ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA EN SISTEMAS
RECUPERACIÓN DE LA INFORMACIÓN
GR1CC

Proyecto I BIMESTRE

Sistema de Recuperación de Información basado en Reuters-21578

AUTORES:

Madelyn Fernández
Kevin Valle
Carlos Sánchez

DOCENTE:
PhD. Iván Carrera

PERIODO:
2024-A

QUITO-ECUADOR

Contenido

Objetivos

Diseñar, construir, programar y desplegar un Sistema de Recuperación de Información (SRI) utilizando el corpus Reuters-21578.
Aplicar conceptos teóricos a retos prácticos de diseño y crear interfaces innovadoras y fáciles de usar para sistemas complejos.

Introducción

El conjunto de datos Reuters-21578 es uno de los más utilizados en la investigación sobre recuperación de información y clasificación de textos. Este informe presenta el diseño, implementación y evaluación de un sistema de recuperación de información basado en Reuters-21578. Se analizarán los métodos de preprocesamiento de texto, técnicas de indexación y modelos de búsqueda utilizados para maximizar la precisión y eficiencia del sistema. El objetivo es demostrar la efectividad de los métodos de representación de datos en este corpus para el desarrollo sistemas de recuperación de información avanzados.

Desarrollo

Adquisición de Datos

A continuación, se presenta y obtiene los datos necesarios para la realización del Proyecto, la cual es el corpus Reuters-21578.

Importamos *os* el cual permite el manejo de archivos para empezar con el preprocesamiento, se imprime la ubicación y el nombre de archivo para comprobar su correcto funcionamiento.

```
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Configuramos el Directorio en el cual se ubica el corpus.

```
# Configuración del directorio del corpus de Reuters
corpus_root = '/kaggle/input/reuters/reuters'
```

Se realiza la carga del corpus utilizando la función *CategorizedPlaintextCorpusReader*, el cual permite la manipulación y analisis de datos de texto.

```
# Cargar el corpus de Reuters
reuters = CategorizedPlaintextCorpusReader(
    corpus_root,
    r'(training|test).*',
    cat_file='cats.txt',
    encoding='ISO-8859-2'
)
```

Preprocesamiento

Se utiliza un conjunto de herramientas para limpiar y preprocesar documentos de texto, con opciones para eliminar stopwords y realizar stemming la cual es una técnica esencial en NLP y recuperación de información que permite agrupar variantes morfológicas de las palabras para mejorar el análisis y la búsqueda de información.

Se define la función *load_stopwords* en la cual cargamos el archivo que contiene las palabras que no se deben incluir en el procesamiento ya que son artículos, pronombres palabras conectoras.

```
# Load stopwords
def load_stopwords(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        stopwords = file.read().splitlines()
    return stopwords
```

Cargamos los *stopwords* con la función definida anteriormente.

```
# Cargar stopwords
STOPWORDS = load_stopwords('/kaggle/input/reuters/reuters/stopwords')
```

Se define un array llamado *codelist* el cual contiene saltos de texto y línea los cuales también se omiten al momento del preprocesado.

Definimos una función llamada *parse_doc* la cual recibe los textos del corpus y se realiza diferentes acciones que permiten que el procesado se realice de mejor manera, por ejemplo, se define que todo el texto se encuentre en minúsculas, eliminación de caracteres no ASCII, etc.

```
codelist = ['\r', '\n', '\t']

def parse_doc(text):
    text = text.lower()
    text = re.sub(r'&(.)+', "", text)  # no & references
    text = re.sub(r'pct', 'percent', text)  # replace pct abreviation
    text = re.sub(r"[^\w\d'\s]+", '', text)  # no punct except single
quote
    text = re.sub(r'[^\x00-\x7f]',r'', text)  # no non-ASCII strings
    if text.isdigit(): text = ""  # omit words that are all digits
    for code in codelist:
        text = re.sub(code, ' ', text)  # get rid of escape codes
    # replace multiple spacess with one space
    text = re.sub('\s+', ' ', text)
    return text
```

Se define dos variables una que sirve para la verificación de la eliminación de las stopwords, y la otra para decidir si aplicar el stemming.

```
DROP_STOPWORDS = True  # Prueba True y luego False para ver el impacto en el análisis
STEMMING = False  # Decisión sobre si aplicar stemming
```

Luego definimos la función *parse_words* la cual toma una cadena de texto y realiza un procesamiento adicional a nivel de palabras, realiza el filtrado de por ejemplo palabras que no sean alfabéticas, palabras que contienen menos de 3 caracteres y luego de esto realiza el Stemming y reconstruye el texto a partir de las palabras procesadas.

```
def parse_words(text):
    # split document into individual words
    tokens=text.split()
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    tokens = [re_punc.sub('', w) for w in tokens]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # filter out tokens that are one or two characters long
    tokens = [word for word in tokens if len(word) > 2]
    # filter out tokens that are more than twenty characters long
    tokens = [word for word in tokens if len(word) < 21]
    # filter out stop words if requested
    if DROP_STOPWORDS:
        tokens = [w for w in tokens if not w in STOPWORDS]
    # perform word stemming if requested
    if STEMMING:
        ps = PorterStemmer()
        tokens = [ps.stem(word) for word in tokens]
    # recreate the document string from parsed words
    text = ''
    for token in tokens:
        text = text + ' ' + token
    return text
```

Se define la función *generate_processed_texts*, la cual devuelve el texto procesado con las funciones definidas anteriormente.

```
# Generador para procesar textos
def generate_processed_texts(docs):
    for doc in docs:
        text_string = parse_doc(doc)
        text_string = parse_words(text_string)
        yield text_string
```

Se extrae los textos completos y las categorías correspondientes para los documentos de entrenamiento *train_documents*.

```
# Procesamiento de textos de entrenamiento y prueba
train_documents = [reuters.raw(fileid) for fileid in reuters.fileids() if
fileid.startswith('training/')]
test_documents = [reuters.raw(fileid) for fileid in reuters.fileids() if
fileid.startswith('test/')]
```

Se utiliza la función para generar los textos procesados del entrenamiento.

```
# Procesamiento de textos de entrenamiento y prueba usando el generador
train_texts_generator = generate_processed_texts(train_documents)
test_texts_generator = generate_processed_texts(test_documents)
```

Representación de Datos en Espacio Vectorial

Las técnicas para esta representación que se usaron son la **Bolsa de Palabras** (Bag of Words, BoW) y la **Frecuencia de Término - Frecuencia Inversa de Documento** (Term Frequency - Inverse Document Frequency, TF-IDF).

Bolsa de Palabras (BoW)

La técnica de Bolsa de Palabras representa un documento como un vector en el que cada dimensión corresponde a una palabra única del vocabulario del corpus. La principal característica de BoW es que se ignora el orden de las palabras, considerándose únicamente la frecuencia de aparición de cada palabra en el documento. Una vez con los datos preprocesados, se realiza la representación BoW:

- **Construcción del Vocabulario**: Se compila un conjunto de palabras únicas a partir de todos los documentos del corpus.
- **Vectorización**: Cada documento se representa como un vector en el que cada entrada corresponde al conteo de la frecuencia de una palabra del vocabulario en ese documento

TF-IDF

La técnica **TF-IDF** mejora la representación **BoW** al considerar no solo la frecuencia de las palabras en un documento, sino también su **importancia en el contexto del corpus** completo. Esta técnica pondera cada palabra con un valor que refleja su importancia relativa. La fórmula para calcular el peso de una palabra ***t*** en un documento ***d*** es:

$$TF - IDF(t, d) = TF(t, f) \times IDF(t)$$

Donde:

- **TF (Frecuencia de Término):** Es el número de veces que la palabra ***t*** aparece en el documento ***d***.
- **IDF (Frecuencia Inversa de Documento)**: Es una medida de la importancia de la palabra ***t*** en el corpus.

$$IDF(t) = \log\left(\frac{Número\ total\ de\ documentos}{Número\ de\ documentos\ que\ contienen\ \boldsymbol{t}}\right)$$

Una vez realizado estos cálculos para cada uno de los documentos del corpus, se construyen matrices BoW y TF-IDF que contienen la información de todos los documentos en el corpus.

Implementación

Se hace uso de la biblioteca "sklearn" de Python, donde se harán uso de las clases *CountVectorizer* para crear la matriz BoW y *TfidfVectorizer* para la matriz TF-IDF

- BoW

```
# BOW
BOW_vectorizer = CountVectorizer()
train_vectors_bow = BOW_vectorizer.fit_transform(train_texts_generator)
```

- TF-IDF

```
# Reiniciar el generador para TF-IDF
train_texts_generator = generate_processed_texts(train_documents)

# TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer()
train_vectors_tfidf =
tfidf_vectorizer.fit_transform(train_texts_generator)
```

Indexación

La indexación juega un papel crucial en los sistemas de recuperación de información al organizar y estructurar documentos para facilitar búsquedas eficientes. Estos índices son estructuras de datos donde cada término del vocabulario apunta a una lista de documentos que lo contienen. Este enfoque optimiza las búsquedas al limitar el espacio de búsqueda a documentos relevantes y facilita el cálculo de la relevancia mediante técnicas como la similitud coseno, asegurando resultados precisos y eficientes para las consultas.

Implementación

Obtiene el vocabulario de palabras utilizado por el vectorizador y se crea un diccionario donde cada palabra del vocabulario tiene una lista de índices de documentos que la contienen.

- Bow

```
# Índice invertido BOW
indice_invertido_bow = defaultdict(list)

vocabulario_bow = BOW_vectorizer.get_feature_names_out()

for i, documento in enumerate(train_documents):
```

```
    palabras_indices = train_vectors_bow[i].nonzero()[1]
    for indice_palabra in palabras_indices:
        palabra = vocabulario_bow[indice_palabra]
        indice_invertido_bow[palabra].append(i)
```

- TF-IDF

```
# Índice invertido TF-IDF
vocabulario_tfidf = tfidf_vectorizer.get_feature_names_out()
indice_invertido_tfidf = defaultdict(list)

for i, documento in enumerate(train_documents):
    palabras_indices = train_vectors_tfidf[i].nonzero()[1]
    for indice_palabra in palabras_indices:
        palabra = vocabulario_tfidf[indice_palabra]
        indice_invertido_tfidf[palabra].append(i)
```

Diseño del Motor de Búsqueda

El diseño de un motor de búsqueda eficiente y preciso es crucial para facilitar la recuperación de información relevante a partir de grandes volúmenes de datos. Se usa el cálculo de similitud coseno para evaluar la relevancia de los documentos recuperados y poder ranquearlos con respecto a las consultas.

Implementación:

Se hace uso de la biblioteca "sklearn" de Python, donde se harán uso de la clase *cosine_similarity* para evaluar la similitud de los documentos con la consulta. Antes de evaluar, se vectoriza la consulta con el vectorizador correspondiente (BoW o TF-IDF) y se obtienen los documentos relevantes con el índice invertido. Por tanto, *cosine_similarity* recibe como atributos la consulta vectorizada y los documentos relevantes vectorizados.

```
# Procesamiento de consulta y ranking de documentos
def procesar_consulta(consulta, vectorizador, indice_invertido,
train_vectors):
    consulta_procesada=parse_words(parse_doc(consulta))
    # consulta_procesada=consulta
    consulta_vectorizada = vectorizador.transform([consulta_procesada])
    consulta_indices = consulta_vectorizada.nonzero()[1]
    consulta_terminos = [vectorizador.get_feature_names_out()[i] for i in
consulta_indices]

    documentos_relevantes = set()
    for termino in consulta_terminos:
        if termino in indice_invertido:
            documentos_relevantes.update(indice_invertido[termino])
```

```
    documentos_relevantes = list(documentos_relevantes)
    if not documentos_relevantes:
        return []

    relevantes = train_vectors[documentos_relevantes]
    similitudes = cosine_similarity(consulta_vectorizada,
relevantes).flatten()

    return [(doc, similitud) for doc, similitud in
zip(documentos_relevantes, similitudes)]
```

Además de obtener la similitud se ranquean los resultados de mayor a menor relevancia, obtenido los 5 documentos más relevantes

```
def rankear_documentos(documentos_similitudes, nombres_archivos,
top_n=5):
    documentos_similitudes.sort(key=lambda x: x[1], reverse=True)
    documentos_rankeados = [(nombres_archivos[doc], similitud) for doc,
similitud in documentos_similitudes[:top_n]]
    return documentos_rankeados
```

```
# Obtener nombres de documentos de entrenamiento y prueba
train_docs = [os.path.basename(fileid) for fileid in reuters.fileids() if
fileid.startswith('training/')]
test_docs = [os.path.basename(fileid) for fileid in reuters.fileids() if
fileid.startswith('test/')]
```

Evaluación del Sistema

Para la evaluación del sistema se usaron las diferentes métricas fundamentales para estos sistemas como la precisión, recall y F1. Además, debemos clasificar los resultados para poder medir estas métricas:

- True Positives (TP): El número de documentos que fueron correctamente clasificadas como positivas.
- False Positives (FP): El número de documentos que fueron incorrectamente clasificadas como positivas.
- True Negatives (TN): El número de documentos que fueron correctamente clasificadas como negativas.
- False Negatives (FN): El número de documentos que fueron incorrectamente clasificadas como negativas.

Precisión

Esta métrica que mide la proporción de TP (True Positives) respecto al total de documentos recuperados. Es una medida de exactitud que indica qué tan relevante es la información recuperada por el sistema. Se calcula de la siguiente manera

$$Precisión = \frac{TP}{TP + FP}$$

Recall

El recall o sensibilidad, mide la proporción de TP respecto al total de documentos relevantes que existen en la colección. Es una medida de exhaustividad que indica la capacidad del sistema para recuperar todos los documentos relevantes. Se calcula como:

$$recall = \frac{TP}{TP + FN}$$

F1 Score

Es la media armónica de la precisión y el recall, proporcionando una métrica balanceada que considera tanto la exactitud como la exhaustividad del sistema. Es especialmente útil cuando se busca un equilibrio entre estas dos métricas. Se calcula como:

$$F1 = 2 \times \frac{Precisión \times recall}{Precisión + recall}$$

Implementación

Para realizar la evaluación se obtienen las querys que se usaran para realizar las consultas y poder medir las métricas de evaluación para BoW y TF-IDF.

```
# Crear diccionario para almacenar las categorías reales y sus documentos solo para el conjunto de entrenamiento
categorias_reales = {}

# Recorrer todas las categorías en el corpus de Reuters
for categoria in reuters.categories():
    # Filtrar y extraer el número de documento de entrenamiento para la categoría actual
    categorias_reales[categoria]= set(int(fileid.split('/')[1]) for fileid in reuters.fileids(categoria) if fileid.startswith('training/'))
```

Se realizan las consultas para obtener los resultados para evaluar BoW y TF-IDF

```
def evaluar_modelo(categorias_reales, vectorizer, indice_invertido, train_vectors, train_docs):

    test_search = {}

    for categoria in categorias_reales:
```

```
        consulta_usuario = procesar_consulta(categoria, vectorizer,
indice_invertido, train_vectors)
        if categoria not in test_search:
            test_search[categoria] = set()
        for resultado in consulta_usuario:
            test_search[categoria].add(int(train_docs[resultado[0]]))

    return test_search

# BoW
bow_test_search = evaluar_modelo(categorias_reales, BOW_vectorizer,
indice_invertido_bow, train_vectors_bow, train_docs)

#TF-IDF
tfidf_test_search = evaluar_modelo(categorias_reales, tfidf_vectorizer,
indice_invertido_tfidf, train_vectors_tfidf, train_docs)
```

Se clasifican los resultados y se calculan las métricas para BoW y TF-IDF. Además, se saca el promedio de los resultados de cada métrica para obtener las métricas globales.

```
def calcular_metricas(categorias_diccionario, test_seach):

    metrics_by_category = defaultdict(dict)
    total_precision = 0
    total_recall = 0
    total_f1 = 0

    # Calcular métricas por categoría
    for categoria in categorias_diccionario:
        if categoria in test_seach:
            documentos_recuperados = test_seach[categoria]
            documentos_reales = categorias_diccionario[categoria]

            # Verdaderos Positivos (TP): Documentos recuperados que son
relevantes
            tp =
len(documentos_recuperados.intersection(documentos_reales))

            # Falsos Positivos (FP): Documentos recuperados que no son
relevantes
            fp = len(documentos_recuperados - documentos_reales)

            # Falsos Negativos (FN): Documentos relevantes que no fueron
recuperados
            fn = len(documentos_reales - documentos_recuperados)
```

```
            # Precisión: TP / (TP + FP)
            precision = tp / (tp + fp) if (tp + fp) > 0 else 0

            # Recall: TP / (TP + FN)
            recall = tp / (tp + fn) if (tp + fn) > 0 else 0

            # F1-score: 2 * (Precisión * Recall) / (Precisión + Recall)
            f1_score = 2 * (precision * recall) / (precision + recall) if
(precision + recall) > 0 else 0

            # Guardar métricas por categoría
            metrics_by_category[categoria]['TP'] = tp
            metrics_by_category[categoria]['FP'] = fp
            metrics_by_category[categoria]['FN'] = fn
            metrics_by_category[categoria]['Precision'] = precision
            metrics_by_category[categoria]['Recall'] = recall
            metrics_by_category[categoria]['F1-score'] = f1_score

            # las métricas globales
            total_precision += precision
            total_recall += recall
            total_f1 += f1_score

    # Calcular métricas globales
    precision_global = total_precision / len(categorias_diccionario)
    recall_global = total_recall / len(categorias_diccionario)
    f1_score_global = total_f1 / len(categorias_diccionario)

    return metrics_by_category, precision_global, recall_global,
f1_score_global
```

```
# Llamada a la función para calcular métricas

#BOW
metrics_by_category_bow, precision_global_bow, recall_global_bow,
f1_score_global_bow =
calcular_metricas(categorias_reales,bow_test_search)

#TF-IDF
metrics_by_category_tfidf, precision_global_tfidf, recall_global_tfidf,
f1_score_global_tfidf = calcular_metricas(categorias_reales,
tfidf_test_search)
```

Por último, se presentan los resultados obtenidos y las métricas para cada una de las querys, como también las métricas globales.

```python
def imprimir_metricas_por_categoria(metrics_by_category):
    for categoria, metrics in metrics_by_category.items():
        print(f"Categoría: {categoria}")
        print(f"   TP: {metrics['TP']}")
        print(f"   FP: {metrics['FP']}")
        print(f"   FN: {metrics['FN']}")
        print(f"   Precisión: {metrics['Precision']:.4f}")
        print(f"   Recall: {metrics['Recall']:.4f}")
        print(f"   F1-score: {metrics['F1-score']:.4f}")
        print()
```

```python
def imprimir_metricas_globales(precision_global, recall_global,
f1_score_global):
    print("Métricas globales:")
    print(f"   Precisión Global: {precision_global:.4f}")
    print(f"   Recall Global: {recall_global:.4f}")
    print(f"   F1-score Global: {f1_score_global:.4f}")
    print()
```

Resultados

Las evaluaciones del sistema de recuperación de información utilizando las métricas de precisión, recall y F1 Score obtuvieron los siguientes resultados en ambos métodos (BoW y TF-IDF) con el mismo conjunto de querys:

- **Precisión Global:** 0.3458
- **Recall Global**: 0.4423
- **F1-score Global**: 0.3352

Precisión Global

La precisión global obtenida fue de 34.58%, lo que nos indica que, de todos los documentos recuperados por el sistema, el 34.58% eran relevantes. Esta precisión nos muestra un número significativo de falsos positivos (FP), es decir, documentos no relevantes que fueron recuperados por el sistema.

Recall Global

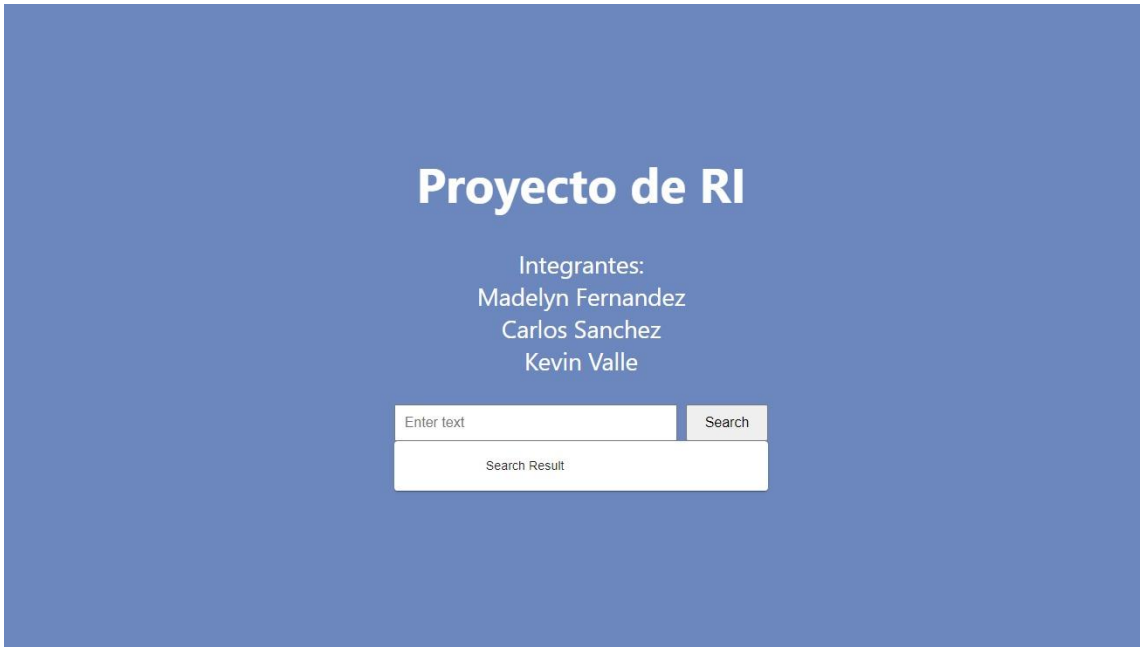
El recall global obtenido fue de 44.23%. indicando que el sistema fue capaz de recuperar el 44.23% de todos los documentos relevantes disponibles en el corpus. Esto muestra que el sistema está perdiendo más de la mitad de los documentos relevantes (falsos negativos, FN), lo que afecta su capacidad de recuperación exhaustiva.

F1-score Global

El F1-score global fue de 33.52. Al ser la media armónica de la precisión y el recall, sugiere que el sistema tiene dificultades para equilibrar ambas métricas. Un F1-score de este nivel indica que

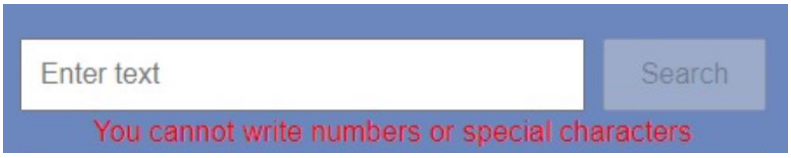
el sistema necesita mejoras tanto en precisión como en recall para obtener un mejor equilibrio y rendimiento general.

Interfaz Web de Usuario

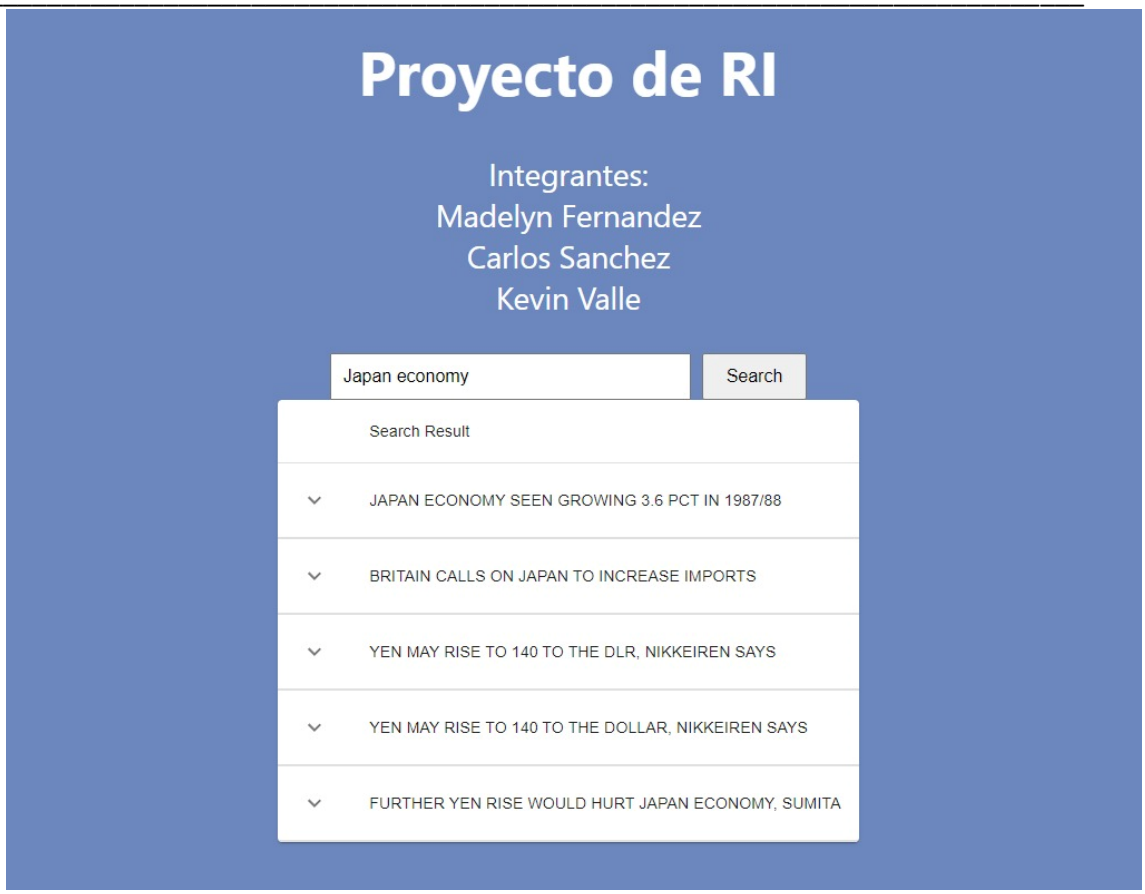


La interfaz de búsqueda es simple y minimalista. Aquí están los componentes visibles:

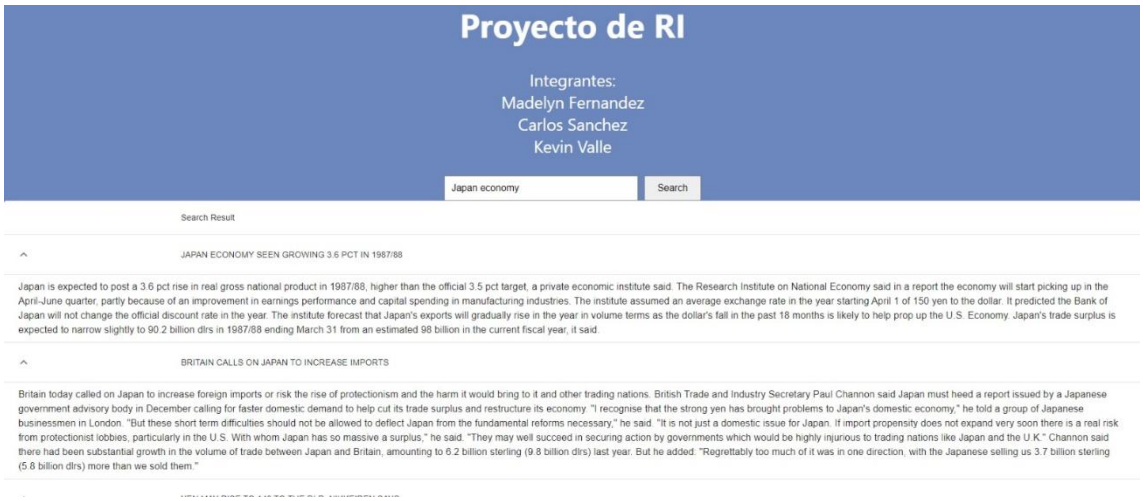
1. **Título de la página**: "Proyecto de RI"
2. **Lista de Integrantes**: Nombres de los miembros del equipo listados bajo el título "Integrantes":
    - Madelyn Fernández
    - Carlos Sánchez
    - Kevin Valle
3. **Campo de búsqueda**: Un cuadro de texto etiquetado "Enter text" permite al usuario ingresar términos de búsqueda. El usuario solo puede ingresar letras, el sistema se bloquea cuando el usuario intenta ingresar números o caracteres especiales.



4. **Botón de búsqueda**: Un botón etiquetado "Search", inicia la búsqueda basada en el texto ingresado.
5. **Área de resultado de búsqueda**: Se muestra una sección con los resultados para la consulta con el top 5 de los documentos más relevantes con respecto al texto de consulta ingresado.



Hay dos artículos listados en este ejemplo en el que se muestra la noticia completa relevante:



El diseño utiliza un esquema de colores simple con fondo azul y texto en blanco, creando un contraste alto para facilitar la lectura.

Anexos

Link Repositorio del Proyecto:

https://github.com/KevinValle97/Proyecto-RI-IB-2024A

Link Procesamiento en Python:

https://www.kaggle.com/code/madelynfernndez/sri-reuters-21578/