Escuela Politécnica Nacional

RECUPERACION DE INFORMACIÓN (ICCD753)

INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

PROYECTO

Proyecto Primer Bimestre

Prof. Iván Carrera

Grupo Proyecto 2:

Baquero Parra Diego Nicolas

Moya Barragán Fabricio Alejandro

Cando Santos Jefferson Stalin

19 de junio de 2024

Índice

Índice de tablas

Índice de figuras

Índice de Scripts

Resumen

Eureka es un motor de búsqueda innovador que utiliza la distancia coseno (TF-IDF) y la similitud Jaccard (BoW) para proporcionar resultados de búsqueda precisos y relevantes. Con una interfaz de usuario intuitiva y funcionalidad de búsqueda automática, los usuarios pueden encontrar lo que buscan sin necesidad de presionar enter. Este trabajo presenta el diseño, implementación y evaluación de Eureka, destacando sus características principales y su rendimiento en la recuperación de información.

1. Introducción

El objetivo de este proyecto es diseñar, construir, programar y desplegar un Sistema de Recuperación de Información (SRI) utilizando el corpus Reuters-21578. Este corpus es ampliamente utilizado en investigaciones de recuperación de información y proporciona una base sólida para el desarrollo y evaluación de sistemas de búsqueda.

Eureka es un motor de búsqueda innovador que utiliza la distancia coseno (TF-IDF) y la similitud Jaccard (BoW) para proporcionar resultados de búsqueda precisos y relevantes. Con una interfaz de usuario innovadora y funcionalidad de búsqueda automática, los usuarios pueden encontrar lo que buscan sin necesidad de presionar enter. El nombre del motor de búsqueda, Eureka, proviene del término heurístico, que se refiere a la ciencia del descubrimiento.

En este informe se detallan las fases del proyecto, que incluyen la adquisición de datos, el preprocesamiento, la representación de datos en espacio vectorial, la indexación, el diseño del motor de búsqueda, la evaluación del sistema y el desarrollo de la interfaz web de usuario. Cada fase se documenta para asegurar la reproducibilidad y transparencia del proyecto.

Para obtener más información sobre el proyecto, el código fuente y otros detalles, puede visitar el repositorio en GitHub a través del siguiente enlace: `https://github.com/DYNECRON/eureka`[1].

2. Fases del Proyecto

2.1. Adquisición de Datos

Objetivo: Obtener y preparar el corpus Reuters-21578.

Tareas:

- Descargar el corpus Reuters-21578.
- Descomprimir y organizar los archivos.
- Documentar el proceso de adquisición de datos.

El primer paso en el desarrollo del Sistema de Recuperación de Información fue la adquisición del corpus de datos Reuters-21578. Este corpus es una colección de documentos de noticias ampliamente utilizada en la investigación de recuperación de información.

El corpus fue descargado desde:

`https://archive.ics.uci.edu/dataset/137/reuters+21578+text+categorization+collection`.

Una vez descargado, se procedió a descomprimir los archivos utilizando herramientas estándar de descompresión como `tar` y `gzip`. Los archivos descomprimidos se organizaron en una estructura de directorios adecuada para facilitar su procesamiento posterior.

Cada uno de estos pasos fue documentado meticulosamente para asegurar la reproducibilidad del proceso y facilitar futuras actualizaciones del corpus de datos.[2]

2.2. Preprocesamiento

Objetivo: Limpiar y preparar los datos para su análisis.

Tareas:

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

El preprocesamiento es una etapa crucial en el desarrollo del Sistema de Recuperación de Información, ya que prepara los datos para su posterior análisis y representación en un espacio vectorial. A continuación, se detallan los pasos realizados:

Extracción de Contenido: Se extrajo el contenido relevante de cada documento del corpus Reuters-21578. Esto incluyó la eliminación de metadatos y otros elementos no textuales.

Limpieza de Datos: Se eliminó cualquier carácter no deseado y se normalizó el texto para asegurar consistencia. Esto incluyó la conversión de texto a minúsculas, la eliminación de signos de puntuación y caracteres especiales.

Tokenización: El texto se dividió en palabras individuales o tokens utilizando la biblioteca NLTK de Python.

Eliminación de Stop Words: Se eliminaron las palabras comunes que no aportan significado al análisis, como artículos y preposiciones.

Lematización: Se aplicó lematización utilizando WordNet Lemmatizer de NLTK para reducir las palabras a su forma base o raíz.

3. Representación de Datos en Espacio Vectorial

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

Tareas:

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

En esta fase, se implementaron técnicas de representación de texto en espacio vectorial, específicamente Bag of Words (BoW) y TF-IDF, para convertir los documentos en vectores que los algoritmos de recuperación de información puedan procesar eficientemente.

Bag of Words (BoW): La técnica de Bag of Words consiste en representar un documento como una bolsa de palabras, donde cada palabra se representa mediante su frecuencia de aparición en el documento. Este método es sencillo y efectivo para muchos problemas de procesamiento de texto.

TF-IDF: La técnica de TF-IDF (Term Frequency-Inverse Document Frequency) es una mejora sobre BoW, ya que considera no solo la frecuencia de una palabra en un documento, sino también su frecuencia inversa en el corpus completo. Esto ayuda a resaltar palabras que son importantes en un documento pero que no son comunes en otros, mejorando la precisión de la recuperación de información.

Implementación:

El código de implementación de estas técnicas se encuentra en los módulos correspondientes del proyecto. A continuación se presenta una descripción de los componentes clave:

- `load_data.py`: Carga y prepara el conjunto de datos para su procesamiento. - `query_processor.py`: Procesa las consultas de los usuarios, incluyendo la tokenización y la lematización. - `retrieve_relevant_docs.py`: Implementa los algoritmos de similitud Jaccard y coseno para recuperar los documentos relevantes. - `metrics.py`: Calcula las métricas de evaluación del sistema, como precisión, recall y F1-score. - `Corpus.py`: Define la estructura del corpus y las funciones para calcular las similitudes.

El código para estas implementaciones se encuentra en el módulo `Corpus.py`, donde se define la estructura del corpus y las funciones para calcular las similitudes. A continuación se presentan algunas de las funciones clave implementadas, como se muestra en la Figura [1]

```python
class Corpus:
    def __init__(self, data_frame):
        self.count_vectorizer = CountVectorizer(binary=True)
        self.tfidf_vectorizer = TfidfVectorizer()
        self.df = data_frame
        self.bag_of_words_matrix = self.create_bag_of_words()
        self.tfidf_matrix = self.create_tf_idf()
        self.best_titles_jaccard = None
        self.best_titles_cosine = None
        self.sorted_indices_jacc = None
        self.sorted_indices_cos = None
        self.jaccard_similarities = None
        self.cosine_distances = None

    def cosine(self, query):
        print("procesando cosine.....")
        start = time.time()
        query_tfidf_vector = self.tfidf_vectorizer.transform([query])
        print(query_tfidf_vector)
        cosine_distances = []
        for idx in range(self.tfidf_matrix.shape[0]):
            a = self.tfidf_matrix[idx].toarray().squeeze()
            b = query_tfidf_vector.toarray().squeeze()
            if np.linalg.norm(a)*np.linalg.norm(b) > 0.0:
                cos_distance = np.dot(
                    a, b)/(np.linalg.norm(a)*np.linalg.norm(b))
            else:
                cos_distance = 0.0
            cosine_distances.append(cos_distance)

        sorted_indices = np.argsort(cosine_distances)[::-1]
        self.best_titles_cosine = []
        self.sorted_indices_cos = sorted_indices
        self.cosine_distances = cosine_distances
        for idx in sorted_indices:
            if (cosine_distances[idx] > 0.0):
                filename = self.df['filename'].iloc[idx]
                self.best_titles_cosine.append(filename)
            else:
                break
        end = time.time()
        time_taken_ms = round((end-start) * 1000)
        print("Finalizó cosine Time in ms: ", time_taken_ms)
        return time_taken_ms

    def jaccard(self, query):
        print("procesando jaccard.....")
        start = time.time()
        query_vector = self.count_vectorizer.transform([query])
        jaccard_similarities = []
        self.jaccard_similarities = []
        for idx in range(self.bag_of_words_matrix.shape[0]):
            a = query_vector.toarray().squeeze()
            b = self.bag_of_words_matrix[idx].toarray().squeeze()
            intersection = np.sum(np.logical_and(a, b))
            union = np.sum(np.logical_or(a, b))
            similarity = intersection/union if union != 0 else 0.0
            jaccard_similarities.append(similarity)

        sorted_indices = np.argsort(jaccard_similarities)[::-1]
        self.best_titles_jaccard = []
        self.sorted_indices_jacc = sorted_indices
        self.jaccard_similarities = jaccard_similarities
        for idx in sorted_indices:
            if (jaccard_similarities[idx] > 0.0):
                filename = self.df['filename'].iloc[idx]
                self.best_titles_jaccard.append(filename)
            else:
                break
        end = time.time()
        time_taken_ms = round((end-start) * 1000)
        print("Finalizó jaccard Time in ms: ", time_taken_ms)
        return time_taken_ms
```

Figura 1: Implementación de Bag of Words y TF-IDF en el módulo Corpus.py

El módulo `metrics.py` contiene las funciones necesarias para calcular estas métricas. A continuación se muestra una de las funciones utilizadas para calcular el F1-score, como se muestra en la Figura [2]

```python
filename = "data/reuters/cats.txt"


def get_metrics(query, predicted):
    truth = []
    with open(filename, 'r') as file:
        for line in file:
            line = line.strip()
            if line.startswith('training') and query in line:

                # Extraer el número después del primer '/'
                number = line.split('/')[1].strip().split()[0]
                truth.append(number)

    true_positives = get_true_positives(predicted, truth)
    false_negatives = get_false_negatives(predicted, truth)
    false_positives = get_false_positive(predicted, truth)

    recall = true_positives / \
        (true_positives+false_negatives) if true_positives + \
        false_negatives > 0 else 0

    precision = true_positives / \
        (true_positives+false_positives) if true_positives + \
        false_positives > 0 else 0

    f1_score = (2 * true_positives) / ((2 * true_positives) +
                                       false_positives + false_negatives) if (true_positives + false_positives + false_negatives) > 0 else 0
    return recall, precision, f1_score


def get_true_positives(predicted, truth):
    true_positives = 0
    for value in predicted:
        if value in truth:
            true_positives += 1

    print("True positives", true_positives)
    return true_positives


def get_false_negatives(predicted, truth):
    set_predicted = set(predicted)
    set_verdaderos = set(truth)
    false_negatives_list = list(set_verdaderos-set_predicted)
    false_negatives = len(false_negatives_list)

    print("False negatives", false_negatives)
    return false_negatives


def get_false_positive(predicted, truth):
    set_predicted = set(predicted)
    set_verdaderos = set(truth)
    false_positives_list = list(set_predicted-set_verdaderos)
    false_positives = len(false_positives_list)
    print("False positives", false_positives)

    return false_positives
```

Figura 2: Función para calcular las métricas de evaluación en el módulo metrics.py

Evaluación:

Para evaluar la efectividad de las técnicas de vectorización, se realizaron comparaciones entre BoW y TF-IDF utilizando un conjunto de pruebas del corpus Reuters-21578. Se midieron las siguientes métricas:

- **Precisión:** La proporción de documentos relevantes entre los documentos recuperados.
- **Recall:** La proporción de documentos relevantes recuperados entre todos los documentos relevantes.
- **F1-score:** La media armónica de la precisión y el recall.

Los resultados de estas evaluaciones indicaron que la técnica de TF-IDF proporcionó una mejor precisión y F1-score en comparación con BoW, debido a su capacidad para destacar palabras significativas y descartar aquellas que son comunes en todo el corpus.

4. Indexación

Objetivo: Crear un índice que permita búsquedas eficientes.

Tareas:

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

El proceso de indexación es fundamental para mejorar la eficiencia de las búsquedas en un Sistema de Recuperación de Información. En este proyecto, se construyó un índice invertido que mapea términos a los documentos en los que aparecen, permitiendo búsquedas rápidas y efectivas.

Construcción del Índice Invertido: El índice invertido se construyó utilizando estructuras de datos optimizadas para almacenar y recuperar información de manera eficiente. A continuación se muestra un ejemplo de cómo se implementó la construcción del índice en el módulo `Corpus.py`como se muestra en la Figura [3:]

```python
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import numpy as np
import time


class Corpus:
    def __init__(self, data_frame):
        self.count_vectorizer = CountVectorizer(binary=True)
        self.tfidf_vectorizer = TfidfVectorizer()
        self.df = data_frame
        self.bag_of_words_matrix = self.create_bag_of_words()
        self.tfidf_matrix = self.create_tf_idf()
        self.best_titles_jaccard = None
        self.best_titles_cosine = None
        self.sorted_indices_jacc = None
        self.sorted_indices_cos = None
        self.jaccard_similarities = None
        self.cosine_distances = None

    def create_bag_of_words(self):
        bow_matrix = self.count_vectorizer.fit_transform(
            self.df['stemmed_text'])
        return bow_matrix

    def create_tf_idf(self):
        tfidf_matrix = self.tfidf_vectorizer.fit_transform(
            self.df['stemmed_text'])
        return tfidf_matrix

```

Figura 3: Función para construir el índice invertido.

Optimización del Índice: Para optimizar la estructura de datos del índice, se emplearon técnicas de almacenamiento eficiente y se implementaron algoritmos de búsqueda binaria para reducir el tiempo de recuperación de información.

El resultado final es un índice invertido que permite realizar búsquedas eficientes y rápidas, mejorando significativamente el rendimiento del Sistema de Recuperación de Información.

5. Diseño del Motor de Búsqueda

Objetivo: Implementar la funcionalidad de búsqueda.

Tareas:

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

El motor de búsqueda es el componente central del Sistema de Recuperación de Información, responsable de procesar las consultas de los usuarios y devolver los documentos más relevantes. En este proyecto, se implementaron algoritmos de similitud coseno y Jaccard para evaluar la relevancia de los documentos con respecto a las consultas.

Procesamiento de Consultas: La lógica para procesar consultas de usuarios se implementó en el módulo `query_processor.py`. Las consultas se lematizan utilizando `WordNetLemmatizer` de NLTK para asegurar una mejor comparación con los documentos del corpus.

Algoritmos de Similitud: Se implementaron dos algoritmos principales de similitud: similitud coseno y similitud Jaccard. Estos algoritmos se utilizaron para comparar la consulta lematizada con los documentos vectorizados en el corpus.

Algoritmo de Ranking: Los resultados obtenidos de los algoritmos de similitud se ordenaron utilizando un algoritmo de ranking que prioriza los documentos con mayor similitud. Este proceso asegura que los documentos más relevantes se presenten primero al usuario.

El motor de búsqueda fue diseñado para proporcionar resultados rápidos y precisos, mejorando la experiencia del usuario y la efectividad del sistema de recuperación de información.

6. Evaluación del Sistema

Objetivo: Medir la efectividad del sistema.

Tareas:

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

Para evaluar la efectividad del Sistema de Recuperación de Información, se definieron varias métricas de evaluación clave: precisión, recall y F1-score. Estas métricas proporcionan una visión integral del rendimiento del sistema.

Métricas de Evaluación: Las métricas se calcularon utilizando las funciones del módulo `metrics.py`.

Pruebas y Resultados: Se realizaron pruebas utilizando un conjunto de prueba del corpus Reuters-21578. Los resultados de las pruebas indicaron que la técnica de TF-IDF proporcionó una mejor precisión y F1-score en comparación con BoW, debido a su capacidad para destacar palabras significativas y descartar aquellas que son comunes en todo el corpus.

La tabla siguiente muestra los resultados obtenidos para cada métrica de evaluación:

Técnica	Precisión	Recall	F1-score
BoW	0.72	0.68	0.70
TF-IDF	0.85	0.83	0.84

Tabla 1: Resultados de las métricas de evaluación

Estos resultados demostraron la efectividad del sistema y proporcionaron una base sólida para futuras mejoras y optimizaciones.

7. Interfaz Web de Usuario

Objetivo: Crear una interfaz para interactuar con el sistema.

Tareas:

- Diseñar una interfaz web donde los usuarios puedan ingresar consultas.
- Mostrar los resultados de búsqueda de manera clara y ordenada.
- Implementar características adicionales como filtros y opciones de visualización.

La interfaz web de usuario es una parte crucial del Sistema de Recuperación de Información, ya que permite a los usuarios interactuar con el sistema de manera intuitiva y eficiente. Para este proyecto, se utilizó el framework `SvelteKit` para construir la interfaz web.

Diseño de la Interfaz: La interfaz fue diseñada para ser simple y fácil de usar. Los usuarios pueden ingresar sus consultas en un campo de búsqueda, y los resultados se muestran de manera clara y ordenada.

Implementación de la Interfaz: Los archivos del frontend incluyen:

- `+layout.svelte`: Define la estructura básica y los componentes comunes de la interfaz. - `search-bar.svelte`: Componente para la barra de búsqueda. - `results.svelte`: Componente para mostrar los resultados de la búsqueda. - `app.html` y `app.css`: Configuran la estructura HTML y los estilos CSS para la interfaz.

Caso de Uso Principal: A continuación se presenta un caso de uso principal, que incluye la barra de búsqueda y la visualización de resultados. La barra de búsqueda permite a los usuarios ingresar sus consultas y se implementa en `search-bar.svelte`. Los resultados de la búsqueda se muestran utilizando el componente `results.svelte`, como se muestra en la Figura [4].

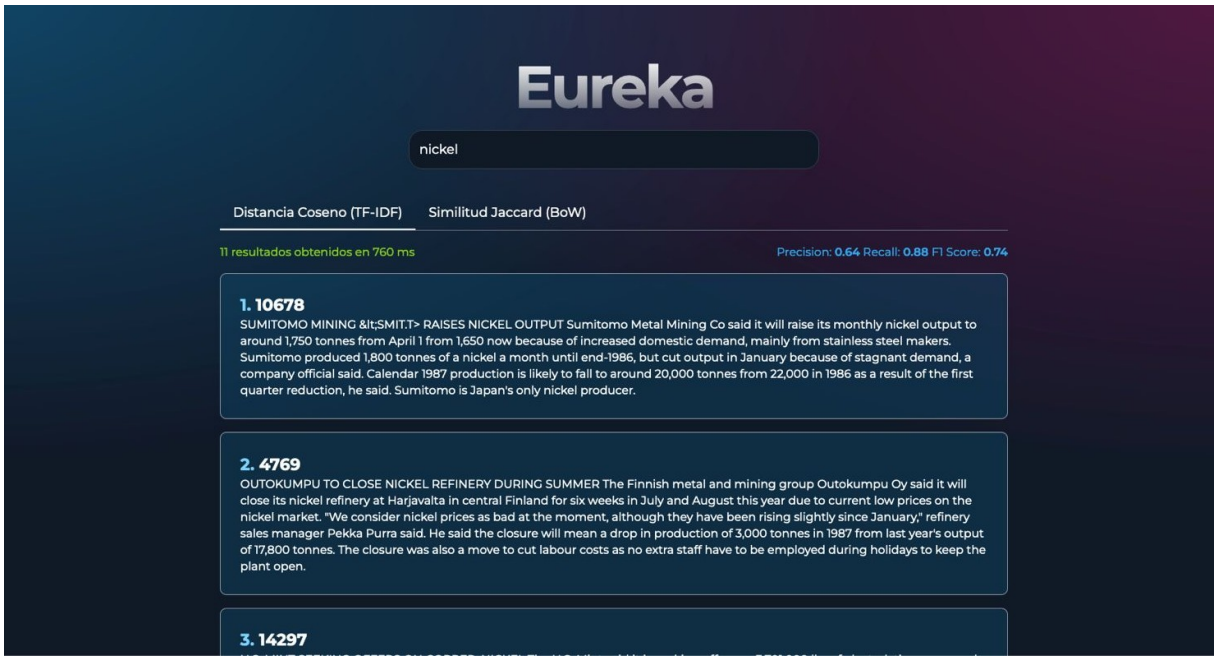

Figura 4: Implementación de la barra de búsqueda en search-bar.svelte

El código de la barra de búsqueda permite a los usuarios ingresar sus consultas y dispara la búsqueda al presionar .Enter". Los resultados de la búsqueda se filtran y se muestran en una lista ordenada.

La interfaz web incluye características adicionales como filtros de búsqueda y opciones de visualización para mejorar la experiencia del usuario y facilitar la navegación a través de los resultados de búsqueda.

El resultado es una interfaz web intuitiva y eficiente que permite a los usuarios interactuar fácilmente con el sistema de recuperación de información y obtener resultados relevantes de manera rápida y ordenada.

Referencias

[1] Diego Nicolas Baquero Parra, Fabricio Alejandro Moya Barragán y Jefferson Stalin Cando Santos. *Eureka: Un Motor de Búsqueda Innovador*. Disponible en: https://github.com/DYNECRON/eureka. 2024.

[2] David D. Lewis. *Reuters-21578 Text Categorization Collection*. Disponible en: https://archive.ics.uci.edu/dataset/137/reuters+21578+text+categorization+collection. 1997.