



ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

Recuperación de la Información

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

INTEGRANTES:

Allan Adrian Molina Erraez
Dylan Santiago Villarroel Celi

Quito, Ecuador

Tabla de Ilustraciones

Ilustración 1 Archivos del Corpus descargados	4
Ilustración 2 Archivos descomprimidos y transformados	4
Ilustración 3 Librerías del preprocesamiento	4
Ilustración 4 Rutas de los elementos para preprocesamiento	5
Ilustración 5 Función para preprocesamiento	5
Ilustración 6 Llamada a los archivos y funciones de preprocesamiento	6
Ilustración 7 Cargar los Documentos pre-procesados	7
Ilustración 8 BoW Vectorización	7
Ilustración 9 TF-IDF Vectorización	8
Ilustración 10 Función Índice Invertido	9
Ilustración 11 Funcion que guarda el indice invertido	10
Ilustración 12 Función carga el Índice Invertido para usarlo	11
Ilustración 13 Resultado de TF_IDF	12
Ilustración 14 Resultado de BoW	13
Ilustración 15 Funciones de Similitud Jaccard y Coseno	13
Ilustración 16 Función de Procesamiento de Query y Función de ranking de resultados	15
Ilustración 17 Creacion de diccionario a partir del archivo de categorias	17
Ilustración 18 Se compara los resultados predichos con las etiquetas reales	17
Ilustración 19 Funciones para evaluar los resultados de BoW y TF-IDF	18
Ilustración 20 Matriz confusión BoW	19
Ilustración 21 Reporte clasificación BoW	19
Ilustración 22 Matriz confusión TF-IDF	20
Ilustración 23 Reporte clasificación TF-IDF	20
Ilustración 24 Librerías para App	21
Ilustración 25 Rutas para la App	21
Ilustración 26 Carga de Stopwords	21
Ilustración 27 Carga de datos pre-procesados	22
Ilustración 28 Vectorización BoW	22
Ilustración 29 Vectorización TF-IDF	22
Ilustración 30 Carga Índices Invertidos	23
Ilustración 31 Preprocesamiento de Querys	23
Ilustración 32 Función calcula similitud Jaccard	24
Ilustración 33 Función calcula similitud Coseno	24
Ilustración 34 Función de búsqueda BoW	24
Ilustración 35 Función búsqueda TF-IDF	25
Ilustración 36 Ruta de la página principal	25
Ilustración 37 Ruta y funciones de la página de Resultados	25
Ilustración 38. Ejecución inicial de la aplicación	26
Ilustración 39. Documento HTML de la página inicial	26

Ilustración 40. Documento HTML de la página de resultados	27
Ilustración 41. Hoja de estilos de la aplicación web.....	28
Ilustración 42. Página de inicio.....	29
Ilustración 43. Página de resultados	29

1. Adquisición de Datos Objetivo:

Obtener y preparar el corpus Reuters-21578.

- Se descargó todo el corpus de **Reuters-21578** de la siguiente página:
[ir24a/proj01/data/reuters.rar at main · ivan-carrera/ir24a \(github.com\)](https://github.com/ivan-carrera/ir24a/blob/main/proj01/data/reuters.rar)
- Posterior a la descarga se descomprimió el archivo rar con todas las carpetas necesarias para el proyecto

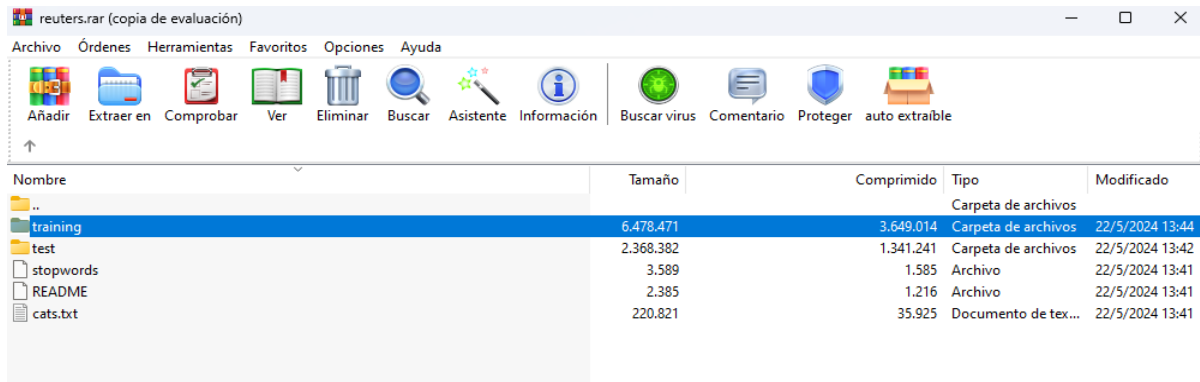


Ilustración 1 Archivos del Corpus descargados

training	12/6/2024 19:21	Carpeta de archivos	
cats	2/6/2024 16:59	Documento de te...	163 KB
stopwords	22/5/2024 13:41	Archivo	4 KB
stopwords	8/6/2024 23:27	Documento de te...	4 KB

Ilustración 2 Archivos descomprimidos y transformados

2. Preprocesamiento

Objetivo: Limpiar y preparar los datos para su análisis:

Para esta parte se decidió crear un script aparte para poder guardar todos los documentos ya preprocesados. Este archivo se llama **preprocessing.py**.

Importaciones de Librerías:

El código utiliza las siguientes librerías estándar de Python y de terceros:

```
import os
import re
from nltk.stem import PorterStemmer
```

Ilustración 3 Librerías del preprocesamiento

- `os` : Para la manipulación de rutas de archivos y creación de directorios.
- `re` : Para el manejo de expresiones regulares, utilizado en la limpieza de texto.
- `PorterStemmer` de `nltk.stem` : Un algoritmo de stemming que reduce palabras a su raíz base.

Definición de Rutas y Lectura de Stopwords:

```
# Definir la ruta del corpus y las stopwords en tu sistema local
REUTERS_PATH = "reuters"
STOPWORDS_PATH = os.path.join(REUTERS_PATH, "stopwords.txt")
TRAINING_PATH = os.path.join(REUTERS_PATH, "training")
PROCESSED_PATH = os.path.join(REUTERS_PATH, "processed")

# Leer las stopwords desde el archivo
with open(STOPWORDS_PATH, 'r', encoding='ascii') as file:
    stop_words = set(word.strip() for word in file.readlines())
```

Ilustración 4 Rutas de los elementos para preprocesamiento

Se definen rutas a directorios y archivos relevantes dentro del sistema de archivos local. Además, se lee un archivo de stopwords para su posterior uso en el preprocesamiento de texto.

- REUTERS_PATH: Ruta base donde se encuentran los datos de Reuters.
- STOPWORDS_PATH: Ruta al archivo de stopwords (`stopwords.txt`) dentro de la carpeta `reuters`.
- TRAINING_PATH: Ruta al directorio que contiene los documentos originales que se van a preprocesar.
- PROCESSED_PATH: Ruta al directorio donde se guardarán los documentos preprocesados después de aplicar el proceso de limpieza y stemming.
- stop_words: Conjunto de stopwords cargado desde el archivo `stopwords.txt` para su posterior eliminación en el preprocesamiento.

Función de Preprocesamiento (`imp`):

```
# Función de preprocesamiento
def imp(texto):
    # Normalización
    cleaned_text = re.sub(r'^\w\s', '', texto)
    cleaned_text = cleaned_text.lower()
    words = cleaned_text.split()
    # Stemming
    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(word) for word in words]
    # Eliminar stopwords
    filtered_words = [word for word in stemmed_words if word not in stop_words]
    cleaned_text = ' '.join(filtered_words)
    return cleaned_text
```

Ilustración 5 Función para preprocesamiento

La función `lmp` realiza varias etapas de procesamiento de texto sobre un texto de entrada:

1. Normalización: Elimina caracteres no alfanuméricos, convierte el texto a minúsculas y divide en palabras.
2. Stemming: Utiliza la librería “PorterStemmer” para reducir cada palabra a su forma base.
3. Eliminación de stopwords: Filtra las palabras que están en la lista de stopwords cargada previamente.

Preprocesamiento y Guardado de Documentos:

```
# Crear la carpeta si no existe para guardar los documentos preprocesados
if not os.path.exists(PROCESSED_PATH):
    os.makedirs(PROCESSED_PATH)

# Preprocesar y guardar documentos
for filename in os.listdir(TRAINING_PATH):
    input_filepath = os.path.join(TRAINING_PATH, filename)
    output_filepath = os.path.join(PROCESSED_PATH, filename)

    with open(input_filepath, 'r', encoding='ascii') as input_file:
        text = input_file.read()
        processed_text = lmp(text)

    with open(output_filepath, 'w', encoding='utf-8') as output_file:
        output_file.write(processed_text)

print("Documentos preprocesados y guardados en la carpeta 'processed'.")
```

Ilustración 6 Llamada a los archivos y funciones de preprocesamiento

Este bloque de código crea una carpeta para guardar los documentos preprocesados si no existe, luego itera sobre los archivos en el directorio `TRAINING_PATH`:

1. Creación de Carpeta: Verifica si el directorio “processed” existe; si no, lo crea.
2. Iteración sobre Documentos: Para cada archivo en `TRAINING_PATH`, lee el contenido, aplica la función “lmp” para preprocesar el texto y guarda el texto preprocesado en un archivo nuevo dentro de `PROCESSED_PATH`.

3. Mensaje de Confirmación: Imprime un mensaje indicando que los documentos han sido preprocesados y guardados exitosamente.

Cargar documentos preprocesados:

```
# Cargar documentos preprocesados
documentos = {}
for filename in os.listdir(PROCESSED_PATH):
    filepath = os.path.join(PROCESSED_PATH, filename)
    with open(filepath, 'r', encoding='utf-8') as file:
        text = file.read()
        documentos[filename] = text
```

Ilustración 7 Cargar los Documentos pre-procesados

Este fragmento de código se encuentre dentro de **app/app.py**

- PROCESSED_PATH: Ruta al directorio donde se encuentran almacenados los documentos preprocesados.
- documentos: Diccionario que almacenará los textos preprocesados de los documentos. Las claves son los nombres de archivo y los valores son los textos preprocesados.

3. Representación de Datos en Espacio Vectorial

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

Estos fragmentos de código se encuentran dentro de **app/app.py**

Vectorización Bag of Words (BoW)

```
# Vectorización Bag of Words
corpus = list(documentos.values())
vectorizer_bow = CountVectorizer()
X_bow = vectorizer_bow.fit_transform(corpus)
df_bow = pd.DataFrame(X_bow.toarray(), columns=vectorizer_bow.get_feature_names_out(), index=documentos.keys())
```

Ilustración 8 BoW Vectorización

- corpus: Lista de textos preprocesados de los documentos.
- CountVectorizer: Objeto utilizado para convertir una colección de documentos de texto en una matriz de conteo de términos (Bag of Words).
- X_bow: Matriz de BoW resultante después de aplicar fit_transform al corpus.
- df_bow: DataFrame de pandas que contiene la matriz BoW con columnas etiquetadas por nombres de características y filas etiquetadas por nombres de documentos.

Vectorización TF-IDF

```
# Vectorización TF-IDF
vectorizer_tfidf = TfidfVectorizer()
X_tfidf = vectorizer_tfidf.fit_transform(corpus)
df_tf_idf = pd.DataFrame(X_tfidf.toarray(), columns=vectorizer_tfidf.get_feature_names_out(), index=documentos.keys())
```

Ilustración 9 TF-IDF Vectorización

- **TfidfVectorizer:** Objeto utilizado para convertir una colección de documentos de texto en una matriz TF-IDF (Term Frequency-Inverse Document Frequency).
- **X_tfidf:** Matriz TF-IDF resultante después de aplicar `fit_transform` al corpus.
- **df_tf_idf:** DataFrame de pandas que contiene la matriz TF-IDF con columnas etiquetadas por nombres de características y filas etiquetadas por nombres de documentos.

Evaluación de los métodos de Vectorización

Para evaluar ambos métodos de vectorización se utiliza 2 métricas esenciales las cuales son tiempo de ejecución y peso computacional.

Bag of Words (BoW): Tuvo un tiempo de ejecución bajo de 1 segundo, y su peso computacionalmente hablando es bajo.

Sin embargo, no considera la importancia de las palabras en el corpus y puede dar mucho peso a palabras muy comunes, pero poco informativas.

TF-IDF: El tiempo de ejecución fue mayor al de BoW siendo de 6 segundos, y el peso computacional también es mayor al de BoW.

Al contrario de BoW, TF-IDF pondera las palabras menos comunes que pueden ser más significativas y reduce el peso de palabras comunes.

4. Indexación

Objetivo: Crear un índice que permita búsquedas eficientes.

Estos fragmentos de código se encuentran dentro de Proyecto RI.ipynb

Función `indice_inver`:

```
def indice_inver(df):
    indice_invertido = {}
    for columna in df.columns:
        for index, value in df[columna].items():
            if value != 0:
                if columna not in indice_invertido:
                    indice_invertido[columna] = []
                indice_invertido[columna].append((index, value))
    return indice_invertido
```

Ilustración 10 Función Índice Invertido

Esta función guarda un diccionario de índice invertido en un archivo de texto estructurado.

Argumentos:

- `indice_invertido`: Diccionario que contiene el índice invertido a guardar.
- `directory`: Ruta del directorio donde se guardará el archivo.
- `filename`: Nombre del archivo donde se guardará el índice invertido.

Funcionamiento:

- Verifica si el directorio especificado (`directory`) existe. Si no existe, lo crea utilizando `os.makedirs(directory)`.
- Construye el camino completo al archivo utilizando `os.path.join(directory, filename)`.
- Abre el archivo en modo escritura ('w') y escribe cada término (`termino`) del diccionario `indice_invertido`.
- Para cada término, escribe cada documento (`documento`) y su frecuencia (`frecuencia`) asociada en líneas separadas.
- Incluye una línea en blanco después de cada conjunto de documentos asociados a un término para una mejor legibilidad del archivo guardado.

Función save_ind_inver

```
def save_ind_inver(indice_invertido, directory, filename):  
    if not os.path.exists(directory):  
        os.makedirs(directory)  
    filepath = os.path.join(directory, filename)  
    with open(filepath, 'w') as file:  
        for termino, documentos in indice_invertido.items():  
            file.write(f"Termino: {termino}\n")  
            for documento, frecuencia in documentos:  
                file.write(f"Documento: {documento}, Frecuencia: {frecuencia}\n")  
            file.write("\n")
```

Ilustración 11 Funcion que guarda el indice invertido

Esta función guarda un diccionario de índice invertido en un archivo de texto estructurado.

Argumentos:

- `indice_invertido`: Diccionario que contiene el índice invertido a guardar.
- `directory`: Ruta del directorio donde se guardará el archivo.
- `filename`: Nombre del archivo donde se guardará el índice invertido.

Funcionamiento:

- Verifica si el directorio especificado (`directory`) existe. Si no existe, lo crea utilizando `os.makedirs(directory)`.
- Construye el camino completo al archivo utilizando `os.path.join(directory, filename)`.
- Abre el archivo en modo escritura ('w') y escribe cada término (`termino`) del diccionario `indice_invertido`.
- Para cada término, escribe cada documento (`documento`) y su frecuencia (`frecuencia`) asociada en líneas separadas.
- Incluye una línea en blanco después de cada conjunto de documentos asociados a un término para una mejor legibilidad del archivo guardado.

Función `load_inverted_index_from_txt`:

```
def load_inverted_index_from_txt(filepath):
    inverted_index = {}
    with open(filepath, 'r', encoding='utf-8') as file:
        current_term = None
        for line in file:
            line = line.strip()
            if line.startswith("Termino:"):
                current_term = line.split("Termino: ")[1]
                inverted_index[current_term] = []
            elif line.startswith("Documento:"):
                doc_info = line.split("Documento: ")[1]
                doc_name, weight = doc_info.split(", Frecuencia: ")
                inverted_index[current_term].append((doc_name, float(weight)))
        return inverted_index

inverted_index_bow_loaded = load_inverted_index_from_txt('/content/drive/MyDrive/reuters/bow/indice_invertido_bow.txt')
inverted_index_tf_idf_loaded = load_inverted_index_from_txt('/content/drive/MyDrive/reuters/tf_idf/indice_invertido_tf_idf.txt')
```

Ilustración 12 Función carga el Índice Invertido para usarlo

Esta función carga un índice invertido desde un archivo de texto previamente guardado.

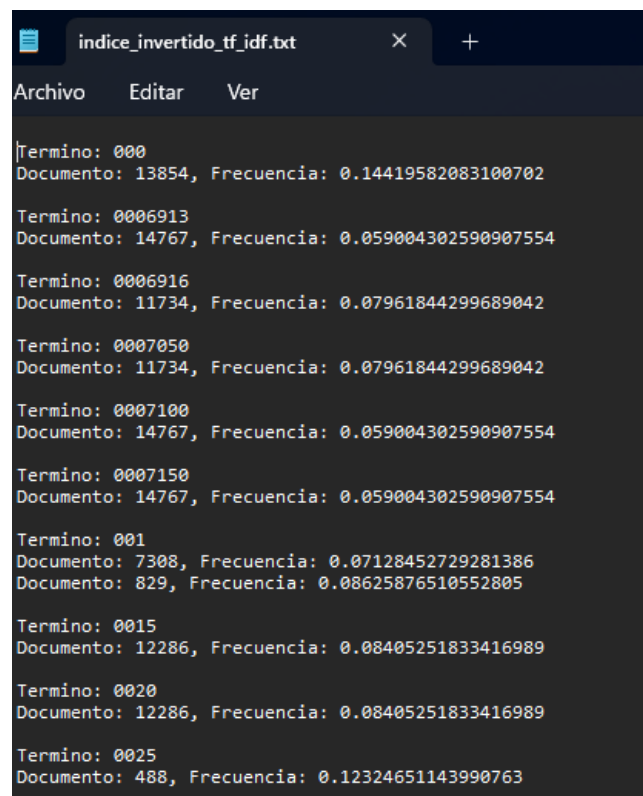
Argumentos:

- filepath: Ruta del archivo de texto que contiene el índice invertido.

Funcionamiento:

- Abre el archivo especificado (filepath) en modo lectura ('r') con codificación UTF-8.
- Itera sobre cada línea (line) en el archivo.
- Elimina los espacios en blanco iniciales y finales de cada línea utilizando strip().
- Si la línea comienza con "Término:", extrae el término actual (current_term) de la línea y lo establece como clave en el diccionario inverted_index, inicializando una lista vacía como valor asociado.
- Si la línea comienza con "Documento:", extrae la información del documento (doc_info), que contiene el nombre del documento (doc_name) y su frecuencia (weight). Divide esta información para obtener el nombre del documento y la frecuencia, convirtiendo la frecuencia a tipo float.
- Agrega el par (documento, frecuencia) a la lista asociada al término actual (current_term) en el diccionario inverted_index.
- Retorna el diccionario inverted_index completo una vez que se ha procesado todo el archivo.

Resultados de las funciones:



```
Archivo  Editar  Ver

Termino: 000
Documento: 13854, Frecuencia: 0.14419582083100702

Termino: 0006913
Documento: 14767, Frecuencia: 0.059004302590907554

Termino: 0006916
Documento: 11734, Frecuencia: 0.07961844299689042

Termino: 0007050
Documento: 11734, Frecuencia: 0.07961844299689042

Termino: 0007100
Documento: 14767, Frecuencia: 0.059004302590907554

Termino: 0007150
Documento: 14767, Frecuencia: 0.059004302590907554

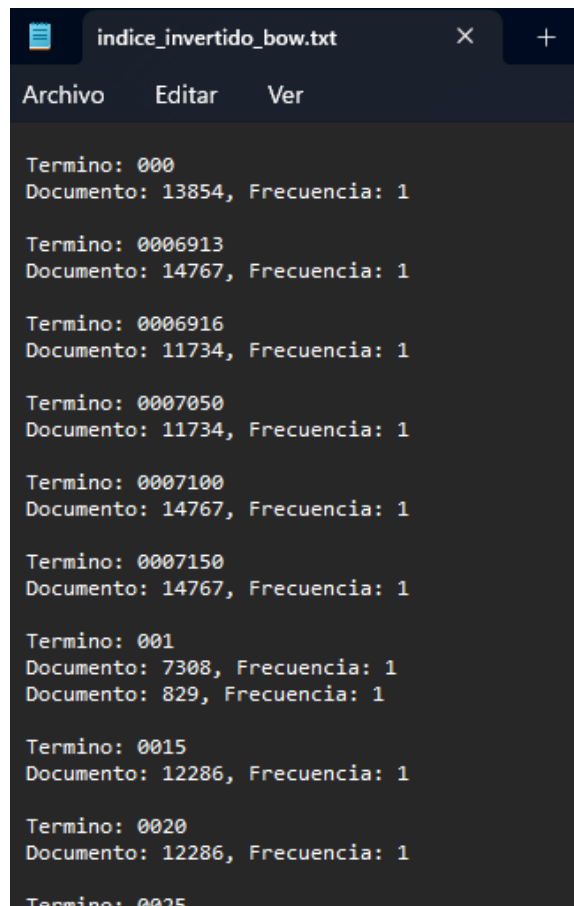
Termino: 001
Documento: 7308, Frecuencia: 0.07128452729281386
Documento: 829, Frecuencia: 0.08625876510552805

Termino: 0015
Documento: 12286, Frecuencia: 0.08405251833416989

Termino: 0020
Documento: 12286, Frecuencia: 0.08405251833416989

Termino: 0025
Documento: 488, Frecuencia: 0.12324651143990763
```

Ilustración 13 Resultado de TF_IDF



```
indice_invertido_bow.txt
Archivo  Editar  Ver

Termino: 000
Documento: 13854, Frecuencia: 1

Termino: 0006913
Documento: 14767, Frecuencia: 1

Termino: 0006916
Documento: 11734, Frecuencia: 1

Termino: 0007050
Documento: 11734, Frecuencia: 1

Termino: 0007100
Documento: 14767, Frecuencia: 1

Termino: 0007150
Documento: 14767, Frecuencia: 1

Termino: 001
Documento: 7308, Frecuencia: 1
Documento: 829, Frecuencia: 1

Termino: 0015
Documento: 12286, Frecuencia: 1

Termino: 0020
Documento: 12286, Frecuencia: 1

Termino: 0025
```

Ilustración 14 Resultado de BoW

5. Diseño del Motor de Búsqueda

Implementar algoritmos de similitud como similitud coseno o Jaccard.

```
def jaccard_similarity(query_tokens, document_tokens):
    intersection = len(set(query_tokens) & set(document_tokens))
    union = len(set(query_tokens) | set(document_tokens))
    return intersection / union if union != 0 else 0 # Avoid division by zero

def cosine_similarity_score(vector1, vector2):
    return cosine_similarity([vector1], [vector2])[0][0]
```

Ilustración 15 Funciones de Similitud Jaccard y Coseno

La **función jaccard_similarity** mide la similitud entre dos conjuntos de tokens (palabras o términos) utilizando el coeficiente de Jaccard.

Proceso:

- **Intersección:** Calcula el número de elementos comunes entre `query_tokens` y `document_tokens` utilizando la operación de intersección de conjuntos (`&`).

- **Unión:** Calcula el número total de elementos únicos presentes en cualquiera de los dos conjuntos utilizando la operación de unión de conjuntos (\cup).
- **Coefficiente de Jaccard:** Divide el tamaño de la intersección por el tamaño de la unión para obtener el coeficiente de Jaccard. Si la unión es cero, la función devuelve 0 para evitar la división por cero.

Salida:

Un valor entre 0 y 1 que indica la similitud entre los dos conjuntos de tokens. Un valor de 1 indica que los conjuntos son idénticos, mientras que un valor de 0 indica que no tienen elementos en común.

La **función cosine_similarity_score** mide la similitud entre dos vectores utilizando la similitud del coseno.

Proceso:

- **Similitud del Coseno:** Utiliza la función `cosine_similarity` de `sklearn.metrics.pairwise` para calcular la similitud del coseno entre los dos vectores.
- **Formato de Entrada:** La función `cosine_similarity` espera una lista de vectores como entrada, por lo que se envuelven `vector1` y `vector2` en listas.
- **Extracción del Resultado:** La función devuelve un valor entre -1 y 1, donde 1 indica que los vectores son idénticos, 0 indica que no tienen correlación y -1 indica que son diametralmente opuestos.

Salida:

Un valor entre -1 y 1 que indica la similitud entre los dos vectores. Un valor cercano a 1 indica una alta similitud, mientras que un valor cercano a 0 indica baja similitud.

Desarrollo de la lógica para procesar consultas de usuarios y un algoritmo de ranking para ordenar los resultados

```

def process_query(query):
    cleaned_query = lmp(query)
    return cleaned_query.split()

def search_with_bow(query, inverted_index_bow, documents):
    query_tokens = process_query(query)
    doc_tokens = {doc_id: documents[doc_id].split() for doc_id in documents}
    scores = {}
    for doc_id in doc_tokens:
        score = jaccard_similarity(query_tokens, doc_tokens[doc_id])
        if score > 0:
            scores[doc_id] = score
    ranked_results = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    return ranked_results

def search_with_tfidf(query, tfidf_matrix, vectorizer_tfidf, documents):
    query_tokens = process_query(query)
    query_vector = vectorizer_tfidf.transform([' '.join(query_tokens)]).toarray()[0]
    scores = {}
    for idx, doc_id in enumerate(documents.keys()):
        doc_vector = tfidf_matrix[idx].toarray()[0]
        score = cosine_similarity_score(query_vector, doc_vector)
        if score > 0:
            scores[doc_id] = score
    ranked_results = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    return ranked_results

```

Ilustración 16 Función de Procesamiento de Query y Función de ranking de resultados

El siguiente código implementa dos funciones de búsqueda, `search_with_bow` y `search_with_tfidf`, que utilizan Bag of Words (BoW) y TF-IDF, respectivamente, para encontrar documentos relevantes basados en una consulta de texto. Ambas funciones emplean `process_query` para limpiar y tokenizar la consulta, convirtiéndola en una lista de tokens.

Función `search_with_bow`

La función `search_with_bow` busca documentos utilizando el modelo Bag of Words. Primero, tokeniza la consulta y los documentos. Luego, calcula la similitud de Jaccard entre los tokens de la consulta y los tokens de cada documento. Los documentos con similitud mayor a cero se guardan con sus puntuaciones en un diccionario, que posteriormente se ordena en orden descendente para devolver los documentos más relevantes.

Función `search_with_tfidf`

La función `search_with_tfidf` busca documentos utilizando el modelo TF-IDF. Primero, tokeniza la consulta y la convierte en un vector TF-IDF. Luego, calcula la similitud del coseno entre el vector de la consulta y los vectores TF-IDF de cada documento. Los documentos con una similitud mayor a cero se guardan con sus puntuaciones en un diccionario, que posteriormente se ordena en orden descendente para devolver los documentos más relevantes.

6. Evaluación del Sistema

Objetivo: Medir la efectividad del sistema.

Definir un conjunto de métricas de evaluación: Las métricas de evaluación que se utilizarán son:

- Precisión (Precision)
- Exhaustividad (Recall)
- F1-score

Realizar pruebas utilizando el conjunto de prueba del corpus: Utilizaremos un conjunto de documentos categorizados para evaluar el rendimiento del sistema de recuperación de información basado en Bag of Words (BoW) y TF-IDF.

Comparar el rendimiento de diferentes configuraciones del sistema: Compararemos el rendimiento del sistema utilizando BoW y TF-IDF.

Los resultados de las evaluaciones se documentarán mediante matrices de confusión y reportes de clasificación.

Crea dos diccionarios a partir de un archivo de categorías, uno que mapea categorías a documentos y otro que mapea documentos a categorías.


```
def crear_diccionario_categorias(archivo):
    # Creamos un diccionario para almacenar las categorías y sus documentos
    categorias = collections.defaultdict(list)
    with open(archivo, 'r') as file:
        for linea in file:
            # Separamos la línea en la ruta del documento y las categorías
            ruta, *cats = linea.strip().split()
            # Extraemos el número del documento de la ruta
            numero_documento = ruta.split('/')[1]
            # Añadimos el número del documento a cada categoría correspondiente
            for cat in cats:
                categorias[cat].append(numero_documento)

    # Crear un diccionario de documentos a categorías (invirtiendo el anterior)
    documentos_categorias = collections.defaultdict(list)
    for categoria, docs in categorias.items():
        for doc in docs:
            documentos_categorias[doc].append(categoria)

    return documentos_categorias, list(categorias.keys())
```

Ilustración 17 Creacion de diccionario a partir del archivo de categorias

Generan las etiquetas verdaderas y predichas necesarias para la evaluación del rendimiento del modelo.

```
def obtener_etiquetas_verdaderas(documentos, query_categoria):
    etiquetas_verdaderas = []
    for doc_id in documentos:
        if query_categoria in documentos_categorias[doc_id]:
            etiquetas_verdaderas.append(0)
        else:
            etiquetas_verdaderas.append(1)
    return etiquetas_verdaderas

def obtener_etiquetas_predichas(resultados):
    return [0 if score > 0 else 1 for _, score in resultados]
```

Ilustración 18 Se compara los resultados predichos con las etiquetas reales

Evalúan el sistema utilizando BoW y TF-IDF respectivamente, devolviendo las etiquetas verdaderas y predichas.

```

def evaluar_bow(query, query_categoria, inverted_index_bow, documentos):
    resultados = search_with_bow(query, inverted_index_bow, documentos)
    documentos_resultados = [doc_id for doc_id, _ in resultados]
    etiquetas_verdaderas = obtener_etiquetas_verdaderas(documentos_resultados, query_categoria)
    etiquetas_predichas = obtener_etiquetas_predichas(resultados)
    return etiquetas_verdaderas, etiquetas_predichas

def evaluar_tfidf(query, query_categoria, tfidf_matrix, vectorizer_tfidf, documentos):
    resultados = search_with_tfidf(query, tfidf_matrix, vectorizer_tfidf, documentos)
    documentos_resultados = [doc_id for doc_id, _ in resultados]
    etiquetas_verdaderas = obtener_etiquetas_verdaderas(documentos_resultados, query_categoria)
    etiquetas_predichas = obtener_etiquetas_predichas(resultados)
    return etiquetas_verdaderas, etiquetas_predichas

```

Ilustración 19 Funciones para evaluar los resultados de BoW y TF-IDF

Proceso de Evaluación:

- Para cada categoría, se realiza una búsqueda con BoW y TF-IDF.
- Las etiquetas verdaderas y predichas se almacenan para su posterior análisis.

Matriz de Confusión y Reporte de Clasificación:

Se generan matrices de confusión y reportes de clasificación para visualizar y evaluar el rendimiento global del sistema.

Los resultados de usar BoW:



Ilustración 20 Matriz confusión BoW

Reporte de clasificación global para Bag of Words:

	precision	recall	f1-score	support
Relevante	0.28	0.30	0.29	9584
No Relevante	0.99	0.99	0.99	689626
accuracy			0.98	699210
macro avg	0.63	0.64	0.64	699210
weighted avg	0.98	0.98	0.98	699210

Ilustración 21 Reporte clasificación BoW

Los resultados de usar TF-IDF:

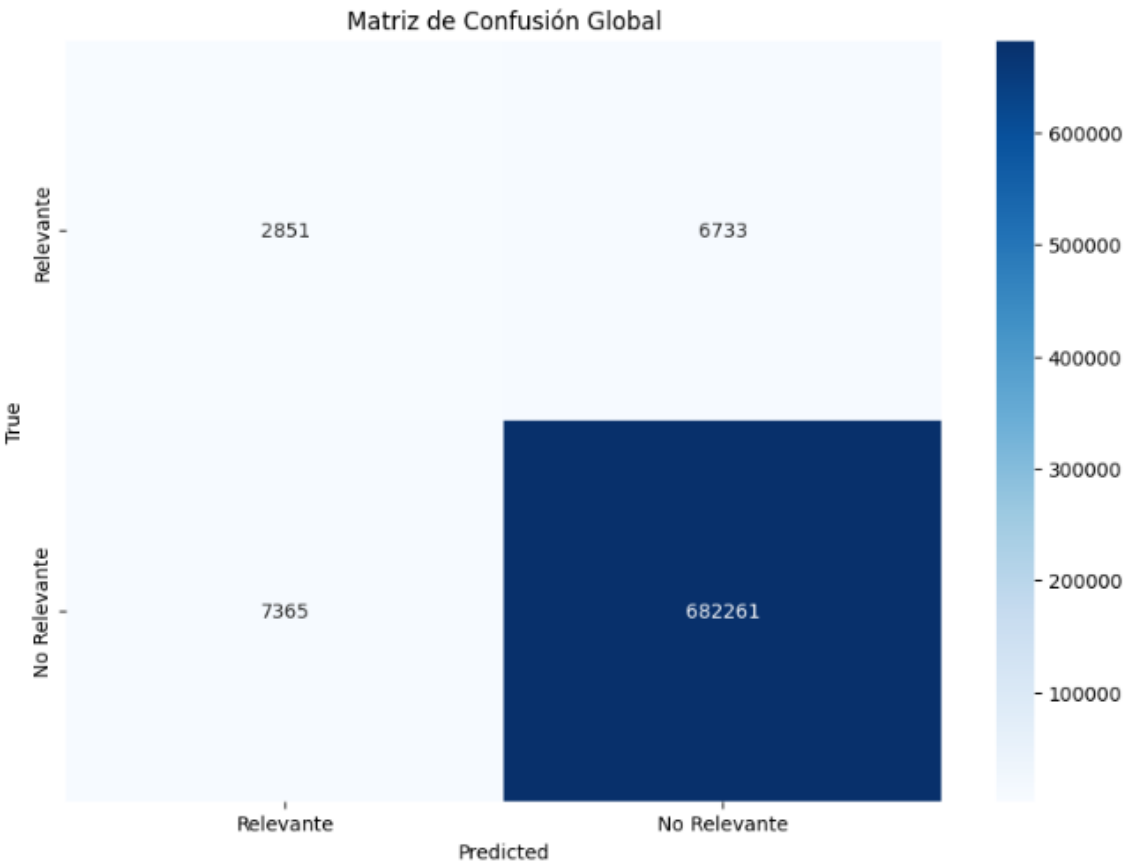


Ilustración 22 Matriz confusión TF-IDF

Reporte de clasificación global para TF-IDF:

	precision	recall	f1-score	support
Relevante	0.28	0.30	0.29	9584
No Relevante	0.99	0.99	0.99	689626
accuracy			0.98	699210
macro avg	0.63	0.64	0.64	699210
weighted avg	0.98	0.98	0.98	699210

Ilustración 23 Reporte clasificación TF-IDF

7. Interfaz Web de Usuario

Objetivo: Crear una interfaz para interactuar con el sistema

API para la página Web:

Todos los archivos se encuentran en app/app.py

Importaciones y Configuración Inicial

```

from flask import Flask, request, render_template
import os
import re
import pandas as pd
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

```

Ilustración 24 Librerías para App

- Flask: Framework web para crear aplicaciones web.
- os, re, pandas: Módulos estándar de Python para operaciones del sistema, expresiones regulares y manipulación de datos.
- nltk, sklearn: Librerías para procesamiento de lenguaje natural y aprendizaje automático.

Rutas del sistema

```

app = Flask(__name__)
|
# Paths
CORPUS_PATH = "reuters/training"
PROCESSED_PATH = "reuters/processed"
STOPWORDS_PATH = "reuters/stopwords"
BOW_INDEX_PATH = "reuters/bow/indice_invertido_bow.txt"
TFIDF_INDEX_PATH = "reuters/tf_idf/indice_invertido_tf_idf.txt"

```

Ilustración 25 Rutas para la App

Variables que almacenan las rutas a los archivos y directorios necesarios.

Carga de Stopwords

```

# Load stopwords
with open(STOPWORDS_PATH, 'r', encoding='ascii') as file:
    stop_words = set(word.strip() for word in file.readlines())

```

Ilustración 26 Carga de Stopwords

- Lee las stopwords desde un archivo y las almacena en un conjunto para eliminar palabras irrelevantes durante el procesamiento de texto.

Carga de Documentos Preprocesados

```
# Load preprocessed documents
documentos = {}
for filename in os.listdir(PROCESSED_PATH):
    filepath = os.path.join(PROCESSED_PATH, filename)
    with open(filepath, 'r', encoding='ascii') as file:
        cleaned_text = file.read()
        documentos[filename] = cleaned_text
```

Ilustración 27 Carga de datos pre-procesados

- Carga los documentos ya preprocesados desde un directorio y los almacena en un diccionario.

Vectorización de Documentos

- **Vectorización Bag of Words (BOW)**

```
corpus = list(documentos.values())
vectorizer_bow = CountVectorizer()
X_bow = vectorizer_bow.fit_transform(corpus)
df_bow = pd.DataFrame(X_bow.toarray(), columns=vectorizer_bow.get_feature_names_out(), index=documentos.keys())
```

Ilustración 28 Vectorización BoW

- CountVectorizer: Convierte la colección de documentos de texto en una matriz de tokens contados.
 - df_bow: DataFrame que contiene la matriz de BOW.
- **Vectorización TF-IDF**

```
vectorizer_tfidf = TfidfVectorizer()
X_tfidf = vectorizer_tfidf.fit_transform(corpus)
df_tf_idf = pd.DataFrame(X_tfidf.toarray(), columns=vectorizer_tfidf.get_feature_names_out(), index=documentos.keys())
```

Ilustración 29 Vectorización TF-IDF

- TfidfVectorizer: Convierte la colección de documentos en una matriz TF-IDF.
 - df_tf_idf: DataFrame que contiene la matriz TF-IDF.

Función para Cargar Índices Invertidos

```
# Functions for loading indices and searching
def load_inverted_index_from_txt(filepath):
    inverted_index = {}
    with open(filepath, 'r', encoding='utf-8') as file:
        current_term = None
        for line in file:
            line = line.strip()
            if line.startswith("Termino:"):
                current_term = line.split("Termino: ")[1]
                inverted_index[current_term] = []
            elif line.startswith("Documento:"):
                doc_info = line.split("Documento: ")[1]
                doc_name, weight = doc_info.split(", Frecuencia: ")
                inverted_index[current_term].append((doc_name, float(weight)))
    return inverted_index

inverted_index_bow_loaded = load_inverted_index_from_txt(BOW_INDEX_PATH)
inverted_index_tfidf_loaded = load_inverted_index_from_txt(TFIDF_INDEX_PATH)
```

Ilustración 30 Carga Índices Invertidos

- Carga un índice invertido desde un archivo de texto. El diccionario resultante tiene términos como claves y listas de tuplas (documento, frecuencia) como valores.
- Utiliza la función definida para cargar los índices invertidos de BOW y TF-IDF.
- filepath (str): Ruta al archivo de texto que contiene el índice invertido.

Procesamiento de Consultas

```
# Funciones de búsqueda
def process_query(query):
    cleaned_query = re.sub(r'^\w\s', '', query.lower())
    words = cleaned_query.split()
    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(word) for word in words]
    filtered_words = [word for word in stemmed_words if word not in stop_words]
    return filtered_words
```

Ilustración 31 Preprocesamiento de Querys

- Normaliza, aplica stemming y filtra stopwords de una consulta de búsqueda.
- query (str): La consulta de búsqueda ingresada por el usuario.
- filtered_words (list): Lista de palabras de la consulta procesada.

Cálculo de Similitud

- **Similitud Jaccard**

```
def jaccard_similarity(query_tokens, document_tokens):
    intersection = len(set(query_tokens) & set(document_tokens))
    union = len(set(query_tokens) | set(document_tokens))
    return intersection / union if union != 0 else 0
```

Ilustración 32 Función calcula similitud Jaccard

- Calcula la similitud Jaccard entre los tokens de la consulta y los tokens del documento realizando la intersección y unión de conjuntos .
- query_tokens (list): Lista de tokens de la consulta.
- document_tokens (list): Lista de tokens del documento.
- **Similitud Coseno**

```
def cosine_similarity_score(vector1, vector2):
    return cosine_similarity([vector1], [vector2])[0][0]
```

Ilustración 33 Función calcula similitud Coseno

- Calcula la similitud coseno entre dos vectores.
- vector1 (ndarray): Primer vector.
- vector2 (ndarray): Segundo vector.

Búsqueda con BOW

```
def search_with_bow(query, inverted_index_bow, documents):
    query_tokens = process_query(query)
    doc_tokens = {doc_id: documentos[doc_id].split() for doc_id in documentos}
    scores = {}
    for doc_id in doc_tokens:
        scores[doc_id] = jaccard_similarity(query_tokens, doc_tokens[doc_id])
    ranked_results = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    return [doc_id for doc_id, score in ranked_results]
```

Ilustración 34 Función de búsqueda BoW

- Realiza una búsqueda utilizando el modelo BOW y devuelve una lista de IDs de documentos ordenados por relevancia basada en la similitud Jaccard.
- query (str): La consulta de búsqueda.
- inverted_index_bow (dict): El índice invertido BOW.
- documents (dict): Diccionario de documentos preprocesados.

Búsqueda con TF-IDF


```
def search_with_tfidf(query, tfidf_matrix, vectorizer_tfidf, documents):
    query_tokens = process_query(query)
    query_vector = vectorizer_tfidf.transform([' '.join(query_tokens)]).toarray()[0]
    scores = {}
    for idx, doc_id in enumerate(documents.keys()):
        doc_vector = tfidf_matrix[idx].toarray()[0]
        scores[doc_id] = cosine_similarity_score(query_vector, doc_vector)
    ranked_results = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    return [doc_id for doc_id, score in ranked_results]
```

Ilustración 35 Función búsqueda TF-IDF

- Realiza una búsqueda utilizando el modelo TF-IDF y devuelve una lista de IDs de documentos ordenados por relevancia basada en la similitud coseno.
- query (str): La consulta de búsqueda.
- tfidf_matrix (sparse matrix): Matriz TF-IDF de los documentos.
- vectorizer_tfidf (TfidfVectorizer): El vectorizador TF-IDF.
- documents (dict): Diccionario de documentos preprocesados.

Rutas de Flask

- **Ruta Principal**

```
@app.route('/')
def index():
    return render_template('index.html')
```

Ilustración 36 Ruta de la página principal

- Renderiza la página principal del sitio web.

- **Ruta de Búsqueda**

```
@app.route('/search', methods=['POST'])
def search():
    query = request.form['query']
    search_type = request.form['search_type']

    if search_type == 'bow':
        results_bow = search_with_bow(query, inverted_index_bow_loaded, documentos)
        return render_template('resultados.html', results_bow=results_bow[:5], results_tfidf=[])
    elif search_type == 'tfidf':
        results_tfidf = search_with_tfidf(query, X_tfidf, vectorizer_tfidf, documentos)
        return render_template('resultados.html', results_bow=[], results_tfidf=results_tfidf[:5])
```

Ilustración 37 Ruta y funciones de la página de Resultados

- Maneja las solicitudes de búsqueda. Realiza una búsqueda utilizando BOW o TF-IDF en función del tipo de búsqueda seleccionado y renderiza los resultados en la plantilla resultados.html.

- **Ejecución de la Aplicación**

```
if __name__ == '__main__':
    app.run(debug=True)
```

Ilustración 38. Ejecución inicial de la aplicación

- Inicia la aplicación Flask en modo de depuración.

Front End:

- **Templates:**

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document Search</title>
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="header">
    <h1>Buscar en documentos</h1>
  </div>
  <div class="container">
    <form action="/search" method="post">
      <div class="form-group">
        <input type="text" class="form-control" name="query" placeholder="Ingresa tu búsqueda" required>
      </div>
      <div class="form-group">
        <button type="submit" class="btn btn-primary" name="search_type" value="bow">Buscar con BOW</button>
        <button type="submit" class="btn btn-primary" name="search_type" value="tfidf">Buscar con TF-IDF</button>
      </div>
    </form>
  </div>
  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
```

Ilustración 39. Documento HTML de la página inicial

- Estructura del Documento:
 - Declara el tipo de documento HTML5 y establece el idioma del contenido.
 - Define la cabecera del documento con meta-información y enlaces a archivos CSS de Bootstrap y un archivo CSS personalizado.
- Cuerpo del Documento:
 - Contiene un título principal "Buscar en documentos" dentro de un contenedor.
 - Un formulario que permite al usuario ingresar una consulta de búsqueda y elegir entre dos métodos de búsqueda (BOW o TF-IDF).
 - Incluye botones estilizados con Bootstrap para enviar la consulta.
- Inclusión de Scripts:
 - Añade scripts para jQuery, Popper.js, y Bootstrap para proporcionar funcionalidad interactiva y estilos.
- Funcionalidad

Formulario de Búsqueda:

- Un campo de entrada de texto para la consulta del usuario.
- Dos botones de envío que determinan si la búsqueda se realizará usando Bag of Words (BOW) o TF-IDF.
- El formulario se envía a la ruta /search del servidor mediante el método POST.

Resultados.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Resultados de búsqueda</title>
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="header">
    <h1>Resultados de búsqueda</h1>
  </div>
  <div class="container">
    <h2>Resultados con Bag of Words:</h2>
    <ul>
      {% for result in results_bow %}
      <li>Documento: {{ result }}</li>
      {% endfor %}
    </ul>

    <h2>Resultados con TF-IDF:</h2>
    <ul>
      {% for result in results_tfidf %}
      <li>Documento: {{ result }}</li>
      {% endfor %}
    </ul>

    <a href="/" class="btn btn-primary">Nueva búsqueda</a>
  </div>
  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

Ilustración 40. Documento HTML de la página de resultados

- Estructura del Documento
 - Se define el tipo de documento como HTML5 y el idioma como inglés.
 - La cabecera incluye meta-información y enlaces a hojas de estilo CSS de Bootstrap y un archivo CSS personalizado.
- Cuerpo del Documento
 - Contiene un encabezado principal "Resultados de búsqueda".
 - Sección de resultados con dos listas:
 - Resultados obtenidos mediante Bag of Words (BOW).
 - Resultados obtenidos mediante TF IDF.
 - Un enlace para regresar a la página principal y realizar una nueva búsqueda.

- Inclusión de Scripts
 - Scripts para jQuery, Popper.js y Bootstrap para funcionalidades interactivas y estilos.
- Funcionalidad
 - Muestra resultados de búsqueda separados en dos listas:
 - Una lista para los resultados usando Bag of Words (BOW).
 - Otra lista para los resultados usando TF IDF.
 - Cada elemento de la lista muestra el nombre del documento encontrado.
- Static

Style.css

```
body {
  padding-top: 20px;
}
.header {
  background-color: #007bff;
  color: white;
  padding: 10px 0;
  text-align: center;
}
.container {
  max-width: 600px;
  margin: auto;
  padding: 20px;
  box-shadow: 0 0 10px rgba(0,0,0,0.1);
  border-radius: 10px;
  background-color: #f8f9fa;
}
h2 {
  margin-top: 30px;
}
ul {
  padding-left: 20px;
}
a {
  display: block;
  text-align: center;
  margin-top: 20px;
}
.form-group {
  margin-bottom: 20px;
}
```

Ilustración 41. Hoja de estilos de la aplicación web

- Estilos para el diseño de las páginas

Resultados:

Buscar en documentos

[Buscar con BOW](#)[Buscar con TF-IDF](#)

Ilustración 42. Página de inicio

Resultados de búsqueda

Resultados con Bag of Words:

- Documento: 12008.txt
- Documento: 1910.txt
- Documento: 13242.txt
- Documento: 2954.txt
- Documento: 9265.txt

Resultados con TF-IDF:

[Nueva búsqueda](#)

Ilustración 43. Página de resultados