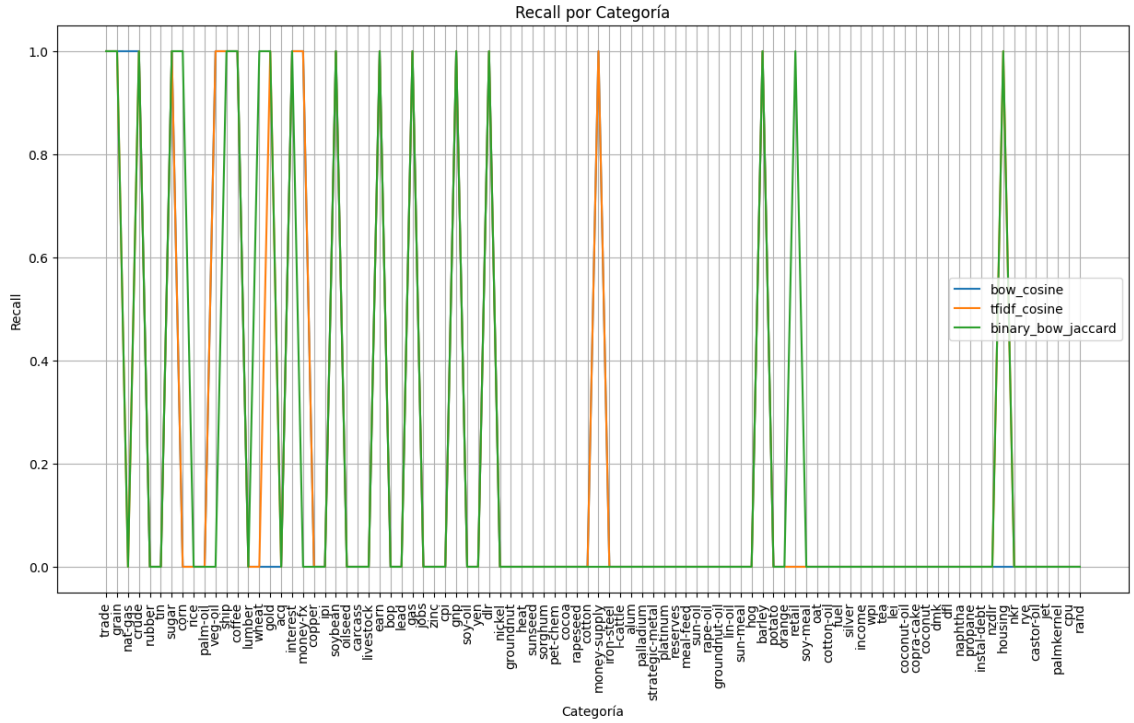
4.2. Resultados Obtenidos

Los resultados se presentan en las siguientes tablas y gráficos, mostrando las métricas de evaluación para cada configuración:

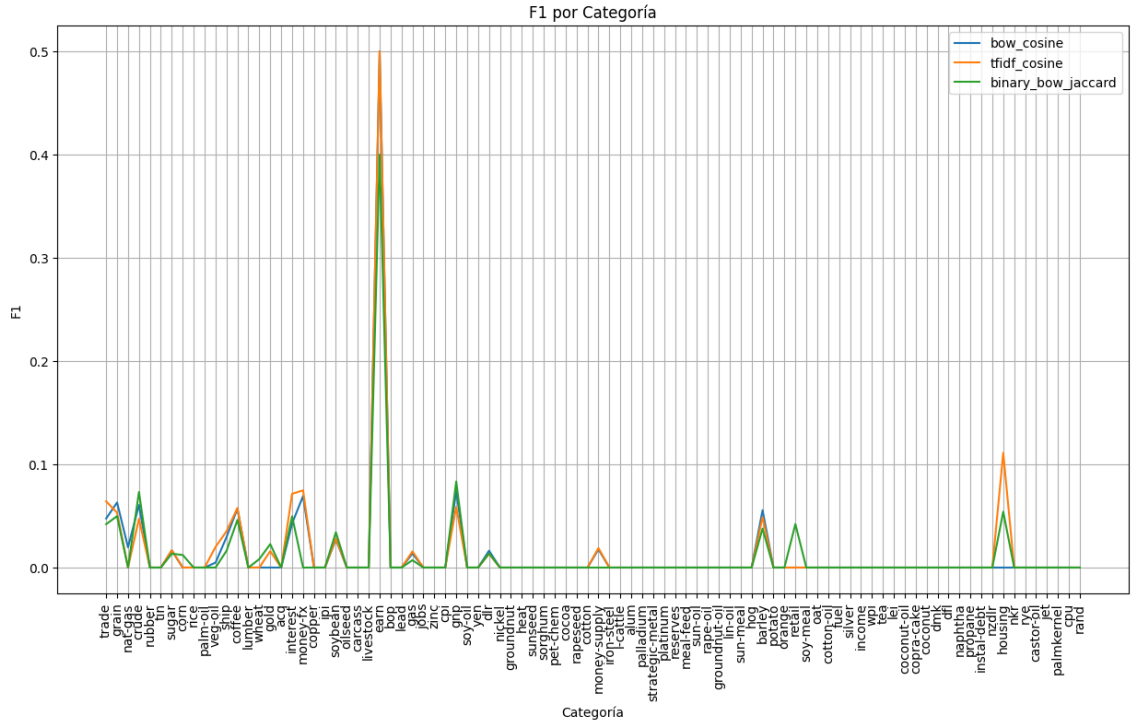
Precisión por Categoría



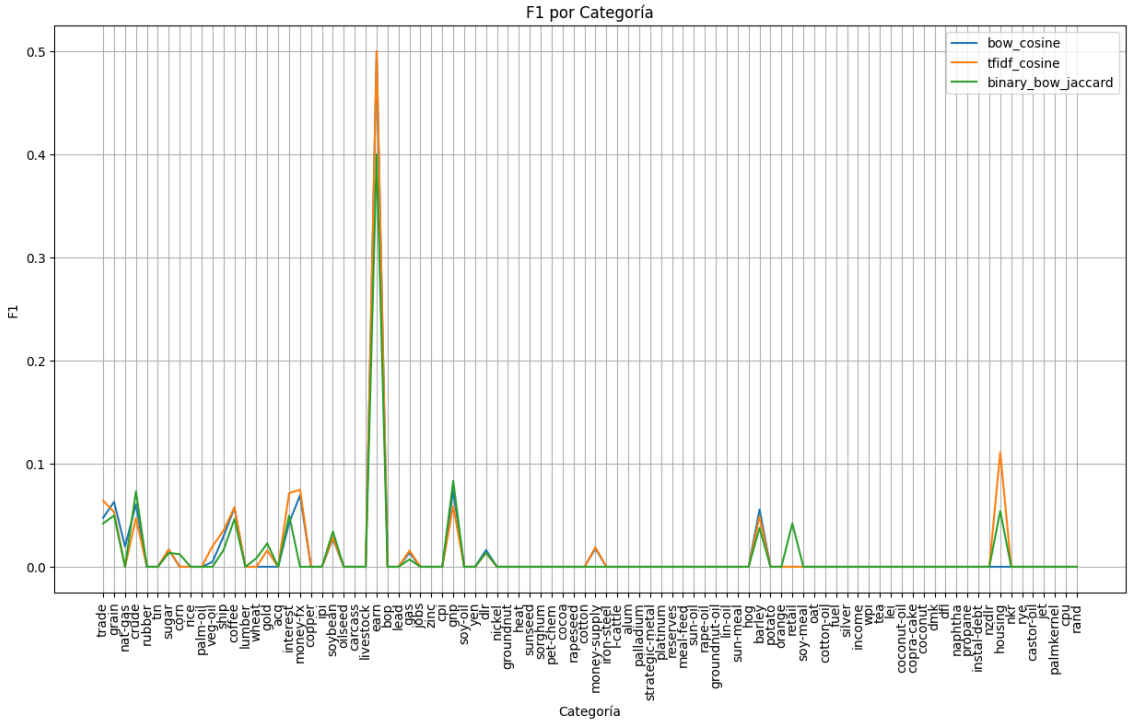
Recall por Categoría



F1 por Categoría



Métricas Promedio por Método



4.3. Comparación y Análisis

- **BoW - Coseno:** Esta 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.

4.4. Decisión tomada

La elección de TF-IDF se fundamenta en su capacidad para proporcionar un equilibrio óptimo entre precisión y recall, su alto F1-Score y su habilidad para representar de manera más efectiva la importancia de los términos en el corpus. Estos factores combinados hacen de TF-IDF la técnica más adecuada para el sistema de recuperación de información basado en el corpus Reuters-21578, asegurando una recuperación de información más relevante y precisa para los usuarios.

5. Conclusiones

5.1. Resumen de los Hallazgos

La evaluación del SRI demostró que la configuración basada en TF-IDF y similitud coseno proporciona el mejor rendimiento en términos de precisión, recall y F1-score.

5.2. Implicaciones de los Resultados

Estos resultados indican que ponderar las palabras según su importancia relativa en el corpus (TF-IDF) y medir la similitud de coseno entre los vectores proporciona una mejor recuperación de información relevante.

5.3. Sugerencias para Futuras Mejoras

- **Mejora del Preprocesamiento:** Experimentar con técnicas avanzadas de preprocesamiento como la eliminación de palabras raras o la detección de entidades nombradas.
- **Ajuste de Parámetros:** Investigar el impacto de diferentes umbrales y parámetros en los algoritmos de similitud.
- **Enriquecimiento del Corpus:** Incorporar más datos de entrenamiento y evaluar el impacto en la efectividad del SRI.