

FACULTAD DE INGENIERÍA DE SISTEMAS
CARRERA DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN



ESCUELA
POLITÉCNICA
NACIONAL

ICCD753 – RECUPERACIÓN DE LA
INFORMACIÓN

SISTEMA DE RECUPERACIÓN DE INFORMACIÓN

INTEGRANTES:

Diego Arias
Sergio Guaman
Anthony Vargas

DOCENTE:

Prof. Iván Carrera

FECHA DE ENTREGA:

2 de diciembre del 2024

Contents

1	Objetivo	2
2	Introducción	2
3	Desarrollo del proyecto	2
3.1	Adquisición de datos	2
3.2	Preprocesamiento	3
3.2.1	Limpieza y normalización de texto	4
3.2.2	Preprocesamiento de consultas	5
3.3	Representación de Datos en Espacio Vectorial	6
3.3.1	Transformación de Datos	6
3.3.2	Evaluación de Técnicas de Vectorización	7
3.4	Construcción del Índice Invertido e Indexación	9
3.4.1	Índice Invertido con TF-IDF	9
3.4.2	Índice Invertido con Word2Vec	10
3.4.3	Construcción del Índice e Inspección	10
3.5	Diseño del Motor de Búsqueda	10
3.6	Evaluación del sistema	11
3.6.1	Función para Calcular Métricas de Evaluación	11
3.6.2	Evaluación para TF-IDF	12
3.6.3	Evaluación para Word2Vec	12
3.7	Interfaz web de usuario	13
4	Conclusión	14

1 Objetivo

El propósito de este proyecto es diseñar un sistema de recuperación de información basado en la base de datos Reuters-21578, que permita realizar consultas eficientes y precisas sobre documentos utilizando técnicas avanzadas de procesamiento de lenguaje natural.

2 Introducción

En el contexto de la recuperación de información, la necesidad de sistemas que procesen grandes volúmenes de datos de texto de manera eficiente se ha incrementado significativamente. Este proyecto aborda la construcción de un sistema de recuperación de información a partir de un corpus bien conocido, aplicando técnicas modernas de procesamiento y representación.

3 Desarrollo del proyecto

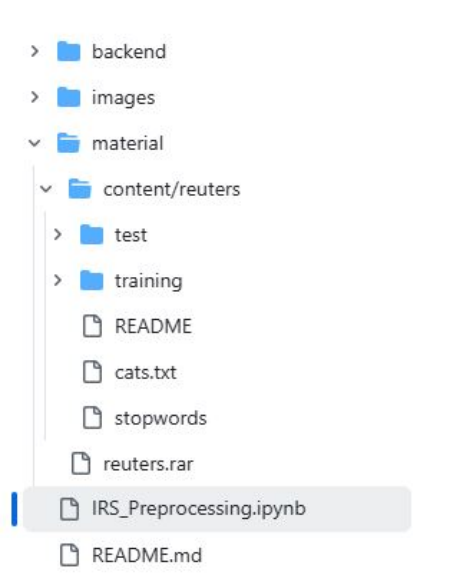
3.1 Adquisición de datos

```
# Librerías necesarias
import os
import re
import nltk
import rarfile
import time # Evaluar tecnicas de vectorización
import psutil # Evaluar tecnicas de vectorización
import gc # Evaluar tecnicas de

# Directorios de documentos
train_dir = 'material/content/reuters/training'
test_dir = 'material/content/reuters/test'
cats_file = 'material/content/reuters/cats.txt'

# Función para extraer contenido
```

Además se va a analizar los archivos que se tiene para poder seguir con el proyecto establecido



3.2 Preprocesamiento

El preprocesamiento es el primer paso en el análisis de texto y se encarga de limpiar, normalizar y estructurar los datos textuales para hacerlos aptos para el procesamiento. Esto incluye la eliminación de caracteres no deseados, la tokenización, la normalización del texto y la eliminación de stop words. Se describe un proceso para descomprimir un archivo RAR que contiene un conjunto de datos (Reuters) y estructurar su contenido en un formato utilizable para análisis posterior. Primero, se configura la herramienta UnRAR para extraer el contenido del archivo RAR y se especifican los directorios de salida. Tras extraer los archivos, se cargan las categorías de los documentos desde un archivo separado, asociando cada documento con su correspondiente etiqueta. Esto se realiza mediante la función cargar categorías, que convierte las líneas del archivo de categorías en un diccionario, y la función extraer texto, que limpia y obtiene el texto de cada archivo de documento.

Posteriormente, se procesan los documentos de entrenamiento y prueba para construir un conjunto de datos consolidado. Las funciones cargar documentos y extraer texto se utilizan para leer el contenido de los archivos y asociar cada documento con sus categorías. Finalmente, los datos se almacenan en DataFrames de pandas, lo que permite analizar el texto y las etiquetas de forma eficiente. Este procedimiento asegura la correcta organización de los datos en un formato estructurado y prepara el camino para su análisis o procesamiento adicional. Se verifica que los documentos se hayan cargado correctamente mostrando algunos ejemplos de texto y categorías en consola.

```
# Directorios de documentos
train_dir = 'material/content/reuters/training'
test_dir = 'material/content/reuters/test'
cats_file = 'material/content/reuters/cats.txt'

# Función para extraer contenido de un archivo de noticias
def extraer_texto(filepath):
    try:
        with open(filepath, 'r', encoding='latin-1') as file:
            contenido = file.read()
            texto_limpio = contenido.strip() # Elimino espacios en blanco iniciales y finales
            return texto_limpio
    except Exception as e:
        print(f"Error al leer el archivo {filepath}: {e}")
        return ""

# Función para cargar las categorías de los documentos
def cargar_categorias(filepath):
    categorias = {}
    try:
        with open(filepath, 'r', encoding='latin-1') as file:
            for linea in file:
                partes = linea.strip().split() # Divide la línea en partes separadas por espacios
                if len(partes) >= 2:
                    doc_id = partes[0] # Primer elemento es el id del documento (test/14826, training/1)
                    etiquetas = partes[1:] # Resto de elementos son categorías
                    categorias[doc_id] = etiquetas
    except Exception as e:
        print(f"Error al leer el archivo de categorías {filepath}: {e}")
        return categorias

# Función para cargar los documentos y asociar categorías
def cargar_documentos(directorio, tipo, categorias_dict):
    data = []
    archivos = os.listdir(directorio)
    if not archivos:
        print(f"No se encontraron archivos en {directorio}")
    for archivo in archivos:
        filepath = os.path.join(directorio, archivo)
        doc_id = f"{tipo}/{archivo}"
        texto = extraer_texto(filepath)
        categorias = categorias_dict.get(doc_id, []) # Obtener categorías, si no hay, devuelve lista vacía
        # Añadir a la lista para crear el dataframe
        data.append({"doc_id": doc_id, "texto": texto, "categorias": categorias})
    return data
```

3.2.1 Limpieza y normalización de texto

En el preprocesamiento del texto, primero se limpió el contenido de los documentos mediante una función que convierte el texto a minúsculas, elimina caracteres especiales y números, y remueve espacios adicionales. Esto garantizó un formato homogéneo y limpio. Luego, se utilizó la tokenización para dividir el texto en palabras individuales (tokens) empleando la herramienta **NLTK**, lo que facilitó la representación y manipulación de los datos en el análisis posterior.

Posteriormente, se eliminaron las *stop words* utilizando una lista personalizada cargada desde un archivo externo. Esto redujo el ruido en el análisis al excluir palabras comunes sin valor semántico relevante, como preposiciones y conectores. Además, se realizó un análisis visual para comparar el impacto de este proceso en la longitud de los textos antes y después de la eliminación de *stop words*, evidenciando una significativa reducción de palabras sin comprometer la relevancia de los datos.

En conjunto, estas etapas optimizaron la preparación de los datos para el análisis. Desde limpiar y tokenizar el texto hasta eliminar palabras irrelevantes, el preprocesamiento aseguró que la información clave de los documentos se mantuviera accesible y estructurada, facilitando pasos posteriores como la clasificación o el modelado de datos.

```
# Cargar el archivo de stopwords proporcionado
ruta_stopwords = 'material/content/reuters/stopwords'
stopwords_personalizadas = set()

with open(ruta_stopwords, 'r') as archivo:
    for linea in archivo:
        palabra = linea.strip() # Removemos espacios y saltos de línea
        if palabra: # Evitamos añadir líneas vacías
            stopwords_personalizadas.add(palabra.lower())

# Verificamos algunas stop words cargadas
print("Ejemplo de Stop Words cargadas:", list(stopwords_personalizadas)[:15])
```

Ejemplo de Stop Words cargadas: ['ie', 'nevertheless', 'anyways', 'beforehand', 'ask', 'who's', 'believe', 'zero', 'according', 'very', 'seen', 'has', 'saying', 'nd', 'wasn't']

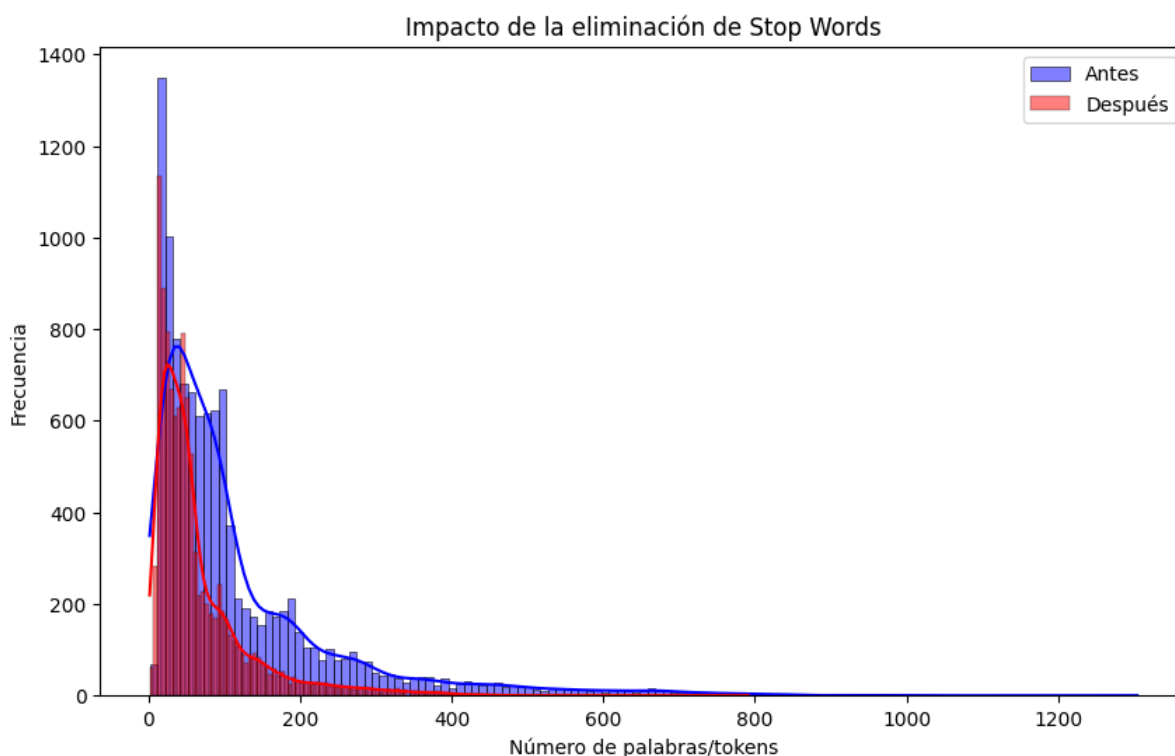
```
# Función para eliminar stop words de los tokens
def eliminar_stopwords(tokens, stopwords):
    # Filtramos los tokens que no están en la lista de stopwords
    tokens_filtrados = [token for token in tokens if token.lower() not in stopwords]
    return tokens_filtrados

# Aplicamos la eliminación de stopwords a la columna 'tokens' del dataframe
df_documentos['tokens'] = df_documentos['tokens'].apply(lambda x: eliminar_stopwords(x, stopwords_personalizadas))

# Verificar el resultado después de eliminar stop words en un documento de ejemplo
ejemplo_doc = df_documentos.iloc[0] # Obtener la primera fila del dataframe
print(f"ID: {ejemplo_doc['doc_id']}")
print(f"Tokens sin Stop Words: {ejemplo_doc['tokens'][:20]}...")
```

Activar Windows
Ve a Configuración para activar W

Luego se analizó el impacto de la eliminación de los stopwords teniendo esta gráfica como resultado



3.2.2 Preprocesamiento de consultas

Para nuestro caso de estudio y aplicación de este proyecto, aplicaremos Stemming.

Como equipo, consideramos más eficiente en términos de recursos porque el stemming utiliza reglas más simples y rápidas para reducir las palabras a su raíz, mientras que la lematización requiere un procesamiento más complejo y el uso de diccionarios para

obtener la forma base correcta de las palabras. A través de la función aplicar stemming, se procesa cada token en la columna tokens del dataframe df documentos, y el resultado es almacenado nuevamente en esta columna. Finalmente, se verifica el resultado mostrando los primeros 20 tokens procesados de un documento de ejemplo.

```
# Inicializar el stemmer
stemmer = PorterStemmer()

# Función para aplicar stemming a los tokens de un documento
def aplicar_stemming(tokens):
    return [stemmer.stem(token) for token in tokens]

# Aplicar el stemming a la columna 'tokens' del dataframe
df_documentos['tokens'] = df_documentos['tokens'].apply(aplicar_stemming)

# Verificar un documento de ejemplo después del stemming
ejemplo_doc = df_documentos.iloc[0] # Obtener la primera fila del dataframe
print(f"ID: {ejemplo_doc['doc_id']}")
print(f"Tokens con Stemming: {ejemplo_doc['tokens'][:20]}...")
```

3.3 Representación de Datos en Espacio Vectorial

En esta sección se abordan las técnicas utilizadas para representar los datos textuales en un espacio vectorial, con el fin de poder aplicar algoritmos de machine learning de manera efectiva. Se emplearon tres métodos comunes: Bag of Words (BoW), TF-IDF y Word2Vec. Estas técnicas permiten transformar el texto preprocesado en vectores numéricos que describen cada documento de manera cuantitativa.

3.3.1 Transformación de Datos

Bag of Words (BoW) Para convertir el texto preprocesado en una representación vectorial utilizando la técnica de Bag of Words (BoW), primero se unificaron los tokens en un solo texto por documento. Posteriormente, se utilizó el `CountVectorizer` de `scikit-learn` para crear la matriz BoW, donde cada fila representa un documento y cada columna representa una palabra del vocabulario. Los valores en la matriz corresponden a la frecuencia de aparición de las palabras en cada documento.

El código inicializa el `CountVectorizer`, lo ajusta a los textos procesados y genera la matriz BoW. Luego se verifica la dimensión de la matriz resultante, que tiene como tamaño el número de documentos por el número de palabras únicas en todo el corpus. Se obtiene y se muestra el vocabulario, así como un vector de ejemplo correspondiente al primer documento. Esta técnica es rápida y útil para representar el texto de manera simple, pero no captura información contextual ni semántica entre las palabras.

TF-IDF (Term Frequency - Inverse Document Frequency) La técnica TF-IDF se utiliza para ponderar la importancia de las palabras en un documento en función de su frecuencia en el corpus general. La idea detrás de TF-IDF es que las palabras frecuentes en un solo documento, pero raras en otros, tienen mayor peso. Para aplicar esta técnica, se emplea el `TfidfVectorizer` de `scikit-learn`, el cual también

genera una matriz de vectores para cada documento, similar a BoW, pero con valores ponderados.

El código aplica el vectorizador de TF-IDF a los documentos procesados y muestra la dimensión de la matriz resultante, así como las primeras 10 palabras del vocabulario. Al igual que en BoW, se inspecciona el vector correspondiente al primer documento, pero en este caso los valores no son simples conteos de palabras, sino ponderaciones que reflejan la importancia relativa de las palabras en el conjunto de documentos. Esta técnica es más precisa que BoW, ya que toma en cuenta la distribución de las palabras en el corpus entero.

Word2Vec La técnica Word2Vec genera representaciones densas de palabras en un espacio vectorial continuo, capturando relaciones semánticas entre las palabras. Para aplicar Word2Vec, se utiliza el modelo **Word2Vec** de la librería **gensim**, que aprende representaciones vectoriales de las palabras a partir de sus contextos en los textos. En este caso, el modelo es entrenado usando los tokens procesados de los documentos.

Después de entrenar el modelo, se obtiene un vector promedio para cada documento. Este vector es el resultado de promediar los vectores de las palabras que aparecen en el documento, lo que permite representar cada documento como un solo vector en el espacio vectorial. La matriz resultante tiene como dimensiones el número de documentos por la cantidad de dimensiones del vector de palabras (en este caso, 100). Se verifica la dimensión de la matriz de vectores promedio y se inspecciona el vector correspondiente al primer documento. Word2Vec es útil para capturar la semántica de las palabras, ya que las palabras con significados similares tienden a tener representaciones vectoriales cercanas.

3.3.2 Evaluación de Técnicas de Vectorización

Con el fin de comparar el rendimiento de las tres técnicas de vectorización (BoW, TF-IDF y Word2Vec), se implementó una evaluación de tiempo de ejecución y uso de memoria para cada técnica. Para esto, se definió una función `medir_tiempo_memoria`, que mide el tiempo de ejecución y la memoria utilizada por una función dada. La función utiliza la librería `psutil` para rastrear el uso de memoria y `time.time()` para medir el tiempo.

Se evaluaron las tres técnicas de vectorización:

- **BoW:** Se aplica el `CountVectorizer` para generar la matriz BoW. Esta técnica es rápida y eficiente en cuanto a memoria, ya que solo cuenta las frecuencias de las palabras en cada documento.
- **TF-IDF:** Se aplica el `TfidfVectorizer`, el cual no solo cuenta las palabras, sino que pondera su frecuencia en función de su aparición en otros documentos. Esta técnica es más lenta que BoW debido al cálculo adicional de los pesos, pero ofrece mejores resultados en términos de importancia de las palabras.
- **Word2Vec:** Se entrena un modelo Word2Vec utilizando los tokens procesados, y luego se calcula el vector promedio para cada documento. Aunque Word2Vec es más preciso en términos de representación semántica, su tiempo de ejecución es significativamente mayor, ya que entrena un modelo sobre el corpus entero.

La evaluación muestra que BoW es la técnica más rápida y eficiente en términos de uso de memoria, lo que la convierte en la opción preferida cuando se requiere un

procesamiento rápido. TF-IDF ofrece un buen equilibrio entre precisión y eficiencia, mientras que Word2Vec, a pesar de ser más lento, captura relaciones semánticas más complejas, lo que lo hace más adecuado para tareas que requieren mayor profundidad semántica en las representaciones de texto.

```
# Convertir tokens preprocesados a texto unificado para vectorización
df_documentos['texto_procesado'] = df_documentos['tokens'].apply(lambda x: ' '.join(x))

# Inicializar el vectorizador BoW
bow_vectorizer = CountVectorizer()

# Aplicar BoW a los textos procesados
bow_matrix = bow_vectorizer.fit_transform(df_documentos['texto_procesado'])

# Verificar la matriz resultante
print("Dimensiones de la matriz BoW:", bow_matrix.shape)

# Mostrar las primeras 10 palabras del vocabulario
vocabulario_bow = bow_vectorizer.get_feature_names_out()
print("Primeras 10 palabras del vocabulario BoW:", vocabulario_bow[:10])

# Inspeccionar una fila de la matriz BoW
print("Vector BoW del primer documento:")
print(bow_matrix[0].toarray())
```

```
# Aplicar TF-IDF a los textos procesados
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(df_documentos['texto_procesado'])

# Verificar la matriz resultante
print("Dimensiones de la matriz TF-IDF:", tfidf_matrix.shape)

# Mostrar las primeras 10 palabras del vocabulario
vocabulario_tfidf = tfidf_vectorizer.get_feature_names_out()
print("Primeras 10 palabras del vocabulario TF-IDF:", vocabulario_tfidf[:10])

# Inspeccionar una fila de la matriz TF-IDF
print("Vector TF-IDF del primer documento:")
print(tfidf_matrix[0].toarray())
```

```
# Crear el modelo Word2Vec
model_w2v = Word2Vec(sentences=df_documentos['tokens'], vector_size=100, window=5, min_count=1, workers=4)

# Función para obtener el vector promedio de un documento
def obtener_vector_promedio(tokens, model):
    # Inicializamos un vector de ceros
    vector_promedio = np.zeros(model.vector_size)
    count = 0
    for token in tokens:
        if token in model.wv:
            vector_promedio += model.wv[token]
            count += 1
    if count > 0:
        vector_promedio /= count # Promediamos si hay tokens en el modelo
    return vector_promedio

# Obtener los vectores promedio para todos los documentos
documentos_w2v = []
for tokens in df_documentos['tokens']:
    vector = obtener_vector_promedio(tokens, model_w2v)
    documentos_w2v.append(vector)

# Convertimos la lista a una matriz
documentos_w2v_matrix = np.array(documentos_w2v)

# Verificar el resultado
print("Dimensiones de la matriz Word2Vec:", documentos_w2v_matrix.shape)

# Inspeccionar el primer vector de documento
print("Vector Word2Vec del primer documento:")
print(documentos_w2v_matrix[0])
```

Activar W
Ve a Configu

3.4 Construcción del Índice Invertido e Indexación

En este apartado, se describe el proceso de construcción del índice invertido utilizando dos técnicas de vectorización: TF-IDF y Word2Vec. El índice invertido es una estructura de datos eficiente que asocia términos con los documentos en los que aparecen, permitiendo realizar búsquedas rápidas y análisis de texto.

3.4.1 Índice Invertido con TF-IDF

El primer método utilizado para construir el índice invertido es mediante la técnica de TF-IDF. Este índice relaciona cada término con los documentos en los que aparece, junto con su peso o importancia, que es calculado utilizando la ponderación de TF-IDF. El código comienza por inicializar un diccionario de listas `indice_invertido`, donde las claves son los términos (palabras) y los valores son listas de tuplas, donde cada tupla contiene el identificador del documento (`doc_id`) y el peso del término en ese documento.

Para construir el índice invertido, el código sigue estos pasos:

- Primero, se obtiene el vocabulario del `TfidfVectorizer` con el método `get_feature_names_out()`, que contiene todos los términos en el corpus.
- Luego, se convierte la matriz de vectores `X_vectores` (obtenida del `TfidfVectorizer`) en un arreglo denso utilizando el método `toarray()`.
- Se recorren los documentos y sus correspondientes vectores. Para cada documento y cada término en el vector, si el peso (valor en el vector) es positivo, se agrega una entrada al índice invertido con el término, el `doc_id` y el peso.

Este proceso asegura que solo los términos relevantes (con peso mayor a cero) sean indexados, lo que hace que el índice sea más eficiente y maneje mejor la memoria.

3.4.2 Índice Invertido con Word2Vec

El segundo método de construcción del índice invertido se realiza utilizando la técnica de Word2Vec. En este caso, el índice invertido no se basa en términos específicos como en TF-IDF, sino que se genera a partir de los vectores promedio de los documentos, calculados a partir de los vectores de las palabras en el documento.

El código funciona de la siguiente manera:

- Se asume que el modelo Word2Vec ya está entrenado y que se pasa como el parámetro `vectorizador`.
- Se define una función auxiliar `obtener_vector_promedio`, que recibe los tokens de un documento y calcula el promedio de los vectores de las palabras en ese documento, utilizando el modelo Word2Vec preentrenado. Si una palabra no está en el modelo, se ignora.
- Se recorre cada documento y sus tokens. Para cada documento, se obtiene el vector promedio de sus palabras.
- Similar a TF-IDF, si algún valor en el vector promedio es mayor a cero, se agrega al índice invertido, pero esta vez el "término" se define como `word2vec_idx`, donde `idx` es el índice de la componente del vector. Esto se hace para representar las componentes del vector en lugar de términos de palabras.

Este método crea un índice invertido que mapea cada componente del vector de Word2Vec (en lugar de palabras individuales) a los documentos en los que la componente tiene un valor significativo.

3.4.3 Construcción del Índice e Inspección

El código muestra cómo construir el índice invertido para ambos métodos:

- Primero, se construye el índice invertido utilizando la técnica de TF-IDF. Esto se realiza llamando a la función `construir_indice_invertido_vectorizado` con el `TfidfVectorizer` y la matriz `tfidf_matrix`.
- Luego, se imprime una muestra de los primeros términos del índice invertido, mostrando solo los primeros dos términos para evitar una salida excesiva.

La construcción del índice invertido con TF-IDF y Word2Vec permite optimizar las búsquedas de términos en grandes colecciones de documentos, al tiempo que se aprovechan las características particulares de cada técnica de vectorización.

Este enfoque es útil para sistemas de recuperación de información y análisis de texto, ya que la indexación permite acceder rápidamente a los documentos que contienen ciertos términos (o componentes vectoriales en el caso de Word2Vec) de interés.

3.5 Diseño del Motor de Búsqueda

El diseño de un motor de búsqueda efectivo implica varios pasos clave para procesar y recuperar documentos relevantes según las consultas de los usuarios. Uno de los componentes esenciales es el procesamiento adecuado de la consulta, que incluye la limpieza del texto, tokenización, eliminación de stopwords y aplicación de técnicas de stemming. La

función `preprocesar_consulta` realiza estos pasos, permitiendo transformar la consulta del usuario en una lista de tokens que serán utilizados en los algoritmos de similitud. Para mejorar la precisión de la búsqueda, también se utiliza un modelo de Word2Vec para obtener una representación vectorial semántica de la consulta, facilitando una comparación más rica entre la consulta y los documentos en términos de contexto semántico.

Una vez preprocesada la consulta, se emplean algoritmos de similitud, como la similitud coseno, para comparar la consulta con los documentos. El enfoque de TF-IDF mide la similitud entre documentos y consulta en función de la frecuencia ponderada de los términos en el corpus. En contraste, Word2Vec compara las consultas y documentos en un espacio vectorial continuo, donde la similitud coseno mide la cercanía semántica entre los vectores de palabras. Ambos métodos permiten determinar qué documentos son más relevantes para una consulta, aunque cada uno tiene un enfoque distinto: TF-IDF se centra en la frecuencia de términos, mientras que Word2Vec captura relaciones semánticas y contextuales.

Finalmente, para ordenar los resultados, se aplica un algoritmo de ranking. En el caso de TF-IDF, los documentos se ordenan según la similitud coseno, mientras que en Word2Vec, la similitud entre los vectores promedio de las palabras también se usa para determinar el ranking. Estos algoritmos aseguran que los documentos más relevantes se presenten primero, mejorando la experiencia del usuario. La diferencia principal entre ambos enfoques radica en su capacidad para manejar la semántica del lenguaje, donde Word2Vec tiene una ventaja al considerar relaciones contextuales entre términos, mientras que TF-IDF se enfoca más en la frecuencia y relevancia de las palabras clave.

3.6 Evaluación del sistema

En esta sección, se calculan las métricas de evaluación, como la precisión, el recall y el F1-score, a partir de los resultados obtenidos utilizando los métodos TF-IDF y Word2Vec.

3.6.1 Función para Calcular Métricas de Evaluación

Se define una función llamada `calcular_metricas` que recibe dos listas: `resultados` y `relevantes`.

- **resultados:** Los documentos recuperados por el sistema para una consulta.
- **relevantes:** Los documentos que son realmente relevantes según el umbral de similitud.

La función realiza los siguientes pasos:

1. Convierte `resultados` y `relevantes` en conjuntos (sets) para eliminar duplicados y facilitar el cálculo de intersección.
2. Calcula la intersección entre `resultados` y `relevantes`, denominada `relevantes_recuperados`.
3. Calcula la **precisión**, que es la proporción de documentos recuperados que son relevantes:

$$\text{precisión} = \frac{\text{len}(\text{relevantes_recuperados})}{\text{len}(\text{resultados})} \quad \text{si los resultados no están vacíos.}$$

4. Calcula el **recall**, que es la proporción de documentos relevantes que fueron recuperados:

$$\text{recall} = \frac{\text{len}(\text{relevantes_recuperados})}{\text{len}(\text{relevantes})} \quad \text{si los relevantes no están vacíos.}$$

5. Calcula el **F1-score**, que es la media armónica de la precisión y el recall:

$$F1 = \frac{2 \times (\text{precisión} \times \text{recall})}{\text{precisión} + \text{recall}} \quad \text{si precisión} + \text{recall} \neq 0.$$

En caso contrario, se asigna un valor de 0.

La función retorna un diccionario con los valores de precisión, recall y F1-score.

3.6.2 Evaluación para TF-IDF

Para el método TF-IDF, se realiza lo siguiente:

- Se define un umbral de similitud (`umbral`).
- Se determinan los documentos relevantes según el umbral de similitud en el ranking de TF-IDF.
- Se calculan las métricas (precisión, recall y F1-score) utilizando la función `calcular_metricas`.

Las métricas obtenidas para cada consulta se muestran en la siguiente tabla (por ejemplo, para la consulta "cocoa"):

Consulta	Precisión	Recall	F1-score
cocoa	0.6282	1.0000	0.7717
yen	0.2660	1.0000	0.4203
⋮	⋮	⋮	⋮

3.6.3 Evaluación para Word2Vec

Para el modelo Word2Vec, el procedimiento es similar:

- Se define un umbral de similitud para Word2Vec (`umbral_w2v`).
- Se determinan los documentos relevantes según el umbral de similitud en el ranking de Word2Vec.
- Se calculan las métricas (precisión, recall y F1-score) utilizando la misma función `calcular_metricas`.

Las métricas obtenidas para cada consulta se muestran en la siguiente tabla (por ejemplo, para la consulta "cocoa"):

Consulta	Precisión	Recall	F1-score
cocoa	0.1916	1.0000	0.3216
yen	0.2225	1.0000	0.3641
⋮	⋮	⋮	⋮

3.7 Interfaz web de usuario

Lo que se hace es definir una interfaz para que el usuario busque por diferentes métodos de búsqueda y la cantidad de resultados que se desea apreciar

Poli-Buscador

Ingrese su consulta:

earn

Método de búsqueda:


☐ TF-IDF

☒ Word2Vec

Número de resultados:

10

Buscar


Procesando su consulta, por favor espere...

Resultado:

Ingrese su consulta:

earn

Método de búsqueda:

☒ TF-IDF

☐ Word2Vec

Número de resultados:

10

Buscar

Resultado:

Documento ID: training/12390

GALVESTON TO ACQUIRE MINE PROPERTY INTERESTS <Galveston Resources Ltd> said it agreed in principle for an option to earn up to a 50 pct interest from <Hemlo Gold Mines Inc> in certain mining properties known as the Interlake Property, subject to regulatory approvals. Galveston said it will earn up to a 50 pct interest by spending a minimum of one mln dlrs in exploration and development work by December 31, 1989. It expects work will commence shortly and continue during the 1987 exploration season. Galveston also said it granted Hemlo Gold options to acquire up to two mln Galveston shares. It said the options can be exercised at 10 dlrs a share up to December 31, 1987, then at 12.50 dlrs a share until December 31, 1988, and then at 15 dlrs a share until December 31, 1989. Separately, Galveston said it agreed in principle with <Noranda Inc> unit Noranda Explorations Ltd for an option to earn up to a 50 pct interest in a major mineral property located at the Bale Verte Peninsula, Newfoundland. Galveston can earn a 50 pct interest by spending six mln dlrs in exploration and development work on the property by December 31, 1989. The company also said it granted Noranda an option to purchase two mln Galveston shares. The options can be exercised at 10 dlrs a share until December 31, 1987, then at 12.50 dlrs a share until December 31, 1988, and then at 15 dlrs a share until December 31, 1989.

Documento ID: test/19409

PACIFIC BASIN TO ACQUIRE 51 PCT OF T.E.A.M. <Pacific Basin Development Corp>, based in Vancouver, British Columbia, said it reached an agreement to buy 51 pct of T.E.A.M. Pacific Corp and its marketing arm for 4.2 mln U.S. dlrs. Pacific also said it expects to earn three mln Canadian dlrs for the year ended June 30, 1988 and 10 mln Canadian dlrs for the year ended June 1989. T.E.A.M., a former Signetics Corp unit, assembles integrated circuits in Southeast Asia and is itself buying an assembler. Pacific said T.E.A.M. expects to earn over 80 mln Canadian dlrs per year when the acquisition is completed.

Poli-Buscador

Ingrese su consulta:

earn

Método de búsqueda:

☐ TF-IDF

☒ Word2Vec

Número de resultados:

10

Buscar

Resultado:

Documento ID: training/1434

GEONEX SEES SALES HURT BY PRODUCTION SHIFT Geonex Corp <GEOX> said BellSouth Corp <BLS> unit Southern Bell Telephone and Telegraph Co's decision to postpone the start up of new conversion assignments at Geonex's Chicago Aerial Survey unit could negatively affect its fiscal 1987 revenues. The company said it had expected higher revenues from the records conversion work, but it now foresees revenues from Southern Bell work at about eight mln dlrs, the same level as last fiscal year. Geonex said Southern Bell will let CAS continue work in progress and it expects to perform mechanized posting and records conversion for Southern Bell through 1989. But, it added that the Southern Bell decision has forced it to pursue opportunities with other telephone companies and municipalities to replace the Southern Bell work.

Documento ID: training/8940

KODAK <EK> TO CUT POLYESTER FIBER OPERATIONS Eastman Kodak Co said it will reduce capacity and employment levels in two polyester fiber operations of its Eastman Chemicals division. A company spokesman said the company will take "some writeoff" in connection with the action in the first quarter and there will probably be a further "carryover" writeoff in the second quarter. The writeoffs will cover the costs of plants and equipment involved, as well as expenses connected with the staff cuts. Kodak said the division will discontinue production of polyester partially-oriented filament yarn, or POY, at its Carolina Eastman Co plant in Columbia, S.C., and will idle 100 mln pounds of older

Poli-Buscador

Ingrese su consulta:

cocoa

Método de búsqueda:

☒ TF-IDF

☐ Word2Vec

Número de resultados:

1

Buscar

Resultado:

Documento ID: test/20005

ASIAN COCOA PRODUCERS EXPAND DESPITE CRITICS Asian cocoa producers are expanding output despite depressed world prices and they dismiss suggestions in the London market that their cocoa is inferior. "Leading cocoa producers are trying to protect their market from our product," said a spokesman for Indonesia's directorate general of plantations. "We're happy about our long-term future." Malaysian growers said they would try to expand sales in Asia and the United States if Malaysian cocoa was not suitable for European tastes. They were responding to comments by London traders that large tonnages of unwanted cocoa beans from Malaysia, Indonesia and Papua New Guinea (PNG) were helping to depress cocoa prices. London traders said the Asian cocoa was considered unsuitable for western palates because of an acrid odour and a high level of free fatty acids. Ng Siew Kee, the chairman of Malaysia's Cocoa Growers' Council, said Malaysia should expand its sales to Asia and the United States if it did not produce a type suitable for Western Europe. A spokesman for the PNG Cocoa Industry Board said the London market was mistaken if it linked PNG cocoa with high-acid Malaysian and Indonesian beans. "When the market is declining, buyers seize on anything to talk down prices," the spokesman said. He said that PNG could sell whatever cocoa it produces. PNG exported 33,000

4 Conclusión

El proyecto demostró la efectividad de aplicar técnicas avanzadas de procesamiento de lenguaje natural para construir un sistema de recuperación de información basado en el corpus Reuters-21578. Desde la adquisición de datos hasta la evaluación del sistema, se implementaron procesos rigurosos, como la limpieza y normalización de texto, la tokenización y la eliminación de *stop words*. Estas etapas garantizaron una representación homogénea y estructurada de los datos, sentando las bases para una indexación eficiente y un análisis posterior.

En el desarrollo, se implementaron técnicas de vectorización como Bag of Words (BoW), TF-IDF y Word2Vec, cada una con sus respectivas ventajas. Mientras que BoW

y TF-IDF ofrecieron representaciones rápidas y fáciles de interpretar, Word2Vec destacó por capturar relaciones semánticas más complejas. Además, la evaluación del sistema permitió comparar estas técnicas, mostrando que TF-IDF es un balance adecuado entre precisión y eficiencia, mientras que Word2Vec sobresale en tareas que requieren un mayor entendimiento contextual.

Finalmente, la integración de un motor de búsqueda basado en similitud coseno y una interfaz de usuario funcional permitió realizar consultas eficaces sobre los documentos. Este sistema es extensible a otros corpus y adaptable a nuevos requisitos, posicionándose como una solución robusta y escalable en el ámbito de la recuperación de información. La metodología y los resultados obtenidos sientan una base sólida para futuras investigaciones y aplicaciones en sistemas de búsqueda avanzados.